POLITECHNIKA GDAŃSKA

The author of the doctoral dissertation: Hammed Adeleye Mojeed

Scientific discipline: Technical Informatics and Telecommunications

DOCTORAL DISSERTATION

Title of doctoral dissertation: An Interactive Machine Learning-Based Multi-objective Optimization Framework for Software Overtime Planning

Title of doctoral dissertation (in Polish): Wielokryterialna, interaktywna i oparta na uczeniu maszynowym metoda optymalizacji planowania nadgodzin w projektach informatycznych

Supervisor *signature*
dr hab. inż. Rafał Szłapczyński

Gdańsk, 2025

POLITECHNIKA GDAŃSKA

STATEMENT

The author of the doctoral dissertation: Hammed Adeleye Mojeed

I, the undersigned, declare that I am aware that in accordance with the provisions of Art. 27 (1) and (2) of the Act of 4th February 1994 on Copyright and Related Rights (Journal of Laws of 2021, item 1062), the university may use my doctoral dissertation entitled:

An Interactive Machine Learning-Based Multi-objective Optimization Framework for Software Overtime Planning

for scientific or didactic purposes.[1]

Gdańsk,..... .....

signature of the PhD student

Aware of criminal liability for violations of the Act of 4th February 1994 on Copyright and Related Rights and disciplinary actions set out in the Law on Higher Education and Science(Journal of Laws 2021, item 478), as well as civil liability, I declare, that the submitted doctoral dissertation is my own work.

I declare, that the submitted doctoral dissertation is my own work performed under and in cooperation with the supervision of dr hab. inż. Rafał Szłapczyński.

This submitted doctoral dissertation has never before been the basis of an official procedure associated with the awarding of a PhD degree.

All the information contained in the above thesis which is derived from written and electronic sources is documented in a list of relevant literature in accordance with Art. 34 of the Copyright and Related Rights Act.

I confirm that this doctoral dissertation is identical to the attached electronic version.

Gdańsk,..... .....

signature of the PhD student

I, the undersigned, agree/~~do not agree~~* to include an electronic version of the above doctoral dissertation in the open, institutional, digital repository of Gdańsk University of Technology.

Gdańsk,..... .....

signature of the PhD student

**delete where appropriate*

[1] Art 27. 1. Educational institutions and entities referred to in art. 7 sec. 1 points 1, 2 and 4–8 of the Act of 20 July 2018 – Law on Higher Education and Science, may use the disseminated works in the original and in translation for the purposes of illustrating the content provided for didactic purposes or in order to conduct research activities, and to reproduce for this purpose disseminated minor works or fragments of larger works.

2. If the works are made available to the public in such a way that everyone can have access to them at the place and time selected by them, as referred to in para. 1, is allowed only for a limited group of people learning, teaching or conducting research, identified by the entities listed in paragraph 1.

POLITECHNIKA GDAŃSKA

DESCRIPTION OF DOCTORAL DISSERTATION

The Author of the doctoral dissertation: Hammed Adeleye Mojeed

Title of doctoral dissertation: An Interactive Machine Learning-Based Multi-objective Optimization Framework for Software Overtime Planning

Title of doctoral dissertation in Polish: Wielokryterialna, interaktywna i oparta na uczeniu maszynowym metoda optymalizacji planowania nadgodzin w projektach informatycznych

Language of doctoral dissertation: English

Supervisor: dr hab. inż. Rafał Szłapczyński

~~**Second supervisor***: <first name, surname >~~

~~**Auxiliary supervisior***: <first name, surname >~~

~~**Cosupervisior***: <first name, surname >~~

Date of doctoral defense:

Keywords of doctoral dissertation in Polish: Optymalizacja interaktywna, optymalizacja wielokryterialna, wielokryterialny algorytm shuffled frog-leaping, uczenie maszynowe, zachłanna walidacja krzyżowa, las losowy.

Keywords of doctoral dissertation in English: Interactive Optimization, Multi-objective Optimization, Multi-objective Shuffled Frog-leaping Algorithm, Machine learning, Greedy Cross-validation, Random Forest.

Summary of doctoral dissertation in Polish:

Problemy z przekraczaniem założonego terminu realizacji projektów informatycznych, wciąż często spotykane w świecie IT, uwidaczniają konieczność zmian w podejściu do zarządzania tego typu projektami. Metody zarządzania powinny w większym stopniu wykorzystywać podejście adaptacyjne oraz zorientowane na członków zespołu informatycznego. Niniejsza rozprawa podejmuje problem ograniczeń w planowaniu nadgodzin w projektach informatycznych przez metody oparte na wyszukiwaniu, które często pomijają subiektywne preferencje kierowników projektów. W pracy zaproponowano nowatorskie, interaktywne środowisko planowania nadgodzin wykorzystujące mechanizmy optymalizacji wielokryterialnej oraz uczenia maszynowego. W rozprawie zaproponowano i przedstawiono algorytm ML-iMOSFLA – interaktywną, wielokryterialną wersję algorytmu Shuffled Frog-Leaping Algorithm - SFLA, wzbogaconą o model regresji lasu losowego wytrenowany na planach nadgodzin oznaczonych przez ekspertów. Zadaniem tego modelu jest sterowanie procesem optymalizacji w celu osiągnięcia rozwiązań nie tylko optymalnych, lecz również preferowanych. Przeprowadzono systematyczny przegląd literatury w celu oceny aktualnego stanu wiedzy i

zastosowań metod optymalizacji wielokryterialnej oraz uczenia maszynowego w planowaniu projektów informatycznych. Wykazano przy tym istotne luki badawcze w zakresie metod planowania nadgodzin o charakterze interaktywnym oraz metod uwzględniających preferencje. Aby je wypełnić, opracowano model predykcyjny wykorzystujący regresję lasu losowego optymalizowaną przy użyciu zachłannego algorytmu przeszukiwania siatki (greedy halving grid search). Model predykcyjny wytrenowano na nowo stworzonym zbiorze danych zawierającym plany nadgodzin, oznaczone przez ekspertów, pochodzące z sześciu rzeczywistych projektów informatycznych. Model ten wykazał wyższą skuteczność we wszystkich analizowanych metrykach i utrzymywał wysoką dokładność predykcji przy różnych poziomach złożoności projektów. Algorytm ML-iMOSFLA został empirycznie porównany z klasycznym algorytmem MOSFLA oraz wariantem tego algorytmu wymagającym udziału człowieka (Human-in-the-Loop). Wyniki pokazały, że algorytm ML-iMOSFLA przewyższa metody bazowe zarówno pod względem wartości uzyskiwanych przez obiektywne wskaźniki jakości, jak i subiektywnej zgodności z preferencjami kierowników projektów, jednocześnie istotnie ograniczając konieczność ciągłej interakcji człowieka. Uzyskane rezultaty potwierdzają zasadność integracji uczenia maszynowego z interaktywnymi środowiskami optymalizacji dla celów planowania nadgodzin w projektach informatycznych. Integracja ta daje możliwość skalowalnego, efektywnego i zgodnego z potrzebami człowieka podejścia do planowania projektów.

Summary of doctoral dissertation in English:

The persistent challenge of project overruns in software development has underscored the need for more adaptive and human-centric planning methodologies. This thesis addresses the limitations of conventional search-based software overtime planning (SOP) approaches, which often neglect the subjective preferences of project managers, by proposing a novel machine learning–based interactive multi-objective optimization framework. The thesis introduces ML-iMOSFLA, an interactive multi-objective shuffled frog-leaping algorithm that integrates a random forest regression model trained on expert-annotated overtime plans to guide the optimization process toward solutions that are not only objectively optimal but also preferred. A comprehensive systematic mapping study was conducted to assess the current landscape of multi-objective optimization and machine learning applications in software project planning, revealing significant gaps in interactive and preference-aware software overtime planning methods. To address these gaps, the thesis develops a predictive model using a greedy halving grid search–optimized random forest regression, trained on a novel dataset of annotated overtime plans derived from six real-world software projects. The model demonstrated superior performance across multiple regression metrics and maintained high predictive accuracy across varying project complexities. The ML-iMOSFLA algorithm was empirically evaluated against traditional MOSFLA and a Human-in-the-Loop variant. Results showed that ML-iMOSFLA consistently outperformed baseline methods in both objective quality indicators and subjective alignment with PM preferences, while significantly reducing the need for continuous human interaction. These findings affirm the viability of integrating ML into interactive optimization frameworks for SOP, offering a scalable, efficient, and human-aligned approach to software project planning.

**delete where appropriate*

EXTENDED ABSTRACT IN ENGLISH

This doctoral thesis presents a novel framework for software overtime planning (SOP) that integrates machine learning (ML) with interactive multi-objective optimization to address persistent challenges in software project management, particularly the misalignment between algorithmically optimal solutions and project managers' (PMs) subjective preferences. The research is motivated by the limitations of existing search-based SOP methods, which often produce technically sound solutions but lack practical usability due to their disregard for human-centric decision-making.

This study conducted a systematic mapping of 71 high-quality studies across software effort estimation (SEE), software project scheduling (SPS), and SOP. This mapping revealed that while multi-objective optimization (MOO) and ML have been extensively applied in SEE and SPS, SOP remains underexplored, especially in interactive contexts. Notably, only four studies addressed SOP using MOO, and none incorporated ML or preference-aware optimization, highlighting a significant research gap.

To address the gap, the thesis introduces ML-iMOSFLA—a Machine Learning–based Interactive Multi-Objective Shuffled Frog Leaping Algorithm. This algorithm integrates a Random Forest Regression (RFR) model trained on a novel dataset of 1,622 annotated overtime plans derived from six real-world software projects (ACAD, WEBMET, PSOA, WEBAMHS, PARM, OPMET). The dataset was constructed through expert annotation by 20 PMs, with outlier removal performed using a clustering-based approach. The RFR model was optimized using a novel greedy halving grid search (gHGS) method, which significantly enhanced its predictive performance and computational efficiency.

Compared to other ML regression models, including linear, kernel-based, neural network, and gradient boosting regressors, RFR achieved the best average performance in terms of mean absolute error, mean square error, root mean square error, mean magnitude of relative error, and R-squared metrics across the six datasets. This formed the basis for its selection in building the interactive model. Moreover, the RFR-gHGS model achieved substantial improvements over the baseline RFR model and RFR optimized with the benchmark hyperparameter optimization methods, including grid search (GS), random search (RS), Bayesian optimization (BO), and the original HGS. The model also maintained stable predictive accuracy across increasing project complexity with Adjusted-R^2 values ranging narrowly from 0.87 to 0.91, underscoring its better scalability compared to the ordinary RFR model (Adjusted-R^2 values ranges from 0.57 to 0.86).

The ML-iMOSFLA algorithm was evaluated through three levels of experimentation. In standalone validation, it produced solutions with low error rates and high alignment with PM preferences, with 45.71% to 64.06% of PMs ranking its top solution as their first choice. Compared to the baseline MOSFLA, ML-iMOSFLA outperformed in contribution, inverted generational distance, and hypervolume indicators, and produced superior results in interactive performance metrics, including Multi-objective Similarity Degree (MoSD), Similarity Factor (MoSF), and Price of Preference (MoPP). When benchmarked against a Human-in-the-Loop variant (HIL-iMOSFLA), ML-iMOSFLA matched or

exceeded performance at higher iteration levels (150–200), while eliminating the need for continuous human interaction.

Key findings from the study include the following:

1. The baseline RFR model demonstrated strong predictive performance but showed sensitivity to project complexity, which was effectively mitigated through gHGS optimization.
2. The RFR-gHGS model consistently outperformed other ML models and hyperparameter tuning methods in both accuracy and computational efficiency.
3. ML-iMOSFLA produced more preferred solutions than traditional MOSFLA and achieved parity with HIL-iMOSFLA at higher iterations, validating its ability to replace continuous human feedback.
4. The framework proved scalable and adaptable across diverse project environments, maintaining high solution quality and preference alignment.

In conclusion, this thesis advances the field of Search-Based Software Project Management (SBSPM) by demonstrating the feasibility and effectiveness of ML-driven interactive optimization for SOP. The proposed framework offers an efficient and project manager-aligned solution to one of the most pressing challenges in software project management—delivering projects on time, within budget, and with high quality. Future work may explore its extension to Agile environments, dynamic learning models, and broader applications in resource-constrained project planning.

ACKNOWLEDGEMENT

All gratitude is due to the Almighty God for granting me the strength, perseverance, and clarity of purpose to bring this doctoral journey to a successful conclusion.

I would like to express my profound appreciation to my supervisor, Professor Rafał Szłapczyński, for his exceptional guidance, unwavering support, and insightful mentorship throughout this research project. His deep knowledge, critical feedback, and encouragement have been instrumental in shaping the quality and direction of this work. I am especially thankful for his patience and for challenging me to think rigorously and independently.

To my beloved wife, Barakat Abdulazeez, I owe a debt of gratitude that words can hardly express. Her unwavering love, patience, and emotional support have been my anchor during the most challenging parts of this journey. Her sacrifices, encouragement, and faith in my potential have continually been a source of strength and motivation.

LIST OF PUBLICATIONS ACHIEVED DURING DOCTORAL STUDIES

The following publications were achieved from the doctoral research topic:

Mojeed, H.A., Szlapczynski, R. (2025). Learning Software Overtime Estimation from Experts' Annotations: A Greedy Cross-Validation-Based Machine Learning Approach. *IEEE Access*, *13*, 150068-150090 doi: https://doi.org/10.1109/ACCESS.2025.3601820 (Impact Factor: 3.6, Polish Ministry Point: 100)

Mojeed, H.A., Szlapczynski, R., Balogun, A.O., Capretz L.F. Machine Learning and Multi-Objective Optimization Algorithms in Software Project Planning: A Systematic Mapping Study on Applications, Interactions and Methodologies: Under review in *Scientific Reports* Journal. (Impact Factor: 3.9, Polish Ministry Point: 140)

Mojeed, H.A. & Szlapczynski, R. (2024). A Machine Learning Approach for Estimating Overtime Allocation in Software Development Projects. In B. Marcinkowski, A. Przybylek, A. Jarzębowicz, N. Iivari, E. Insfran, M. Lang, H. Linger, & C. Schneider (Eds.), *Harnessing Opportunities: Reshaping ISD in the post-COVID-19 and Generative AI Era (ISD2024 Proceedings)*. Poland: ISBN: 978-83-972632-0-8. https://doi.org/10.62036/ISD.2024.4 (CORE Ranking : A, Polish Ministry Point: 140)

Mojeed, H.A. & Szlapczynski, R. (2023). Machine Learning Assisted Interactive Multi-objectives Optimization Framework: A Proposed Formulation and Method for Overtime Planning in Software Development Projects. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds) *Artificial Intelligence and Soft Computing. ICAISC 2023* https://doi.org/10.1007/978-3-031-42505-9_35 (CORE Ranking: National, Polish Ministry Point: 20)

The following publications were achieved from research collaborations:

Shaheed, K., Szczuko, P., Ullah, I., **Mojeed, H.A.**, Balogun, A.O. & Capretz, L.F. (2024). Finger Vein Presentation Attack Detection Method Using a Hybridized Gray-Level Co-Occurrence Matrix Feature with Light-Gradient Boosting Machine Model. In B. Marcinkowski, A. Przybylek, A. Jarzębowicz, N. Iivari, E. Insfran, M. Lang, H. Linger, & C. Schneider (Eds.), *Harnessing Opportunities: Reshaping ISD in the post-COVID-19 and Generative AI Era (ISD2024 Proceedings).* Gdansk, Poland: University of Gdansk. ISBN: 978-83-972632-0-8. https://doi.org/10.62036/ISD.2024.54 (CORE Ranking: A, Polish Ministry Point: 140)

Usman-Hamza F. E., Balogun A. O., Amosa R. T., Capretz L. F., **Mojeed H.A**, Salihu S. A., Akintola A. G., Mabayoje M. A. (2024), Sampling-based novel heterogeneous multi-layer stacking ensemble method for telecom customer churn prediction. *Scientific African*, 24, s.e02223. https://doi.org/10.1016/j.sciaf.2024.e02223 (Impact Factor: 3.3, Polish Ministry Point: 20)

Adewole, K. S., **Mojeed, H. A.**, Ogunmodede, J. A., Gabralla, L. A., Faruk, N., Abdulkarim, A., Ifada, E., Folawiyo, Y. Y., Oloyede, A. A., Olawoyin, L. A., Sikiru, I. A., Nehemiah, M., Gital, A. Y., & Chiroma, H. (2022). Expert System and Decision Support System for Electrocardiogram Interpretation and Diagnosis: Review, Challenges and Research Directions. *Applied Sciences*, *12*(23), 12342. https://doi.org/10.3390/app122312342 (Impact Factor: 2.5, Polish Ministry Point: 100)

Usman-Hamza, F. E., Balogun, A. O., Capretz, L. F., **Mojeed, H. A**., Mahamad, S., Salihu, S. A., Akintola, A. G., Basri, S., Amosa, R. T., & Salahdeen, N. K. (2022). Intelligent Decision Forest Models for Customer Churn Prediction. *Applied Sciences, 12(16), 8270*. https://doi.org/10.3390/app12168270 (Impact Factor: 2.5, Polish Ministry Point: 100)

LIST OF IMPORTANT SYMBOLS AND ABBREVIATIONS

Abbreviation	Meaning
ANN	Artificial Neural Network
CPM	Critical Path Management
DM	Decision Maker
DT	Decision Tree
FP	Function Point
GB	Gradient Boosting
gHGS	Greedy Halving Grid Search
HIL	Human-in-the-Loop
I_C	Contribution Indicator
I_{HV}	Hypervolume Indicator
I_{iGD}	Inverted Generational Distance
iMOSFLA	Interactive Multi-Objective Shuffled Frog Leaping Algorithm
LASO	Least Absolute Shrinkage and Selection Operator
KNN	K-Nearest Neighbors
MAE	Mean Absolute Error
MAR	Margarine Strategy
ML	Machine Learning
ML-iMOSFLA	Machine Learning–based Interactive Multi-Objective Shuffled Frog Leaping Algorithm
MLP	Multi-Layer Perceptron
MLR	Multiple Linear Regression
MMRE	Mean Magnitude of Relative Error
MOO	Multi-Objective Optimization
MoPP	Multi-objective Price of Preference
MoSD	Multi-objective Similarity Degree
MoSF	Multi-objective Similarity Factor
MOSFLA	Multi-Objective Shuffled Frog Leaping Algorithm
MSE	Mean Squared Error
MSLE	Mean Squared Logarithmic Error
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OMS	Overtime Management Strategies
R^2	Coefficient of Determination
RF	Random Forest
RFR	Random Forest Regression
RMSE	Root Mean Squared Error
SBSE	Search-Based Software Engineering
SEE	Software Effort Estimation
SH	Second Half Strategy
SOP	Software Overtime Planning
SPM	Software Project Management
SPP	Software Project Planning
SPS	Software Project Scheduling
SVR	Support Vector Regression
WP	Work Package

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

Software engineering employs systematic methods for developing software. According to IEEE standard [1], it is “the application of a systematic, disciplined, and quantifiable approach to software development, operation, and maintenance.” The process includes converting user needs into software requirements, translating those requirements into design, implementing the design through coding, testing the code, deploying the software, and maintaining the product throughout its lifecycle. The objective is to develop software that is rapidly built, cost-efficient, easily testable, dependable, scalable, and maintainable [2]. Therefore, software development projects require effective management and planning to improve the outcome of software processes. Inappropriate planning and/or poor management can cause unnecessary delays and increased overhead costs, resulting in durations and budgets that are often unacceptable, which can in turn lead to critical business failures.

Software Project Management (SPM) involves several activities critical to a software project’s success. Such activities include, among others, cost/effort estimation, project planning, resource allocation, quality management, and risk management [2,3]. To improve software production processes, these activities often occur in approximately all software lifecycle stages [4]. Nevertheless, several studies have reported a strong link between the success of software projects and management activities conducted at the planning stage [5–7]. This is because significant decisions that affect the subsequent stages are made at that stage. Therefore, developing effective project planning methods is essential to delivering successful software projects. Software Project Planning (SPP) is a field in SPM that seeks to design methods for executing and controlling the progress of software development processes to guarantee timely delivery and reduced cost while maintaining the required quality [5,8,9]. It involves a number of interdependent tasks (major tasks depicted in Figure 1.1) that must be completed to produce a schedule plan and roadmap for executing the software project [4,10]. The size estimation is pivotal in assessing other activities, such as cost (effort) estimation and development time estimation. Resource allocation requires that cost and time estimation be completed, which, along with development time, is necessary to build a schedule for software projects.

