Configuring Parameters of Search-based Software Engineering Tools with Multiple LLM Models

Daniel Nouchi
State University of Maringá
Maringá, Paraná, Brazil
ra123991@uem.br

Willian M. Freire
State University of Maringá
Maringá, Paraná, Brazil
willianmarquesfreire@gmail.com

Aline M. M. M. Amaral
State University of Maringá
Maringá, Paraná, Brazil
ammmamaral@uem.br

Silvia R. Vergilio
DInf, Federal University of Paraná
Curitiba, Paraná, Brazil
silvia@inf.ufpr.br

Thelma E. Colanzi
State University of Maringá
Maringá, Paraná, Brazil
thelma@din.uem.br

ABSTRACT

Search-Based Software Engineering (SBSE) tools offer automatic support by applying search-based techniques to solve different optimization problems of various software engineering areas. However, configuring SBSE tools often requires expertise in optimization techniques, posing a significant barrier for non-specialist users. This task involves selecting suitable algorithms, tuning parameters, choosing appropriate objective functions and search operators, and making decisions that affect optimization quality and performance. To address this challenge, we propose a multiple-LLM approach that integrates Large Language Models (LLMs), specifically ChatGPT-4, into SBSE workflows to provide context-aware and natural language suggestions for parameter configuration. The approach follows a *Divide and Conquer* strategy, in which specialized LLMs are trained for distinct parameter configuration subproblems. As a proof of concept, we implemented OPLA-Wizard that works with OPLA-Tool v2.0, for *Product Line Architecture (PLA)* design. This assistant supports users in configuring algorithm settings, search operators, and objective functions through a user-friendly interface. Preliminary results show that the multiple-LLM approach improves accuracy, explanation clarity, and output format adherence compared to a single-LLM approach. This work contributes to the democratization of SBSE by reducing cognitive load, preventing misconfigurations, and making advanced optimization techniques more accessible to a broader audience.

KEYWORDS

Search-based Software Engineering, Large Language Model, SPL

1 Introduction

Search-Based Software Engineering (SBSE) is the field of software engineering research and practice that applies search-based techniques to solve various optimization problems across different software engineering areas [10]. These problems typically involve competing constraints, ambiguous and imprecise information, and generally do not have exact and analytical solutions.

To support the practical application of SBSE, several specialized tools have been developed. Notable examples include EvoSuite, which automatically generates test data to enhance the effectiveness of software testing [6]; OPLA-Tool, which helps optimize the *Product Line Architecture (PLA)* design to achieve better architectural trade-offs [9]; and OptiJIT, a tool that improves software execution performance by optimizing just-in-time compilation strategies [11].

The use of SBSE tools enables software engineers to automatically obtain solutions for complex and labor-intensive tasks, thereby reducing the efforts and costs associated with software development. However, software engineers do not always have sufficient knowledge in the optimization field and may face some challenges using SBSE tools. One of the main challenges is effectively configuring the parameters required by the tools. The parameter range is generally broad, and misconfigured parameters can severely impact solution quality or computational cost, directly affecting the convergence behavior and performance of the optimization algorithm [3].

In practice, users are often required to make multiple configuration decisions, such as choosing the appropriate objective functions, the most suitable algorithm, and search operators, which may vary significantly depending on the problem's characteristics and optimization goals. This variety increases the cognitive load on non-specialist users and can hinder the practical application of SBSE tools. Defining appropriate parameter values typically requires running multiple experiments to evaluate different configurations empirically. This process is often time-consuming and computationally expensive, even when configuration tools are used. The problem of configuring parameters for SBSE tools is very complex. This makes using SBSE tools unfeasible for many real-world scenarios, particularly in industrial environments.

Recent advancements in computational intelligence, particularly in *Machine Learning (ML)*, have opened new possibilities for addressing such complexities. Among these, *Large Language Models (LLMs)* have emerged as powerful tools capable of processing, interpreting, and generating human-like insights from complex and unstructured data [17]. Their ability to produce context-sensitive, natural language outputs makes them especially well-suited to assist in decision-making tasks, including those inherent in dynamic and domain-rich SBSE scenarios, where traditional techniques may face limitations [15].

