From Brittle to Robust:
Improving LLM Annotations for SE Optimization

Lohith Senthilkumar · **Tim Menzies**

Abstract Software analytics often builds from labeled data. Labeling can be slow, error prone, and expensive. When human expertise is scarce, SE researchers sometimes ask large language models (LLMs) for the missing labels.

While this has been successful in some domains, recent results show that LLM-based labeling has blind spots. Specifically, their labeling is not effective for higher dimensional multi-objective problems.

To address this task, we propose a novel LLM prompting strategy called SynthCore. When one opinion fails, SynthCore's combines multiple separated opinions generated by LLMs (with no knowledge of each others' answers) into an ensemble of few-shot learners. Simpler than other strategies (e.g. chain-of-thought, multi-agent-debate, etc) SynthCore aggregates results from multiple single prompt sessions (with no crossover between them).

SynthCore has been tested on 49 SE multi-objective optimization tasks, handling tasks as diverse as software project management, Makefile configuration, and hyperparameter optimization. SynthCore's ensemble found optimizations that are better than state-of-the-art alternative approaches (Gaussian Process Models, Tree of Parzen Estimators, active learners in both exploration and exploitation mode). Importantly, these optimizations were made using data labeled by LLMs, without any human opinions.

From these experiments, we conclude that ensembles of few shot learners can successfully annotate high dimensional multi-objective tasks. Further, we speculate that other successful few-shot prompting results could be quickly and easily enhanced using SynthCore's ensemble approach.

To support open science, all our data and scripts are available at `https://github.com/lohithsowmiyan/lazy-llm/tree/clusters`.

Keywords Active Learning · Large Language Models · Multi-Objective Optimization

All authors are from Computer Science, North Carolina State University Oval Dr, Raleigh, NC 27606

E-mail: panjal@ncsu.edu, timm@ieee.org

1 Introduction

Software Engineering (SE) data is often costly and labor-intensive to obtain, typically requiring careful curation by subject matter experts (SMEs) to ensure both accuracy and relevance [37]. Yet, as demand grows for large-scale, high-quality datasets across a range of SE tasks, researchers have increasingly turned to automation to reduce annotation costs [3]. Among these, the use of Large Language Models (LLMs) as automated oracles for labeling has emerged as a compelling option [63].

However, LLM-based annotation is not without challenges. In a recent empirical study presented at the MSR 2025 conference, Ahmed et al. [3] systematically evaluated LLM performance on five SE labeling tasks. Their findings reveal a nuanced picture: while LLMs can at times match human-level agreement on straightforward labeling tasks, they frequently fall short on higher dimensional or ambiguous problems. Key issues include hallucinated outputs, unjustified confidence in incorrect answers, and brittleness when applied to unfamiliar or out-of-distribution inputs. Crucially, Ahmed et al. warn that even when LLMs exhibit high inter-annotator or human-model agreement, this is not a reliable proxy for ground-truth correctness-particularly in domains that demand deep software-specific reasoning. They argue that LLMs should be used as conditional collaborators rather than drop-in replacements for human annotators:

> *"We do not claim that LLMs can universally replace human annotators; instead, our findings suggest that they may be viable complements—especially when* **used carefully** *and* **selectively.***.* [3]

`R3c` This paper reports a *careful* and *selective* use of LLMs for annotating examples used in *multi-objective optimization* tasks. By multi-objective optimization, we refer to problems where the goal is not to optimize a single output but to simultaneously balance several (often competing) objectives. This differs from the tasks studied by Ahmed et al., where the output space is typically single-dimensional. In contrast, our optimization setting requires annotations that guide trade-offs across multiple domain constraints. For instance, when evaluating software design options, practitioners may need to find alternatives that allow them to:

- deliver the *most* features,
- in the *least* time,
- at the *lowest* cost,
- with the *fewest* defects.

Other examples of multi-objective problems in SE include:

- *Green engineering* ensures faster responses that use less energy;
- and *Hyperparameter optimization* finds learner parameters that minimize false alarms and maximize recall;
- And any *Configuration task* such as selecting magic control variables within a Makefile.

Modern software urgently needs better tools for automatic configuration. We say this since all to often, software is deployed with suboptimal configurations [34]. For example, STORM's defaults yield $480\times$ performance degradation vs. optimal parameters [30]. Such poor performance is hardly surprising since industrial optimization faces major obstacles:

- Configuration spaces explode exponentially (in 7z: 14 parameters = one million configurations).
- Performance landscapes are rugged and sparse [10, 25, 37], creating local optimal traps.

Evaluating a wide range of different configurations can be very costly; e.g. x264's 11 parameters need $1,536$ hours to explore [61], limiting budgets to dozens of evaluations [9, 46]. *Active learning* can reduce that cost by evaluating only the most informative examples. But the success of any optimizer—including active learning—can depend heavily on the availability of reasonable initial examples. In active learning, the learner selectively queries or acquires labels, but its ability to do so effectively often hinges on a high-quality initial set of annotated samples. These initial samples, commonly referred to as "warm starts," serve as the starting point from which the learner begins exploring the space of candidate solutions.

Generating such warm starts has proven to be challenging. Recent work [54] shows that LLMs often struggle to produce high-quality initial examples, particularly in domains that are unfamiliar or underrepresented in their training corpora. We have previously argued that this limitation arises from insufficient training data coverage:

- LLMs excel when tasks are frequent, well-represented, and structurally consistent with patterns in their training data.
- But when confronted with rare, highly specialized, or high-dimensional optimization problems, their outputs tend to be shallow, generic, or systematically misinformed.
- For details of these findings, see Table 5, 6 7.

To overcome this limitation, this paper proposes a novel prompting strategy called SynthCore. Rather than relying on a single monolithic answer from the LLM, SynthCore prompts the model multiple times—each time under slightly different contexts or with randomized seeds—and then synthesizes the resulting outputs into a composite candidate set. Simpler than other strategies (e.g. chain-of-thought, multi-agent debate, etc) [17, 68, 70] SynthCore aggregates results from multiple single prompt sessions (with no crossover between them). `R2a-1` It also differs substantially from self-consistency prompting strategies that rely on a voting based mechanism to select the majority output while decoding. [2]. Moreover, prior work on seed generation [67] has shown that varying few-shot examples encourages language models to produce more diverse seeds—an insight that we regard as central to our method. By aggregating diverse perspectives from the LLM, SynthCore avoids the pitfalls of overconfident single-shot reasoning and improves the chances of capturing useful problem structure.

We evaluated SynthCore on 49 multi-objective SE optimization tasks spanning domains such as project planning, Makefile tuning, and hyperparameter selection. Across the board, SynthCore's ensemble-based strategy delivered superior performance compared to state-of-the-art techniques, including Gaussian Process Models, Trees of Parzen Estimators, and both exploitative and exploratory active learners. Notably, all labels used in SynthCore's experiments were generated solely by LLMs—no human annotations were included. These results demonstrate that,

> *While individual LLM responses may be unreliable,* **a carefully curated ensemble of few-shot learners** *can collectively overcome their shortcomings.*

Thus, we conclude that with the right orchestration, LLMs can move from brittle annotators to robust contributors in SE automation pipelines.

To introduce SynthCore, this paper explores two research questions

R3e

- **RQ1:** *While standard few-shot learning often struggles in SE Active Learning, does ensemble-based few-shot learning overcome these limitations and achieve better performance?* This is the core question of this research.
- **RQ2:** *Does the effectiveness of ensemble-based few-shot learners hold when dealing with high-dimensional SE data?* This work was motivated by a prior study that reported LLMs failed on higher dimensions. Therefore, in our results, we must pay particular attention to this kind of data.

As seen in the results of this paper:

- **RQ1** Yes, the results demonstrates the superiority of ensemble LLM Learners in SE multi-objective optimization tasks.
- **RQ2** Yes, the results shows that ensembles of few shot learners solve the problem reported in prior work.

The contributions in this paper include:

1. We propose an novel prompting tactic: ensembles of LLM few-shot learners.
2. We test the effectiveness of this approach for 49 SE datasets.
3. We compare this approach with alternative methods for 49 datasets.
4. We offer a reproduction package with all our data and scripts[1].

Based on this work, we speculate that other successful few-short prompting results could be quickly and easily enhanced using SynthCore's ensemble approach. This is not a proven contribution but the results of this paper suggest it would be a useful future research direction for prompt engineering and software analytics.

---

[1] `https://github.com/lohithsowmiyan/lazy-llm/tree/clusters`

1.1 Digression

`R0a` Before beginning, it is important to clearly define the scope of this work. This paper does not investigate which specific LLM architecture is optimal for these optimization tasks. While we have preliminary hypotheses on that matter, a comprehensive exploration of model-specific performance remains a critical direction for future research.

What this paper offers is an important finding regarding LLMs and optimization: contrary to recent pessimistic results [54], LLMs are highly effective for optimization problems, even with high-dimensional data. Our results demonstrate the definitive superiority of LLMs over long-standing, state-of-the-art symbolic methods for this task. This is an important contribution to the Software Engineering (SE) literature, which currently suffers from insufficient comparative evaluation of LLMs. For instance, a recent systematic review of 229 SE papers using LLMs found that only $13/229 \approx 5\%$ compared LLMs to other approaches [28]. It is important to explore this gap since the prevailing hype often obscures the fact that LLMs frequently do not outperform more established methods in many domains [23,27,31,38,57,59]. What we can offer here is good news since we can offer robust evidence that LLMs offer a significant, practical performance advantage over conventional methods in the domain of SE optimization.

2 Literature Review

2.1 Annotations in Software Engineering

Manual annotations are vital for data-driven Software Engineering (SE) tasks, influencing empirical findings [18]. Annotated datasets underpin research in defect prediction, vulnerability detection, sentiment analysis, and static analysis validation [39]. Redundancy, using multiple annotators, ensures reliability and mitigates bias [44]. Examples include 11 annotators for Jupyter Notebook inconsistency [48] and 3 annotators (from 11) for Java method similarity [32]. Manual labeling is costly. Generic automated tools often fail in SE due to specialized terminology [29], leading to the development of domain-specific models like SENTI-STRENGTH (78% accuracy, 85% recall) [29].