The complexity of software projects underscores the importance of using computer-aided tools or methods for effective planning, as research shows that 60-80% of software development projects are completed late [11,12]. As an optimization problem, SPP has garnered significant interest from Search-Based Software Engineering (SBSE) researchers over the past decades, particularly through the application of metaheuristic optimization algorithms to develop project planning tools and methods [13–17]. However, when these tools prove inadequate or the project experiences unforeseen ‘mission creep’ as a result of time constraints and/or late requirement changes, Project Managers (PM) frequently depend on overtime allocation [2]. According to a recent report by Statista [18], more than 50% of developers experience unplanned accumulated overtime up to two days a month, as depicted in Figure 1.2. Moreover, excessive overtime has been consistently reported to result in heightened stress levels among developers [19–22] and in the production of low-quality software [23–27]. Due to this, studies in

search-based SPP consider integrating overtime allocation into the project schedule to mitigate potential uncertainties or risks of failure during the software development process.

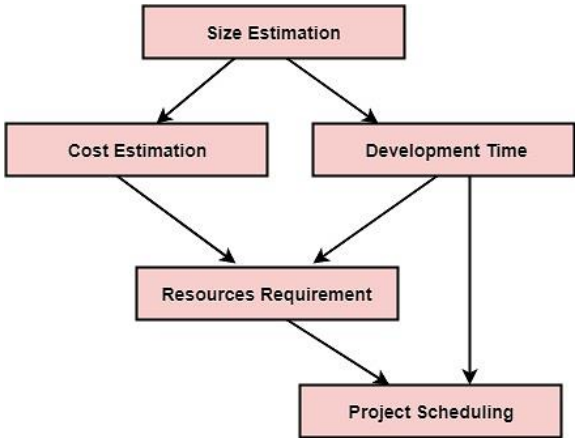


Figure 1.1. Major tasks in SPP and their relationships [28]

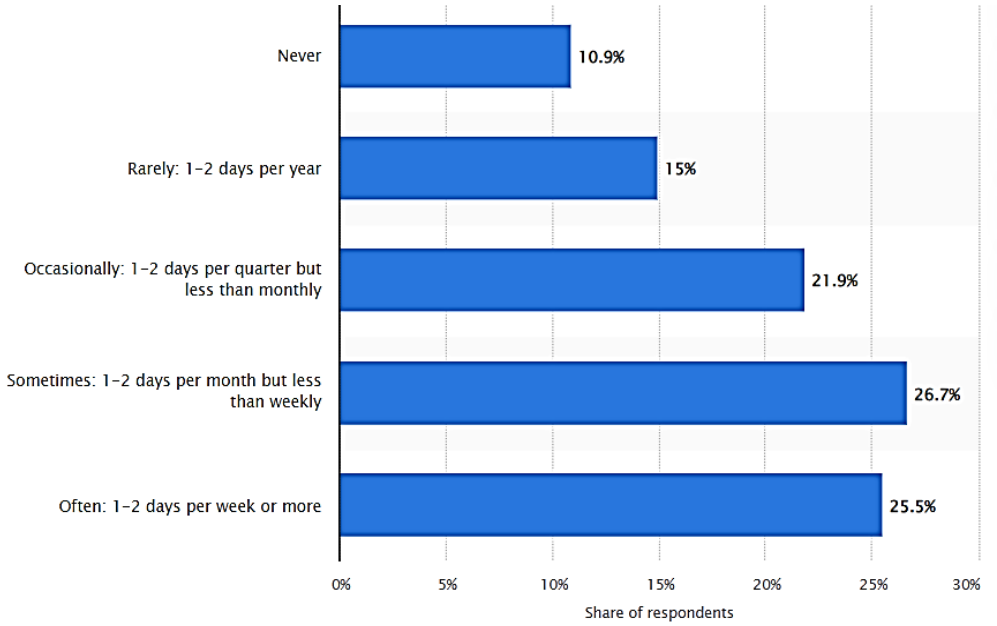


Figure 1.2. Frequency of developers working overtime as of 2020. Source: [18]

In SPM, several Overtime Management Strategies (OMS) are deployed to mitigate project overrun risks. Ferrucci et al. [21] reported three industry-based OMS widely used by PMs: MARgarine (MAR), Critical Path Management (CPM), and Second Half (SH) strategies. MAR evenly distributes overtime hours across all activities in the project schedule. It adopts a structured approach, which is planned a priori and based on a balanced workload. However, it is not effective when the deadline is critical. In CPM, overtime hours are assigned only to the activities covered by the schedule's critical path. It employs a more focused approach, prioritizing critical path tasks. It is applicable only when critical tasks are clearly defined in the project schedule. SH uses a more dynamic and reactive approach

by loading overtime hours only onto the activities in the latter half of the schedule. Considering research in Software Overtime Planning (SOP), recent studies have approached it as a multi-objective optimization problem that incorporates the side effects of overtime as objectives [21,22,29,30]. The studies deployed search-based multi-objective meta-heuristic algorithms to find optimal overtime plans and compare their results with industry-based OMS strategies. The insights derived from the equally good solutions (overtime plans) produced by these algorithms provide PMs with more flexibility in managing overtime in software projects than conventional practices [22,29].

While the planning of overtime is contingent upon the project management methodology adopted in software development, Agile methodologies have emerged as the predominant approach within the field, as evident in the literature [31–33]. Nonetheless, the scalability of Agile practices for large-scale software development teams remains a significant challenge, primarily due to certain demotivating factors identified in prior studies [34,35]. This obstacle has recently influenced the adoption of Agile strategies in the planning of extensive software development projects. The traditional approach remains in use in numerous countries and organizations, particularly in Asia and Africa. In light of these observations and the difficulties encountered in accessing datasets related to Agile-based software projects, this study concentrates on planning overtime within the framework of the classical development methodology.

1.1 Research Problem

State-of-the-art SOP methods incorporate overtime allocation directly into the project schedule, balancing optimal overtime, makespan, costs, and risk of overruns. They often use multi-objective search-based optimization techniques, such as NSGA II [36,37], memetic MOSFLA [30], and Adaptive NSGA II [38]. These algorithms provide suitable solutions for PMs to aid their decisions in overtime planning without considering their subjective preferences. Additionally, they generate many equally optimal solutions, which may necessitate further decision-making analysis, such as Multi-Criteria Decision Making (MCDM). Existing research has demonstrated that relying solely on explicit solutions provided by automated tools for software engineering problems is not a practical approach [2,38,39]. PMs usually prefer to follow their own judgments about the appropriate solutions because managing software projects involves implicit, subjective decisions that automated tools cannot fully handle. This has limited their application in software project management environments. Therefore, there is a need for interactive overtime planning approaches that can bridge the gap between algorithm effectiveness and real-world application.

There are studies in the literature on interactive multi-objective optimization techniques that combine search-based optimization with real-time DM feedback in software engineering problems such as requirement engineering, design, testing, refactoring, and maintenance [39–44]. These studies modeled the interactions between search algorithms and DMs as either direct or semi-indirect. In direct interaction, the human-in-the-loop (HIL) method is used, and the DM provides feedback at each optimization step to guide the algorithm toward preferred regions or solutions. In semi-indirect

interaction, the DM provides initial solutions, and the search algorithm finds the optimal solutions closest to those specified by the DM. These interactions could lead to severe and overwhelming fatigue due to a prolonged physical involvement of the DM in guiding the search for an optimal solution. Additionally, accurate modeling of DM preferences, particularly the subjective ones, remains a persistent problem in interactive multi-objective optimization. There is a need for an interactive approach capable of modeling the subjective preferences of DM without prolonged physical involvement, thereby ensuring indirect interaction.

In this thesis, a framework is designed to fill the identified gaps by incorporating a machine learning model trained on DM's preferences into the optimization steps of a search-based multi-objective evolutionary algorithm. Machine learning has been recognized as an effective approach to enhance SBSE across different applications [45–47]. Furthermore, to the best of my knowledge, no research has been conducted on the development of interactive search-based optimization methods within SPP, particularly in the context of overtime planning. Therefore, this study proposes an interactive multi-objective optimization framework applied to software overtime planning, integrating a memetic search-based optimization algorithm and a Machine learning predictive model to produce solutions that simultaneously optimize explicit objectives, satisfy constraints, and satisfy the PM's preferences.

1.2 Motivation and Justification of the Research Problem

State-of-the-art approaches in SOP have largely relied on multi-objective evolutionary optimization approaches [21,22], [29], and multi-objective memetic optimization algorithms [30] to achieve an optimal allocation of overtime that yields a more balanced project schedule compared to existing OMS. However, despite the technical advancements these methods offer, their adoption in real-world software project environments is limited due to usability issues and a lack of integration with the subjective judgment of PMs.

Recent successes in interactive optimization problems within software engineering have demonstrated that the synergetic combination of machine learning models with multi-objective optimization algorithms can significantly enhance solution quality and relevance [39,40,44,48]. This work will extend the state-of-the-art in overtime planning by incorporating PM subjective judgment into the optimization process of search-based meta-heuristic algorithms using machine learning models. Motivated by these outcomes, this research proposes an extension to the current state-of-the-art by developing an interactive machine learning–based framework that not only optimizes overtime allocation but also incorporates PMs' subjective judgments directly into the optimization process. This integration is expected to bridge the gap between algorithmic efficiency and practical usability, as it allows PMs to adjust and guide the search process according to context-specific criteria, ultimately resulting in solutions that are both optimal and practically acceptable.

By explicitly embedding PM insights into the multi-objective optimization loop, the proposed framework aims to produce overtime allocation plans that better reflect real-world software projects' complexities and dynamic priorities. This interactive approach not only promises to enhance the overall

decision-making process but also to facilitate broader adoption in practice, where human judgment and contextual expertise remain indispensable.

1.3 Research Objectives

This thesis aims to develop a machine learning-based interactive multi-objective algorithm (ML-IMA) for optimal overtime planning in software development projects.

The specific objectives for achieving the aim are to:

1. Conduct an extensive systematic literature review on the application and integration of multi-objective optimization algorithms and machine learning in SPP
2. Design a machine learning-based interactive multi-objective optimization framework for incorporating DM's subjective preferences
3. Develop a machine learning predictive model for estimating the PM's subjective evaluation of overtime allocations
4. Develop an interactive Multi-Objective Shuffled Frog-Leaping Algorithm (iMOSFLA) that integrates the developed machine learning model built in (3) for optimizing SOP
5. Empirically evaluate the developed interactive multi-objective optimization algorithm on different software development projects

1.4 Research Questions

Reflecting on the research objectives and focusing on how a machine learning model can effectively substitute for continuous human-in-the-loop feedback in overtime planning for software projects, the following research questions are posed:

RQ1. How effectively can a machine learning predictive model capture and replicate project managers' subjective preferences in overtime planning?

This question examines the model's ability to learn from historical decisions and reflect the nuanced judgment of PMs.

RQ2. To what extent can the integration of a machine learning model into an interactive multi-objective memetic optimization algorithm replace continuous human-in-the-loop interaction?

This question examines whether the machine learning model can autonomously guide the optimization process without requiring ongoing manual intervention.

*RQ3. How do overtime planning solutions generated by the machine learning-based interactive optimization approach compare with those developed through traditional human-in-the-loop methods in terms of project overtime, cost, and quality metric***?**

This question aims to evaluate the technical performance and practical outcomes of the proposed approach against conventional practices.

RQ4. How scalable and adaptable is the proposed machine learning–driven optimization framework when applied to diverse software development project environments?

This question examines the framework's ability to accommodate varying project complexities and requirements.

1.5 Organization of the Thesis

The remaining part of the thesis is organized as follows:

Chapter two focuses on a review of the state-of-the-art in software overtime planning. It discusses important terminologies in software project management and reports the systematic mapping study carried out on the application of machine learning and multi-objective optimization in software project planning. The chapter concludes by analyzing closely related work in comparison with the proposed approach in this thesis.

Chapter three presents the methodology followed in the realization of the research objectives. Firstly, it describes the proposed machine learning-based interactive multi-objective optimization framework and discusses how the framework is adapted to SOP. It also presents the procedure of a specific interactive algorithm: ML-iMOSFLA, designed for optimal and preference-respecting overtime planning in software projects. The chapter also describes the software data used for evaluation and concludes with the definition of performance metrics used.

Chapter four focuses on the empirical evaluation of the ML component of the proposed algorithm. It discusses the steps followed to produce training datasets for building ML models for overtime estimation. Furthermore, it presents the results of various ML models on the produced dataset, the designed greedy halving grid search hyperparameter optimization method, and its performance comparison against the benchmarked hyperparameter tuning methods. The chapter concludes with the findings drawn from the evaluation results.

Chapter 5 reports the results of the performance evaluation of the proposed ML-iMOSFLA. It discusses the implementation details of the algorithm, the setting of its parameters, the stand alone evaluation of the algorithm. It also compares the performance of the algorithm with traditional MOSFLA and the human-in-the-loop interactive version of MOSFLA. The chapter ends with the findings extracted from the results.

Chapter 6 concludes the thesis by presenting answers to the research questions posed, the summary of key findings, the significance and implication of the findings from the study as well as its limitation. The chapter ends with the overall conclusions of the study and the future works.

CHAPTER 2 LITERATURE REVIEW

Software development is a complex task requiring systematic engineering methodologies to build reliable, scalable, and maintainable systems. According to the most recent CHAOS report [49], only one out of three software projects (based on the analysis of 50,000 projects globally) is completed successfully, and 47% of the projects are delivered with some challenges. This means that most software projects are still challenged with late delivery, are over budget, and deliver fewer-than-expected required features and functionalities, even with the availability of advanced tools and methodologies. This may be because, over time, project and environment complexity have increased, and delivery time has been reduced. Consequently, software projects require proper and efficient management approaches to guarantee success. Software Project Management (SPM) encompasses several critical activities essential to a software project's success. Such activities include, among others, cost/effort estimation, project planning, resource allocation, quality management, and risk management [2,3]. To optimize software production processes, these activities often occur in nearly all stages of the software lifecycle [4]. Nevertheless, several studies have reported a strong link between the success of software projects and management activities conducted during the planning stage [5–7], as significant decisions that affect subsequent stages are made at this stage. Therefore, developing effective project planning methods is essential to delivering successful software projects.

SPP and its various activities have been thoroughly studied by software engineering researchers [2,50]. Two major computational intelligence approaches, search-based optimization algorithms and machine learning, have gained increased application for decades in SPP [4–8,51–53]. A search-based optimization algorithm views SPP as an optimization problem and applies meta-heuristic algorithms guided by a fitness function to search for near-optimal solutions. In contrast, ML views it as an estimation or classification problem and fits machine learning models with historical software data to predict software planning metrics.

SBSE [54] – a research field that approaches software engineering problems as a search optimization problem and aims at finding optimal solutions that satisfy some quality conditions – has attracted a variety of applications in software project management [2,5,55]. Researchers in SBSE have introduced different formulations of the problem and applied different variants of meta-heuristic optimization algorithms [56–61]. An extensive survey on these formulations (single and multi-objective) and algorithms was carried out by Ferrucci et al. [2] in 2014, making the Search-Based Software Project Management field more prominent. Their work described how SBSE had been used to address optimization problems that Project Managers (PMs) deal with, including those related to SPP. Other review/mapping studies identified in this domain focused on one specific task of SPP, such as software project scheduling [5,8,62] and for effort, defect, and change proneness prediction [63]. In contrast to the existing review studies, and because SPP problems are often characterized by many and usually conflicting objectives (such as delivery time and cost), PMs often seek multi-objective solutions to SPP to capture the realistic environment in the industry [21].

2.1 Software Project Scheduling (SPS)

SPS focuses on the optimal allocation of software project tasks to time slots and the optimal assignment of developers to tasks. The goal is to build a schedule for executing the software project in a way that guarantees minimum cost in terms of developer salaries and minimum duration in terms of completion time. The fusion of task ordering and human resource allocation leads to a complex combinatorial optimization problem regarded as NP-hard [3,4,64–66]. Most studies refer to the problem as a Software Project Scheduling Problem(SPSP). Solving the problem manually has been fraught with error, time consumption, and fatigue [66], and finding an exact solution has proved nearly impossible [3,53,67,68]. Therefore, search-based software engineering approaches using meta-heuristic optimization algorithms have been deployed to find near-optimal solutions to the problem [2]. Applying meta-heuristic optimization algorithms to software task scheduling and human resource allocation forms the basis for the search-based software project management scheme presented in Figure 2.1, as proposed by Harman, Mansouri, and Zhang [54] in 2012.

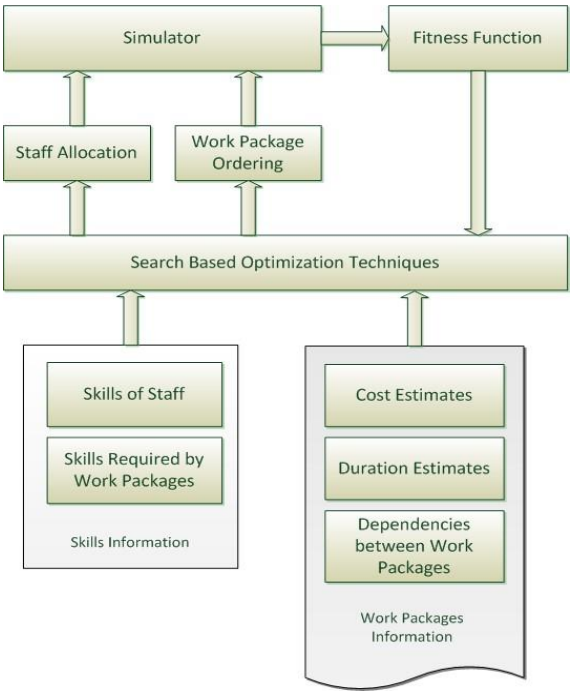


Figure 2.1. Generic search-based software project management scheme for solving task scheduling and staff allocation problems (Source: [54])

Moreover, early studies in this domain formulated the problem as a single objective by aggregating the cost and time objectives using weighing functions. They were based on simulation due to insufficient SPP real-world datasets [66,69]. In this work, however, studies that have focused on investigating the multi-objective formulation of the problem and applying Multi-Objective Optimization (MOO) algorithms to tackle the SPS problem are of interest.

2.1.1 Software Overtime Planning (SOP)

Planning a software project is a complex and highly dynamic task characterized by uncertainties and the risk of overrun in duration and cost. Although PMs are provided with automated project planning tools, software development teams still suffer from unplanned overtime as the only option to mitigate the overrun risk when the project encounters “mission creep” or an excessive change in requirements [2]. However, empirical studies show that disproportionate overtime negatively impacts developers [19,20,24] and software quality [21,23]. These findings have directed researchers’ attention to more proactive overtime planning on project schedules. Three distinctive approaches to incorporating overtime into software project schedules have been considered in the literature.

The first and earliest approach modeled overtime as a soft constraint for the SPSP solution [70,71]. The idea is to achieve zero overtime (if possible) in the project plan. Penalties are applied to solutions that do not satisfy the overtime constraint, and the fittest solutions are the ones with minimum overtime. Some studies in this category, such as [72], set a restrictive maximum allowed overtime for tasks in the schedule, and any solutions where tasks’ overtime is greater than the threshold are considered unworkable. The second approach sets overtime as one of the objectives to be minimized in the project schedule [65,73]. This approach allows the optimization algorithm to search for project schedule solutions with moderate overtime as a separate objective alongside project duration and cost. The idea is to find the optimal schedule with minimal overtime rather than aim for zero overtime, which is realistically not feasible. The third and the most state-of-the-art approach models overtime as a complete multi-objective optimization problem considering its effects on project duration, cost, overrun risk, and quality [21,22,30]. This approach separates overtime planning from the schedule as it is modeled as another optimization layer over the project schedule. In this study, we are interested in the studies that applied MOO and/or ML for overtime planning based on the third approach. Since the optimization-based formulation of overtime planning was introduced in 2013 by Ferrucci et al. [21], subsequent studies have applied MOO to SOP, with diverse objectives such as minimizing total overtime hours spent by developers, project duration, cost, and risk of overrun, as well as maximizing the quality of products.

Although this thesis focuses on developing a machine learning-based interactive multi-objective optimization approach for tackling SOP, it is crucial to understand the current state of research regarding the application of these two methods in the broader field of SPP. Therefore, a systematic mapping study is carried out to investigate the application of machine learning and multi-objective optimization algorithms in SPP.

2.2 Systematic Mapping Study on Application of MOO and ML in Software Project Planning

Optimization is a mathematical problem that involves finding the best solution among all feasible solutions. It is composed of the objective, decision variables, and constraints. The objective is the function to be minimized or maximized, and it is defined over the decision variables. The constraints

are limitations or restrictions on the solution, also defined as functions over the decision variables. The constraints may consist of equality and equality functions. Mathematically, it is defined as:

$$\begin{aligned} &\min_{x \in R^n} \quad f(x) \\ s.t.\ &g(x) \leq 0 \\ &h(x) \leq 0 \\ &x_L \leq x \leq x_U \end{aligned} \tag{2.1}$$

Typical examples of optimization problems are: minimizing cost while keeping efficiency in a manufacturing process, minimizing the duration of a software project schedule, and maximizing customer satisfaction in a software release. Optimization problems are typically solved using linear programming methods, such as the simplex method or dynamic programming approaches. A graphical solution to a hypothetical optimization problem is presented in Figure 2.12. These conventional methods aim to find the exact optimal solution within the solution space, which might be time-consuming when dealing with complex optimization problems.