In the context of search-based PLA design, Freire et al. [8] proposed an approach that utilizes an LLM model trained to suggest objective functions, through the AIssistDM plugin [7]. The approach presented promising results, assisting non-specialist users in configuring this parameter in OPLA-Tool. Nevertheless, we conducted preliminary tests using the AIssistDM plugin to address the configuration problem, considering all the parameters of OPLA-Tool,

and the obtained model resulted in inconsistent and inaccurate suggestions. For example, AIssistDM occasionally recommended using inadequate mutation operators, leading to suboptimal results. These limitations were attributed to the cognitive overload imposed when using a single LLM model for a complex task, such as the parameter configuration of SBSE tools.

To deal with this limitation, this paper proposes a *multiple-LLM approach*, based on the *Divide and Conquer* strategy. The idea is to divide the complex problem of configuring SBSE tools into simpler subproblems and train LLM models separately to improve the quality of the obtained general solution. In this context, each subproblem corresponds to a specific category of parameters commonly found in SBSE tools, such as: (i) algorithm-specific settings (e.g., population size and number of fitness evaluations), (ii) objective functions, and (iii) search operators (e.g., mutation and crossover rates). A distinct LLM model is trained for each category using targeted training material relevant to that parameter group. This enables each model to more accurately capture parameter-specific nuances and produce more precise, context-aware suggestions. By integrating this specialized approach into the SBSE workflow, we aim to effectively bridge the expertise gap frequently encountered by non-specialist users, who are unfamiliar with optimization strategies and domain-specific trade-offs.

As a proof of concept, we evaluated the proposed approach in the domain of PLA design by implementing OPLA-Wizard. This LLM-based assistant supports parameter configuration in OPLA-Tool v2.0, and was implemented on top of AIssistDM [7], which enables the exchange of messages between OPLA-Tool and ChatGPT-4. OPLA-Wizard guides users in configuring parameters by offering an interface embedded in the PLA design environment.

Initial results show that our multiple-LLM approach outperforms the single-LLM approach adopted by AIssistDM and used as a baseline. The specialized models achieved high accuracy when answering parameter-related questions; two of the models, to suggest basic algorithm settings and objective functions, reached 100% accuracy, while the models to suggest crossover and mutation operators achieved an accuracy of 96.6% and 93.2%, respectively. We also observed that the wizard consistently produced answers in the correct output format required by the SBSE tool and provided clear justifications for the recommended parameters.

These findings indicate that the wizard using a multiple-LLM approach not only helps users obtain more accurate suggestions but also delivers explanations that make configuration parameters easier to understand, especially for non-specialist users. By embedding LLM guidance directly into the SBSE workflow and tailoring parameter configurations, the multiple-LLM approach represents a step forward in making SBSE tools more accessible, usable, and adaptable. Ultimately, this approach contributes to the democratization of SBSE by reducing the cognitive overhead of complex configuration processes and by enabling broader adoption among practitioners and researchers.

This paper is organized as follows: Section 2 addresses studies that adopt LLMs to ease SBSE tasks. Section 3 presents our approach to configure the parameters of SBSE tools using multiple LLMs. Section 4 describes implementation aspects. Section 5 presents preliminary results. Finally, Section 6 concludes the paper and outlines future work.

2 Related work

Research has recently begun to explore how LLMs can support software engineering tasks traditionally addressed by SBSE, particularly those involving decision-making and heuristic search. In SBSE, LLMs can be utilized to interpret complex software engineering tasks, automate the generation of coding artifacts, or even assist in developing and maintaining software by providing real-time suggestions and solutions. Their ability to process and synthesize large amounts of unstructured text data can enhance decision-making processes, bridging the gap between vast data handling and actionable software development insights [1].

OrcaLoca [14] enhances fault localization by combining LLM-based reasoning with task priorization, decomposition, and semantic context pruning. SWE-Search [2] integrates LLMs with Monte Carlo Three Search in a multi-agent system to iteratively refine solutions for tasks such as bug fixing. AutoCodeRove [18] combines LLMs with abstract syntax tree (AST) analysis and fault localization to enable cost-efficient automatic code repair.

While these approaches demonstrate the potential of LLMs to assist in reasoning and decision-making, they do not address a key challenge in SBSE: helping users configure parameters (e.g., objective functions and search operators). This gap becomes particularly relevant in more open-ended SBSE applications like PLA design, where configuration decisions are highly context-dependent and require user input.

Some SBSE tools, such as EvoSuite [6], embed default parameter settings based on prior empirical studies, relieving users from manual configuration. However, in tools like OPLA-Tool v2.0, the user must explicitly perform parameter configuration, and no universally optimal configuration applies to all scenarios. This increases the barrier to adoption, especially for non-specialist users.