Annotation quality is crucial but faces challenges:

– **Heuristic Labeling Pitfalls**: Lexical patterns (e.g., "bug", "fix") for defect prediction yield inconsistent ground truth without rigorous validation [62].
– **Validation Discrepancies**: Manual errors are common; 90% of false positives in one technical debt analysis were labeling errors [69].

Manual annotation can be very slow.

– Unlike medical labs where automated agents label based on clearly defined definitions [33], SE annotation is far more subjective.

– Valerdi reports studies with panels of human experts in software effort
  estimation. Those panels could handle only 13.3 examples/hour [60].
– Lustosa reports studies with humans were complex management ranking
  peaked at only 15-20 judgments/hour [37].
– In prior work with industrial subject matter experts, we found expert avail-
  ability was very limited (rarely > 3 hours/week) [40]. Also, when we could
  negotiate for more access, those longer sessions decreased label quality [19]
  (due to cognitive fatigue).

2.2 Can LLMs Replace Manual Annotators in SE?

In theory, automated annotation methods address manual limitations [16].
Large Language Models (LLMs) are promising due to their pre-training on
vast corpora, which allows them to adapt to domain-specific contexts without
extensive local training. LLMs are used in code generation [58] and testing [13].

Ahmed et al. [3] explored if LLMs can replace manual annotators on 10 SE
tasks (e.g., code summarization, semantic similarity, static analysis labeling).
LLMs achieved **moderate to substantial agreement** with humans in **6
out of 10 tasks** (e.g., non-functional requirements) [3]. However, performance
declined significantly in context-intensive tasks (e.g., static analysis warnings),
showing low model-model agreement, suggesting LLMs cannot fully replace
humans, especially for nuanced tasks.

Senthilkumar et al. [54] reached similar conclusions about the limits of
LLMs for labeling. That study used LLM annotations to generate *warm-start*
samples for *active learning* in multi-objective SE tasks. They found LLMs
useful for low-dimensional probems, but not high-dimensional, multi-objective
optimization problems (this conclusion persisted even after applying feature
synthesis to artifically reduce dimensionality).

Senthilkumar et al. used the MOOT repository ([41], see Table 1), a col-
lection of SE multi-objective optimization tasks with x independent variables
(3 to 38) and y goals (1 to 5). Datasets were categorized by x-dimensionality:
Low ($|x| < 6$), *Medium* ($6 \leq |x| \leq 11$), and *High* ($|x| > 11$) [14]. They found:

– For **Low-dimensional tasks**: LLM warm-starts matched or surpassed
  symbolic traditional (Gaussian Process and TPE methods. [5]).
– For **Medium-dimensional problems**: same result;
– For **High-dimensional problems**: LLM warm-starts underperformed, of-
  ten falling below baselines [54].

They concluded that LLMs struggle to generalize to complex, high-dimensional
optimization, limiting their utility beyond trivial problems. That result, which
is very negative for LLM-based research, motivates us to explore alternate
methods (hence this paper).

Table 1 Data used in this study. As per Di Fiore et al. [14], data is labeled low, medium, or high dimensional based on the number of x independent goals. $|x| < 6$ means "low" ; $6 \leq |x| \leq 11$ means "medium" and $|x| > 11$ means "high". For more details on this data, see §3.1.

Groups	File Name	$|\text{rows}|$	$|x|$ / $|y|$	Dimsionality
Configuration	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
Process	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

2.3 Insight into why LLMs fail

LLM success depends on training data diversity. In SE, data can be sparse or biased, leading LLMs to inadequate initial hypotheses [4]. To compensate, *diversity of inference* uses techniques like prompt ensembles to explore a broader spectrum of candidate solutions during inference [65].

Given the last paragraph, it seems appropriate to explore *ensemble learning* [15] where conclusions are formed from multiple experts, each working slightly different versions of the data. Ensemble learning is know to boost robustness and predictive accuracy, especially in few-shot, data-scarce settings [15]. For example, ensembles (e.g., Random Forest [7]) outperform single models by resisting noise and generalizing better on high-dimensional data.

2.4 Prompting Strategies

R2a-2 A wide range of prompting strategies have been explored in recent literature [1,2,35,36,47,55,68]. These strategies vary depending on the nature of the problem space, but several have emerged as particularly influential, including *few-shot prompting*, *chain-of-thought prompting*, *self-consistency*, and *self-reflection* approaches. Our own variant of prompting is described in §3.4.

Few-shot prompting [1,35,47] is the standard practice of providing a small number of labeled examples (typically $n = 1$ to 4–5) within the prompt. This helps large language models generalize to previously unseen tasks. Many advanced prompting techniques can be viewed as extensions or modifications of the few-shot paradigm.

Chain-of-thought (CoT) prompting [36,68] extends few-shot prompting by including step-by-step explanations or reasoning traces along with each example. This encourages the model to follow explicit reasoning structures when solving unseen tasks, often improving interpretability and output quality. However, CoT has a notable drawback: if the provided reasoning path is incorrect or biased, the model may confidently follow and amplify flawed logic.

Self-consistency prompting [2] addresses the limitations of standard CoT. Instead of generating a single explanation–answer pair, the model generates multiple reasoning paths during decoding. These paths are then aggregated, typically via a voting mechanism, to select the most reliable final answer. This reduces sensitivity to any single, potentially incorrect chain of reasoning.

Self-reflection prompting [55] represents a distinct strategy in which the model iteratively refines its responses. In this setting, the model evaluates its previous outputs across a sequence of interactions (question → answer → critique → revised answer), enabling incremental improvement across iterations.

Our proposed *SynthCore* method may appear superficially similar to self-consistency prompting, but as shown in Table 2 there are important distinctions. While both methods utilize multiple inference paths, they are philosophically and mechanically distinct. **Self-Consistency** acts as a consensus engine, filtering out noise to find the most probable "average" truth   [2]. In

Feature	Self-Consistency [2]	SynthCore (this paper)
Core Logic	Consensus: Truth is the most frequent answer.	Evolution: Optimal solutions are rare outliers (mutations).
Driver	Decoding Sampling (Stochastic paths).	Input Variation (Diverse seeds/contexts).
Goal	Minimize variance (Reliability).	Maximize variance (Exploration).
Selection	Majority Vote (Mean/Mode).	Sorting/Ranking (Best of).
Analogy	A jury voting to agree on a verdict.	Biological mutations driving evolution.

Table 2 Comparison of prompting strategies.

contrast, **SynthCore** acts as an evolutionary search engine, actively generating noise (mutations) to discover superior "outlier" solutions.

Also, the two methods take different approaches to variation:

- **Self-Consistency (Consensus):** Views variation as uncertainty to be minimized. It assumes correct reasoning paths converge [2] . It relies on a voting mechanism to discard outliers and select the majority answer.
- **SynthCore (Evolution):** Views variation as a feature to be exploited. Drawing on seed generation theory [67], it uses varied few-shot examples to force diverse outputs. These are treated as "strong mutations" in a search process, designed to escape local optima and find better warm-starts for active learners.

Also there are differences in how the mechanism, specifically decoding vs. input context. This is to say that the methods intervene at different stages of the generation pipeline.

- **Self-Consistency:** Keeps the input constant. It generates diversity solely by sampling multiple reasoning paths during the decoding stage [65].
- **SynthCore:** Varies the input context. It executes multiple independent sessions, each with unique randomized seeds or shuffled examples [2]. This prevents the model from fixating on a single context window's logic.

Finally, the final answer is generated in a different way:

- **Self-Consistency:** Aggregates via **Voting**. The goal is to find the mode of the distribution (the most reliable answer) [2].
- **SynthCore:** Aggregates via **Synthesis**. The goal is to collect all valid samples and sort them by an external objective function (e.g., Chebyshev distance) to find the single best candidate.

While self-consistency focuses on improving the *consistency* of model outputs, our ensemble method emphasizes *diversity*. The goal is to generate a diverse set of high-quality candidate outputs, which can be interpreted as producing strong "mutations" in a search process to move toward near-optimal solutions.

Table 3 Introducing the MOOT Repository. Available on line at `http://tiny.cc/moot`.

MOOT (Multi-Objective Optimization Testing) is a collection of software engineering datasets drawn from tasks such as process tuning, database configuration, and hyperparameter optimization for, e.g. defect prediction models.
Table 3 shows the general form of MOOT data.

```
            x = independent values      | y = dependent values
            ------------------------------|-----------------------
            Spout_wait, Spliters, Counters, | Throughput+, Latency-
                10,        6,        17,  |    23075,    158.68
                 8,        6,        17,  |    22887,    172.74
                 9,        6,        17,  |    22799,    156.83
                 9,        3,        17,  |    22430,    160.14
               ...,      ...,       ...,  |      ...,    ...
             10000,        1,        10,  |   460.81,    8761.6
             10000,        1,        18,  |   402.53,    8797.5
             10000,        1,        12,  |   365.07,    9098.9
             10000,        1,         1,  |   310.06,    9421
```

These tabular data sets are are divided into independent (x) inputs and dependent (y) goals. The first row names each column. Numeric column names start with an uppercase letter. Goal names ending in + or - are to be maximized or minimized, respectively. For example, in the above table, a system configures *Spout_wait, Spliters, Counters* to maximize *Throughput* and minimize *Latency*.
For illustration purposes, rows are sorted from best to worst based on those goals. But note that in our experiments, rows are randomized and goal values (y) are initially hidden.

3 Methods

Drawing inspiration from these established and emerging ensemble methodologies, the rest of this paper investigates the application of ensembles of few-shot prompts for the synthesis of optimal candidate solutions within active learning frameworks.

This section describes the data, algorithms, performance measures used in this study.

3.1 Data

This study explored dozens of software engineering (SE) multi-objective optimization tasks from the MOOT repository [41] shown in Table 1 (with more details in Table 3)[2]. In MOOT, data sets have:

- 1 to 5 numeric optimization y goals;
- 3 to 38 independent x variables;
- 90 to 90,000 data rows.