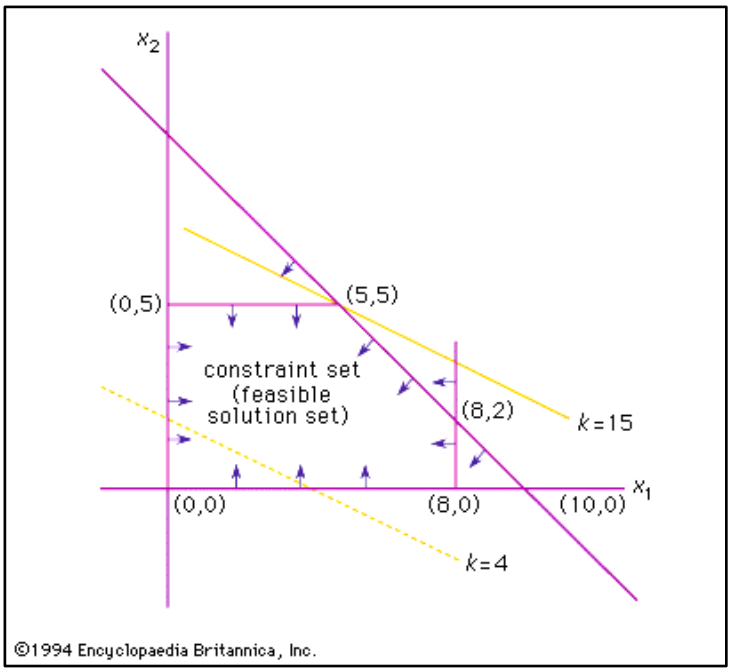


Figure 2.2. Solving Optimization Problem

To address complex optimization problems, search-based metaheuristic algorithms are often employed to find a suboptimal solution within a reasonable timeframe. Evolutionary and swarm intelligence algorithms are the most popular metaheuristics used for solving complex optimization problems. They mimic the natural biological evolution and the social behavior of species to tackle global optimization problems. Examples include genetic algorithms, memetic algorithms, particle swarm optimization, and Ant colony optimization algorithms. These algorithms have been successfully applied to various classical and engineering problems, yielding optimal performance [74]. Real-life optimization problems typically involve multiple and often conflicting objectives, which can only be effectively addressed using multi-objective optimization approaches.

Multi-objective optimization (MOO) is the process of optimizing systematically and simultaneously a collection of objective functions. It involves the concurrent optimization of several objective functions in the presence of some degree of conflict among them (one objective cannot be improved without worsening at least another objective). MOO problems exist in diverse fields such as engineering, economics, and logistics, where optimal decisions require balancing trade-offs among conflicting objectives. For instance, designing a new system might involve minimizing costs while maximizing safety, or selecting a portfolio could aim to maximize expected returns while minimizing risk. MOO is also referred to as vector optimization, multi-objective programming, multi-criteria optimization, multi-attribute optimization, or Pareto optimization.

Typically, there is no single solution that optimizes all objectives simultaneously but a set (possibly infinite) of Pareto optimal solutions. Since no single solution is optimal in this setting, the goal is to identify a set of alternative solutions that offer different trade-offs among the objectives. Without additional preference information, all Pareto optimal solutions are considered equally good. A multi-objective optimization problem can be formulated mathematically as described by [75]. Find the vector $x^{\rightarrow *} = [\, x_1^*, x_2^* \ldots\ldots x_n^* \,]^T$ which will satisfy the *m* inequality constraints: $g_i\,(x^{\rightarrow}) \geq 0 \quad i = 1,2, \ldots\ldots m$ and *p* equality constraints: $h_i\,(x^{\rightarrow}) = 0 \quad i = 1,2, \ldots\ldots p$ and optimize the objective function:

$$f^{\rightarrow}(x^{\rightarrow}) = [\, f_1(x^{\rightarrow}), f_2(x^{\rightarrow}) \ldots\ldots f_k(x^{\rightarrow})]^{\,T} \tag{2.2}$$

Where $x^{\rightarrow} = [\, x_1, x_2 \ldots\ldots x_n \,]^T$ is the vector of decision variables

To identify these solutions, Pareto optimization– a method based on a dominance relation – is employed in [76]. The Pareto dominance specifies that a single solution dominates another if the former is better than the latter in at least one of the objectives and not worse in any of the other objectives. In addition, a set of Pareto optimal solutions called the Pareto front comprises those not dominated by any other solution in the space [77].

Pareto optimality: A point $x^{\rightarrow *} \in X$ is said to be pareto optimal if for every $x^{\rightarrow} \in X$ and $I = \{\, 1, 2, \ldots\ldots K \,\}$

$$\forall_{i \in I} \left(f_i\,(x^{\rightarrow}) = f_i\,(x^{\rightarrow *}) \right), \;\; \exists\, i \in I\colon f_i\,(x^{\rightarrow}) < f_i\,(x^{\rightarrow *}) \tag{2.3}$$

Pareto Dominance: A vector $u^{\rightarrow}$ = (u_1 ……..u_k) is said to dominate $v^{\rightarrow}$ = (v_1 ……..v_k) if and only if :

$$\forall_{i \in \{\, 1, \ldots\ldots, k \},} \;\; ui \;\leq\; vi \;\wedge\; \exists i \in \{\, 1, \ldots\ldots, k \}\colon ui \;<\; vi \tag{2.4}$$

A detailed explanation of MOO can be found in [15,77]. Population-based metaheuristic algorithms such as multi-objective evolutionary genetic algorithms [70,78–81], swarm intelligence optimization algorithms [15,82,83], and multi-objective memetic algorithms [84–86] are prominent methods for solving MOO and have been used widely in SPP [5,55,84,87].

Therefore, identifying the research landscape in applying search-based MOO algorithms to SPP is indispensable for updating the body of knowledge in this field. For instance, understanding these advances of MOO in SPP may be beneficial to project managers in making more informed decisions in resource allocation, scheduling, and overall project planning, leading to better outcomes. Project

managers can adopt strategies that minimize bottlenecks, shorten project timelines, and enhance overall project efficiency. For researchers, staying informed about the often applied optimization algorithms in SPP ensures that software project planning aligns with industry expectations. This alignment is crucial for maintaining competitiveness and delivering projects that meet or exceed industry standards.

On the other hand, Data-driven Software Engineering (DSE) – a subfield of software engineering that utilizes software data generated by software houses or projects to build ML models for estimating software metrics and predicting software characteristics – has also gained many applications in SPM [88]. Research in this area focuses on replacing human expert judgment [89], which is prone to errors, with more accurate ML models [90,91]. ML is a branch of AI that builds algorithms that automatically learn from experience. The goal is to train a system by presenting examples of desired input-output behaviour rather than manually programming it and anticipating desired responses from all possible inputs. ML employs statistical-computational techniques to extract patterns from large volumes of data and uses this to predict the behaviour of systems. Detailed information on ML and its various applications can be found in [92]. A secondary study conducted by Mahd et al. [50] revealed that the majority of ML studies in software project management focus on SPP activities, which calls for an extensive survey on the role of ML in planning successful software projects. To this end, an exploratory literature search was conducted to identify recent reviews and mapping studies that focus on ML applications to the various tasks of SPP. Several review/mapping studies found on ML for SPP [6,93–96] focused solely on effort/cost estimation, even though there is evidence of its application in other activities of SPP [47,97–99]. There is a scarcity of secondary studies examining the application of ML algorithms to various SPP tasks.

Moreover, recent developments in software engineering research have shown the synergistic interplay between search-based MOO and ML as a promising strategy for finding robust and more accurate solutions to software engineering problems [45,48,100]. The studies recommended a systematic combination of the two computational intelligence methods to enhance efficiency and improve accuracy. Investigating how this strategy has been explored in SPP is still an open issue. Therefore, addressing it will provide researchers in this domain with state-of-the-art approaches to tackling SPP problems in its various dimensions.

To address these gaps identified in the existing body of knowledge in SPP, an extensive systematic mapping study on the application and interaction of search-based MOO and ML algorithms in SPP is conducted. The mapping is based on 71 quality studies identified using a robust systematic mapping methodology in software engineering. It covers studies published over the last ten years and is addressed to new and emerging researchers in software project management, serving as a benchmark study for understanding the structure and trends of research in the domain. It will also be helpful to industry practitioners in identifying new methods applicable to their environment.

The major contributions of the systematic mapping are summarized as follows:

1. Identification of actively researched domains in SPP and classification of MOO and ML algorithms used in the various domains of SPP
2. Investigation on the interaction strategies between MOO and ML used in SPP domains
3. Investigation on the software project development/management methodologies mainly considered in SPP studies
4. Highlights of open research issues in SPP and propositions on the future direction

2.2.1 Review Methodology

The adopted systematic mapping methodology follows the latest guidelines for conducting systematic mapping studies in software engineering, as outlined by Petersen, Vakkalanka, and Kuzniarz [101]. These guidelines were created by consolidating knowledge from other established systematic mapping and literature review protocols [102–104]. This systematic mapping aims to explore existing research on the use of MOO and ML in SPP, with the goal of addressing the following questions:

1. *Which SPP activities have been most frequently addressed using ML and MOO techniques?*
2. *What types and classifications of ML and MOO algorithms have been applied to these SPP activities, and how are they distributed across different planning tasks?*
3. *To what extent have studies explored the synergistic integration of ML and MOO in SPP, and what interaction strategies have been employed?*
4. *What software development or project management methodologies are most commonly considered in studies applying ML and MOO to SPP?*

Following the guidelines in [101], the systematic mapping process is organized as described in the following sub-sections.

2.2.1.1 Study Identification

The study identification approach employed in this study combines database search and snowballing to ensure adequate coverage of the research domain. It defines precise inclusion and exclusion criteria, along with quality assessment procedures, to identify high-quality studies. As presented in Figure 2.3, the method considers articles extracted from various selected literature databases and other sources, such as citation and bibliography searches, to identify articles most relevant to the research questions.

Articles were searched across various databases, and the retrieved records were aggregated. The aggregated records were then filtered for duplicates since the searches were conducted independently. After the database search, the following procedures were carried out to identify the most relevant articles that address the established research questions:

1. *Screening of record for eligibility:* The identified articles from databases were screened by examining the title and abstract based on some criteria. Ineligible articles were removed before proceeding to the next step.

2. *Retrieval of Eligible articles:* Articles found eligible from step 1 were downloaded for thorough examination. Articles that were inaccessible or could not be retrieved were removed from the records of eligible articles.
3. *Assessment for Relevance:* In this step, eligible articles were further assessed by thoroughly examining the full text for relevance. Relevance was measured by defining inclusion and exclusion criteria. Relevant articles based on inclusion criteria were retained, while irrelevant articles based on exclusion criteria were removed from the pool.
4. *Assessment for Quality*: The remaining articles in the pool after step 3 were further assessed based on the quality of their methodology and results. Although quality in this context has no agreed-upon definition, Kitchenham and Charters [105] contend that quality can be estimated as the extent of bias minimization and the degree of maximization of internal and external validity in the study. A quality instrument based on this opinion was used to assess the quality of the studies. Articles that did not pass the quality test were removed.
5. *Identification of additional studies from other sources:* In this step, additional articles were identified by applying snowballing to the articles that qualify for review based on step 4. Snowballing iteratively searches for other relevant articles that cite the current article (forward snowballing) and relevant articles within the reference list or bibliography (backward snowballing). This extends the coverage of the systematic mapping and captures articles that the database search might have omitted. New studies identified are added to the existing pool of qualified articles after thorough assessment (i.e., eligibility, relevance, and quality).
6. *Data extraction:* In this step, an instrument is designed to extract usable data from the articles for classification. The instrument is carefully designed to answer all the research questions defined in the study.
7. *Classification:* At the final stage, the information extracted from the previous step was used to classify the primary studies based on the defined research questions.

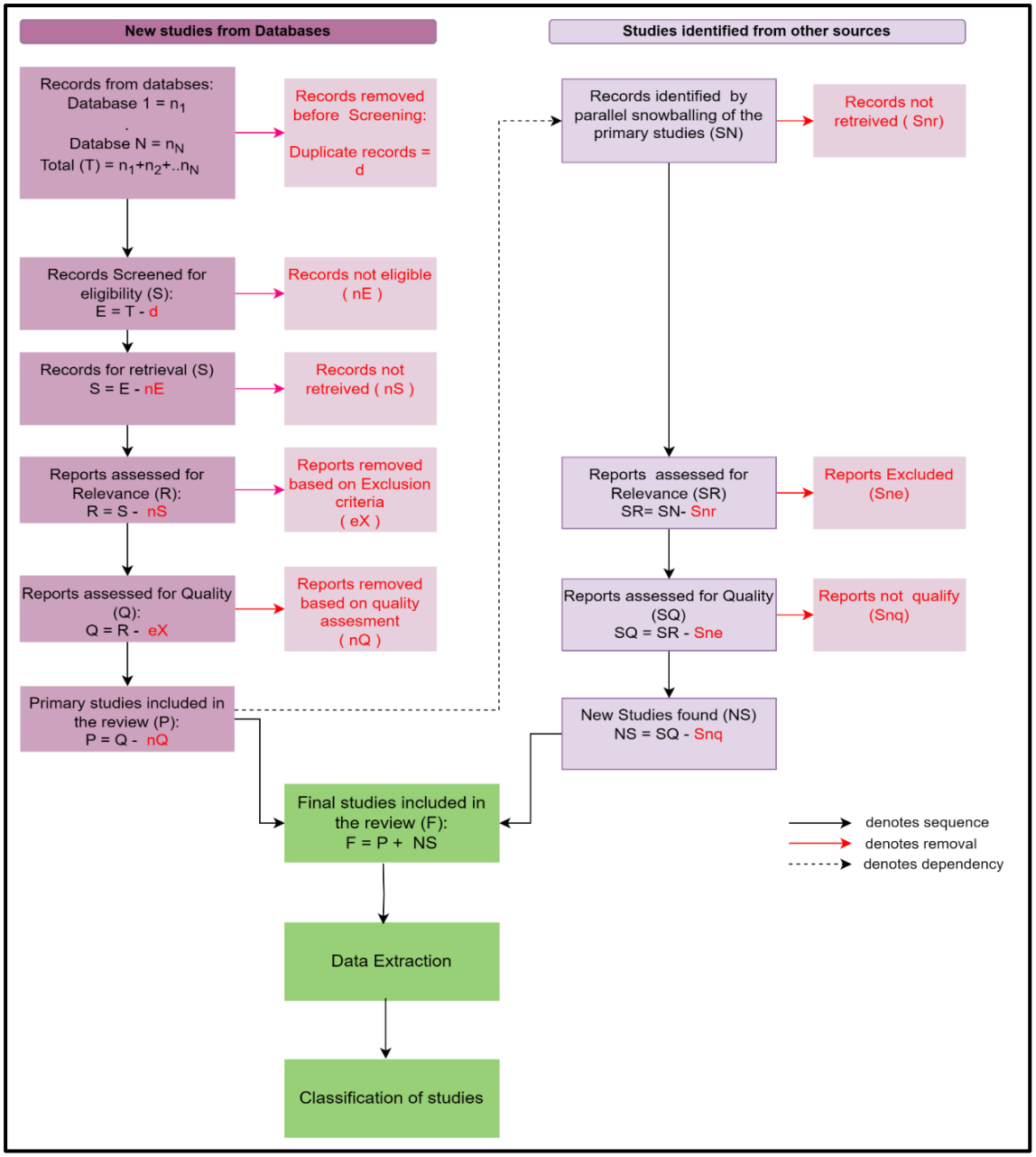


Figure 2.3. The study identification procedure for the mapping

2.2.1.2 Search Strategy

To conduct an exhaustive search for primary studies, several literature databases are often considered as recorded in several systematic review and mapping studies in software engineering [4,47,62,101,106]. This study selected seven popular databases as sources for the literature search, as presented in Table 2.1. The hybrid search strategy proposed by Mourão et al. [107] for systematic reviews and mappings in software engineering was adopted for searching the databases. Their approach recommends combining database search and parallel snowballing for literature searches, as it has proven superior to other strategies in terms of precision and recall. Specifically, the approach applied backward and forward snowballing concurrently to articles identified from the database search

to recover other potential articles that the original search might have missed. To avoid snowballing from irrelevant articles and reduce the complexity of the literature search, parallel snowballing was conducted after the quality assessment of identified articles from the databases.

Table 2.1. List of Literature Databases Used

Database/Source	URL
ACM Digital Library	https://dl.acm.org/
IEEE Xplore	http://ieeexplore.ieee.org
Science Direct	https://www.sciencedirect.com/
Scopus	https://www.scopus.com/
SpringerLink	https://link.springer.com/
Tailors & Francis	https://www.tandfonline.com/
Web of Science	https://www.webofscience.com/wos/woscc/basic-search

Regarding the search formulation, Kitchenham and Charters [104] suggested using PICO (Population, Intervention, Comparison, and Outcomes) from research questions to identify keywords and formulate search strings. However, Petersen, Vakkalanka and Kuzniarz`s guideline [101] reflected that the P and I components of the PICO are most relevant when formulating search strings for systematic mapping studies. In software engineering, population may refer to an application domain or software engineering role; in this case, it is ”software project planning”. Intervention in software engineering may refer to a software technology, tool, or methodology. In this study’s context, it refers to “Multi-objective Optimization” and “machine learning” methods utilized. Moreover, to incorporate synonyms and other related terms of the identified keywords, a search string was formulated based on the recommendation of Brereton et al. [108] by integrating logical search operators into the query. The search string is given as:

(((“search-based” OR “evolutionary” OR “metaheuristic" OR "genetic" OR "swarm intelligence” OR “memetic”) AND “multi-objective”) OR (“machine learning” OR “Data mining” OR “Predictive learning”)) AND (“algorithm” OR “approach” OR “model” OR “technique”) AND ("software project planning" OR "software project management").

2.2.1.3 Filtering Eligible Articles

At this stage, the articles retrieved from the database search were filtered to identify eligible articles worthy of full-text analysis. This initial filtering step ensures the removal of unrelated studies and redundant articles. The filtering process is based on three steps:

1. *Duplicate articles removal:* Due to the use of multiple source databases, there is a high likelihood of having duplicates in the initial list of articles. Thus, before analyzing the retrieved articles, duplicates were removed.

2. *Filtering based on Title:* After removing the duplicates, the remaining articles were analyzed based on their titles. Articles that do not contain any of the keywords in the search sentences were filtered out and discarded. These articles are unrelated to the study or originate from a different domain. The metadata of the remaining articles were loaded into the Mendeley reference management system for further analysis.

3. *Filtering based on Abstract:* The articles were further analyzed on Mendeley based on their abstract, and those that presented no application of MOO or ML algorithms were removed. Also, articles that focused on project management other than software were removed

2.2.1.4 Inclusion and Exclusion Criteria for Selecting Relevant Articles

After filtering, the potential articles were downloaded and thoroughly examined based on the full text for inclusion and exclusion. The inclusion and exclusion criteria utilized for selecting relevant articles are presented below.

Inclusion criteria (IC) cover all articles that:

IC1: applied search-based MOO or ML algorithms to any activities of software project planning/management

IC2: employed both search-based MOO and ML algorithms in any activities of software project planning/management

IC3: are research / technical papers

IC4: were published in the English language

IC5: were published between 2012 and 2022 inclusive. The systematic mapping was carried out in 2023.

Exclusion criteria (EC) eliminate articles that:

EC1: applied search-based MOO and ML algorithms to software engineering problems outside SPP

EC2: employed other techniques/algorithms apart from search-based MOO and ML algorithms for the activities of SPP

EC3: carried out comparative analyses of already used search-based MOO and ML algorithms in SPP

EC4: carried out replication studies on previously published works in search-based MOO and ML algorithms for SPP

EC5: focus on education in or teaching of SPM/SPP

EC6: had an extended version in the pool (preference is given to the most recent version of the articles)

EC7: were published in languages other than English

EC8: could not be accessed or downloaded.

2.2.1.5 Quality Assessment

To select only high-quality studies with evidence of usefulness in the SPP domains from the relevant articles, a quality assessment procedure was designed, adopting three key points as outlined in [109]: relevance to the goal of this systematic mapping, credibility of the results, and quality of the dissemination medium. Based on these points, the following quality criteria (QC) along with a scoring system adapted from [110,111] were established:

QC1: Does the paper propose a novel and specific MOO/ML algorithm/model/approach to a software project planning problem? (Yes +1 / No +0).