The studies conducted by Freire et al. [7, 8] are initiatives to attack this problem. The plugin AIssistDM integrates LLMs, specifically ChatGPT-4, into an SBSE workflow to support non-specialist users [7]. AIssistDM works with OPLA-Tool and focuses on assisting users in configuring problem-domain parameters, such as objective functions and search operators, through natural language recommendations. While AIssistDM shares our motivation of enhancing SBSE usability for non-expert users through LLMs, our work adopts an innovative approach based on multiple LLMs. The configuration problem is partitioned into sub-problems, and multiple LLM models are obtained by training on a narrower group of related parameters. This results in more accurate, consistent, and context-aware suggestions. This specialization improves the quality of suggestions and facilitates more precise explanations and future expansion of the approach to support additional SBSE domains and tools. In addition, existing studies that employ multiple LLMs are not suitable for the problem tackled in our work [5, 12].

Existing studies that employ multiple LLMs, such as those described in [6, 13], typically address scenarios involving general multi-objective optimization or code generation tasks, which differ significantly from the parameter configuration challenges in SBSE contexts, where parameter groups have explicit dependencies and require highly domain-specific expertise. Therefore, their methods are not directly applicable or effective in our scenario.

3 Proposed approach

Our multiple-LLM approach for configuring SBSE tools parameters is based on the *Divide and Conquer* strategy, as illustrated in Figure 1. The complex problem to be attacked is partitioned into simpler sub-problems. For each sub-problem, an LLM model is trained with specialized data, which contributes to increasing performance.

The first question that can arise is how to divide our problem. For this end, we resort to the main ingredients required for an SBSE solution proposed by Harman et al [10]: the representation for the problem and search operators to be manipulated by the search algorithm, and a function to evaluate the quality of the solutions. Moreover, we can use the parameter categories mentioned in the work of Arcuri and Fraser [3].

SBSE tools often deal with a broad spectrum of parameters, ranging from straightforward algorithmic settings to complex problem-domain configurations that significantly influence the outcomes of software engineering tasks. For instance, algorithm-specific parameters of evolutionary algorithms include population size, number of generations, mutation rate, and crossover probability, which directly affect the behavior and performance of the search process. On the other hand, domain-specific parameters involve objective functions, which articulate the evaluation criteria used to assess and rank candidate solutions.

To illustrate this decomposition, consider a typical SBSE scenario, such as search-based PLA design using OPLA-Tool [9]. In this context, the configuration process is partitioned into configuring different sets of related parameters. For instance, the user must configure (i) basic algorithm settings: select an *optimization algorithm*, define its basic settings (e.g., population size, number of runs, number of generations or evaluations, and so on), (ii) search operators: choose appropriate *mutation and crossover operators* from a predefined set tailored to PLA evolution, and (iii) specify a subset of *objective functions* (e.g., minimize coupling, maximize cohesion, improve feature modularization). Each of these decisions impacts the optimization behavior in different contexts and requires different types of knowledge.

According to the proposed approach, we assign each subproblem to a specialized LLM model. The model trained for algorithm settings focuses only on understanding and suggesting configurations, such as algorithm choice and run parameters. Another model specializes in choosing mutation and crossover operators, while a third model suggests objective functions. This division allows each LLM to focus on a narrower context, improving suggestion accuracy, reducing ambiguity, and making the overall system more interpretable and maintainable.

The multiple-LLM approach aims to bridge the knowledge gap frequently encountered by non-specialist users when configuring complex optimization parameters. The idea of the approach is to rely on natural language interaction to provide human-like suggestions, instead of requiring manual interpretation of technical settings. These suggestions are generated dynamically, based on contextual information extracted from the user about the software engineering problem, thereby supporting decision-making with minimal prior knowledge of optimization or SBSE techniques. We present the implementation aspects of our approach in the next section.

4 Implementation aspects

This section describes OPLA-Wizard, an assistant that implements our multiple-LLM approach for automating the suggestion of parameter configurations for OPLA-Tool-v2.0. The PLA design domain is inherently complex and has a multi-objective nature, which requires balancing competing quality attributes, such as coupling, cohesion, and feature modularization, within a vast design space composed of interdependent architectural elements [16]; hence, PLA optimization involves context-specific decisions that vary across designs. OPLA-Tool utilizes various evolutionary algorithms that require configuration. The user must configure a subset of 20 custom objective functions. There are also three crossover operators and seven mutation operators specific to the PLA design domain that the user must configure before running the optimization algorithm.