Following a recommendation from Di Fiore et al. [14], the MOOT datasets are categorized based on their input (x) dimensionality:

- *Low:* 12 datasets with $|x| < 6$ independent variables;

---

[2] MOOT = *Multi-Objective Optimization Tasks*, a collection of recent SE benchmark problems [41]. `http://tiny.cc/moot`.

  – *Medium:* 14 datasets with $6 \le |x| \le 11$;
  – *High:* 19 datasets with $|x| > 11$

(Aside: We acknowledge that "high-dimensional" is defined differently in other domains. For example, text mining may be considered low-dimensional relative to image processing. In defense of our categorization, we note that different studies report that active learning algorithms exhibit very different performance across data sets divided in this way [14, 54].)

This MOOT data divides into four groups:

1. The *Config* directory consists of datasets derived from software engineering literature, specifically those labeled with "SS-*" [45]. These datasets comprehensively capture configurations for various tasks, such as video encoding, with primary objectives including query time and run time. Additionally, Config includes datasets related to database configurations, such as Apache_AllMeasurements.csv, SQL_ALLMeasurements.csv, and X264_AllMeasurements.csv.

2. The *HPO* directory houses datasets from the hyperparameter optimization domain [37]. These datasets document the outcomes of random forest regression models trained to forecast metrics such as commits, closed issues, and closed pull requests over a 12-month period for open-source projects hosted on GitHub. The Y-values in these datasets represent the error and accuracy of the model under different hyperparameter settings.

3. The *Process* directory contains datasets originating from software process modeling research [26,42,43,49]. The "pom*" datasets capture insights into agile development as studied by [6]. Specifically, the POM3 model represents requirements as a tree of dependencies that emerge dynamically, akin to elements surfacing from water. This model tracks completion rates, idle times, and development effort as teams navigate evolving tasks. Additionally, the "nasdem" dataset provides real-world data, while "osp2" follows the USC Cocomo model format, offering predictions for development effort, defect rates, and risks in waterfall-style software projects [42].

4. The *Misc* directory includes non-software engineering datasets, such as auto93 and WineQuality, which serve as demonstration tools for presenting MOOT to a broader audience.

We provide other meta data on the datasets externally. [3]

3.2 Performance Measure

Our active learners returned a row of options. To measure the effectiveness of that row, we use the *Chebyshev Distance*. This is the maximum distance between any y value of a row to its ideal y value in the dataset.

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

---

[3] R4b , R4c Details of the parameters for every dataset are provided here: https://github.com/lohithsowmiyan/lazy-llm/blob/clusters/docs/datasets.pdf

For this calculation, we normalized the y values for each goal to 0..1 min..max. For the data set shown in Table 3, those ideal y values are associated with maximal *Throughput* and minimal *Latency*; i.e. their ideal y values are:

$$ideal\ \{Throughput, Latency\} = \{1, 0\}$$

We use Chebyshev since:

– It is also used by other prominent multi-objective algorithms [71];
– It is a "cruel critique" that punishes failure for any optimization goal;
– We have run all our results with other evaluation measures (e.g. average distance of row goals to the ideal) and our main result (that SynthCore works for higher dimensional data) still persists.

3.3 Active Learners

Given n initially labeled examples, active learning proceeds as follows [8, 37].:

1. Acquire the next most informative example based on the current model.
2. Label this example and add it to the training set.
3. Update the predictive model.
4. Repeat until the labeling budget (B) is exhausted.
5. Return the **best** example (where "best" is defined as per Equation 1).

Due to their prominence in the literature, we focus on two active learners:

3.3.1 Gaussian Process Models (GPM)

GPMs generate estimates by passing the available data through a range of kernels. In this way, they can generate a mean (μ) and standard deviation (σ) for each prediction [8, 66]. The Upper Confidence Bound (UCB) acquisition function cab use μ, σ to guide the selection of the next example to label. UCB recommends labeling the example x that maximizes:

$$UCB(x) = \arg\max_x(\mu(x) + \kappa\sigma(x)) \tag{2}$$

(where κ is a constant). Early in the reasoning, when little is known, the variances are large and the $\kappa\sigma(x)$ term dominates. Later, as more data reduces the variance, UCB converges to just $\mu(x)$. In this way, UCB adapts from *exploring* regions of large variance to *exploiting* regions with best mean prediction.

3.3.2 Tree-structured Parzen Estimator (TPE)

The Tree-structured Parzen Estimator (TPE) differs from Gaussian Process methods by separately modeling the conditional distributions $p(x|y)$ and the marginal distribution $p(y)$ [5]. TPE splits the data based on objective values

into two distinct groups, defined by a threshold y^*. Specifically, the conditional probability is represented as:

$$p(x|y) = \begin{cases} l(x), & \text{if } y < y^* \\ g(x), & \text{if } y \geq y^* \end{cases} \tag{3}$$

Here, $l(x)$ represents the distribution of high-performing observations, whereas $g(x)$ corresponds to the lower-performing observations. The threshold y^* is determined using a quantile γ, ensuring $p(y < y^*) = \gamma$. The primary goal of TPE is to select configurations that maximize the probability under $l(x)$ and minimize it under $g(x)$, optimizing:

$$\arg\min_{x} \frac{g(x)}{l(x)} \tag{4}$$

3.3.3 Model Selection

R2c In this study, our objective is to compare Synthcore (an LLM-based synthesis method) against symbolic baselines (UCB, TPE, etc), rather than to benchmark different LLMs against one another. Accordingly, the goal is not to identify which LLM performs best, but to ensure that the LLM used in the evaluation is sufficiently capable to provide a fair and meaningful comparison.

Hence we select Gemini 1.5 Pro, a state-of-the-art model at the time of the study, whose large context window (1M tokens) and strong factual reasoning abilities enable it to handle the rich metadata, multi-example prompts, and high-dimensional structures required for our task.

(Aside: In the prior experiments from [54], both gpt-o1 and gemini-1.5-pro consistently produced similar results for all categories of the datasets (low, medium, & high dimensional). Therefore, rather than focusing more on the performance of different models, we focus on comparing LLMs collectively versus other baseline methods.)

For completeness, we also looked into other models, but Gemma-3 and Llama3.1 have two orders of magnitude fewer variables than Gemini 1.5 Pro, and they performed poorly on most of the datasets[4]. Also , DeepSeek-R1 has achieved much recent publicity, but we found we could not run it locally. Furthermore, the quotes we received for on-line experimentation of that model were prohibitively expensive[5].

---

[4] This study was conducted before the release of Gemini 2.5 on March 21, 2025.

[5] Our experiments require 20 repeats of a 10-way ensemble for 49 data sets. Total quotes we received from that kind of inference ranged from \$5,000 to \$20,000. Our lab supports a dozen graduate students, each trying to write, at most, 3 papers per year. Supporting this kind of inference would hence cost up to $(3 * 12 * 20,000) = \$720,000$, annually.

Table 4 Prompt template for few shot learning, here the two variables are `meta` which is a table containing the meta data of the dataset (name of column, min and max values, mean or mode, standard deviation or entropy). `table` which are all the rows of that dataset.

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:
Generate an examples that is Better:
This should outperform the given "Best" examples by optimizing the relevant
features to better 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:

3.4 Prompting Strategies with LLMs

In an attempt to emulate human reasoning, we used LLMs to generate the warm starts needed for our active learners. These warm starts are created using the N-shot prompt tactic of Table 4. In summary:

- The initial *System Message* introduces the LLM to attribute names of a data set.
- The subsequent *Human Message* then shows the LLM N examples, at random, and divide them into half "best" and "rest" (using Equation 1).
- The *Task* prompt asks the LLM to synthesize an example better than all the 'best" and "rest" seen so far,
- Label and return the row nearest this better synthetic example.

We call this an N-shot learners where N is the number of labelled examples given as part of the prompt . For example, if the *Human Message* shows $N = 6$ labelled rows, then we would call that a 6-shot learner.

SynthCore is a

$$L + M * N$$

shot learner where M is the number of repeated trials of our $N + 1$-shot learners. Unlike Chain of Thought or MAD (Multi-Agent Debates [56]), each M trial has no knowledge of the results of the other trials. Rather:

- After labeling L randomly selected rows,
- Repeat M times
  - Reset the context window of the LLM.

- − Apply the N-shot learning of Table 4. In that process, the LLM is given all the column names and all the x values of the unlabeled rows.
- − All the $L + M * N$ labels generated by the this loop are then sorted via Equation 1.
- − The best example in that sort is then returned.

We recommend this approach for two reasons:

1. Basic N-shot learner fails for higher-dimensional optimization problems. On the other hand, as shown in our *Results* section, SynthCore's $L + M * N$ approach performs very well,
2. SynthCore is very simple to implement: it is just a wrapper around the established prompt strategy of Table 4.

Further to the second point, since SynthCore is so simple, it is worth asking if it merits publication at venues like this journal.

The software engineering and AI literature has many examples where something can be very simple, yet still be very significant[6] [7]. While a research *result* might be quite simple, the research *effort* to achieve that result can be considerable. SynthCore's current simplicity is the result of months of systematic exploration and overcoming obstacles:

- − Our initial results from Table 4 where obtained using $N = 4$ examples. Progressively increasing N did not yield the desired performance gain. Furthermore, since the LLM had to reflect on all rows at each step of its *Task*, this approach became excessively slow.
- − Chain-of-thought (CoT) prompting was investigated, which typically requires the LLM to generate step-by-step reasoning. However, developing an explanation module that allowed the LLM to offer genuinely insightful reflections on our specific tasks proved to be challenging. This experience aligns with findings from other researchers who note that tables can be particularly difficult for models primarily trained on textual sequences, and that complex reasoning over them remains a hurdle even with techniques like normalization [50].
- − We also explored Multi-Agent Debates (MAD) [56], but the computational costs associated with running multiple agents were prohibitive.
  - − Our experiments were applied to 49 data sets.