QC2: Does the paper validate the approach it proposed?

The possible options are: empirical validation by case study, survey, or experiment (+1)/ No validation (+0).

QC3: Does the paper compare its results with the existing state-of-the-art approaches/models in the literature? (yes +1 / No +0).

QC4: Does the paper present a threat to the validity plan or limitations?

Options are: It provides internal and external validity, and limitations/biases (+1) / partially (e.g., provides just limitations or biases) (+0.5) / No type of validity plan (+0).

QC5: Was the paper published in a highly-ranked journal or conference?

The Scimago Journal Rank (SJR2022) [112] was used for journal ranking as it is based on journals indexed in Scopus, the largest journal indexing database. For conference papers, the CORE ranking [113] was employed. The available options are: Q1 or A* (+2) / Q2 or A (+1.5) / Q3 or B (+1) / Q4 or C (+0.5) / Unranked (+0)

The cumulative quality score of each article was computed using QC1 to QC5, and articles with a minimum score of 3.5 over a 6.0 total score were selected as part of the final studies to be considered for the systematic mapping in this work.

2.2.1.6 Data Extraction and Classification

The final set of articles contained useful data for answering the research question. The data were extracted using the template presented in Table 2.2. It details the extracted data, their expected values, and the research questions they address.

The extracted data were then organized in a tabular form, ordered by the project planning activity covered, to analyze the algorithms' taxonomy, frequency of use, and the project management approaches employed. Studies in each of the major categories identified were counted and extracted into separate tables. The generated tables were further analyzed to answer the research questions and identify trends or limitations. The results from the analysis of the extracted data were used to classify domains, algorithms, interaction strategies, and project management methodologies. Elsevier's

Mendeley bibliometric management system was used to manage the large volumes of returned papers and remove duplicates. A spreadsheet application was used to extract data and analyze results. Bar charts and pie plots were utilized as the visualization tools to present the classifications.

Table 2.2. Data Extracted from the Final studies

Data extracted	Value	Research question addressed
Title	Text	-
Author(s)	Text	-
Year of publication	Integer (2012 – 2022)	-
Software project planning activity covered	Text	RQ1
Software project management/development methodology explored	Traditional/Agile/Hybrid	RQ4
Does the study apply MOO?	Binary (0/1)	RQ2, RQ3
Does the study apply ML?	Binary (0/1)	RQ2, RQ3
The main research issue or objective	Text	RQ1, RQ2, RQ3
Algorithm/Method applied	Text	RQ2
Sources of dataset used	Simulated/Private/Public [with list of datasets]	RQ2, RQ3, RQ4
Performance evaluation method(s) used	Text	RQ2, RQ3
Summary of results/finding	Text	RQ2, RQ3
Quality score	Real number (0.0 - 6.0)	-

2.2.2 Study Identification Results

After applying the study identification procedure, 71 high-quality articles were identified (see Appendix A for the list of included articles), which were deemed highly relevant to the mapping and helpful in answering the research questions. Table 2.3 presents the results of the number of articles obtained after each step of the study identification process.

Table 2.3. Result of the Literature Search

Sources	Study identification step	Articles identified	Articles removed	Articles retained
Literature Databases	Total Records from database search (T)	3,335		
	Duplicate records		-865	
	Records screened for eligibility (E)	2,470		
	Records not eligible (nE)		-2,142	
	Records sought for Retrieval (S)	328		
	Records not retrieved (nS)		-11	
	Articles assessed for Relevance (R)	317		
	Articles removed based on Exclusion Criteria (eX)		-216	
	Articles assessed for Quality (Q)	101		
	Articles not quality (nQ)		-43	
	Primary studies found (P)			58
Other sources	Articles identified by parallel snowballing of primary studies (SN)	35		
	Articles excluded (Sne)		-16	
	Articles assessed for Quality (SQ)	19		
	Articles not quality (Snq)		-6	
	additional article found (NS)			13
	Final articles included in the mapping (F)			71

From the initial 3,335 records returned by the database search, 865 duplicates were found and removed. The remaining 2,470 were screened for eligibility based on title and abstract, and 2,142 records were found to be ineligible and were deleted from the collection, leaving 328 articles for consideration. Out of these articles, 317 articles were retrieved for complete text analysis. Applying the inclusion and exclusion criteria, 216 articles irrelevant to the mapping's goals were excluded. Of the remaining 101 articles, 58 were found to pass the quality criteria and included as primary studies for the mapping. Exploring other sources, 35 additional studies were identified through snowballing of the primary studies previously identified. Out of these, 16 were excluded due to the inclusion and exclusion criteria, and six were excluded due to their low quality. The remaining 13 were added to the primary studies, resulting in 71 final articles used in this systematic mapping study. The list of the final articles included in the mapping study is presented in Table 2.4.

Table 2.4. List of articles included in the mapping

S/N	Article	Year of publication	Quality Score
1.	[114]	2014	3.5
2.	[115]	2012	4.5
3.	[70]	2013	3.5
4.	[73]	2014	4.5
5.	[116]	2021	4.0
6.	[117]	2019	3.5
7.	[118]	2021	4.0
8.	[119]	2021	4.0
9.	[64]	2018	5.0
10.	[120]	2021	5.0
11.	[121]	2021	5.0
12.	[29]	2017	6.0
13.	[122]	2017	3.5
14.	[123]	2018	6.0
15.	[124]	2019	5.0
16.	[125]	2022	4.0
17.	[126]	2020	4.0
18.	[127]	2015	4.5
19.	[128]	2019	4.5
20.	[129]	2021	3.5
21.	[130]	2021	6.0
22.	[131]	2012	3.5
23.	[132]	2016	5.0
24.	[133]	2020	5.0
25.	[134]	2016	3.5
26.	[135]	2020	4.0
27.	[65]	2016	5.5
28.	[136]	2016	4.5
29.	[137]	2021	4.5
30.	[138]	2017	5.0
31.	[71]	2015	4.0
32.	[139]	2021	3.5
33.	[140]	2021	5.0
34.	[141]	2021	3.5
35.	[142]	2012	3.5
36.	[143]	2021	6.0
37.	[144]	2016	3.5
38.	[145]	2018	3.5

39.	[146]	2012	5.0
40.	[147]	2021	4.0
41.	[72]	2022	5.0
42.	[148]	2020	3.5
43.	[149]	2015	6.0
44.	[150]	2016	5.0
45.	[151]	2013	5.0
46.	[22]	2016	4.5
47.	[30]	2019	4.0
48.	[87]	2022	5.0
49.	[152]	2016	6.0
50.	[153]	2013	5.0
51.	[154]	2015	5.5
52.	[21]	2013	5.5
53.	[155]	2012	6.0
54.	[82]	2018	3.5
55.	[156]	2021	4.0
56.	[157]	2021	3.5
57.	[158]	2012	3.5
58.	[159]	2017	5.0
59.	[160]	2022	6.0
60.	[161]	2012	3.5
61.	[162]	2019	5.0
62.	[163]	2013	5.0
63.	[164]	2013	5.0
64.	[165]	2015	4.5
65.	[166]	2019	6.0
66.	[167]	2020	4.0
67.	[168]	2018	5.5
68.	[7]	2021	5.0
69.	[169]	2020	4.0
70.	[170]	2021	3.5
71.	[171]	2017	4.5

2.2.3 Results on Classification and Analysis of Studies

Based on the data extraction method discussed earlier (Section 2.2.1.6), the classification and analysis of the studies included in the mapping are conducted in relation to the research questions. The results are organized to capture the SPP domains or activities addressed by the identified MOO and ML studies, the specific algorithms employed, the interaction between the two methods, and the software project management methodologies considered.

2.2.3.1 *Which SPP activities have been most frequently addressed using ML and MOO techniques?*

To identify project planning activities covered in the primary studies, the major SPP activities tackled with MOO or ML were extracted from the final articles. Three software project planning activities were identified as having attracted the attention of MOO and ML researchers in the last decade. These activities include software effort estimation, software project scheduling and software overtime planning. The distribution of studies based on these SPP activities is presented in Figure 2.4.

SEE involves estimating the size of the software to be built and determining the human effort required to develop the software fully [142]. SEE is typically carried out at the initial stage of software

development, and an incorrect estimation can lead to significant time and cost constraints in subsequent phases, which may eventually result in project failure [82,120,172,173]. Furthermore, a precise and dependable estimate can significantly assist PMs in the effective scheduling of software development tasks, robust allocation of tasks to developers, and, more importantly, efficient execution of software projects on time and within budget [174]. Focusing on recent developments in SEE using ML and MOO methods, this study has identified 53 high-quality studies that proposed and critically evaluated new learning algorithms, achieving significant results against benchmarks. Analysis revealed that most studies (44) employed machine learning. While only six studies employed MOO, the remaining three used both ML and MOO. Figure 2.5 illustrates the proportion of studies that applied ML, MOO, and both to SEE over the last decade.

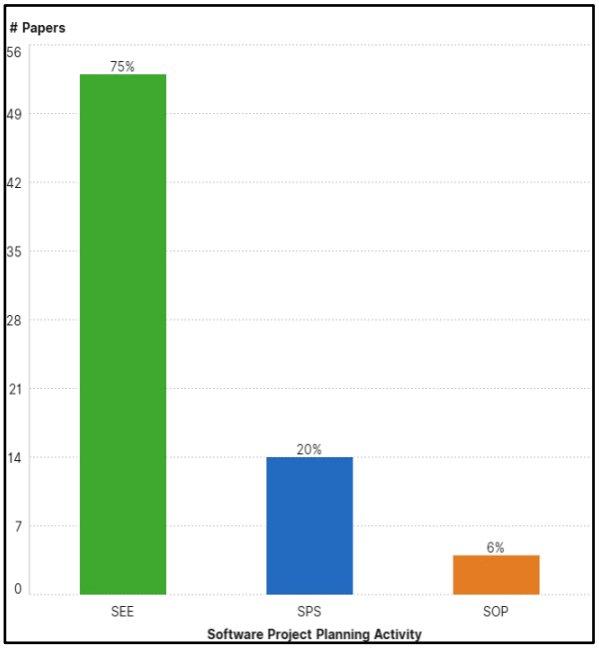


Figure 2.4. Distribution of articles based on SPP activities

Regarding SPS, a total of 14 high-quality studies were identified; nine of them utilized MOO, while the remaining five employed ML algorithms. All identified studies applied MOO or ML individually, and none combined the two methods in a model or framework. The proportion of the studies that applied ML and MOO to SPS in the last decade is presented in Figure 2.6.

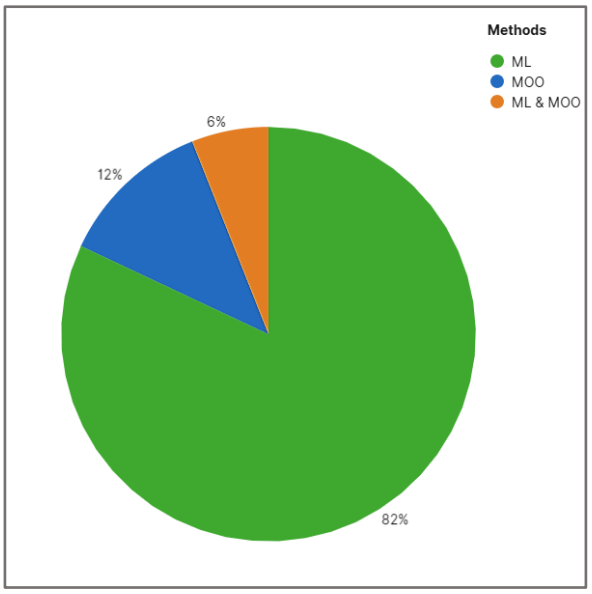


Figure 2.5. Proportion of studies in SEE based on methods applied

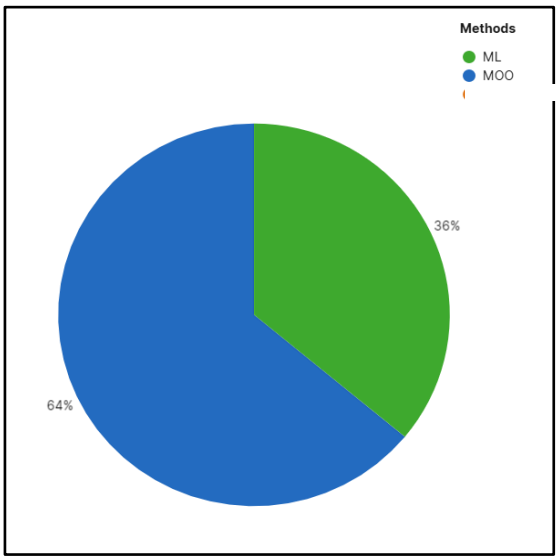


Figure 2.6. Proportion of studies in SPS based on method applied

Concerning SOP, a multi-objective optimization-based formulation was introduced in 2013 by Ferrucci et al. [21]. This study identified four articles that applied MOO to develop optimal overtime plans in software development projects, with diverse objectives including minimizing total overtime hours, project duration, cost, and risk of overrun, as well as maximizing product quality. No work was found that applies ML to software overtime planning. This may be due to the limited publicly available datasets for building overtime estimation models and the difficulty of obtaining software project datasets from industries, as they are classified as confidential, which in turn affects the ability to carry out rigorous research in this area.

2.2.3.2 *What types and classifications of ML and MOO algorithms have been applied to these SPP activities, and how are they distributed across different planning tasks?*

To answer this question, the classification of algorithms applied under the MOO and ML methods vis-à-vis SPP activities covered is constructed. The results are presented according to the method employed.

MOO algorithms in SPP activities

In SEE, MOO algorithms have been sparingly applied, despite achieving convincing results with these algorithms [120,152,163,175]. Nine studies employing ten different MOO algorithms were identified, considering those that applied MOO alone or in combination with other methods. The mapping of primary studies in SEE to MOO algorithms, along with their classification, is presented in Table 2.5. Figure 2.7 also shows the frequency of application of the algorithms. Based on the classification, three classes of MOO algorithms were identified for SEE.

1. *Multi-Objective Evolutionary Algorithms (MOEA):* MOO algorithms that simulate the natural evolution of offspring from parents.
2. *Multi-Objective Swarm Intelligence Algorithms (MOSIA):* MOO algorithms that mimic the intelligent social behaviors of groups of organisms or entities.
3. *Multi-objective Bio-Inspired Optimization Algorithms (MBIOA):* MOO algorithms inspired by an individual organism's naturally intelligent behavior in solving problems in its environment.

Table 2.5. Mapping of studies in SEE to MOO algorithms

Classification	Algorithm	Studies
MOEA	MODE	[120]
	NSGA-II	[152]
	HaD-MOEA	[163]
	MOEA-D	[120]
	NSGA-III	[120]
MOSIA	MOPSO	[82], [114], [176], [120]
	MOWOA	[120]
	FOA	[126]
MBIOA	SBO	[159]
	FPA	[117]

MOEAs have been explored to tackle the problem of effort/cost estimation in software project management. The study in [152] employed MOAEs to search for the optimal and most accurate estimates of a specific ML model. In contrast, other studies, such as [120,163], used MOAEs to find the optimal estimation models. Sarro, Petrozziello, and Harman [152] introduced and evaluated a bi-objective SEE approach, Confidence Guided Effort Estimator (CoGEE), which incorporates confidence intervals and the sum of absolute errors to guide NSGA-II algorithm to optimal effort estimates. Minku and Yao [163] proposed HaD-MOEA as an improved version of NSGA-II to search for the optimal MLP models using three different performance measures - MMRE, LSD, and Pred(25) - simultaneously and explicitly as objectives. The algorithm produced a Pareto ensemble of MLP models with optimal

estimation performance. A multi-objective version of DE called HFMODE was designed by Shailendra et al. [120] and used for the automatic prediction of software cost/effort models. The algorithm incorporates a homeostasis factor-based mutation operator to address the problem of diversity loss and low convergence rates inherent in existing multi-objective optimization (MOO) methods. The proposed algorithm achieved more accurate estimation values than other implemented MOEAs, such as MOEA-D and NSGA-III

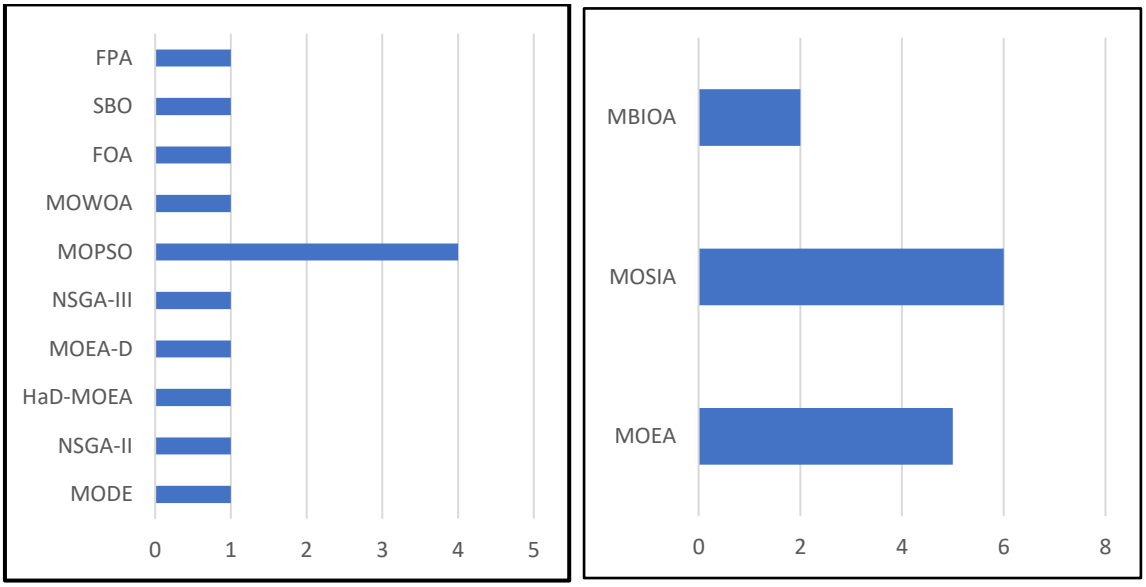


Figure 2.7. (A) MOO algorithms used in SEE studies (B) usage by classification

MOSIA is another class of MOO that has recorded several applications in SEE. Studies in this category employed MOSIA individually or in combination with another approach to optimize software development efforts. MOPSO was used in [114] individually for case-based effort estimation. Specifically, the study aimed at optimizing feature set, case weights, and the number of nearest analogies in a single model. The approach proved superior to other existing CBR approaches. Likewise, in [120], it was employed to find the optimal estimation models on two-objective and three-objective software cost estimation formulations. MOPSO has also been used for hyperparameter optimization of ML-based effort estimation models. The study in [82] used MOPSO to optimize COCOMO II model parameters for development effort and time, along with fuzzy logic for calibration. Similarly, it was used in [176] to optimize the hyperparameter selection procedure of SMO-based SVR, modeled for imputing missing data in fuzzy analogy-based SEE. In another study, Resmi and Vijayalaksmi [126], employed FOA to optimize the parameters of a clustering-based fuzzy analogy estimation model, producing a better estimation accuracy than the fuzzy analogy approach without optimization. Nonetheless, a study [120] applied MOWOA—a member of MOSIAs—to produce optimal effort estimation models solely.

There have also been applications of MBIOA in software development effort/cost estimation, mainly to support other parametric and ML-based estimation models. Ullah et al. [117] applied bi-objectives FPA to optimize the COCOMO-II cost estimation model's parameters based on MMRE and

MD. Additionally, Moosavi and Bardsiri [159] optimized the parameters and structure of an ANFIS framework using the SBO algorithm to achieve more accurate estimation of software development effort. Existing evidence shows that MOO algorithms have been significantly effective in producing better development effort estimates than their ML counterparts [27, 36, 38]. Additionally, MOO algorithms have been beneficial in enhancing the estimation accuracy of other parametric and machine learning models when used to optimize hyperparameters [82,117,126,159,168]. This evidence demonstrates the robustness of MOO algorithms in addressing complex problems and their guided approach to finding optimal solutions.

In SPS, optimization algorithms are ideally employed due to the nature of the activity, which requires optimal allocation of tasks to time slots and human resources to tasks. Nine quality studies applying MOO were identified in this category, and they have applied 16 different algorithms. Table 2.6 presents the mapping of primary studies on SPS to MOO algorithms and their classification, and Figure 2.8 presents the statistics on their frequency of application.