Thus, PLA design provides a compelling environment for assessing the wizard's aptitude in generating intelligent, context-sensitive suggestions and facilitating decision-making in the SBSE context.

OPLA-Wizard was implemented on top of AIssistDM, as illustrated in Figure 2. Four LLM models are generated. The **OPLA-Settings** model assists users in configuring core optimization parameters, such as the algorithm type, number of fitness evaluations, population size, and number of runs. The **OPLA-ObjFN** model suggests appropriate objective functions based on the user's input. Finally, the **OPLA-Mutation** and **OPLA-Crossover** models recommend suitable mutation and crossover operators, respectively, tailored to the characteristics of the PLA design task.

AIssistDM plugin [7] allows the exchange of messages between OPLA-Tool v2.0 and ChatGPT-4. When someone asks for a suggestion of parameters in the wizard, the question is sent to ChatGPT-4 through the plugin. When ChatGPT-4 gives back the suggestion, the AIssistDM plugin collects the answer and returns to OPLA-Wizard.

Although LLMs typically produce human-readable text, this output is not ideal for integration with software tools, as it is ambiguous and complex to parse automatically. Structured formats, such as JSON, are preferred for reliable integration and programmatic interpretation. Therefore, we defined a standardized JSON structure for the responses, enabling programmatic interpretation and seamless integration with OPLA-Tool.

Figure 3 presents an example of the answer of each model. Every prompt created for the respective model contains an instruction specifying the output, as shown in `Answer in the JSON-Format "values": [...], suggestion: ".."`, where values contain the objective functions suggested, and suggestion explains why they were selected.

The main functionalities of OPLA-Wizard include the automated suggestion of parameter configurations for OPLA-Tool v2.0 and interactive visualization of these suggestions. The core feature consists of generating suggestions for parameters such as objective functions, search algorithms, and operators. ChatGPT-4 was trained using a dataset comprising the available parameters and usage examples, enabling it to provide accurate and context-aware suggestions. Additionally, users can interact with the wizard through a graphical interface embedded in OPLA-Tool, which displays the suggested configurations and their justifications. This interface enhances user understanding of the trade-offs involved in each configuration decision and supports informed parameter selection.

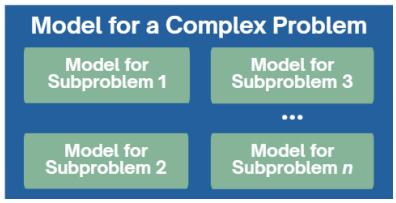

Figure 1: Multiple-LLM approach

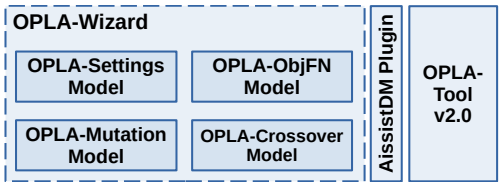

Figure 2: OPLA-Wizard Architecture

```
// OPLA-Settings Model
"settings": {
  "algorithm": "NSGAII",
  "runs": 30,
  "evaluations": "3000",
  "population": "20",
  "suggestion": "The choice was made because..."
}
```

Figure 3: JSON example

These mechanisms collectively reduce cognitive load, mitigate configuration errors, and streamline decision-making. Unlike traditional tools that rely on rigid menus or form-based inputs, the approach enhances usability through conversational guidance, improving accessibility and transparency for users of intricate SBSE tasks. Next, we present a usage example of OPLA-Wizard.

4.1 Wizard´s usage

OPLA-Wizard offers a *Graphical User Interface* (GUI) for enhancing the configuration of parameters in OPLA-Tool v2.0, utilizing AI-driven capabilities. This subsection provides a detailed usage example, demonstrating how users can leverage OPLA-Tool using SBSE techniques augmented by LLMs.

Figure 4 presents an excerpt of OPLA-Tool v2.0. Due to space limitations, some settings, such as the number of runs, were omitted. Nevertheless, as can be seen, many configurations can be made. The first card in the figure contains the main settings, e.g., Optimization Algorithm and Population Size. The second one includes the objective functions available for use. The third and fourth cards contain the search operators for crossover and mutation operations. This is only an excerpt, but OPLA-Tool front-end can be found in the official repository.