---

[6] Code reviews is "just" asking that at least one other developer examines and approves changes before they are merged. Rigby and Bird [51] demonstrated that this simple addition to a DevOps pipeline yields substantial benefits to organizational knowledge, increasing programmers' understanding of the broader system by 60% to 150%. Furthermore, in the projects they studied, this change transformed code reviews from a defect-finding task into a collaborative problem-solving activity.

[7] Boosting is "just" resampling technique where learner $i + 1$ places increased focus on examples misclassified by learner i [22]. Its co-inventor, Yoav Freund, famously remarked: *Boosting is the best 20-line shell script ever written*. The AdaBoost script, while very short, catalyzed a revolution in machine learning. With minimal logic, AdaBoost demonstrated how to transform weak learners into strong ones, laying the groundwork for modern ensemble methods such as XGBoost and LightGBM.

- In each experiment, $M * N$ times, an LLM has to reflect on the x-values on the unlabeled rows. As shown in Figure 1, the MOOT problems can range up to 90,000 rows.
- To ensure statistical validity and account for the non-deterministic behavior of the LLM, all our experiments were repeated 20 times using different random seeds.

In the end what lead us to SynthCore was the observation was that through all the above, our methods did sometimes stumble on the optimal solution[8]. This observation called to mind all the research on ensemble methods which lead us to try SynthCore's $L + M * N$ ensemble approach.

Before continuing, we make one technical point. After labeling N items, the N-shot learner has to label one more item (the best example seen so far). Accordingly, the total number of labels required for these methods is $L + M * (N + 1)$.

3.5 Experimental Rig

R4d Our choices of the initial annotation budget L and the later sampling budget M follow directly from the Near Enough Optimization (NEO) framework. NEO adopts the view that (a) "near enough is good enough," and (b) solutions within a small effect-size ε of the optimum are indistinguishable. Under this assumption, sampling is not required to locate the global optimum, but only to encounter *any* solution lying within an ε-neighbourhood of it.

If solutions are randomly shuffled in the dataset (Assumption 1) and values within ε are indistinguishable (Assumption 2), then the probability of drawing an ε-optimal solution in a single trial is proportional to ε. The waiting time to the first success can therefore be modelled as a geometric process [52]. After n draws, the probability of having observed at least one success is:

$$1 - (1 - \varepsilon)^n \geq C.$$

Solving for n yields:

$$n(C, \varepsilon) = \frac{\log(1 - C)}{\log(1 - \varepsilon)}.$$

Cohen's rule [12] is that results are statistically distinguishable by more than a small effect if they different by half a standard deviation. Assuming a normal distribution with the span ± 3 then $\varepsilon = 0.5/6$. Hence, to be 95% sure that we have found solutions that are indistinguishable from the best, we need the following number of samples:

$$n(C = 0.95) \approx 60$$

---

[8] Given data like Table 1, a solution is "optimal" if it is the row closest to the ideal point, as judged by Equation 1.

The NEO analysis indicates that ≈ 60 samples are sufficient to reach an ε-near-optimal region. However, once the initial L samples anchor the search, the remaining adaptive rounds need not individually reach the full theoretical requirement. Instead, they serve as incremental refinement steps.

We therefore select $M = 20$ & $M = 30$ as a deliberate engineering compromise:

 - $M = 10$ was too small to provide meaningful refinement in most datasets.
 - $M = 40$ offered marginal improvements while increasing labeling costs.
 - $M = 20$ & $M = 30$ consistently achieved the best trade-off between performance and annotation expense, whereas $M = 20$ performed better for low-dimensionality problems, while $M = 30$ performed better in medium and high-dimensionality problems.

Using the nomenclature from the previous section, we say:

 - $B = L + M * (N + 1)$
 - $L = 0.6B$
 - $M = 20$ i.e., perform 20 repeats to test whether our results hold across a range of randomly selected initial conditions.

We ran all our active learners with budgets in the range of 20..100 for 20 repetitions to obtain the statistics for the Scott-Knott test. Budgets up to 100 were chosen since, as seen below, no performance improvements were observed over 50 samples.

3.6 Statistical Methods (Scott-Knott)

We run Scott-Knott tests for all the datasets across different treatments, summarize the results in the above subsections, and report the percentage of times a treatment (e.g., "SynthCore") appears in rank $|n|$.

Results were ranked using a combination of the Scott-Knott, bootstrap, and Cliff's Delta procedure [53]. Scott-Knott recursively partitions sorted treatment means (derived from their performance metric, e.g., Chebyshev distances from the repeats) into statistically distinct, non-overlapping groups [24]. Each partition maximizes the between-group sum of squares:

$$SS_B = n_1(\bar{x}_1 - \bar{x})^2 + n_2(\bar{x}_2 - \bar{x})^2 \tag{5}$$

where n_1, n_2 are sub-group sizes with means $\bar{x}_1, \bar{x}_2$, and $\bar{x}$ is the combined group mean. A key advantage of the Scott-Knott method is that it creates statistically distinct, non-overlapping groups of means. This simplifies interpretation, as there is no ambiguity about which group has what treatment.

Scott-Knott recursively splits treatments if a significance test and an effect size test report that groups are (a) statistically distinguishable and (b) different by more than a small effect. Cliff's Delta (δ) [11] is a non-parametric effect size that quantifies the magnitude of the difference between treatment performance distributions. It measures the probability of one distribution being stochastically greater than another [11].
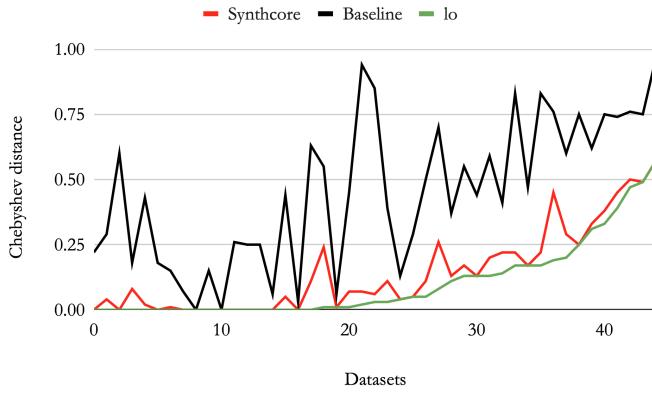
Fig. 1 Performance of Synthcore 20, (shown in red) fall very close to the best possible optimal (shown in green).

Bootstrap resampling [20] assesses the statistical significance of the observed differences. This non-parametric technique creates empirical sampling distributions by resampling with replacement from each treatment's 20 performance values. It tests if the observed deviations from random variation are significant, which is valuable for hypothesis testing and model validation, particularly with limited or non-normally distributed data [21].

4 Results

Before conducting comparative evaluations of different methods, it is prudent to first document the baseline performance of our proposed approach. Why? Reporting only relative results, without first establishing an absolute baseline, undermines the significance of the results and should be avoided. All too often, we read research papers where authors overlook this step and then report seemingly impressive improvements—such as a X% gain— which is misleading when the baseline itself is unacceptably low.

Figure 1 shows mean results from twenty runs of Synthcore (budget $B = 20$ labels, LLM-generated warm start):

 – The green curve sorts data by their best row that is closest to "heaven" (see Equation 1).
 – The black curve represents the untreated "before" condition of our data. This curve shows the average distance to heaven if all rows. If we just picked random rows, we would usually achieve the results shown in black.
 – The red curve shows the rows found by Synthcore.

By our design, every optimization falls between the black (mean) and green (minimum) lines. Hence, the closer the red curve falls to the green curve, the better the performance. The red curve's proximity to green shows that Synthcore delivers strong results.
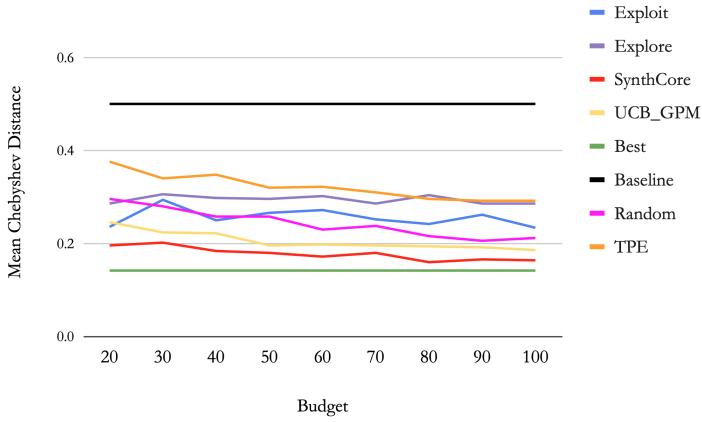
Fig. 2 Performance of different active learners w.r.to budgets. Best is the average heaven values for all the datasets
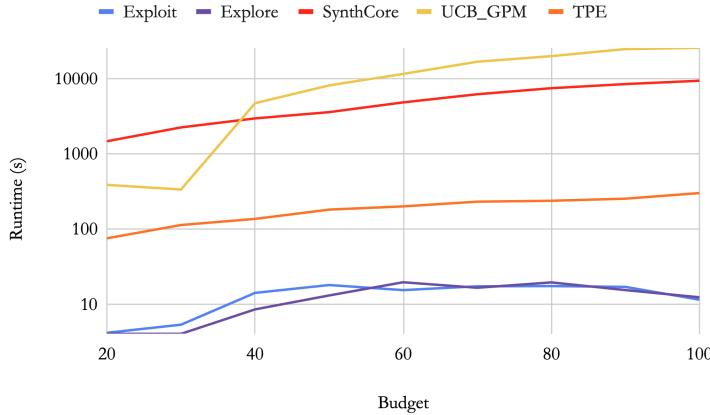

Fig. 3 Average Run times of different treatments, y axis is time in seconds scaled to logarithmic values and x axis is budget ranging from 20 to 100

(Aside: not shown in Figure 1 are statistical tests that rank Synthcore's performance against other state-of-the-art methods. For those statistical comparisons, see below.)

Figure 1 showed Synthcore's benefits. Those benefits came at some cost:

- The cost of collecting labels;
- The CPU runtime cost.