Table 2.6. Mapping of SPS studies based on MOO algorithms applied

classification	Algorithm	Studies
MOEA	MoCell	[150]
	NSGA-II	[70], [150], [73], [71], [87]
	PEAs	[150]
	SPEA2	[150]
	MOGA	[73], [72]
	CCMOGA	[133]
	ε-domination-based MOEA (dε-MOEA)	[65], [64]
	COEA	[65]
	MOEA-D	[71], [64]
	Two-Archive2 Algorithm	[87]
	BCE	[87]
	NSGA-III	[64], [87]
Memetic Multi-Objective Optimization Algorithm (MMOOA)	MOTAMA	[64]
MOSIA	OMOPSO	[87]
	SMPSO	[87]
	MOACO	[71]

SPS has also been addressed using MOEA/D, which approaches multi-objective optimization by decomposing the multi-objective problem into several single-objective subproblems using an aggregation function and then optimizing them in parallel using a collaborative method. In [71], Xiao et al. introduced MOEA/D-ACO to SPS as a Multi-Objective Ant Colony Optimization (MOACO). The proposed algorithm was implemented to optimize duration, cost, and developers' overwork. Compared to NSGA-II, MOEA/D did not perform better in most complex instances but produced reasonable results in a shorter time. Therefore, the authors recommended improved tuning of the proposed algorithm. With regards to the Cooperative Coevolution (CC) based MOO strategy, Shen et al. [133] proposed a variant of CCMOGA with an improved computational resource allocation (CCMOGA-ICRA) to solve a modeled

mathematical formulation of the large-scale multi-objective SPSP (LSMOSPSP), which takes into account several tasks and employee properties by optimizing four objectives of cost, duration, employees' satisfaction and robustness in the presence of three practical constraints. The proposed CCMOGA-ICRA showed improved convergence results compared to other evolutionary algorithms and exhibited good scalability in large-scale SPS.

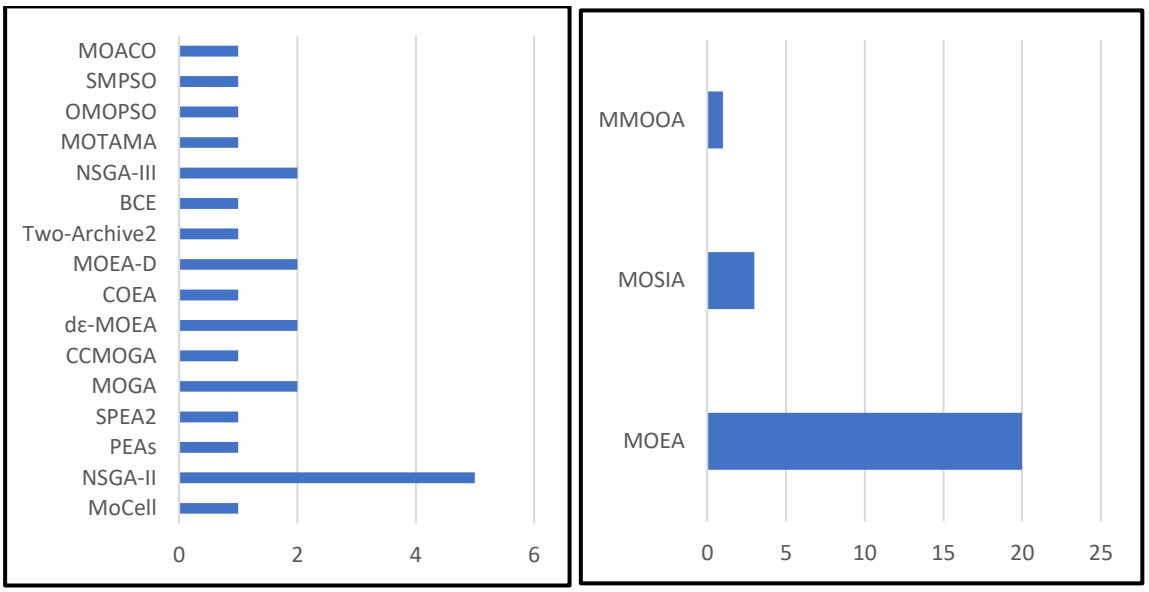


Figure 2.8. (A) MOO algorithms used in SEE studies (B) usage by classification

Applying memetic MOO in a dynamic SPS environment, Shen et al. [64] proposed a novel formulation for dynamic SPSP that relates the rate of increase in skill competency to human factors (such as motivation and learning ability) and skill level difficulty. The model defined employee satisfaction as an additional objective to project cost, duration, stability, and robustness, considering various constraints during rescheduling. A Q-learning-based multi-objective two-archive memetic algorithm (MOTAMAQ) was designed to solve the formulated MODSPSP through proactive rescheduling. Experimental results on benchmark dynamic project scheduling datasets and three real-world datasets demonstrated the superiority of MOTAMAQ over NSGA-II, MOEA/D-DE, and dε-MOEA.

In SOP, only a few studies have employed MOO algorithms. This is primarily due to its status as an emerging field in SPP, with the first known attempt to tackle it dating back to 2013. Four quality studies were identified that used two different MOO algorithms for SOP. Table 2.7 presents the mapping of studies to the MOO algorithms applied, their classification, and frequency of use.

Table 2.7. Mapping of SOP studies based on MOO algorithms applied

Classification	Algorithm	Studies	Freq.
MOEA	NSGA-II	[21], [22], [29]	3
MMOOA	MOSFLA	[30]	1

MOEAs are the most widely applied MOO method in SOP, with three out of the four studies identified using it. Ferruci et al. [21] were the first to formulate SOP as a MOO problem and study the effects of overtime allocations on the project schedule to minimize the project duration and the risk of overrun. The authors applied NSGA-IIv, a variant of NSGA-II, to find the optimal overtime allocations, and it performed significantly better than conventional industry-based overtime planning methods. To improve performance, Sarro et al. [29] introduced an adaptive method for selecting meta-heuristic operators in NSGA-II, using the same problem formulation as in [21]. A variant of NSGA-II (Adaptivevsc), which efficiently combines the crossover operator used in [21] with adaptive genetic operators proposed by Nebro et al. [177], was introduced. The proposed adaptive method outperformed the standard NSGA-II by a margin of 93%. In another study, De Barros and De Araujo [22] extended the formulation of [21] by incorporating the already established effect of overtime on software quality into the formulation. To study overtime dynamics, the authors simulated the defects introduced by developers spending overtime as they affect project cost and duration within the optimization steps of NSGA-II. The objective was to minimize the project duration, cost, and overtime hours. Empirical results showed the effectiveness of using the formulation with NSGA-II, which outperformed existing overtime planning strategies in the industry.

MMOOA has also been used once in SOP with the application of MOSFLA [30]. Using the same experimental setting and datasets as in [22], the authors evaluated the performance of the proposed memetic algorithm, and the experimental results indicated its superiority over the mainly applied NSGA-II in this field.

ML algorithms in SPP activities

Regarding SEE, ML algorithms have been widely applied over the last decade, with significant advancements and improvements in results. After quality assessment, 47 studies employing up to 30 ML algorithms to advance effort estimation were identified. 44 of them applied only ML algorithms (individually or hybridized with other ML methods), while the remaining three combined ML with MOO. A mapping of the identified studies to the ML algorithm implemented and their classification is presented in Table 2.8. As shown in Figure 2.9 and 2.10, voting, ANN, and bagging are the most frequently used ML algorithms for SEE, while ensemble, neural-based, and hybrid methods are the most commonly used ML methods.

Table 2.8. Mapping of SEE studies based on ML algorithms and their classification

Classification	Algorithms	Studies
Tree-based	Regression Tree (RT)	[171]
	LMT	[134]
	REPTree	[131]
	Decision Tree	[134]
Distance-based	KNN	[135]
Probability-based	BBN	[153]
	Kernel density function	[151]
Regression function-based	Linear Regression	[156], [127]
	Logistic Regression	[156]
	Ridge regression	[156]

	SVR	[126], [137]
Fuzzy-based	Fuzzy Logic regression	[162]
Neural-based	ANN	[158], [142], [122], [116], [119]
	FLANN	[165]
	ELM	[143]
	LSTM	[160]
	FFDNN	[121]
Homogenous Ensemble	Bagging	[134], [164], [139], [136]
	Boosting	[166], [138], [118]
Heterogenous Ensemble	Voting	[167], [157], [155], [148], [123], [124]
	Stacking	[147], [141]
	Dynamic voting	[145], [140]
Hybrid Model	Linear regression with non-parametric SVM, ANN ABE models	[149]
	hybrid Modified GA with MRL perceptron	[146]
	Canonical Correlation Analysis with Restricted Boltzmann Machines (MCR)	[144]
	SVR-Reptree	[132]
	K-means Clustering with ML algorithms (CHAID, C-R Tree, and Generalized Linear Regression)	[128]
	Fuzzy-logic enhanced Decision Tree	[115]
	Neuro-Fuzzy Network	[161], [125]
Automated Machine Learning (AutoML)	AutoSklearn	[129]

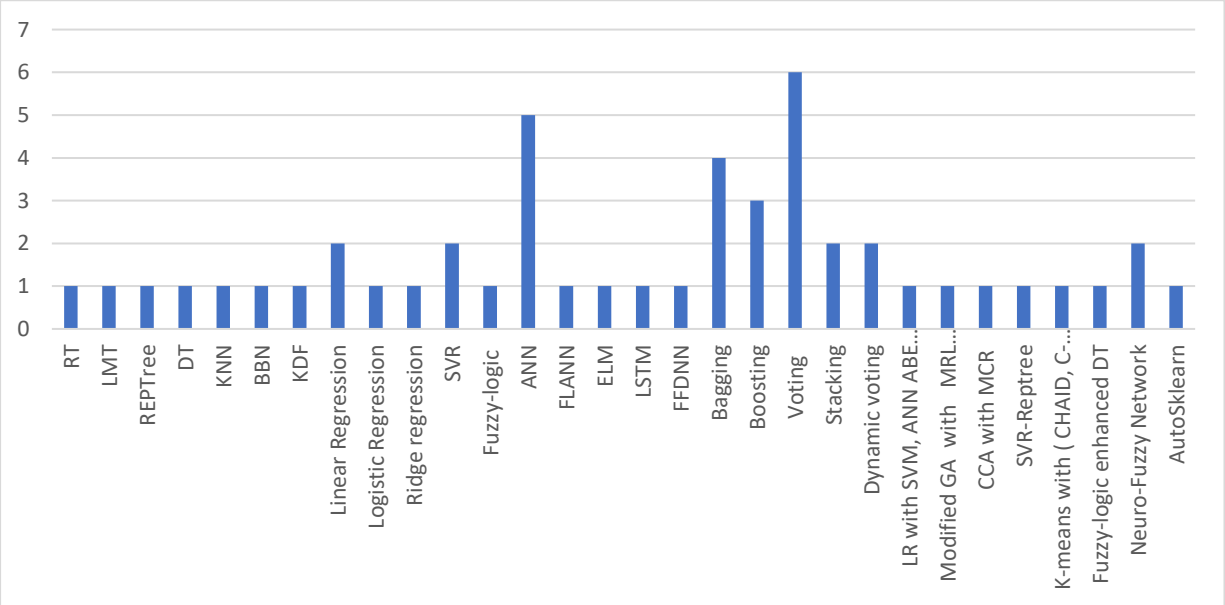


Figure 2.9. ML algorithms used in SEE studies

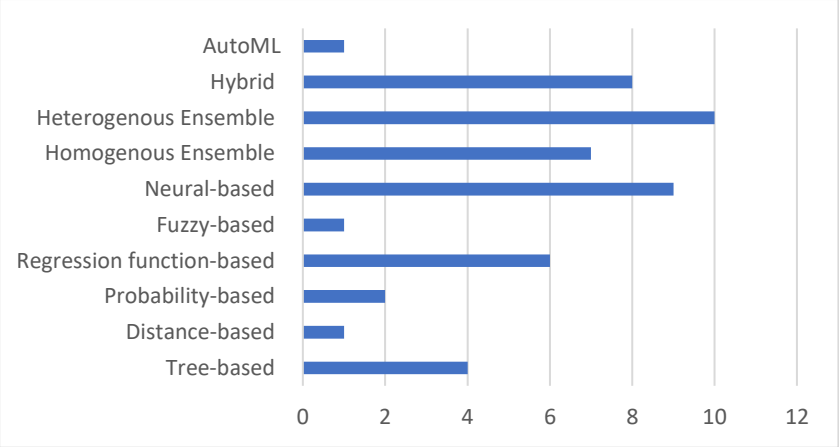


Figure 2.10. ML methods by classification in SEE

The earliest studies in machine learning (ML) for software effort estimation (SEE) utilized regression function models to automatically estimate software effort, thereby enhancing the expert judgment used in software development houses and industries. Notable regression-based models identified within the scope of this mapping study include multiple linear regression used for estimating the effort of individual tasks in software projects [127], and SVR used for estimating software construction phase effort [137]. These studies confirmed that software development efforts can be precisely estimated if the effort is broken into phases of development. Additionally, linear, ridge, and logistic regression models were employed in [156] to estimate the efforts of agile software projects. With the distance-based model, Karna et al. [135] employed KNN to predict the effort of sprints in the later phase of an agile project based on models built iteratively from the sprints of the initial phase. Probability-based models have been sparingly employed for SEE; Fuentetaja et al. [153] applied multi-step BBN to address the uncertainty aspects of software development through semi-automatically generated estimation models, while Kocaguneli et al. [151] implemented a kernel density function as a non-uniform weighting strategy for analogy-based software effort estimation. Tree-based ML models are primarily employed in cross-company and agile SEE studies. Those studies include RT for Dynamic Cross-company Learning (DCL) [171], REPtree for online learning in cross-company effort estimation [131], and LMT and J48b decision trees to enhance the human estimate from planning poker in agile effort estimation [134]. Neural-based models are the most frequently used ML algorithm for SEE; their applications are in three forms: simple ANN, enhanced variants of ANN, and deep learning models.

A simple ANN based on the Post-Architecture Level of the COCOMO II model was developed in [158] to improve the accuracy of estimating software development efforts. Similarly, an MLP-based architecture of ANN was implemented in [142] for estimating software development efforts using use case points. Regarding agile software projects, ANN with text vectorization techniques was employed in [122] for predicting software effort based on text analysis of user stories, while in [119] an MLP-based ANN was used to predict sprint effort based on extracted project and human-related features. Other enhanced variants of ANN identified include FLAAN, used in [165] for estimating software development effort with active learning; ELM, a feed-forward ANN employed in [143] to model software development

effort estimation; and Orthogonal Array Tuning Method (OATM)-optimized ANN introduced in [116] to improve the learning speed and estimation accuracy of an SEE model. The results of these studies demonstrated the superiority of the enhanced variants over the simple ANN and other ML models for SEE. Focusing on deep neural architectures, [121] proposed using FFDNN integrated with the binary search algorithm for optimal hyperparameter configuration in SEE, and LSTM was used by Favero et al. [160] on pre-trained word embedding models to estimate software effort based solely on requirements texts.

Apart from individual ML models, ensemble and hybrid ML models have also been widely employed in SEE. In the ensemble method, homogenous and heterogenous models have been implemented in SEE studies identified. With the homogenous ensemble approach, bagging and boosting are the most prominent in SEE. For instance, bagged ensemble of ANN models was used in [164] on collected effort estimation datasets; the Synthetic Bootstrap ensemble of Relevance Vector Machines (SynB-RVM), which simultaneously provides both point prediction and prediction interval estimation with confidence levels, was implemented in [166] implemented and evaluated on publicly available SEE datasets; a tree-based bagging ensemble of DTs method was employed in [139] in combination with feature extraction and recursive feature elimination to build a robust SEE model; RF bagging ensemble was also used in [136] to investigate the early-stage UCP-based effort estimation. Empirical analyses from these studies revealed the superiority of the ensemble models over the individual ML models. Concerning boosting ensemble models for SEE, [118] applied a gradient-boosting model for estimating large-scale software development. The method outperformed other ensemble learning and regression models when applied to Cocommo81 and China datasets. [138] implemented and evaluated the Stochastic Gradient Boosting (SGB) ensemble method to estimate the development effort of agile software based on story points. An empirical assessment of 21 agile software projects indicated that the SGB technique outperformed other already-used ensemble methods, such as RF and ML techniques, such as DT. Three heterogeneous ensemble methods were employed: Voting, Dynamic voting, and Stacking. These heterogeneous ensemble methods have been applied in both traditional software development [123,140,141,147,148,155,157] and Agile methodologies [124,167], covering function point, use case point, analogy, and story point-based SEE methods.

Regarding the traditional method, Kocaguneli et al. [155] investigated the performance of ensemble methods built from multiple most stable solo ML-based effort estimation models across different performance metrics to predict the software development effort accurately. Following this, numerous studies have explored heterogeneous voting and stacking ensembles to build more robust and stable models for SEE. In [157], the Generalized Linear Model (GLM) and MLP are combined using the average voting method to build an ensemble SEE model. Average voting was also used in [123] to build a SEE model, producing a more accurate estimate. In [148] ensemble of three different effort estimation approaches: Use Case Point, Expert judgment, and analogy-based CBR was implemented to support software development firms. The parallel approaches' estimates were aggregated using the combination rules: mean, median, and inverse rank weighted mean. An enhanced version of voting

method, dynamic voting, was introduced to SEE in [145] and [140] by building a heterogenous ensemble of dynamically selected best individual estimators based on mean and median aggregation. Some other studies employed the stacking method, combining multiple base estimators and use a metamodel for final prediction. In [147], a stacking regression ensemble model composed of LR, MLP, Random Forest Regressor, and AdaBoost Regressor as base learners and SVR as meta-learner was employed for estimating software development effort. In [141], a stacking ensemble model comprising SVM, DT, ANN, elastic net regression (EN), LASSO regression, ridge regression, and deep net (DN) as base models and Random forest as metamodel was implemented. The results of these ensemble models proved more effective than those of other ensemble approaches, such as average voting, weighted averaging voting, bagging, and boosting models.

Considering Agile software development effort, a voting ensemble of SVR, Gradient Boosting Regression, RF regression, and MLP were used in [167] to estimate effort in Scrum projects based on story points. This approach aggregates efforts of sprints within phases and efforts of phases within the project to arrive at the final estimation. Similarly, the study in [124] predicted the effort of agile sprints based on temporal data from previous sprints using an ensemble of seven individual ML algorithms. Individual predictions were aggregated based on their weights, and the ensemble model made the final prediction.

Hybrid ML systematically integrates two or more individual ML models to build a more effective model. Although they may be complex, their robustness and effectiveness in producing more accurate software effort estimates have been consistently reported. The study in [161] presented SEffEst, a hybrid framework combining fuzzy logic and ANN models for estimating software effort. Fuzzy logic was employed to fuzzify uncertain features, and different ANN architectures were tested for the model building. A similar hybrid architecture was employed in [125] but with an optimized neuro-fuzzy network. In another study [115], fuzzy logic and DT were hybridized to handle the unexplainability and uncertainty that often characterize machine learning estimation models; a fuzzy inference system was incorporated to enhance the classification accuracy of DT. These fuzzy-based hybrid frameworks achieved a significant performance improvement compared to other existing works, yielding accurate estimations in most cases. In a unique approach, Mittas et al. [149] proposed a semi-parametric method that hybridized a parametric method, Least Squares (LS) linear regression, with the non-parametric methods, Estimation by Analogy (EbA), Locally Weighed Regression (LOESS), ANN, and SVM to build a more robust software cost estimation models. The approach shows improvement in the performance of both parametric and non-parametric counterparts as it produces better results.