To request a new suggestion from OPLA-Wizard, the user clicks the "Get a suggestion" button. The process begins with the input of an XML file containing the PLA design, which serves as the starting point for the wizard. This XML encodes the PLA model using UML (Unified Modeling Language), and its content is analyzed by the wizard to extract relevant architectural information, including the number of classes and interfaces, relationships among components, the number of attributes, and methods. These extracted elements compose a contextual prompt sent to ChatGPT-4. Additionally, the user is prompted to describe their goals through a dialog interface, as illustrated in Figure 5. This user input is also included in the final prompt, ensuring that the parameter suggestions returned by ChatGPT-4 are both architecture-aware and goal-oriented.

ChatGPT-4 receives the prompt and gives an answer based on its training. Since the wizard is composed of four independent LLM models, each responsible for a specific group of parameters, the prompt is dispatched simultaneously to all models. Each model generates a suggestion tailored to its respective parameter group (e.g., objective functions or search settings). OPLA-Wizard then collects the responses, aggregates them into a unified suggestion, and displays the combined result to the user in a single dialog. Additionally, in this case, the answer from one model does not affect the answers from the other models.

Finally, the suggested parameters are displayed in a dialog to the user as an example, as shown in Figure 6. The first part presents the suggested objective function, which was obtained through the OPLA-ObjFN model. The second one contains settings from the OPLA-Settings model, such as optimization algorithm, runs, and individuals. Finally, the third and fourth parts show the crossover operators suggested by the OPLA-Crossover model and the mutation operators offered by the OPLA-Mutation model, respectively.

In addition to simply suggesting values, the wizard provides justifications for each suggestion, helping the user understand the rationale behind the configuration. This explanation is crucial for enhancing transparency and facilitating informed decision-making. Moreover, the wizard assists in configuring other parameters, such as mutation and crossover probabilities, which directly influence the application of the respective operators during the search process.

When the user clicks on the "That's it" button (Figure 6), the parameters are selected on OPLA-Tool for optimization. On the other hand, the button "Get new suggestion" allows the user to return to the previous step and send a new prompt to ChatGPT-4.

Currently, the proposed architecture operates independently for each parameter group, without explicitly sharing context across models. A mechanism to propagate contextual choices between models could be explored in future work, ensuring that parameter dependencies are adequately considered to avoid potential suboptimal or conflicting recommendations.

It is worth mentioning that since the underlying LLM (ChatGPT-4) is not deterministic by nature, identical user inputs might lead to slight variations in suggestions.

4.2 Training ChatGPT-4

The full potential of the multiple-LLM approach lies in the data that the optimization tool user supplies to ChatGPT-4. In this sense, it is necessary to tailor the underlying LLM to better understand and respond to the specific needs of software engineering tasks. Training ChatGPT-4 involves fine-tuning it with datasets relevant to its functions, including diverse software engineering problems and optimization scenarios. For each model encompassed in the approach, the training process typically involves the steps:

(1) Data Collection: Gathering a comprehensive dataset that includes various software engineering texts, such as code snippets, documentation, and problem descriptions.
(2) Preprocessing: Cleaning and preparing the data to ensure it is suitable for training, which may involve removing irrelevant information, correcting errors, and formatting.
(3) Model Training: Feeding the prepared dataset to ChatGPT-4 allows it to learn from the specific patterns, terminology,