Recall that the Figure 1 achieved at a cost of just a few labels ($B = 20$). Using Figure 2 and Figure 3 we can now justify that budget:

- Figure 2 shows mean distance to heaven across all data sets in twenty repeated trials of our different optimizers. The bottom curve shows the mean distance of heaven of the best row in each data set. The x-axis shows

how performance changes as the budget increases. The statistical tests of §3.6 shows no statistical different between the $B = 20, 30, 50, 100$ results.

– Figure 3 shows the runtimes associated with budget sizes associated with varying budgets. Comparing budgets of 20 and 100, we note that Synthcore is much slower for larger budgets; e.g. at $B = 1000$ it is five times slower than at $B = 20$.

In summary, the (a) runtimes are faster at smaller budgets; and (b) there is no statistically significant optimization improvement at larger budgets. Accordingly, the rest of out results will be reported using $B = 20$.

Before continuing, we answer one frequently asked question: Why the orders of difference in run times between the lower and upper curved of Figure 3? The two lower curves from simple Bayesian methods that can update their models for every row, in linear time. The upper two curves come from methods that are far more reflective. Specifically, the upper methods made decision by (a) reflecting over a large space of possible models (UCB_GPM), or (b) reflecting over a large context window that struggled to pay attention to hundreds to thousands of rows (SythnCore). Hence it is hardly surprising that that more reflective methods (UCB_GPM and SynthCore) are far slower than the other methods.

4.1 Answers to Research Questions

Have documented the absolute performance of SynthCore, we now move to statistical comparisons between different methods.

These comparisons are shown in Tables 5, 6, 7 and Tables 8, 9, 10 Each cell in this table shows the percentage of times (in 20 repeats) that a particular treatment was ranked best (i.e. rank=0), or some other rank, by the statistical methods of §3.6. The most important part of these tables are the rank=0 results. For example, top left of Table 5, we see that 100% of the time, an LLM method achieved a top rank.

The red results in Tables 5, 6, 7 come from prior work by Senthilkumar et al. [54] that ran the few-shot procedure of Table 4, only once (no ensembles). Here we see the negative prior result that inspired this study:

– Tables 5, 6 that LLM warm start method did as well, or better than anything else.
– But in Table 7, LLMs were only half as good as the best method (UCB_GPM).

The blue results in Tables 8, 9, 10 show the performance improvements achieved by SynthCore's ensemble methods. Across all our data, SynthCore performed better than everything else. Hence we say:

RQ1: *Is ensemble few shot learning better than standard few shot learning for SE Active Learning?* Answer: **Yes**.

Column 1
Prior results, from [54].
(no ensembles).

Column 2
Our results
(with few-shot ensembles).

	Scott-Knott Rankings						
	0	1	2	3	4	5	6
LLM Warms	100						
Exploit	73	18	9				
UCB_GPM	55	27		9			
TPE	27	27		9	18	9	
Explore		9	18	18	27		
random	36		9	9	9	9	
Baseline						9	18

Table 5 Low dimensional ($x < 6$).

	Scott-Knott Rankings						
	0	1	2	3	4	5	6
SynthCore	86	7					
Exploit	71	7	7	7	7		
UCB_GPM	43	29	7	7	14		
TPE	7	29	7	14	14		
Explore	7	7	29	36	14	7	7
random	7	36	29	14	7	7	
Baseline					9	21	

Table 8 Low dimensional ($x < 6$).

	Scott-Knott Rankings						
	0	1	2	3	4	5	6
UCB_GPM	64	18	9	9			9
LLM Warms	64	9	18				
Exploit	45	18	27	18	10		
TPE	36	18	18	18		9	
random	27	18	27	18			
Explore	27	9	18	20		10	10
Baseline	9	9	9				18

Table 6 Medium dim. ($5 < x < 11$).

	Scott-Knott Rankings						
	0	1	2	3	4	5	6
SynthCore	85			8			
UCB_GPM	77	8	8				
Exploit	69		8	8	8		
TPE	39	8	8	15	15		
random	31	15	31	15			
Explore	9	9	41	23		8	8
Baseline	8		15	8	15	15	8

Table 9 Medium dim. ($5 < x < 11$).

	Scott-Knott Rankings						
	0	1	2	3	4	5	6
UCB_GPM	100						
LLM Warms	53	13	7	13			
Exploit	47	13	13	13			
Explore	7		13	20	20		
TPE	40	13	7	13	7	7	
random		27	27	20	13	13	
Baseline			7	13		9	27

Table 7 High dimensional ($x > 10$).

	Scott-Knott Rankings						
	0	1	2	3	4	5	6
SynthCore	71	5	11	8			
UCB_GPM	68	11	11			5	
Explore	37	11	21	5	16		
TPE	32	16	16	5	5	5	
Exploit	16	16	32	11	5	5	5
random		21	21	16	11	11	
Baseline			5	11	11	26	5

Table 10 High dimensional ($x > 10$).

Moving on to our second research question, this work was motivated by a prior study that reported LLMs failed on higher-dimensional. Therefore, in our results, we must pay particular attention to this kind of data.

RQ2: *Does this improvement hold for high dimensional data?* Comparing Table 7 and Table 10, we see, for high-dimensional, ensembles thrive where solo-based reasoning fails. Hence for this question we answer **Yes**.

To say all this another way, for this kind of optimization, LLMs need to be encouraged to divulge their knowledge. A single prompt is not enough, nor is simple few shot learning. Rather, it is needed to run multiple sessions with an LLM, then aggregate the results.

5 Discussion

SynthCore demonstrates that large language models, though unreliable as single annotators, can become powerful contributors when treated as ensembles of weak, diverse learners. While past studies have shown that few-shot prompts struggle in high-dimensional SE tasks, our results indicate that diversity in prompting can significantly mitigate this limitation. Across 49 datasets, SynthCore achieved consistent gains over traditional optimization baselines and even surpassed adaptive Bayesian learners in many cases.

One reason SynthCore works is that ensemble prompting reduces overfitting to prompt phrasing or context. Each prompt acts as a sample from the LLM's latent capability space, and their aggregation offers resilience against individual prompt failures. This is especially valuable in SE optimization, where initial warm-starts are crucial to guiding the search process. Rather than placing all trust in a single LLM prediction, SynthCore leverages variation to increase coverage of plausible solutions early in the learning loop.

However, the benefit of ensembling is not uniform across all tasks. Our experiments show that ensemble gains are greatest in moderate-dimensional spaces with clear trade-offs, such as project planning or Makefile tuning. In simpler tasks, single-shot learners often suffice. Few-shot prompting typically shows the highest returns when moving from 0 to 1 shot, with diminishing returns after 4 shots. Ensembles counteract this drop-off, particularly in higher-dimensional settings.

Crucially, SynthCore avoids complex multi-agent orchestration. Unlike chain-of-thought prompting or debate-style sampling, it does not rely on model introspection or self-consistency checking. Instead, it uses independent prompts with no cross-talk. This simplicity makes SynthCore faster to run, easier to adapt to new domains, and more robust to failures in reasoning depth or hallucinated explanations. In practice, this makes SynthCore more applicable in real-world SE pipelines where latency and predictability matter.

`R3d` For practitioners, we recommend using SynthCore following the workflow shown in Figure 4. Our results indicate that SynthCore offers the largest performance gains in complex, high-dimensional SE optimization tasks where human heuristics are difficult to encode. For SE optimization problems, we
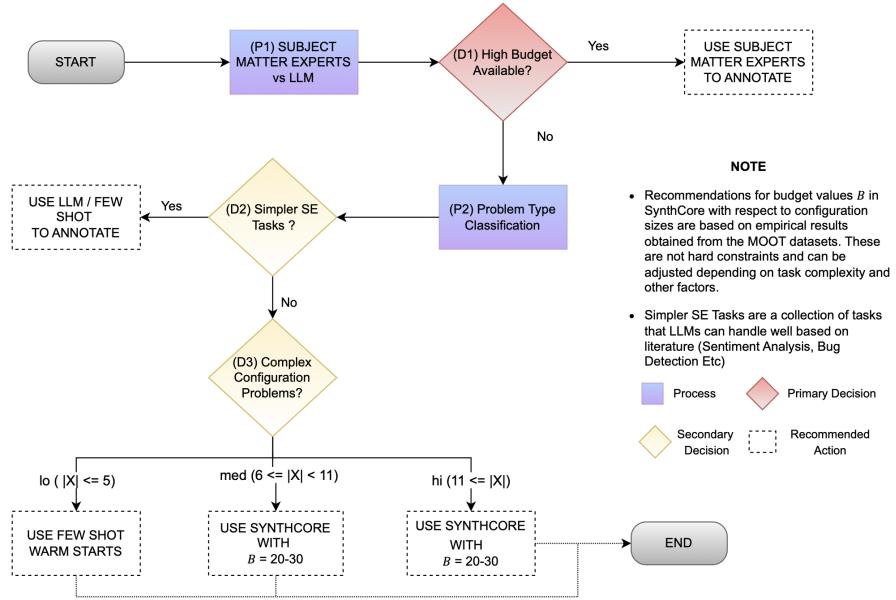
Fig. 4 R3d-2 Recommendations to practitioners.

advise scaling the evaluation budget with the complexity of the data—using the number of independent variables as a practical indicator—and increasing SynthCore's budget accordingly. However, SynthCore is not universally optimal. When strong SME expertise is readily available, relying on a human SME may be more cost-effective than invoking LLM ensembles. Likewise, for highly structured and common tasks such as sentiment analysis or bug detection, standard LLM prompting already performs well, and the additional API cost and runtime of ensemble prompting may not be justified.

SynthCore is also efficient. Compared to multi-stage LLM pipelines that require dialog, argument tracking, or self-refinement, SynthCore relies on multiple single-prompt calls—each simple, stateless, and fast. This architecture is well-suited to deployment within CI/CD workflows or as part of configuration tuning tools. Our results show that ensembles of size 3–5 provide most of the performance gains, making this approach viable even under strict budget constraints.