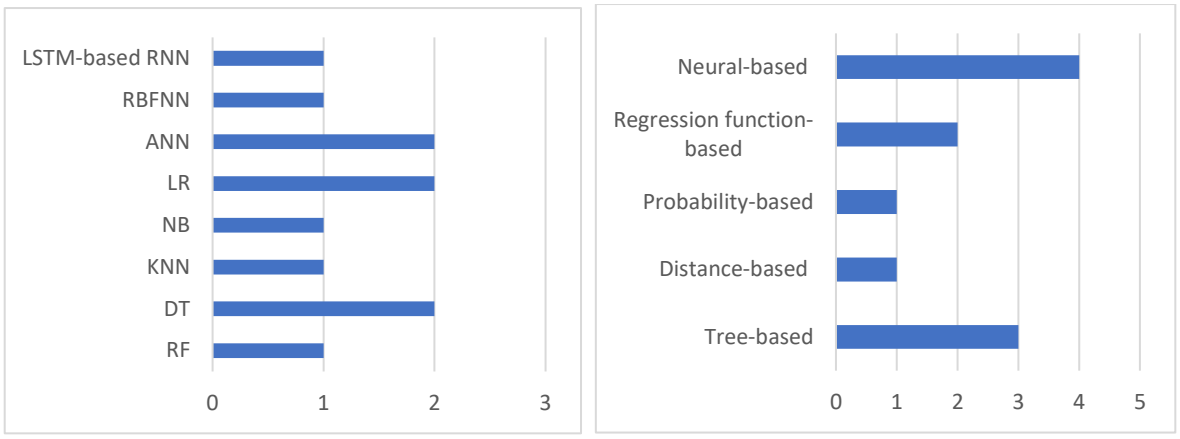
Hybrid ML models have also shown significant performance in cross-company estimation. In [144] Canonical Correlation Analysis (CCA) and Restricted Boltzmann Machines (MCR) were hybridized based on with transfer learning for heterogeneous cross-company effort estimation. They proposed a unified metric representation to transform heterogeneous SEE data into homogenous data. Addressing the comprehensibility issue in the complex black box SVR model and performance issues in the simple white box REPTree model simultaneously, [132] proposed an active learning rule

extraction-based model ALPA-R. The model extracted rules from well-performing Support Vector Regression (SVRs) to build trees that mimic the black-box model as closely as possible using REPTree. The model outperformed the original Reptree in terms of reliability and surpassed SVR in accuracy. In [128], an unsupervised clustering method was hybridized with supervised classification models for analogy-based SEE. The study employed the K-means algorithm to cluster past projects into groups. Then, machine learning models: generalized linear regression (GenLin), chi-squared automatic interaction detection classification (CHAID), and classification and regression tree (CART) models were trained using clusters closest to the new project to be estimated. The findings proved that the proposed approach is suitable for application in software development effort estimation in a real-life environment. More recently, AutoML—which automatically searches for the best-performing ML model to deploy for the task at hand by optimizing and tuning various ML algorithms—has been introduced to SEE. The study in [129] applied Auto-Sklearn to software effort estimation and compared its performance with control models: Linear multiple regression, random forest, and elastic Net. A logarithm transformation was applied to categorical variables before Auto-Sklearn was used, and the search time of the model was also adjusted to measure its influence. Results from empirical experiments emphasized the superiority of auto-sklearn with a logarithm transformation over other models, and it also exhibited the best search time, ranging from 20 to 30 minutes for SEE. This work has opened research exploration into the area of auto-ML for SEE.

Regarding the application of ML algorithms in SPS, this mapping study found only five research articles that applied eight different ML algorithms specifically for team recommendation and software project duration prediction. Table 2.9 presents the mapping of the studies to the machine learning algorithms used in software project scheduling over the last decade, along with their classification. Figure 2.11 illustrates the usage statistics by individual ML algorithms and by classification.

Table 2.9. ML algorithms used in SPS

Classification	Algorithms	Studies
Tree-based	RF	[130]
	DT	[130], [169]
Distance-based	KNN	[130]
Probability-based	NB	[130]
Regression function-based	Logistic Regression	[130], [170]
Neural-based	ANN	[130], [154]
	RBFNN	[154]
	LSTM-based RNN	[7]



A B

Figure 2.11. (A) ML algorithms used in SPS studies (B) usage by classification

Tree-based, regression-based, and neural-based ML algorithms were used more frequently in software project scheduling than distance-based and probability-based models, which were primarily employed as baseline methods for comparison in studies. Moreover, ML methods were applied mainly in team recommendation and project duration prediction as subtasks of software project scheduling, except for the work in [174], which predicted the success or failure of a software engineering project based on team distribution, considering both local and global teamwork. The DT model was trained with a team distribution dataset to predict failure in the process and the final product at specific intervals. Evaluation results revealed that globally distributed projects produce fewer failures than locally distributed ones from a process perspective.

For team recommendation, [130] proposed an intelligent software team configuration recommendation system called RECAST to satisfy the role requirements, teamwork compatibility, and technical skills. Taking a task description and a task assignee as inputs, RECAST employed the Max-Logit algorithm to rank software teams based on the team fitness scores computed by the RF model. A similar approach was used in [170], where a modified Max-Logit algorithm based on logistic regression was employed to predict the top K best teams for specific tasks in software development. The proposed method extended the model in [97] by incorporating win experience, role experience, win rate, and team closeness as features to recommend and use team strength as the weighing function. Considering agile software project scheduling and resource allocation, [7] introduced TaskAllocator, which employs deep learning to predict the optimal team role for incoming tasks based on knowledge from historical projects in a distributed agile software development scenario. LSTM, a variant of RNN with pretraining on embedding, was trained on tokens extracted from textual task descriptions to predict the task's appropriate role. Evaluation of case study projects from Taigo.io indicated the superiority of the approach over other existing deep learning architectures and ML models for accurate task allocation in agile software project planning. In the context of duration prediction, [154] investigated the performance of MLP and RBF neural networks in comparison to the baseline MLR approach for estimating the duration of new software projects. In their work, functional size and maximum team size were used as

the features to predict the duration. With superior performance, their work underscores the effectiveness of neural network models in predicting software development project duration.

Regarding the application of ML algorithms in SOP, this mapping study found that no work has been published in the last decade on the topic. It is, therefore, essential for future studies to explore this direction, given the potential of ML methods in modeling experts' preferences and patterns for the effective planning of software overtime.

2.2.3.3 *To what extent have studies explored the synergistic integration of ML and MOO in SPP, and what interaction strategies have been employed?*

An intrinsic synergistic relationship between ML and search-based metaheuristic optimization algorithms has been established and well-studied within the software engineering and data mining communities [45,46]. It is observed that ML can help optimization algorithms identify suitable regions in the solution space, enabling them to find the best solutions more efficiently. Additionally, metaheuristic algorithms could aid in finding superior parameter settings and the most discriminating subset of features for an ML model. Researchers have explored this synergistic relationship to solve various software engineering problems, such as software defect prediction [100,178] and requirement engineering [48], by drawing insights from ML-based software analytics — often referred to as Mining Software Repositories (MSR) — and SBSE. This has led to a subfield of SBSE called Data-driven Search-Based Software Engineering (DSBSE) [45]. This study extends the exploration to SPP by investigating the synergistic interaction between ML and search-based MOO in the SPP activities already identified. Based on findings from existing studies, two interaction strategies between ML and MOO are determined as follows:

1. *MOO-based ML*: MOO is utilized within the learning framework of an ML model to enhance its capabilities; the goal is to classify, predict, or estimate.
2. *ML-based MOO*: Machine learning (ML) is incorporated into the optimization framework of a multi-objective optimization (MOO) algorithm to accelerate its search; the goal is to optimize processes.

To address the question on synergistic interaction, studies that employed MOO and ML within a single framework or architecture were extracted from all identified articles and classified according to the interaction strategy.

First, the proportion of all identified studies that interactively combined MOO and ML for the SPP activities was found to be only four studies (5.63%). This suggests that this interactive strategy has not received sufficient attention from researchers in SPP despite its envisaged superior performance. It was observed that the studies applying this strategy originated from the SEE domain, where most studies are ML-based. In the other domains of SPS and SOP, where most of the studies are MOO-based, no study employed the interactive strategy. These findings confirm the existence of open research issues in the domains of SPP. Table 2.10 presents the identified studies that applied the

interaction strategies between MOO and ML in SEE. It details the interaction mode, the combined MOO and ML algorithms, and the problem addressed.

MOO-based ML approaches have been introduced in software engineering to address the reported scalability drawback of the Bayesian optimization method commonly used by the ML community for parameter tuning [46]. Bayesian optimization relies heavily on Gaussian Process Models (GPM), which do not scale well beyond a dozen variables, as reported in [45]. As a result, researchers in software engineering often substitute the Bayesian optimization method with metaheuristic MOO algorithms that can scale effectively for many variable combinations. In SEE, this work found two studies that implemented this approach. The study in [168] employed MOPSO to optimize the hyperparameters of SVR, while the study in [126] utilized the Firefly optimization algorithm for optimal tuning of the fuzzy analogy model. These studies reported improved performance compared to other parameter-tuning methods commonly used in the ML community. Apart from parameter tuning, MOO has also been employed for optimal feature selection in ML models for SEE. The study in [159] utilized SBO to identify the optimal feature subset for an ANFIS model for effort estimation. The study's results indicated the superiority of the SBO compared to other feature selection models. These convincing results, which demonstrate the effectiveness of using MOO to complement ML and optimize its performance, confirm the synergistic interaction between MOO and ML.

Table 2.10. Studies on the interaction between MOO and ML in SEE

Studies	MOO Algorithm	ML Algorithm	Interaction Mode	Problem addressed
[163]	HaD-MOEA	MLP	ML-based MOO	Finding optimal MLP models for software effort estimation
[168]	MOPSO	SVR	MOO-based ML	Finding the optimal hyperparameter setting of SVR for building an imputation model in SEE
[159]	Satin Bowerbird Optimization (SBO)	ANFIS	MOO-based ML	Finding the optimal structure and feature of an ANFIS framework for software effort estimation
[126]	Firefly Optimization Algorithm	Fuzzy analogy	MOO-based ML	Finding optimal parameters for a clustering-based fuzzy analogy model for SEE

On the other hand, ML-based MOO has not been actively studied in SEE. This study found only one article [163] that employed this strategy. In the article, the authors approach SEE as a multi-objective optimization problem, incorporating an ML model within the MOO framework. Instead of using the mathematical formulation for modeling effort estimation as usually done by other MOO studies in SEE [82,114,120,152], the authors used MLP to build estimation models and applied HaD-MOEA to search for optimal models that minimize MMRE, LSD, and maximize Pred(25) as objectives. Their model generated a Pareto ensemble of MLP models, exhibiting optimal performance in estimation. Their strategy of replacing the mathematical model with an MLP for multi-objective effort estimation yielded

a more practical approach to addressing the SEE problem, offering greater flexibility for project managers.

The two-way synergistic interactions between MOO and ML have been explored only in SEE. Most studies used MOO within the ML framework to optimally select features and tune hyperparameters—only one study employed ML within the MOO framework. This study found no studies on SPS and SOP that implement the synergistic interaction between MOO and ML.

2.2.3.4 *What software development or project management methodologies are most commonly considered in studies applying ML and MOO to SPP?*

Depending on project size, team size, criticality, and complexity, software project management can be approached using Traditional, Agile, and Hybrid methodologies. The traditional method follows the generic waterfall life cycle, where project tasks are carried out in succession until the product is completed and delivered. The agile approach focuses on customer satisfaction and rapid development, where workable versions of the product are delivered in intervals, known as sprints. The hybrid approach combines the two former approaches to handle complexity while achieving speed simultaneously. To answer this question, the software project management methodologies considered in the identified articles were extracted, and the proportion of studies focusing on traditional, Agile, and hybrid approaches was computed. Nine articles (12.68%) out of 71 were found to have applied MOO and/or ML to SPP problems in an agile environment. Only one article (1.41%) considered the hybrid project management methodology, and the remaining 61 (85.91%) were based on traditional software project management methodologies. Figure 2.12 presents a detailed analysis of these results.

It is evident from the results that the majority of MOO and ML studies in software project planning are still based on traditional approaches to software project development or management. A small percentage considered the agile approach, despite having some evidence of its efficiency and acceptability in the software development industry. The limited research in agile software project planning using MOO and ML may be due to the lack of publicly available datasets and project instances for testing intelligent learning algorithms. Moreover, the sheer idea of agile SPP makes it harder to plan and optimize as it accentuates short-term tasks, frequent updates of plans, and adapting to the current situation. Considering analysis based on the identified software project planning activities, SEE has attracted more studies on Agile approaches (eight articles). SPS only attracted one study on agile and one on a hybrid approach. No agile or hybrid approach was found in SOP.

Further analysis of studies that applied ML and MOO algorithms in Agile software project planning was also conducted. As presented in Table 2.11, it was observed that most works on Agile software project planning employed ML, primarily in SEE; only one study applied ML in agile-based SPS for task allocation. Studies in agile-based SEE, as identified in this work, have explored individual ML algorithms [135,156], homogeneous ensembles [134,138], heterogeneous ensembles [124,167], and neural-based ML algorithms [119,122]. The studies were primarily based on private datasets collected from software houses, as in [119,122,124,134,135,138,156]; only one study [167] utilized

project datasets from a public repository. Moreover, the contributions of the studies can be categorized into three: those that predicted the effort of sprints [119,124,135,167], those that predicted the effort of the entire agile project based on story points [138,156], and those that supported human experts in the estimation of story points [122,134].

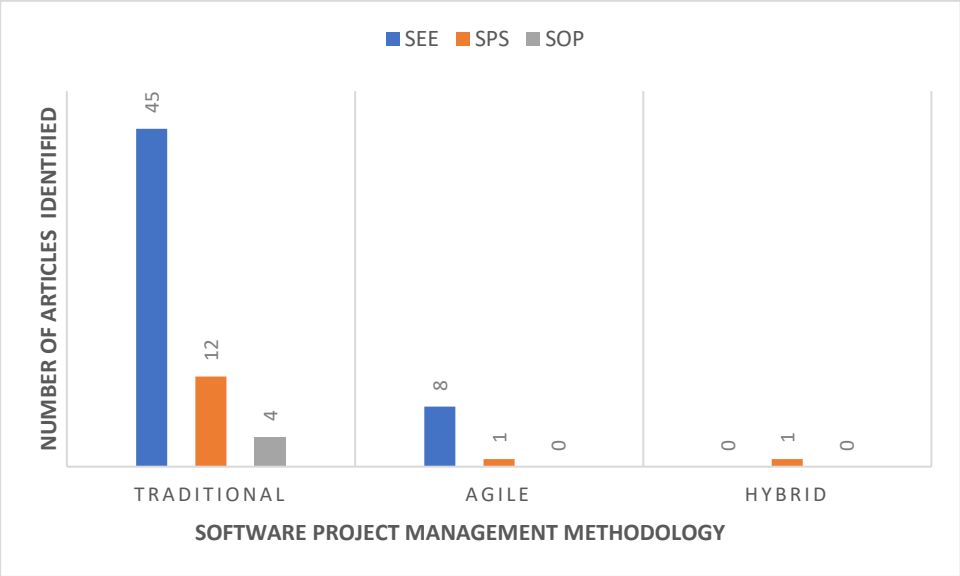


Figure 2.12. The proportion of identified articles based on software project management methodologies

Table 2.11. ML studies in agile-based software project planning

Studies	Algorithm	SPP activity	Main Contribution	Data Source
[135]	KNN	SEE	predict the effort of sprints in the later phase of an agile project based on models built from previous sprints to improve on expert estimation of agile project	Private telco project data
[167]	Heterogenous ensemble of SVR, GBR, RFR and MLP	SEE	estimate effort in scrum projects using story points by aggregating efforts of sprints within phases	Open source project scrum dataset (Apache, JBoss, JIRA, MongoDB)
[124]	Heterogenous ensemble of seven ML algorithms	SEE	predict the effort of agile sprints based on temporal data of previous sprints	Private dataset from the IT department of a University
[138]	Stochastic gradient boosting	SEE	Estimate the development effort of an agile software project based on story points.	Private agile software project dataset from a software house
[156]	Linear, Ridge, and Logistic regression	SEE	estimate the effort of agile software projects from a story point	private datasets on agile development projects from 6 software houses

[122]	ANN with word embedding	SEE	Extract keywords from the agile story card with word embeddings to build a feature set for training an ANN model in agile effort estimation.	Private local dataset from a software company
[119]	MLP	SEE	predict effort of sprint in agile software project based on project and human-related features	Simulated dataset and private dataset from a company
[134]	DT, RF, and LMT	SEE	enhance the human estimate from planning poker of agile effort estimation through auto-estimation with ML algorithms	Data from 110 teams using IBM Rational Team Concert
[7]	LSTM	SPS	predict the perfect team role for incoming tasks based on knowledge from historical projects in a distributed agile software development scenario using a deep learning	10 case study projects from Taiga.io

The study in [7] (the only research on agile-based SPS identified in this work) employed an advanced ML algorithm based on LSTM to allocate tasks to a team of developers using selected case study projects from a public repository of agile project management tool: Taiga.io. The work introduced a deep learning approach to task allocation in a distributed agile environment.

A study [72] was identified applying MOO to SPS in a hybrid software project management environment combining traditional and agile methodologies. The study employed a mixed-integer multi-objective genetic algorithm that considers the pairwise synergies among developers and handles the flexibility of the software project structure to adapt agile project features. However, the project dataset was based on the simulated project networks generated from the multi-skill resource-constrained project scheduling problem generator created by Myszkowski et al. [179].

In summary, the findings from this study revealed a limited body of research in agile-based software project planning. The few studies identified focused mainly on SEE and sparingly on SPS. In addition, ML algorithms dominate MOO when considering applications in agile-based SEE, while both were used equally in SPS. No record of Agile-based approaches was found in SOP. The limited research in this domain may be due to the scarcity of publicly available project data on agile-based software development, as well as the difficulty of obtaining private datasets from software companies, which is often hindered by confidentiality and bureaucratic constraints.

2.2.4 Existing Gap and Open Research Issues

Significant advancements have been achieved in the application of MOO and ML to SPP over the past decade. Nonetheless, several unexplored and unresolved research issues remain that are essential for progressing the current state-of-the-art in SPP. The identified open issues are outlined as follows.

(i) *Search-based multi-objective effort estimation***:** search-based SEE has been extensively researched over the past decade. Nonetheless, most existing methods primarily generate point estimates and focus on a single objective. Among the limited number of multi-objective studies identified, many have concentrated on enhancing point estimate accuracy by exploring trade-offs among various performance metrics as objectives. There is a need for research that incorporates both point and interval estimates to achieve more robust and reliable estimation. Future investigations should also aim to extend current bi-objective frameworks into many-objective formulations, thus enabling the application of more sophisticated search-based multi-objective optimization algorithms within SEE.

(ii) *ML-based software project scheduling:* SPS has primarily been addressed through search-based optimization techniques, including MOO; however, few studies have utilized ML methods. Most existing ML research in SPS centers on team recommendations. Accurately predicting the duration of individual tasks in software project schedules remains difficult, mainly due to the lack of available datasets. The only study on duration prediction applies its model to entire software projects. Future research could focus on creating valuable datasets using software project management tools and repositories. Additionally, applying advanced machine learning models to these datasets for task duration prediction could lead to more accurate and informed project schedules.

(iii) *ML-based software overtime estimation:* In recent times, there has been limited focus on overtime planning in software development projects among SPP researchers. The few existing studies approached it as a search-based multi-objective optimization problem, which faces challenges related to complexity and industry acceptance. However, with appropriate data such as project and developer details, ML models can be created to predict the overtime required in software projects by learning from project managers' expertise. Collecting and generating such datasets remains a significant challenge in SOP, and further research in this area is highly valuable. Additionally, industry practitioners are likely to adopt and trust ML-based overtime estimates because these models will incorporate project managers' subjective preferences..

(iv) *ML-driven MOO for SPP:* This study found that MOO is dominant in SPS and SOP, while ML leads in SEE. Combining these two computational intelligence methods offers promising potential but has received limited research attention. Although some studies have integrated MOO within ML models for SEE, ML-driven MOO approaches are rarely used in SPP. Research in other software engineering fields, such as requirement prioritization, design, and defect prediction, shows that ML can effectively improve MOO algorithms by helping to identify optimal regions in the search space and validating solutions. Future research should explore this synergy in SPP. For example, ML can be integrated into search-based SPS frameworks for task allocation and duration estimation, enabling MOO to plan schedules more effectively. This improves search efficiency and prevents the generation of infeasible solutions. Similarly, in SOP, ML can incorporate the experience of project managers to predict the acceptability of

overtime plans generated by MOO, ensuring solutions are both optimal and practical for PMs. A comparable approach can be applied in SEE by developing a search-based effort estimation method that uses ML-built models instead of traditional mathematical formulations, aiming for more accurate and effective estimates.

(v) *Agile-based SPP:* The agile approach has become the standard in the software industry. SEE has been approached with several agile methods that are useful in practice. However, a gap remains between research on software project planning (especially in SPS and SOP) and current industry practices. To bridge this divide, researchers need to develop scheduling and overtime planning methods suited for an agile environment. Additionally, existing research on agile-based methods for SPP relied on private datasets, as seen in agile SEE studies. This is a challenge for SPS and SOP, since software companies are often reluctant to share planning data for empirical model testing. It would be valuable for the research community if new efforts focused on collecting data necessary to develop and validate Agile-based planning models..

2.3 Interactive Multi-Objective Optimization Techniques in Software Engineering

In some fields of engineering, adhering to regular Pareto dominance in multi-objective optimization has the drawback of producing several undesired solutions from the DM's perspective and may result in a lengthy optimization process [180]. This has led to the introduction of preference-based multi-objective optimization techniques, which incorporate DM preferences into the optimization process to guide the search. The primary goal is to ensure the search algorithm prioritizes solutions most relevant to the DM. This is achieved through several methods, including weight vectors, reference points, regions of interest, and tradeoffs. The preferences can be incorporated in three ways: before the optimization process begins (a priori), after the main optimization process (a posteriori), and during the optimization process (interactive). In situations where the DM can be engaged to provide their subjective evaluations, the interactive approach is most recommended [181] because it allows DM to adapt their preferences to newly obtained solutions and their coordinates in the objective space.