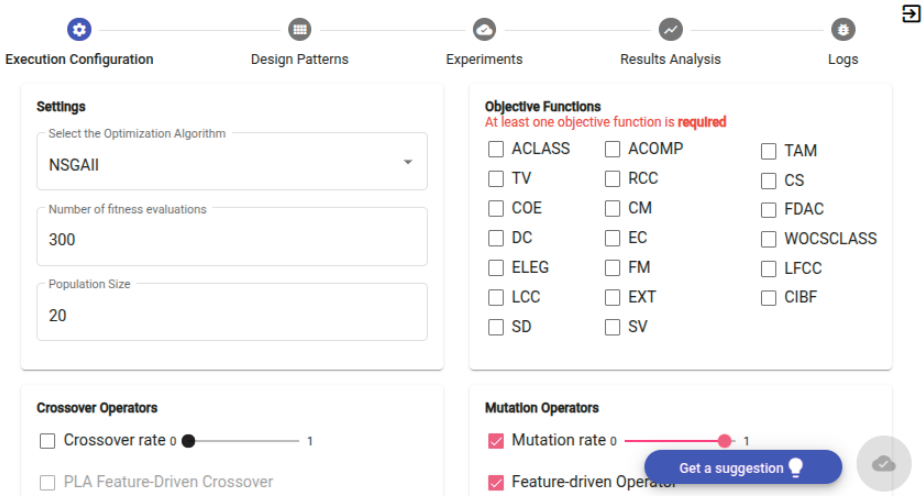

Figure 4: Excerpt of OPLA-Tool v2.0

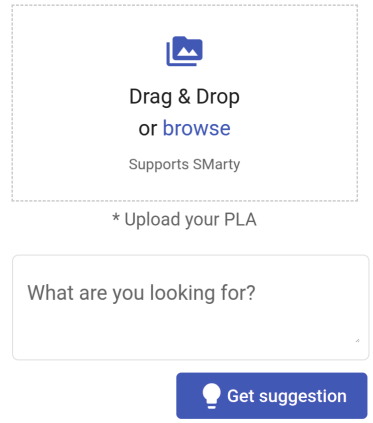

Figure 5: Uploading and proceeding with the solution.

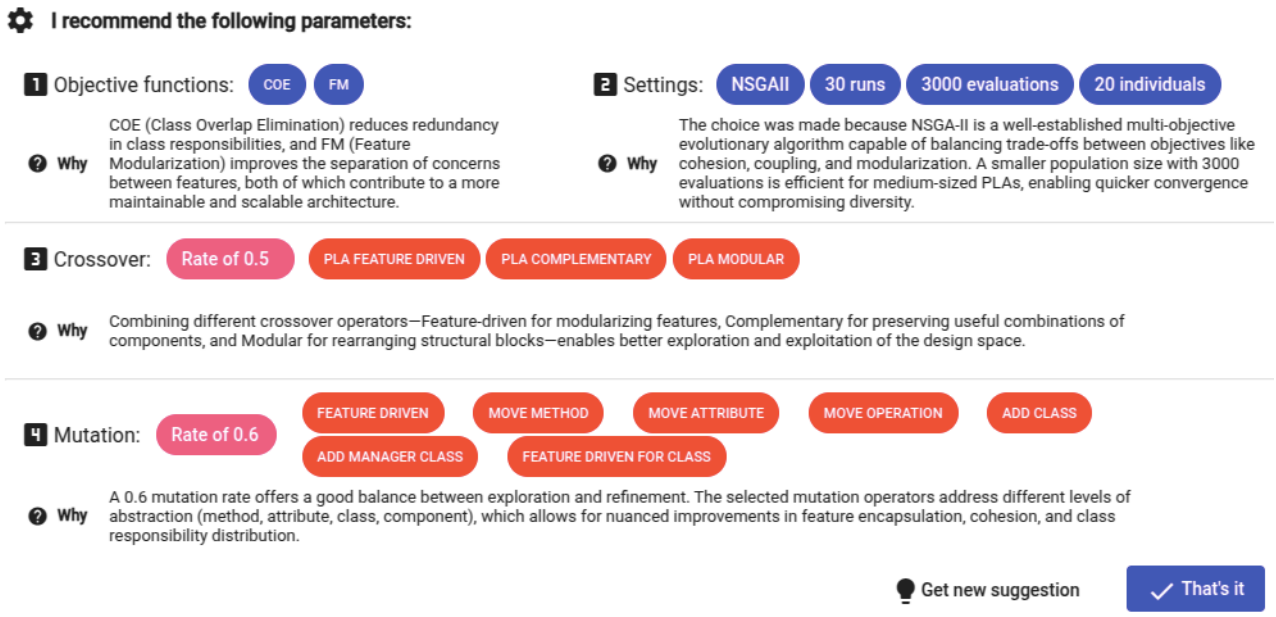

Figure 6: Getting suggestion

and contexts used in software engineering. Each LLM model has an independent training file. For our study, we trained ChatGPT-4 with all the information available about each set of parameters, compiled into a PDF (Portable Document Format) format. These training files, including the exact PDF documents and the prompts used during evaluation, are available and can be accessed through the repository provided in [13]. The information in these files includes definitions of mutation operators, crossover operators, objective functions, and basic settings of evolutionary algorithms, along with their performance in previous experiments with various configurations. In addition, the dataset includes distinct PLA projects; however, the evaluation in this preliminary study was based on prompts derived from the training data. Future evaluations should utilize entirely independent datasets.

(4) Evaluation: Testing the trained model on new, unseen data to assess its understanding and ability to generate accurate and relevant responses.
(5) Iteration: Based on performance, additional adjustments and refinements may be made to improve the model's accuracy and responsiveness.