Beyond performance, SynthCore also surfaces a deeper point about LLM limitations. Prior work (e.g., Ahmed et al. at MSR 2025) has shown that LLM agreement with human annotators is not always a good proxy for truth. Our work shifts the question: rather than ask whether an LLM is "correct," we ask whether a group of diverse LLM prompts can generate a fertile search

space. Seen in this way, LLMs are not oracles—they are generative seeds for optimization.

R3g More generally, we say that SynthCore functions as a subject matter expert amplifier, not a replacement. Its primary role in the pipeline is to compress the human expert's review surface. Instead of requiring an SME to laboriously generate and evaluate a large set of candidates—many of which are suboptimal—SynthCore filters and surfaces a small, high-quality candidate set for validation. This fundamentally shifts the SME's task from slow, exhaustive *generation* to rapid, high-leverage *validation*.

This transformation provides concrete benefits for efficiency and iteration speed. Without SynthCore, an SME might label over 100 examples to identify the 5 truly optimal configurations. With SynthCore, the SME's effort is focused on reviewing approximately 20 top candidates, validating, or making minor adjustments. This reduces the SME's cognitive load significantly, as evaluating five promising designs is an intrinsically different task from shifting through fifty unknowns. Furthermore, human judgment is thus applied precisely where it is most needed: assessing ambiguous trade-offs, applying domain constraints the LLM may not possess, and integrating specific organizational context.

Regarding the need for human SME input, SynthCore reduces the required volume of input for a given quality threshold. It facilitates the democratization of the initial design phase. Junior engineers, leveraging SynthCore, can rapidly produce reasonable first drafts and solution proposals that previously required the intervention of a senior SME. Senior SMEs can then dedicate their limited, high-value time not to initial exploration, but to final refinement and tackling complex, boundary-pushing engineering challenges.

R3d-b However, we do not claim SynthCore to be a universal solution for annotation in multi-objective optimization problems. While text and code are indeed high-dimensional modalities, modern LLMs handle them remarkably well due to their training objectives and architectural design. This is also reflected in the tasks explored by Ahmed et al., which were predominantly text- and code-centric (e.g., sentiment classification), where LLMs naturally excel. In contrast, our focus is on tabular multi-objective optimization problems, a setting where LLMs typically struggle. Tabular data lacks the sequential structure and semantic regularities that LLMs exploit in language or code, making annotation and performance estimation significantly more challenging. For this reason, SynthCore is specifically designed for such structured, non-textual domains.

At the same time, we acknowledge that SynthCore may not generalize well to multimodal, ultra–high-dimensional domains such as audio or images, which fall outside the native capabilities of current LLMs. Addressing these more complex modalities would likely require future extensions that incorporate multimodal foundation models [64] capable of reasoning jointly over text, vision, and other signals.

5.1 Future Work

We plan to explore transferability across iterations. Currently, SynthCore treats each ensemble as independent, but a sequential execution model—where insights from early prompts inform later ones—could reduce sampling cost while improving guidance. This strategy may enable higher efficiency with fewer prompt calls.

A natural next application is Next-Release Planning (NACP), where trade-offs among value, effort, and delivery constraints align with SynthCore's strengths in navigating sparse, multi-objective spaces. Here, ensembles could support decisions under uncertainty with little labeled data.

Another direction is extending SynthCore to structured domains such as patch generation, bug triage, or Makefile tuning. These require prompts that account for semantics and ordering. Ensemble diversity may help span different reasoning paths over these structures.

We also intend to benchmark SynthCore against expert-curated annotations—not just to evaluate predictive accuracy, but to assess alignment with human judgment and trust. This could clarify the role of LLMs as surrogates or collaborators.

Explainability remains a key open challenge. While SynthCore improves sampling, it does not make LLM reasoning more transparent. Post-hoc interpretation of prompt choices or traceable rationales could make the system more suitable for regulated or high-stakes domains.

Finally, we plan to automate prompt selection using clustering or meta-learning. Rather than relying on static prompt sets, SynthCore could dynamically adjust ensemble makeup based on task feedback or historical performance, further improving generality and sample efficiency.

5.2 Threats to Validity

Construct Validity. While our evaluations use standard multi-objective metrics, these may not capture qualities important to practitioners, such as long-term maintainability or developer trust. Further studies could compare SynthCore-generated configs against expert-labeled benchmarks or measure downstream effects on developer workflow.

Internal Validity. Though care was taken to verify our implementations, there may be latent bugs or assumptions in our active learning and acquisition functions. Hyperparameters for GPM, TPE, and LLM prompting were chosen empirically, and small changes may affect outcomes. We encourage replication and sensitivity testing using our reproduction package.

External Validity. Our 49 datasets cover a broad range of SE optimization problems, but not all real-world tasks. Tasks with strong semantic constraints—like

patch generation or prioritization based on security context—may demand additional domain knowledge or hybrid LLM-human annotation schemes. Also, while we used open-source models, performance may differ for closed commercial APIs.

6 Conclusion

This work presents SynthCore, a simple but powerful ensemble prompting technique for LLM-based annotation and optimization in software engineering. By combining multiple few-shot prompts with no coordination or chain-of-thought overhead, SynthCore achieves state-of-the-art performance on high-dimensional SE optimization tasks—outperforming Bayesian methods, active learners, and single-prompt baselines. All results were achieved using only LLM-generated data, without human intervention.

Our findings reframe how LLMs should be used in SE. Rather than expecting correctness from a single prompt, we should treat LLMs as stochastic samplers of reasoning fragments. When these samples are aggregated via SynthCore, they yield diverse and effective search spaces for optimization—even on tasks where single prompts fail.

SynthCore's value lies in its generality. It requires no fine-tuning, no deep prompt engineering, and no supervision. It is easy to implement, extensible to many task types, and efficient enough to use in budget-constrained settings. This makes it a practical building block for next-generation SE tooling.

Declarations

Conflict of Interest:

The authors declared that they have no conflict of interest. The authors have no competing interests to declare that are relevant to the content of this article.

Clinical Trial Number:

Clinical trial number: not applicable.

References

1. Toufique Ahmed and Premkumar Devanbu. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*, pages 1–5, 2022.
2. Toufique Ahmed and Premkumar Devanbu. Better patching using llm prompting, via self-consistency. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1742–1746. IEEE, 2023.
3. Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. Can llms replace manual annotation of software engineering artifacts? In *MSR'25*, 2025.
4. Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? *ACM Conference on Fairness, Accountability, and Transparency (FAccT '21)*, pages 610–623, 2021.
5. James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.
6. Barry W Boehm and Richard Turner. *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional, 2004.
7. Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
8. Eric Brochu, Vlad M Cora, and Nando De Freitas. 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.
9. Junjie Chen, Ningxin Xu, Peiqi Chen, and Hongyu Zhang. Efficient compiler autotuning via bayesian optimization. In *43rd IEEE/ACM Int. Conf. Softw. Eng. ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 1198–1209, 2021.
10. Pengzhou Chen and Tao Chen. Promisetune: Unveiling causally promising and explainable configuration tuning. In *Proc. of the 48th IEEE/ACM Int. Conf. Softw. Eng.*, 2026.
11. Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3):494–509, 1993.
12. Jacob Cohen. A power primer. 2016.
13. Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*, pages 423–435, 2023.
14. F. Di Fiore, M. Nardelli, and L. Mainini. Active learning and bayesian optimization: A unified perspective to learn with a goal. *Archives of Computational Methods in Engineering*, pages 1–29, 2024.
15. Thomas G Dietterich. Ensemble methods in machine learning. *Multiple classifier systems: First international workshop, MCS 2000 Cagliari, Italy, June 21-23, 2000 Proceedings 1*, pages 1–15, 2000.
16. Markus Dollmann and Michaela Geierhos. On-and off-topic classification and semantic annotation of user-generated software requirements. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1807–1816, 2016.
17. Zhuoyun Du, Chen Qian, Wei Liu, Zihao Xie, Yifei Wang, Yufan Dang, Weize Chen, and Cheng Yang. Multi-agent software development through cross-team collaboration. *arXiv preprint arXiv:2406.08979*, 2024.
18. Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. *Guide to advanced empirical software engineering*, pages 285–311, 2008.

19. Mark Easterby-Smith. The design, analysis and interpretation of repertory grids. *International Journal of Man-Machine Studies*, 13(1):3–24, 1980.

20. Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

21. Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 1993.

22. Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

23. Wei Fu and Tim Menzies. Revisiting unsupervised learning for defect prediction. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 72–83, New York, NY, USA, 2017. ACM.

24. C. E. Gates and J. D. Bilbro. Illustration of a cluster analysis method for mean separation. *HortScience*, 26(8):1070–1070, 1991.

25. J. Gong and T. Chen. Deep configuration performance learning: A systematic survey and taxonomy. *ACM Trans. Softw. Eng. Methodol.*, 2025.

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

27. Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS'22*, 2022.

28. Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024.

29. Md Rakibul Islam and Minhaz F. Zibran. Leveraging automated sentiment analysis in software engineering. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 203–214, 2017.

30. P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *Proc. 24th IEEE Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2016.

31. Brittany Johnson and Tim Menzies. Ai over-hype: A dangerous threat (and how to fix it). *IEEE Software*, 41(6):131–138, 2024.

32. Marius Kamp, Patrick Kreutzer, and Michael Philippsen. Sesame: A data set of semantically similar java methods. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 529–533. IEEE, 2019.

33. Matthew D Krasowski, Scott R Davis, Denny Drees, Cory Morris, Jeff Kulhavy, Cheri Crone, Tami Bebber, Iwa Clark, David L Nelson, Sharon Teul, et al. Autoverification in a core clinical chemistry laboratory at an academic medical center. *Journal of pathology informatics*, 5(1):13, 2014.

34. Rahul Krishna, Zhe Yu, Amritanshu Agrawal, Manuel Dominguez, and David Wolf. The "bigse" project: Lessons learned from validating industrial text mining. In *Proceedings of the 2Nd International Workshop on BIG Data Software Engineering*, BIGDSE '16, pages 65–71, 2016.

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

36. Jia Li, Ge Li, Yongmin Li, and Zhi Jin. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology*, 34(2):1–23, 2025.