Interactive multi-objective optimization integrates real-time DM feedback into the optimization steps of metaheuristic algorithms to guide the search toward preferred solutions. It is implemented using two approaches: goal programming and Human-In-the-Loop (HIL). In goal programming, the optimization algorithm attempts to minimize the deviation of the search from the DM-defined reference points adaptively. Specifically, the DM dynamically updates reference points, and the algorithm finds solutions closest to them. An illustration of this approach as presented in [182] is depicted in Figure 2.13. The questions of *why* a reference point has been mapped to a specific solution and *how* the reference point could be changed to achieve a desired result are highlighted. In the HIL approach, DM provides feedback in the form of evaluation of solutions iteratively at each optimization step until desired solutions are found. It follows the simple procedure described as follow:

1. Algorithm generates initial Pareto front.
2. DM provides feedback (e.g. subjective evaluation).

3. Algorithm refines solutions based on feedback.
4. Repeat until DM is satisfied.

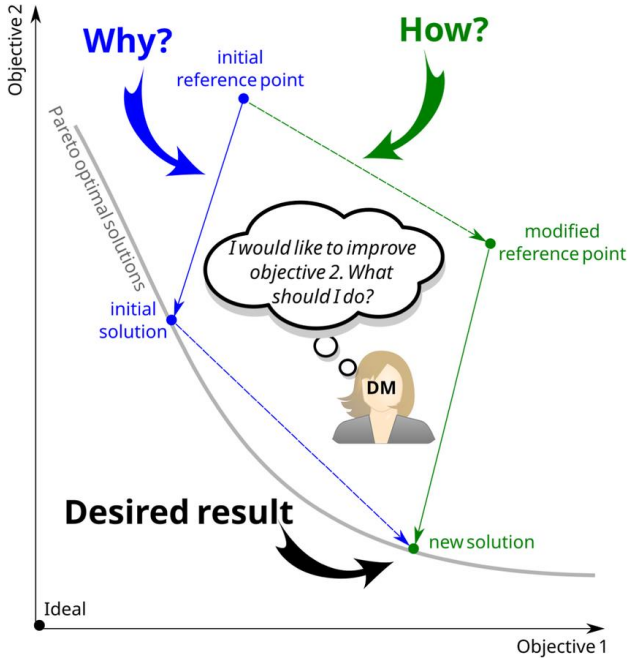


Figure 2.13. Reference point-based interactive multi-objective optimization (source: [182])

HIL is highly popular in interactive multi-objective optimization for two reasons. First, it can find solutions that best meet the DM's subjective needs [183]. Second, it results in a reduction of up to 50% in optimization iterations [184], leading to faster convergence. HIL is commonly used in SBSE studies to integrate developers' subjective preferences into the optimization procedure of search-based multi-objective evolutionary algorithms. This is because software engineering problems often involve human-centric processes that objective optimization alone may not be able to handle. Since this thesis is based on an interactive multi-objective optimization approach, it is imperative to review studies in software engineering that have implemented the strategy. Several interactive search-based optimization algorithms have been developed and applied across various subfields of software engineering. The interaction mode in these studies can be categorized into two: direct and semi-indirect interaction.

In direct interaction, the DM is physically involved in the optimization process to provide subjective evaluation. [185] introduced a novel interactive ant colony optimization framework by embedding user preferences directly into the optimization process for the Next Release Problem. By directly translating qualitative user input into quantifiable preference metrics, the approach ensures that the resulting software release plans more accurately reflect the end users' priorities. In their experimental evaluation, the authors applied their enhanced algorithm to a set of benchmark instances that simulated realistic software release planning scenarios. The experiments demonstrated that this interactive approach produced release plans with significantly better alignment to stakeholder

expectations. [44] proposed an interactive software release planning method that uses a Preferences Base provided by the PM during the search process of an IGA. This approach incorporates the PM's preferences as penalties for candidate solutions, depending on the importance of each preference that was not satisfied. Regarding software design optimization, [40] explored interactive Ant Colony Optimization (iACO) for the software design process, where both objective and subjective factors influence the search. The user participates by providing a numeric evaluation (from 1 to 100) for feasible solutions, represented as UML diagrams. Similarly, [43] suggested involving developers in software re-modularization tasks using IGA. This algorithm employs a fitness function that combines modularization quality with a human-assessed factor, determining whether two components should reside in the same module, thereby acting as a constraint to penalize solutions. The findings demonstrate the approach's effectiveness in improving cohesion.

The direct interaction approach has also been applied in software testing and maintenance. [186] introduced a Differential Evolution algorithm for testing embedded software, utilizing a technique that largely automates test data generation while allowing domain experts to contribute their knowledge and experience. To direct the search, the domain expert specifies the relative importance of the quality objectives. In terms of software maintenance, [187] proposed an IGA method to identify suitable refactoring suggestions based on a set of examples. The fitness function combines the objective value of refactoring solutions during the IGA process with the developer's ratings. Results indicated that this method is consistently accurate.

In the semi-indirect approach, DM partially interacts with the optimization algorithms in some initial iteration steps, after which a machine learning model is trained on the interaction pattern to replace the DM in subsequent iterations. [39] proposed an architecture for incorporating human experience and preferences in the search process of an Interactive Genetic Algorithm (IGA) for requirements engineering. The architecture utilizes a machine learning model to replace PM's interactions with the search process when it becomes overwhelming. Similarly, [188] proposed neural network-based fitness evaluation within the IGA framework to model the subjective fitness function in search-based software refactoring. The software engineers manually evaluate the suggested refactoring solutions by GA for a few iterations. Then, an ANN uses the evaluation examples provided by the engineers to assess the refactoring solutions for the remaining iterations. Their case study demonstrates that the adaptive fitness function leads to improved refactoring outcomes, achieving a better balance between conflicting objectives such as code quality and maintainability while also enhancing the efficiency of the search procedure.

2.4 Analysis of Related Works

The analysis of the limitations of existing studies on software overtime planning and interactive optimization for software engineering problems, in comparison with the proposed study, is presented in Table 2.12.

The table presents a comparative analysis of existing studies, highlighting their use of overtime planning, multi-objective formulation, and interactive approaches, as well as the specific modes of interaction and incorporation of machine learning. Several studies (e.g., [37], [36], [30]) focus on overtime planning and multi-objective formulations without deploying interactive approaches or machine learning, while others (e.g., [40], [186], [43]) integrate direct or semi-indirect interactive modes. Notably, the study in [38] distinguishes itself by incorporating all three features, including a partial interactive approach, and [39] employs a semi-indirect mode with machine learning integration. In contrast, the proposed method advances the current research landscape by combining overtime planning, multi-objective formulation, a fully interactive approach via an indirect mode, and machine learning, thereby suggesting a more holistic and enriched framework.

Table 2.12. Gab Analysis of the Existing Studies

Studies	Overtime planning	Multi-objective formulation	Interactive approach	Interaction Mode	Machine Learning based
[37]	*	*		None	
[40]		*	*	Direct	
[44]			*	Direct	
[36]	*	*		None	
[186]		*	*	Direct	
[38]	*	*	*	Partial	
[48]			*	Semi-indirect	*
[30]	*	*		None	
[185]			*	Direct	
[188]		*	*	Semi-indirect	*
[43]		*	*	Direct	
Proposed	*****	*****	*****	**Indirect**	*****

CHAPTER 3 METHODOLOGY

In this chapter, a novel interactive multi-objective optimization framework is designed, and the methodology for implementing it to solve the software overtime planning problem is presented. The methodology describes the materials (software project datasets), methods (problem formulation, algorithms, and models), and metrics (performance measures) to be used in assessing the proposed framework.

3.1 Machine Learning-Based Interactive Multi-objective Optimization Framework

The machine learning-based interactive memetic multi-objective optimization framework proposed in this study, as illustrated in Figure 3.1, comprises three main steps: Problem Modelling/Formulation, development of the machine learning-based interactive memetic algorithm, and Evaluation. The interactive algorithm based on machine learning consists of three interconnected components: the memetic multi-objective search optimization algorithm, the interaction module, and the machine learning model.

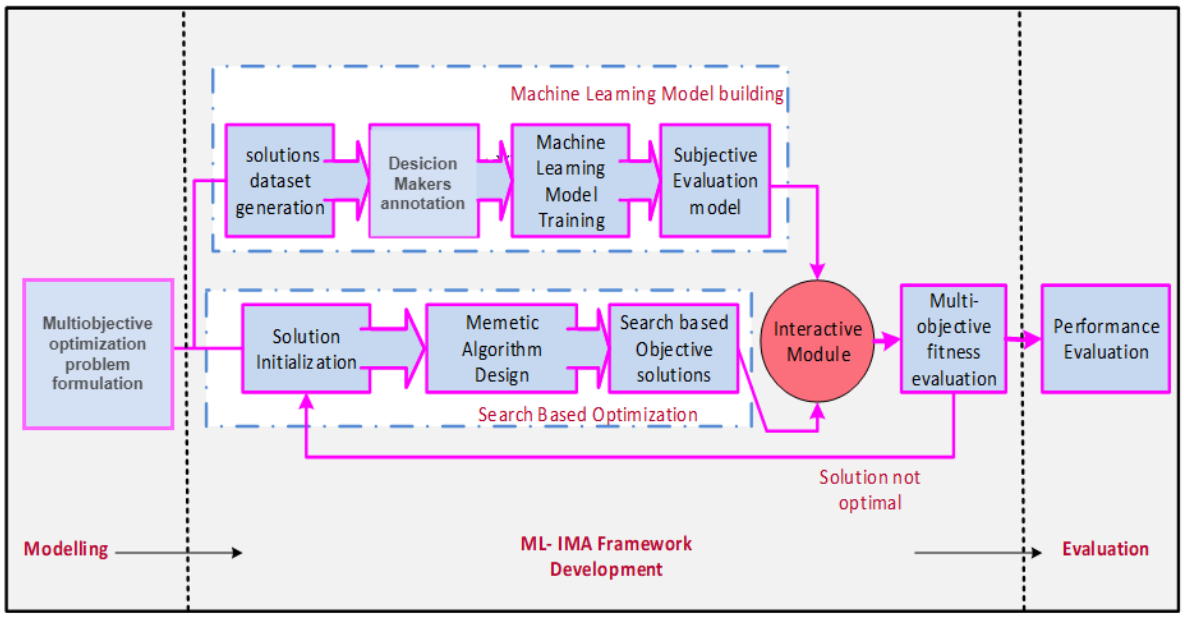


Figure 3.1. ML-based interactive multi-objective optimization framework

At first, an interactive multi-objective optimization problem is formulated for a software engineering application and modelled to facilitate interaction with a machine learning model. Using the formulation, instances of initial optimal solutions are generated based on real-world software project datasets collected. The solutions are then presented to DMs for annotation, which in turn is used to build the ML model that can learn the subjective preferences of the DMs. A memetic multi-objective optimization algorithm is developed to find an optimal solution to the problem explicitly. Then, the ML model, built a priori, is integrated with the algorithm via an interaction module to guide the search algorithm interactively to the preferred region of interest or solutions. In the last phase, the solutions

produced by the proposed algorithm are evaluated based on multi-objective solution quality metrics and interactive solution measures.

3.1.1 Machine Learning (ML) model

In the proposed framework, the ML model is developed separately before initiating the search-based optimization process. Creating an effective and precise predictive model requires training data, which is often limited, particularly in software project management. A machine learning model suited for working with a moderately small dataset would be more appropriate here. Initially, a large set of solutions will be generated based on problem modeling and empirical data. Subsequently, software engineering experts will evaluate these solutions subjectively, based on their experience with similar issues, assigning a numeric score (ranging from 1 to 100) to indicate the solution's applicability, as referenced in [40]. This annotated data will then be used to train a predictive model to assess the solutions produced by the optimization algorithm.

Given the significant efforts required from the PMs, which may result in a limited annotated dataset and anticipated high dimensionality contingent upon the complexity of the software projects, an optimized tree-based learning model is recommended due to its documented performance in mitigating bias and overfitting. Furthermore, the inherently numeric attributes and the continuous nature of the predicted value, coupled with a relatively small sample size, suggest that a regression-based approach is more appropriate for the machine learning model. Consequently, an appropriate method is to implement an ensemble-based regression tree model—a meta-estimator that can fit several regression trees on various sub-samples of the dataset, using averaging to improve predictive accuracy and control over-fitting—for estimating DM satisfaction with the multi-objective solution produced by the search-based multi-objective optimization algorithm.

3.1.2 Interactive memetic multi-objective optimization algorithm

Considering the multi-objective nature and inherent complexity of most problems in software engineering, a memetic approach is proposed for the framework. This approach integrates a population-based global technique with localized search performed by each individual within the population to direct the search toward a global optimum [85,189]. Memetic algorithms are renowned for their efficient exploration of the solution space and their robust exploitation of promising regions within it [3]. They have been consistently reported to outperform genetic algorithms in terms of both speed and solution quality [30,80,85,86]. Memetic algorithms integrate local evolution with global optimization strategies adopted from the genetic algorithm. This characteristic nature of memetic algorithms makes it possible to implement an interactive algorithm with them in two ways: during local evolution steps and at the global optimization stage. The two methods can be implemented in the proposed framework, each with its advantages and drawbacks.

[85,189]. Memetic algorithms are renowned for their efficient exploration of the solution space and their robust exploitation of promising regions within it [3].

For instance, the ML model could be integrated during the local search to identify and accept only solutions that satisfy the DM's preferences in the neighborhood of the individual solutions. In this way, the preference of the DM is better exploited but at the expense of numerous interaction instances which may lead to slow convergence of the interactive algorithm. Also, it might significantly increase the complexity of the algorithm and complicate its implementation. On the other hand, the ML model can be integrated during global evolution to find the best solutions that satisfy DM's preferences at each global iteration, leading to a faster approach with reduced interaction instances. However, this method may miss some optimal and preferred solutions, as the DM's preferences are activated after the optimal solutions at each step are identified, thereby missing a better trade-off between objectively good solutions and those preferred by the DM. The choice of method to use depends on the application areas and the goal of the interactive approach.

3.1.3 Interaction Module

The interaction module functions as a communication link between the ML model and the memetic algorithm. This study extends the interaction model from [39] for the Next Release Problem (NRP) by eliminating the PM's direct role during optimization, replacing him entirely with a trained predictive machine learning model. Specifically, the module receives an individual solution from the memetic algorithm and sends it to the pretrained ML model for subjective assessment. The resulting evaluation score is then integrated into the solution's fitness score to help select the best solutions. As a result, the interaction module connects the memetic multi-objective optimization algorithm, the predictive model, and the fitness function, which uses a preference-based dominance relation to determine the superiority of candidate solutions.

In this thesis, the proposed framework is implemented for interactive software overtime planning based on real-life software projects. The problem modeling, interactive memetic algorithm design, and performance evaluation of this implementation are presented in the following sections.

3.2 Interactive Overtime Planning Problem Formulation

A software project schedule is typically depicted using a Direct Acyclic Graph (DAG), which comprises a set of nodes, denoted as $WP = \{wp_1, wp_2, \ldots . . wp_n\}$ representing the collection of work packages, along with a set of edges, denoted as $DP = \{ (wp_i, wp_j) : i, j \leq n\}$ indicating dependencies. Each work package is characterized by the effort required (measured in function points), estimated duration, and associated costs (for example, in USD). The development of an effective software project schedule has been extensively studied within the scholarly literature [3,10,14,53,70,71,76,78]. This study does not focus on constructing such schedules but instead analyzes the impact of overtime allocations on an existing schedule. For this purpose, the Critical Path (CP) method, a widely adopted planning technique for decades, is employed. An overtime plan for the schedule can be represented by a linear, finite set of *N* (the total number of work packages) integer values within the interval [0, 4], each indicating the daily overtime hours allocated to a specific work package. An example of an overtime plan for a project comprising ten packages is provided below.

0	2	1	4	0	2	3	4	2	3
0	1	2	3	4	5	6	7	8	9

In the example, the second work package is allocated 2 hours of overtime daily. The aim is to create an optimal overtime plan that aligns with the set objectives. The interactive overtime planning problem addressed here is formulated as a multi-objective optimization with four distinct goals. Since the formulation is interactive, the first three goals—total overtime, cost, and code quality—are assessed based on explicit characteristics of project schedules. The fourth goal, the project manager's (PM's) satisfaction, is predicted using a pre-trained machine learning model. Notably, this is the first instance where code quality is included as an objective, integrating machine learning into the formulation. The key objectives targeted in this overtime planning model are listed below.

1. *Total overtime:* refers to the sum of overtime hours across all work packages required to complete the project, as determined by the optimization algorithm. A well-optimized solution should have reduced overall overtime hours. Its calculation is described by Equation 3.1.

   Minimize:

$$OH = \sum_{i=1}^{n} O_{wp_i} \cdot D_{duration(wp_i)} \tag{3.1}$$

Where O_{wp_i} represents the amount of overtime spent on i[th] work package and $D_{duration(wp_i)}$ is its duration.

2. *Cost:* is the total amount of money spent to complete the project. The cost of each work package is calculated using the function points-based approach, which aligns with industry practice. Using the productivity value of 27.8 FP/developer-month, as estimated by [190] for software projects and previously used in existing studies on software overtime planning [22], 5.76 developer-hour is required to complete one function point. To calculate the final cost, the average hourly rate of developers needs to be determined. The average hourly rate for software developers varies significantly based on factors such as location, expertise, and project complexity. Based on recent data and global standards, a balanced rate that reflects both quality and cost-effectiveness would be around $75 per hour [191,192]. So the cost of each WP is computed as:

$$C_{wp} = 75 \times 5.76 \times FP_{wp} \tag{3.2}$$

   The impact of overtime hours on project cost is modeled according to Brazilian cost law [22]. If a regular working hour costs X, then the first two overtime hours cost 120% of X, and the subsequent two hours cost 150% of X. The candidate solution aims to minimize this cost, which is calculated as specified in Equation 3.3.

   Minimize:

$$Cost = \sum_{i=1}^{n} \begin{cases} C_{wp_i} + \frac{6C_{wp_i}}{5}, & if\ O_{wp_i} \leq 2 \\ C_{wp_i} + \frac{3C_{wp_i}}{2}, & if\ O_{wp_i} > 2 \end{cases} \quad (3.3)$$

Where C_{wp_i} is the regular cost of the work packages.

3. *Quality of code*: The quality of the software code is measured by the error rate caused by overtime, as modeled in [36]. The model predicts a 20% increase in errors for developers working 2 hours of overtime daily and a 50% increase for those working 4 extra hours per day. The lower the error rate, the higher the code quality. Therefore, the error rate should be as low as possible in the ideal solution.

   Minimize:

$$Error_{overtime} = \sum_{i=1}^{n} \begin{cases} \frac{E_{wp_i}}{5}, & if\ O_{wp_i} \leq 2 \\ \frac{E_{wp_i}}{2}, & if\ O_{wp_i} > 2 \end{cases} \quad (3.4)$$

4. *PM's satisfaction:* The PM's subjective evaluation of a solution is treated as a separate objective to be maximized. PM satisfaction is estimated using a trained machine learning model that relies on solutions annotated by the PM.

Constraint:

The above-listed objectives are constrained by the maximum allowed daily overtime constraint:

$$0 \leq O_{wp_i} \leq maxovertime \quad (3.5)$$

3.3 ML-based Interactive Memetic Algorithm for Overtime Planning

To solve the interactive software overtime planning problem modelled in section 3.2, an ML-based interactive MOSFLA algorithm (ML-iMOSFLA) is designed. MOSFLA is chosen as the optimization algorithm for the framework because it has been shown in a previous study [30] to outperform NSGA-II, which is mainly used in the existing studies on software overtime planning [36–38]. The interactive algorithm integrates a pretrained RFR model to capture the subjective PM evaluation of solutions during global evolution and for ranking non-dominated solutions. Additionally, an interactive multi-objective fitness function is employed to assess the fitness of each solution. Figure 3.2 depicts the flowchart for the proposed ML-iMOSFLA.