5 Preliminary results

This section describes the preliminary evaluation of the proposed approach using OPLA-Wizard. The main goal is to evaluate the performance of the multiple-LLM approach in comparison with a single-LLM approach implemented by AIssistDM. For this end we adopt the following *Research Question (RQ)*: *How does the multiple-LLM approach compare to the single-LLM approach?*

For the multiple-LLM approach, four distinct ChatGPT-4 models were trained: OPLA-Settings, OPLA-ObjFN, OPLA-Mutation, and OPLA-Crossover . Each one specializes in recommending a specific group of parameters for OPLA-Tool: search settings, objective functions, mutation operators, and crossover operators, respectively. For the single LLM approach, only a single general-purpose ChatGPT-4 model was trained on all parameter groups to recommend the whole set of parameter configurations for OPLA-Tool.

First, we assessed the accuracy of the parameter suggestions. In this study, we define **accuracy** as the proportion of correct responses provided by the OPLA-Wizard models to the total number of questions submitted during evaluation. For each model, we submitted a set of predefined prompts and compared the model's responses with the expected answers based on the training material. Expected answers were derived from expert annotations based on prior experimental validation provided in the training documents. A response was considered *correct* if the suggested parameters were appropriate for the prompt and aligned with the training data. Otherwise, it was marked as *incorrect*. Achieving 100% accuracy indicates no deviations or errors in the parameter suggestions provided by the models. All evaluations were carried out by the researchers involved in this study.

For the multiple-LLM approach, we tested each specialized model with 10 prompts: 8 base file questions and 2 optimization questions. A base file question refers to factual knowledge explicitly in the training documents. For example: *"What crossover operators improve feature modularization?"* An optimization question requires contextual inference based on a scenario or PLA design. For example: *"Given a PLA focused on minimizing class coupling and improving feature cohesion, which search algorithm should I use?"*

Two LLM models, OPLA-Settings and OPLA-ObjFN, achieved an accuracy of 100% across the tested questions. The OPLA-Crossover model achieved an accuracy of 96,6%, while the OPLA-Mutation model maintained a consistent accuracy of 93,2%. These results were obtained after iterative refinements in the training data.

Moreover, we examined three aspects of each answer: (i) whether the output adhered to the required JSON structure, (ii) whether the explanation accompanying the suggestion was coherent and aligned with the prompt, and (iii) the clarity of the justification provided for each parameter. This evaluation also involved manual review by the researchers, who followed a checklist to ensure consistency across models. The checklist used in the review is available in [13].

We observed that all specialized models returned outputs in the correct JSON format, as required by the tool's integration layer. The explanations provided were generally aligned with the training context and offered clear reasoning for each suggestion, especially in the models focused on search settings and objective functions.

For the single-LLM approach, the model was tested with 23 prompts: 19 base file questions and four optimization questions. This model achieved 91,78% accuracy. We also observed frequent inconsistencies, including suggestions to configure some parameters with invalid values. For instance, in one scenario, the single-LLM model suggested a non-existent mutation operator in OPLA-Tool, leading to an invalid configuration. These problems persisted across replications of questions on basic algorithm settings.

> **Answer to our RQ**: The multiple-LLM approach outperformed the single LLM approach. The preliminary results demonstrated improvements in the accuracy and coherence of responses when using the multiple-LLM approach. Unlike the general-purpose model, which frequently suggested invalid parameter values, the specialized models consistently produced more precise and contextually aligned suggestions. These findings strongly support the effectiveness of the multiple-LLM approach, particularly in the complex context of parameter configuration within SBSE tools.

6 Concluding remarks

This work introduced a multiple-LLM approach to simplify SBSE tool parameter configuration for non-specialists. The approach is based on the *Divide and Conquer* strategy. The idea is to divide the complex problem of configuring SBSE tools into simpler subproblems and train LLM models separately to improve the quality of the obtained solution.

We presented OPLA-Wizard, implemented over AIssistDM, dividing parameter configuration into four specialized models: objective functions, algorithmic settings, crossover, and mutation operators. Preliminary results confirm the multiple-LLM approach significantly outperforms single-LLM, demonstrating the potential to simplify decision-making, reduce cognitive load, and increase SBSE tool accessibility.

In this work, we validated our approach using OPLA-Tool and ChatGPT. Although the core idea could be expanded to other SBSE tools and different software engineering tasks, adapting the multiple-LLM approach to these contexts will require significant additional domain-specific effort, such as defining new parameter taxonomies and preparing curated training documentation. Other LLM models could be explored in future adaptations. Furthermore, it is not strictly necessary to employ the same LLM for every sub-problem.