37. Andre Lustosa and Tim Menzies. 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):1–22, 2024.

38. Suvodeep Majumder, Nikhila Balaji, Katie Brey, Wei Fu, and Tim Menzies. 500+ times faster than deep learning. In *Proceedings of the 15th International Conference on Mining Software Repositories. ACM*, 2018.

39. Tim Menzies. Retrospective: Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 2025.

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

41. Tim Menzies and Tao Chen. Moot repository of multi-objective optimization tests, 2025.

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

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

44. Janice M Morse, Michael Barrett, Maria Mayan, Karin Olson, and Jude Spiers. Verification strategies for establishing reliability and validity in qualitative research. *International journal of qualitative methods*, 1(2):13–22, 2002.

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

46. Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. Finding faster configurations using FLASH. *IEEE Trans. Software Eng.*, 46(7):794–811, 2020.

47. Noor Nashid, Mifta Sintaha, and Ali Mesbah. Retrieval-based prompt selection for code-related few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2450–2462. IEEE, 2023.

48. Jibesh Patra and Michael Pradel. Nalin: learning from runtime behavior to find name-value inconsistencies in jupyter notebooks. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1469–1481, 2022.

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

50. Moein Pourreza and Davood Rafiei. NormTab: Improving symbolic reasoning in LLMs through tabular data normalization. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11025–11038. Association for Computational Linguistics, 2023.

51. Peter C. Rigby and Christian Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 202–212, New York, NY, USA, 2013. Association for Computing Machinery.

52. Sheldon M Ross. *Introduction to Probability Models*. Academic Press, 11 edition, 2014.

53. C. W. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30(3):507–512, 1974.

54. Lohith Senthilkumar and Tim Menzies. Can large language models improve se active learning via warm-starts? *arXiv preprint arXiv:2501.00125*, 2024.

55. Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

56. Andries Petrus Smit, Paul Duckworth, Nathan Grinsztajn, Kale-ab Tessera, Thomas D Barrett, and Arnu Pretorius. Are we going mad? benchmarking multi-agent debate between language models for medical q&a. In *Deep Generative Models for Health Workshop NeurIPS 2023*, 2023.

57. Shriyank Somvanshi, Subasish Das, Syed Aaqib Javed, Gian Antariksa, and Ahmed Hossain. A survey on deep tabular learning. *arXiv preprint arXiv:2410.12034*, 2024.

58. Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 1433–1443, 2020.

59. Vali Tawosi, Rebecca Moussa, and Federica Sarro. Agile effort estimation: Have we solved the problem yet? insights from a replication study. *IEEE TSE*, 49(4):2677–2697, 2023.

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

61. Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. Transferring performance prediction models across different hardware platforms. In *Proc. of the 8th ACM/SPEC on Int. Conf. Perf. Eng.*, pages 39–50, 2017.

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

63. Mengting Wan, Tara Safavi, Sujay Kumar Jauhar, Yujin Kim, Scott Counts, Jennifer Neville, Siddharth Suri, Chirag Shah, Ryen W White, Longqi Yang, et al. Tnt-llm: Text mining at scale with large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5836–5847, 2024.

64. Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, et al. Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *Advances in Neural Information Processing Systems*, 36:61501–61513, 2023.

65. Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

66. Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, 8, 1995.

67. Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

68. Guang Yang, Yu Zhou, Xiang Chen, Xiangyu Zhang, Terry Yue Zhuo, and Taolue Chen. Chain-of-thought in neural code generation: From and for lightweight language models. *IEEE Transactions on Software Engineering*, 2024.

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

70. Dawei Yuan, Guocang Yang, and Tao Zhang. Ui2html: utilizing llm agents with chain of thought to convert ui into html code. *Automated Software Engineering*, 32(2):1–24, 2025.

71. Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

Response to Reviewers for EMSE-D-25-00425

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.

Comments for the authors

This paper is an interesting idea which aims at important research field. However, there are several issues in this paper that need to be improved.

Comment 1: The evaluation is limited. Many LLMs are mentioned without evaluation.

Thank your for that note. Please forgive us, that was our fault, we were not clear on the overall goals of those paper. This study does not attempt to determine which specific LLM architecture is optimal for optimization tasks; that question remains for future work. Instead, it establishes a clear empirical result: **LLMs are highly effective warm-start mechanisms for software engineering optimization**, outperforming long-standing symbolic baselines even in high-dimensional settings. This contribution helps fill a documented gap in the SE literature, where only ∼5% of LLM papers compare against non-LLM approaches [28], and where several recent studies report pessimistic assessments of LLM performance. Our findings provide robust evidence to the contrary, showing that LLMs deliver a substantial and practical advantage when initializing SE optimization workflows.

We have added this note to the paper R0a on page 5 and R2c on page 13.

Comment 2: Lack of context and background. Much information is not introduced very well.

Thank you for that note. We agree with you that this paper needs much careful expansion. As shown in the BLUE text throughout this revision, this draft contains many expanded points.

Comment 3: Value and highlight potential limitations of the approach are not discussed.

You make a good point. Please see our new discussion section where these matters are reviewed.

Response to Reviewer 1

We received no comments from reviewer1.

Response to Reviewer 2

Comment 1: While the paper addresses an important challenge in SE optimization and presents extensive experiments, several serious issues limit its contribution in its current form.
The proposed SynthCore is essentially a repetition of a fixed prompt template with multiple independent runs and aggregation of outputs. This is conceptually very close to established methods such as Self-Consistency prompting and other voting-based ensembles. The paper does not provide a theoretical rationale for why SynthCore is fundamentally different, nor does it include empirical comparisons against these related baselines.

Thank you for the comment. In the previous draft, we did not clearly articulate how our approach differs from self-consistency prompting. Now, in R2a-1 on page 3 & R2a-2 on page 8 9and Table 2 we provide a detailed explanation of these distinctions and clarify why SynthCore is fundamentally different both conceptually and operationally.

Comment 2: All experiments are conducted on MOOT-like tabular datasets involving multi-objective optimization problems. While these datasets are relevant to certain SE optimization tasks, they do not cover the diversity of labeling problems in software engineering. This narrow scope limits how far we can generalize of findings and weakens the broader impact of the proposed method.

Please excuse us, we do not mean to be argumentative, but "just" restricting the result to tabular higher dimensional multi-objective problems actually covers ain interesting space of problems. While working on this revision, we served as ICSE'26 and FSE'26 reviewers. In that work, we were assigned four ICSE'26 and FSE'26 submissions that all used a small subset of the data explored here. Those papers explored ten data sets, we explore 49. Which is to say that while we "just" explore optimization, in that sub-arena of SE, our experiments are far more extensive than very recent state-of-the-art submissions to leading venues.
Also, we argue that the analysis of tabular data is not simplistic or outdated. Tabular data is widely used. Somvanshi et al. (https://arxiv.org/abs/2410.12034) call it "the most

commonly used data format in many industries...finance, and transportation.". Commercial use in data synthesis/privatization for health/government (https://ieeexplore.ieee.org/abstract/document/10438420) GitHub-scale code analytics uses tabular data (e.g., Commit-Guru5, used in many SE papers). Also, returning to those ICSE'26 and FSE'26 papers we mentioned above, all those papers use case studies expressed as tabular data.

Further, the processing of tabular data is challenging, even for LLMs. Somvanshi et al. (https://arxiv.org/abs/2410.12034) report that "despite deep learning's success in image and text domains, tree-based models like XGBoost and Random Forests continue to outperform neural networks on medium-sized tabular datasets. This performance gap persists even after extensive hyperparameter tuning."

Comment 3: The evaluation is dominated by results from a single LLM (i.e., Gemini 1.5 Pro). Although other models (e.g., GPT, LLaMA) are briefly mentioned, their results are neither systematically evaluated nor reported in detail. This raises concerns about whether the proposed approach is specific to one model's capabilities. Cross-model experiments are essential to demonstrate that the approach is robust across different LLMs.

Thank you for the comment. We agree with your observation and apologize for not making our objective clear in the previous draft. Our goal is not to benchmark the performance of different LLMs for this task, but rather to evaluate how well our LLM-based method, SynthCore, compares against symbolic baselines. To ensure a fair comparison, we selected Gemini 1.5 Pro, a sufficiently capable state-of-the-art model at the time of the study. We have clarified this study design rationale in the revised manuscript at `R0a` on page 5 and `R2c` on page 13.

Comment 4: The related work section is disproportionately long and includes extensive repetition of prior results, which could be summarized more concisely. This pushes the presentation of the method to page 12 of a 25-page body, which makes it difficult for readers to quickly grasp the main technical contribution.

We agree, the old paper was too verbose. Here, we have reduced that 8 page section to under 3. Which means our method is introduced far earlier in the paper.

Comment 5: The results section presents many tables and statistical rankings, but the key takeaways are not clearly distilled. For example, while the data might indicate that SynthCore improves performance in high-dimensional settings, this is not

clearly summarized in prose with supporting evidence and effect sizes.

Thank you for the comment. We agree that the previous draft did not sufficiently discuss the implications of our results. In this revision, we have added a dedicated recommendation for practitioners (see `R3d` on page 22). These recommendations clarify when our methods should or should not be used, given constraints such as available budget, task characteristics, and data dimensionality. We believe these additions provide practical guidance for industry practitioners working on configuration tasks.

Comment 6: Other minor issues: Page 3, Line 12, a reference is missing Page 9, Referenced another Table (i.e., Table 2) in Table 1

Thank you for the comment. We have fixed the above issues

Response to Reviewer 3

Comment 1: The topic of the paper is highly relevant to both empirical SE and AI-for-SE. The experimental setup is rigorous. The authors compare SynthCore to strong baselines (Gaussian Process Models, Tree of Parzen Estimators, standard few-shot LLMs), and the results are convincing when it comes to demonstrating performance improvements in high-dimensional tasks. The approach is methodologically solid. SynthCore is simple yet effective, and relies on well-motivated ensemble learning principles and validated through statistical analysis (Scott-Knott, bootstrapping, Cliff's Delta). Moreover, the authors were good at motivating their approach throughout the paper and the relevance of their methods despite its simplicity.