Therefore, several promising directions for future research emerge from our study. Specifically, we plan to: (i) conduct a more rigorous evaluation; (ii) assess the generalization to additional SBSE domains; (iii) conduct user studies evaluating practical impacts; (iv) expand evaluations with structured metrics (e.g., RAGAS framework [4]); and (v) enhance model robustness with broader datasets.

ARTIFACT AVAILABILITY

The source code, documentation, and instructions for building, installing, and usage can be obtained online [13].

ACKNOWLEDGMENTS

This research is supported by CAPES (Grant: 88887.941765/2024-00), and CNPq (Grants: 310034/2022-1, 404027/2023-7, 306774/2025-9), and Fundação Araucária (Grant PRD2023361000135).

REFERENCES

[1] Baleegh Ahmad, Shailja Thakur, Benjamin Tan, Ramesh Karri, and Hammond Pearce. 2024. On Hardware Security Bug Code Fixes By Prompting Large Language Models. *IEEE Transactions on Information Forensics and Security* (2024). doi:10.1109/TIFS.2024.3374558

[2] Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. 2024. SWE-Search: Enhancing Software Agents with Monte Carlo Tree Search and Iterative Refinement. *arXiv preprint arXiv:2410.20285* (2024).

[3] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* 18 (2013), 594–623. doi:10.1007/s10664-013-9249-9

[4] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. 150–158. doi:10.48550/arXiv.2309.15217

[5] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.

[6] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419. doi:10.1145/2025113.2025179

[7] Willian M Freire, Murilo Boccardo, Daniel Nouchi, Aline MMM Amaral, Silvia R Vergilio, Thiago Ferreira, and Thelma E Colanzi. 2024. AIssistDM: A Plugin to Assist Non-specialist Decision-Makers in Search-Based Software Engineering Tools. In *Brazilian Symposium on Software Engineering*. SBC, 734–740. doi:10.5753/sbes.2024.3567

[8] Willian M Freire, Murilo Boccardo, Daniel Nouchi, Aline MMM Amaral, Silvia R Vergilio, Thiago Ferreira, and Thelma E Colanzi. 2024. Large Language Model-based suggestion of objective functions for search-based Product Line Architecture design. In *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*. SBC, 21–30.

[9] Willian Marques Freire, Mamoru Massago, Arthur Cattaneo Zavadski, Aline Maria Malachini, Miotto Amaral, and Thelma Elita Colanzi. 2020. OPLA-Tool v2. 0: a tool for product line architecture design optimization. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*. 818–823. doi:10.1145/3422392.3422498

[10] Mark Harman and Bryan F Jones. 2007. The current state and future of search based software engineering. *Future of Software Engineering* (2007), 342–357.

[11] William B. Langdon and Mark Harman. 2015. Optimizing Existing Software With Genetic Programming. *IEEE Transactions on Evolutionary Computation* 19, 1 (2015), 118–135. doi:10.1109/TEVC.2013.2281544

[12] Bingdong Li, Zixiang Di, Yanting Yang, Hong Qian, Peng Yang, Hao Hao, Ke Tang, and Aimin Zhou. 2024. It's Morphing Time: Unleashing the Potential of Multiple LLMs via Multi-objective Optimization. *arXiv preprint arXiv:2407.00487* (2024).

[13] Daniel Nouchi, Willian Marques Freire, Aline Maria Malachini Miotto Amaral, Silvia Regina Vergilio, and Thelma Elita Colanzi. 2025. Complementary Material. (2025). https://doi.org/10.6084/m9.figshare.28930388.v2

[14] Yanyan Peng, Xinjie Wang, Ziyang Chen, Hong Liu, Ge Li, Xin Xia, and Zhi Jin. 2024. OrcaLoca: Enhancing LLM Agents for Code Fault Localization. In *Proceedings of the 46th International Conference on Software Engineering (ICSE)*.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008. doi:10.48550/arXiv.1706.03762

[16] Yenisei D. Verdecia, Thelma E. Colanzi, Silvia R. Vergilio, and Marcelo C.B. dos Santos. 2017. An Enhanced Evaluation Model for Search-based Product Line Architecture Design.. In *CIbSE*. 155–168.

[17] Chen Yang, Junjie Chen, Bin Lin, Jianyi Zhou, and Ziqi Wang. 2024. Enhancing LLM-based Test Generation for Hard-to-Cover Branches via Program Analysis. *arXiv preprint arXiv:2404.04966* (2024). doi:10.48550/arXiv.2404.04966

[18] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1592–1604.