Thank you for those kind words

Comment 2: The introduction does not prepare the reader for the core problem space of the paper. Challenges of annotation with LLMs are outlined, but the central focus of the paper: multi-objective optimization, is only briefly mentioned. The authors should better motivate why optimization in SE is important.

Thank you for that comment. You are quite correct, we needed to add more material in that area. Please see `R3c` on page 2.

Comment 3: Many critical concepts (multi-objective reasoning, active learning, warm-starts) are only introduced in depth in the related work section. This makes the paper hard to follow, especially for readers that are unfamiliar with the optimization literature. These concepts should be introduced earlier in the

paper with concrete examples to help contextualize the contribution.

Thank your for emphasizing this point. As mentioned above, we have added brief explanations for the above concepts in the introduction section of this paper. kindly refer [R3c] on page 2.

Comment 4: The results are well-structured and clearly presented, but the paper does not fully unpack their implications. The discussion section needs more depth in interpreting the findings: What does it mean for practitioners?

Thank you for the follow-up comment. As noted in our response above, the earlier draft did not sufficiently discuss the practical implications of our findings. In addition to those revisions, we have now included a clearer recommendation-for-practitioners section that explains when and where SynthCore should be used. Please refer to [R3c] on page 2.

Under what conditions might SynthCore not work well?

Thank you for raising this point. SynthCore is not designed to work well in certain settings, and we explicitly acknowledge these limitations in the revised draft refer [R3d-b] on page 24. In particular, we have not explored SynthCore on ultra–high-dimensional domains such as audio, or image data, where feature spaces can be extremely large and structured.

How should researchers integrate SynthCore into real-world SE pipelines? For example, could SynthCore reduce the need for human SME input in industry?

Thank you for asking these questions. In response to your query, we offer a proposed workflow [R3d-2] on page 23 and soem general notes on the impact of SynthCore at [R3g] on page 24

Comment 5: Is it feasible to deploy this as part of continuous SE processes?

We think so. Please see our notes on that at [R3d-2] on page 23.

Comment 6: Overall, the results should be discussed with respect to the nature of high-dimensional optimization tasks. Provide guidance on when to use SynthCore. Discuss the tradeoffs involved in using SynthCore (e.g., does it have increased runtime and LLM costs?) versus simpler prompting approaches.

Thank you for the comment. As addressed in earlier responses, we have included a recommendation diagram indicating when our methods should be used under specific constraints (see [R3c] on page 2). Our methods are also an order of magnitude faster than state-of-the-art symbolic approaches such as Gaussian Process models (see 3). Based on these results, we argue that practitioners can use our guidelines to select and

adapt faster methods for multi-objective tasks effectively (see `R3d` on page 22).

Comment 7: Are there scenarios where standard few-shot would be good enough even in high-dimensional?

Our results suggest not.

Comment 8: Make results more actionable. The paper presents rankings and performance curves, but readers would also benefit from concrete takeaways or heuristics (e.g., "if your SE optimization task has more than X features, use SynthCore with budget ≥ 20). What do the results imply for practitioners? For researchers?

Comment 9: Thank you for the comment. We agree that the implications of our results were not presented clearly in the previous draft. In this revision, we have added explicit recommendations for practitioners, supported by a recommendation diagram (Figure 4) that clearly indicates when to prefer *SynthCore* over standard few-shot learners. The conclusions in this diagram are grounded in the summarized empirical results reported in `R3d` on page 22.

Comment 10: Additional comments for the Introduction section: The first few lines of the first paragraph need citations.

Good point. Now added.

Comment 11: Page 2, line 12: "LLMs appear well-positioned to streamline SE data collection pipelines" - Would "data labelling be more fitting in the context of the paper?

We have now dropped that sentence.

Comment 12: Page 3, line 5: "see later in this paper" Where? Be more specific. What section?

We have now added pointers at that point of the paper to Table 5, 6 7.

Comment 13: Page 3, line 13: Missing citation.

We apologize for that, it is rectified.

Comment 14: Page 3, line 32: "this paper explores three research questions" There are only two research questions.

We apologize for that, we have corrected in this version to two research questions .

Comment 15: For each RQ introduced, it is good practice to mention a brief motivation for it, and present an overview of the results/findings. RQ2 phrasing assumes the answer to RQ1 is "yes". Consider reframing it so that it is independent of the previous RQ's outcome.

Thank you for the suggestion, we have changed the RQs as mentioned. Please refer `R3e` on page 4.

Comment 16: Page 4, line 8: "and relatively easy" Based on what? Either drop the statement or explain why.

We have decided to drop the statement.

Comment 17: Key concepts introduced too late. Move essential context to the introduction or early background. Discuss them in more depth in this section. Use concrete examples to contextualize concepts. Definitions of terms like "multi-objective optimization" and "active learning" are given, you give an example in Table 1, but it is still abstract.
Thank you, your comment is quite correct. We have added the required material, see R3c on page 2.

Comment 18: Add short explanations to the definitions. For example, if you are discussing warm-starts, you can add "e.g., pre-selecting configurations that are likely to perform well when tuning. . . " Clarify assumptions and terminology earlier. The distinction between "LLM annotations", "few-shot learners", and "ensemble learners" becomes clear only deep into the method sections. Summarize them in a terminology paragraph early.
Thank you for the suggestion. We have added brief explanations for the above concepts in the introduction section of this paper. Kindly refer R3c on page 2.

Comment 19: Additional comments on Literature Review section: Page 4, line 24: You mention MSR in this case as the concept of "mining software repositories" but earlier in the paper, it is used as the conference. You should define it clearly, some readers may not be familiar with this concept or venue.
Thank you for pointing this out. We have fixed it

Comment 20: Page 5, line 25: "...this tools three sets of three hour meetings hours" Needs rephrasing. That clumsy section was removed in our shortenning of the overly-long related work section.

Comment 21: Page 6, line 37: Typo "trainign" → training.
We apologize, we have fixed it.

Comment 22: Page 6, line 46: "The Ahmed et al. study" → Suggestion: Ahmed et al.
Thank you for the suggestion, we have made the change.

Comment 23: 2.3 Results from Related Work: Very confusing section. A lot of background concepts are discussed, some parts of the experimental setup of the paper, comparing studies results. Consider reorganizing.
We agree the structure was confusing. We have significantly reorganized that section to separate background concepts from the comparison of related work results, making the experimental context clearer. As for earlier backgrpund, please see R3c on page 2.

Comment 24: Page 9, line 36: "subject matter experts" has already been defined as SMEs. Use the abbreviations when introducing them.
Thank you for pointing it out, we have fixed the issue.

Comment 25: Page 21, line 17: Possible typo "the two lower curves can from simple" That sentence was been reorganized and removed.

Comment 26: Page 21, line 45: Possible typo "Tables... show what be achieved by..."
Thank you for pointing it out, we have fixed the issue.

Comment 27: Are the results presented in this paper directly tied to the selected model? It seems to me that they would be very dependent on the model's size, parameters, context size, etc. The justification of the model selection was very clear, but it makes me wonder how practical the solution is if it requires the best performing state-of-the-art model to be implemented. Would you consider repeating your experiment using other models and compare the results?
As we say in R0a on page 5 our point is not that LLM1 is better than LLM2 but rather it is that LLM us better than symbolic. This draft now clarifies the gains are methodological, not just model dependent.

Response to Reviewer 4

Comment 1: The reasons for selecting Gemini 1.5 Pro are stated as "a 1-million-token context window and low cost," yet no baseline comparison data with other mainstream models (e.g., GPT-4o) is provided. For example, what are the differences in single-prompt annotation accuracy and runtime between Gemini 1.5 Pro and GPT-4o in low-, medium-, and high-dimensional tasks? The paper only mentions that "gpt-o1 and gemini-1.5pro yield similar results" without providing specific values (such as the mean Chebyshev distance or rank proportion), making it impossible to verify the rationality of the model selection.
Thank you for the comment. We apologize for not making our objective clear in the previous draft. Our goal is not to compare or benchmark different LLMs, but rather to compare LLMs with other baseline symbolic methods (UCB, TPE, Bayes Explore/Exploit, etc.). We have clarified this in R2c on page 13; kindly refer to it for details.

Comment 2: The paper notes that the MOOT repository contains 49 tasks, including "configuration, Hyperparameter Optimization (HPO), and process," but fails to clarify the specific optimization objectives and evaluation criteria for each individual task. For instance, datasets like "SS-A" and "SS-B" in Table 2 only label "x/y dimensions" without explaining the actual optimization objectives corresponding to the y-values (e.g.,

whether it is "minimizing runtime" or "maximizing system throughput").

Thank you for the comments. While we initially planned to include the complete parameter specifications for all 49 datasets, space constraints made this infeasible. Consequently, we have provided these details as an external reference. kindly refer R4b on page 11

Comment 3: Additionally, the paper does not supplementarily explain what specific configuration parameters the "38 x-variables" in the "SQL AllMeasurements" dataset refer to. This prevents readers from judging the representativeness of the datasets for SE tasks.

We appologize for not including these details. However due to space contraints of including 38 varialbe names inside the manuscript we have provided an external reference for these details. kindly refer R4c on page 11.

Comment 4: The paper sets L=0.6B (where L is the number of initial randomly annotated samples) and M=20, but does not explain the rationality of these parameter values. For example: Why is L set to 0.6B instead of 0.5B or 0.7B? Is M=20 determined through experimental validation (e.g., performance comparison with M=10 or M=30) or based on empirical settings?

We apologize for the earlier lack of clarity. According to the Near Enough Optimization (NEO) framework (see R4d on page 16) "near-enough" solutions are effectively indistinguishable from the true optimum. As shown by the mathematical analysis in R4d on page 16, encountering such a near-optimal solution requires sampling only about 60 times. Our choice of setting $M = 20$ follows from the ambition of staying well below this 60-sample NEO threshold. In our experiments, sampling 20 or 30 instances consistently performed as well as sampling any number above this point, with only minimal gains beyond that threshold. Thus, selecting $M = 20$ reflects an engineering decision that remains faithful to the NEO theory while keeping the annotation cost low.