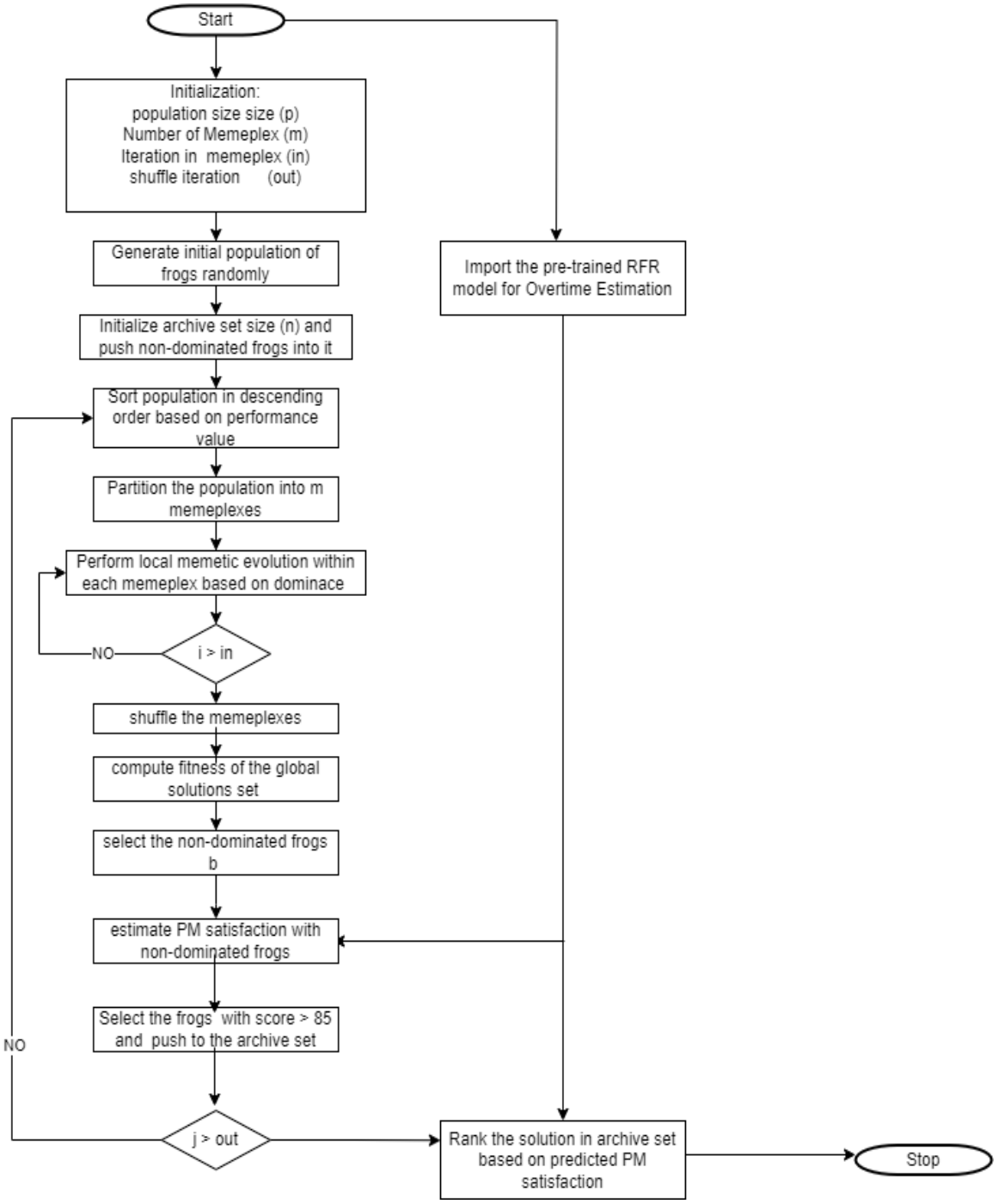


Figure 3.2. Flow chart of the ML-iMOSFLA algorithm

The MOSFLA, designed explicitly for software overtime planning by [30], is adopted in this study. It extended the original SFLA [193] (a simple framework given in Figure 3.3) by incorporating an archiving strategy with the self-adaptive niche method proposed in [194] to maintain the non-dominated solutions. It also includes an improved population sorting method and a memetic evolution process to adapt to MOO problems. Due to the group evolution mechanism employed by MOSFLA, solutions can evolve in different directions, which informs its selection in this study.

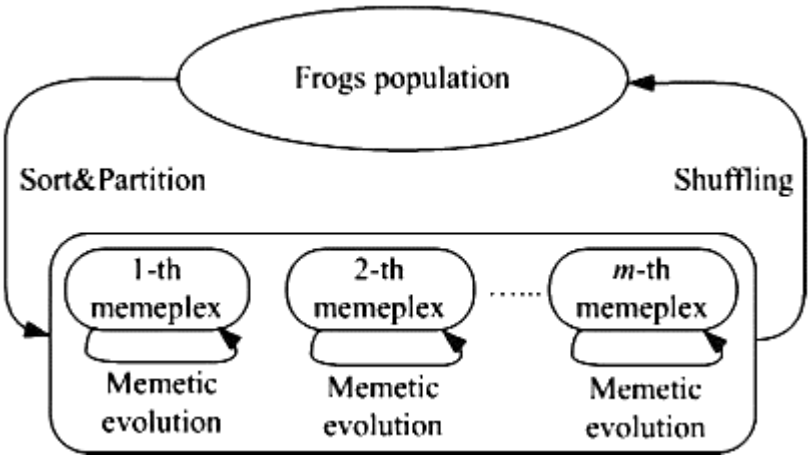


Figure 3.3. Framework of SFLA algorithm

To adapt MOSFLA to interactive overtime planning, the algorithm is updated as follows.

i. Archiving Strategy

In many efficient MOO algorithms, the archiving strategy plays a key role in maintaining multiple non-dominated solutions, while the niche technique effectively preserves the diversity among these solutions. In a niche-based archiving approach, the niche radius is utilized to compute the sharing fitness of non-dominated solutions. The sharing fitness is calculated as defined in equations 3.6 and 3.7.

$$SF(i) = \frac{1}{\sum_{j=1}^{q} sh(d_{ij})} \tag{3.6}$$

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^{\alpha} & d_{ij} < \sigma_{share} \\ 0 & d_{ij} > \sigma_{share} \end{cases} \tag{3.7}$$

Where $SF(i)$ is the sharing fitness of *i* th non-dominated solution, q is the number of solutions in the archive set, $sh(d_{ij})$ is the sharing function of *i* th and *j* th non-dominated solutions, d_{ij} is Euclidean distance of objective space between *i* th and *j* th non-dominated solutions, αα is constant coefficient; and σ_{share} is the niche radius.

σ_{share} directly affects $SF(i)$ and an unfit σ_{share} can lead to uneven distribution of the non-dominated solutions. Because specifying σ_{share} beforehand is challenging and it depends on the size and distribution of the archive set, a self-adaptive method for calculating σ_{share} is adopted. This method dynamically calculates and adjusts σ_{share} during each iteration, as described in [194]. The formulas for this calculation are shown in equations 3.8 and 3.9.

$$\sigma_{share} = \begin{cases} C & if\ q < 2 \\ \sum_{i=1}^{q} {}^{d_i}/_{q} & if\ q \geq 2 \end{cases} \quad (3.8)$$

$$d_i = \min\left(\left\| F_i(x) - F_j(x) \right\| \right) for\ \ i, j = 1,2, \ldots\ldots, q : \ j \neq i, \quad (3.9)$$

q is the number of solutions in the archive set, d_i is the minimum Euclidean distance of objective space between *i* th non-dominated solutions and others, and *C* is a positive constant usually set to 1. The niche radius is calculated as the mean of all non-dominated solutions' d_i values in the archive..

ii. Population Creation and Sorting Strategy

MOSFLA is a stochastic algorithm where the initial frog population is randomly generated. The acceptance or rejection of solutions depends on maximum overtime constraints. To sort the population, various methods have been used, such as Pareto dominance, crowding distance of solutions, a combination of crowding distance within non-dominated solutions, and the Hamming distance between dominated and non-dominated solutions [80,189]. In this study, sorting is based on both the rank determined by Pareto fronts and the crowding distance of individual frogs. The ranking process is as follows:

1. Non-dominated solutions in the initial population are assigned to the first rank (rank 0) and removed.
2. From the remaining solutions, non-dominated solutions are identified and assigned to the next rank (rank 1).
3. This process continues until no solutions remain.

Following ranking, crowding distance (Cd) is calculated for all solutions within the same rank using equation 3.10.

$$Cd = \sum_{k=1}^{r} |P[i+1] \cdot fk - P[i-1] \cdot fk| \quad (3.10)$$

$P[i+1] \cdot fk$, $P[i-1] \cdot fk$ are the k[th] objective function values of two adjacent frogs, and *r* is the number of objectives. The final multi-objective fitness function is calculated using equation 3.11 as proposed in [195].

$$MOFit = \frac{1}{2^{rank} + \frac{1}{1 + C_d}} \quad (3.11)$$

iii. Memplex formation

The sorted frogs are stored in an array X = {*P(i), f(i), i = 1, . . . ,n},* in the order of their fitness values. The array is then partitioned into *m* memeplexes $Y^1, Y^2, \ldots, Y^m$, each containing *n* frogs based on equation 3.12:

$$Y^k = [\, P(i)^k \mid P(i)^k = P[\, k + m(i-1)\,]\,], \quad i = 1,..n \;\; k = 1,..m \tag{3.12}$$

iv. Memetic evolution within Memeplex

Within each memeplex, the frogs undergo local memetic evolution. To do this, local best frog in each memeplex (x_b) and global best frog (x_g) are used systematically to improve the worst frog (x_w) within each memeplex. The following steps are taken to achieve the local evolution.

Step 1: x_b and x_w are set as $Y^k[1]$ and $Y^k[n]$ of k^{th} memeplex, respectively. For the frogs to evolve toward Pareto optimal, x_g is randomly selected from the archive set.

Step 2: improve worst frog x_w with local best frog x_b using equation 3.13.

$$d = 2 * rand * (x_b - x_w), \quad newx_w = oldx_w + d \tag{3.13}$$

Recompute the objective function values and compare $newx_w$ and $oldx_w$ using the Pareto dominance relation. If $newx_w$ dominates $oldx_w$, then replace $Y^k[n]$ with $newx_w$ and go to Step 5. Otherwise, go to step 3.

Step 3: improve worst frog x_w with the global best frog x_g using equation 3.14

$$d = 2 * rand * (x_g - x_w), \quad newx_w = oldx_w + d \tag{3.14}$$

Recompute the objective function values and compare $newx_w$ and $oldx_w$ using the Pareto dominance relation. If $newx_w$ dominates $oldx_w$, then replace $Y^k[n]$ with $newx_w$ and go to Step 5. Otherwise proceed to step 4.

Step 4: a new frog is generated to replace x_w by randomly using two methods:

1. Randomly generate a new frog in the neighborhood of x_g
2. Locally mutate x_g to generate a new frog. This is done by randomly selecting two segments of the same length in x_g and then copying the front segment to the last segment or vice versa

Step 5: Update the memeplex after the improvement on x_w and sort Y^k in decreasing order of fitness value.

Step 6: Repeat Steps 1 to 5 for several iterations in each memeplex.

v. Shuffling and preference integration

After several memetic evolution iterations, the Memeplexes are shuffled for global optimization, and the generated frogs are sorted in descending order of their MOfit value. To adapt the algorithm to the proposed interactive optimization framework, the PM's preferences are evaluated on the Non-dominated frogs produced at each global optimization iteration step using a pretrained RFR model. The RFR model estimates PM's satisfaction with the overtime plans presented by the non-dominated

solutions; only solutions with an estimated score greater than a specified threshold (85 is proposed in this study) are added to the archive using the defined archiving strategy. The memetic evolution and global optimization steps continue for a specified number of iterations to ensure convergence.

vi. Ranking of final solutions

The interactive MOSFLA generates several preferred solutions (although they may be very limited in number) for PM. The algorithm is designed to prioritize overtime planning solutions using a ranking approach. The ranking is achieved using the score predicted by the RFR model for each solution. With this approach, the most probable preferred solution, based on the ML model prediction, is recommended for the PM.

3.3.1 Random Forest Regression

Random Forest Regression (RFR) is a supervised learning algorithm that employs an ensemble learning method to construct a regression model [196,197]. The ensemble learning method combines predictions from multiple models to produce a more accurate prediction than an individual model [198–200]. RFR works by building several Decision Trees (DTs) during training and produces the output as the average prediction of all the DTs, as depicted in Figure 3.4. It combines the concepts of random subspaces and bagging to build a decision forest algorithm trained on multiple decision trees, each driven by slightly different subsets of data. A simple procedure, as described in Algorithm 1, is used by RFR to build a stable ensemble model.

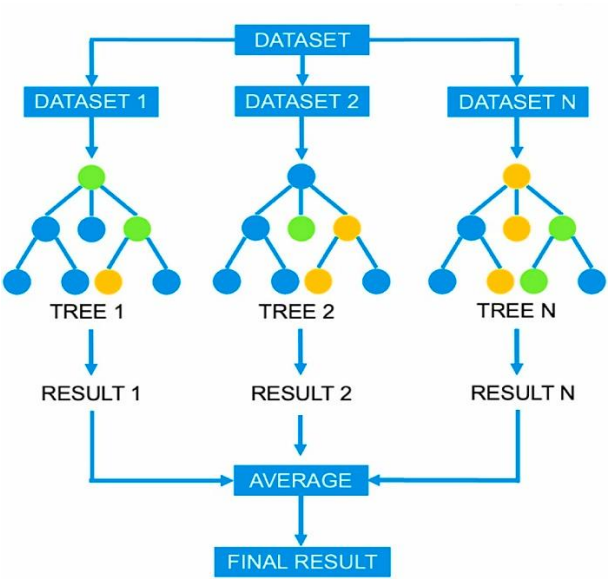


Figure 3.4. Structure of Random Forest Regression [201]

RFR is chosen for the study due to its ability to effectively handle non-linear relationships in data and its robustness against overfitting, achieved through averaging across multiple trees. Moreover, RFR has been successfully applied to solve estimation problems in software engineering tasks such as effort estimation [136,138,141]. In this study, RFR is employed to develop an estimation model for

learning the subject preferences of PMs, which is trained on evaluation scores provided by project management experts for overtime plans generated by a multi-objective optimization model.

Algorithm 1: Random Forest Regression

Input: Training data D, Number of Trees $N - trees$

1: Initialize the number of trees in the forest, denoted as N-trees
2: For each tree in N-trees:
3: Randomly sample a subset of the data (with replacement) to create a bootstrap sample
4: Build a decision tree using the bootstrap sample
5: Repeat the process until the tree is fully grown (or reaches a specified depth)
6: End for
7: For a new datapoint:
8: Estimate the output value from all the N-trees
9: Return the mean of the outputs across all N-trees as the final output

Output: Average estimation

3.4 Software Project Data Used

Six real-world software project data collected by [22] were obtained from https://github.com/luizaraujojr/GECCO2016 are used to evaluate the effectiveness and efficiency of the proposed ML-iMOSLA algorithm for optimal overtime planning. The software projects are described below.

1. ACAD: manages university academic administration through a portal system, encompassing registration, student management, classes, and teachers' records.
2. WEBMET: stores meteorological information in a database.
3. PSOA: manages users' authorization and authentication for enterprise systems
4. WEBAMHS: controls air traffic routing system for airlines.
5. PARM: is a configuration management application. It manages configuration settings for multiple systems, enabling the rapid configuration of user profiles and sharing configurations across different software.
6. OPMET: handles the storage, management, and accurate delivery of meteorological information

Each project's data (specified in XML) defines the project's activities, their transaction functions, the data functions associated with the activities, and dependencies between activities. Additionally, related activities in the projects are organized into groups of Work Packages (WP) that form the schedule. Table 3.1 presents the quantitative description of the software projects.

Table 3.1. Description of Software Projects used

Project	ACT	TF	DF	DEP	FP
ACAD	40	39	7	39	185
WEBMET	44	48	7	33	225
PSOA	72	65	9	84	290
WEBAMHS	60	67	8	45	381
PARM	108	98	21	91	451
OPMET	84	129	22	63	635

ACT is the number of activities in a project, TF is the number of transaction functions, DF is the number of data functions, DEP is the number of activity dependencies, and FP is the number of function points denoting the size of the project.

3.5 Performance Evaluation Metrics

The performance evaluation metrics used in this study are categorized into three groups based on the components of the proposed framework: regression-based metrics for the ML model, quality indicator metrics for the multi-objective optimization algorithm, and interactive algorithm metrics for evaluating the effectiveness of the proposed interaction method. The performance of the ML model is evaluated using widely used regression-based performance metrics, as defined in Equations 3.15 to 3.21.

Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of predictions, without considering their direction, ensuring equal contributions from all errors. Its value is the average of the absolute errors between the actual and the predicted PM's scores.

$$\mathrm{MAE} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}} |\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i}| \tag{3.15}$$

Where:
$\boldsymbol{n}$ = number of observations
$\boldsymbol{y_i}$ = the actual PM`s score
$\boldsymbol{\hat{y}_i}$ = the predicted PM`s score

Mean Squared Error (MSE) measures the average of the squares of the errors—the average squared difference between the actual and predicted PM's score. Unlike MAE, it heavily penalizes large errors and is highly sensitive to outliers, making it useful to find out whether the models minimize large errors.

$$\mathrm{MSE} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2 \tag{3.16}$$

Root Mean Squared Error (RMSE) [118] is the standard deviation of the residuals. Residuals are the distances between data points and the regression line, representing the prediction error. It measures the dispersion of these residuals. Like MSE, RMSE is also based on a squared error to remove the bias towards the large errors. However, it is more helpful in understanding the average magnitude of prediction errors in the same units as the original data by computing the root of the square errors, giving a more intuitive grasp of the model's performance.

$$RMSE = \sqrt{\frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} (\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^{\mathbf{2}}} \tag{3.17}$$

Mean Squared Logarithm Error (MSLE) [202] computes the average squared logarithmic difference between the predicted and actual PM scores. It gives more attention to the relative difference between the predicted and actual scores than the absolute difference. Furthermore, unlike MSE and RMSE, it is generally more stable with large error values since the log function reduces the impact of large

differences.

$$\mathrm{MSLE} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\log(1 + \mathrm{y_i}) - \log(1 + \hat{\mathrm{y}}_\mathrm{i})\right)^2 \quad (3.18)$$

Mean Magnitude of Relative Error (MMRE) [203] measures the average relative error between the predicted and actual PM's score. Like MSLE, it provides a relative error measure, making it suitable for comparing models across different datasets with varying scales. It is widely used to evaluate the accuracy of estimation models in software engineering tasks such as effort estimation [125,158,163,165]. A lower MMRE value means a better estimation performance, indicating that the predictions more accurately reflect the actual scores on average.

$$\mathrm{MMRE} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\left|\frac{\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i}}{\mathrm{y_i}}\right| \quad (3.19)$$

R^2 measures the percentage of variance in the PM's scores that can be explained collectively by the predictors (overtime allocations). It measures the strength of the relationship between the regression model and the dependent variable. R^2 is scale-independent because it is a relative measure of fit. It indicates how well the model explains the variability of the response data regardless of the scale of the data. Its computation is based on the sum of the squares of residuals and the total sum of squares, and its value ranges from 0 to 1.

$$\mathrm{R}^2 = 1 - \left(\frac{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2}{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{y_i} - \bar{\mathrm{y}})^2}\right) \quad (3.20)$$

$\bar{\boldsymbol{y}}$ = the mean of the actual PM`s scores

Adjusted-R^2 [204] modifies the R^2 value based on the number of predictors present in the model. It gives a more accurate estimation of the goodness of fit, especially when comparing models across datasets with diverse features. To tackle the limitation of R^2 in terms of sensitivity to the dataset, adjusted-R^2 takes into account the number of predictors in the model. It is computed as:

$$\mathrm{R}^2_{Adjusted} = 1 - \left(\frac{(1 - \mathrm{R}^2)(n - 1)}{n - p - 1}\right) \quad (3.21)$$

$\boldsymbol{p}$ is the number of predictors. It penalizes the addition of irrelevant predictors by decreasing the R^2 value if the new predictors do not lead to any significant performance improvement. Additionally, it measures how well the model generalizes to new, unseen data while adjusting for the number of predictors.

Concerning the multi-objective optimization performance evaluation, Contribution (I_C), Inverted Generational Distance (I_{iGD}), and Hyper-Volume (I_{HV}) are utilized. They have been used in previous studies on SOP [30,36–38]. They are defined in equations 3.22 to 3.25

I_C: This is a convergence measure that computes the proportion of solutions from a reference front RF produced by an algorithm. It is the ratio of the solutions in RF produced by algorithm A [26].

$$I_C = \frac{\frac{|C|}{2} + |W_A| + |N_A|}{|RF|} \quad (3.22)$$

C is the Pareto set common to both A and RF, W_A is the solution set in A, which dominates other solutions in RF. N_A is the set of solutions in *A* having no dominant relation with solutions in *RF.* A good Pareto front should have a high I_C value and contribute significantly to the reference front.

I_{iGD}: measures the average distance from a set of true Pareto-optimal solutions (reference front (RF)) to the obtained solution set *S* from an algorithm [30]. It evaluates both the convergence and diversity of the solutions as it estimates how well the obtained solutions represent the entire Pareto front. The distance from *RF* to *S* in an N-objective space is computed as the average N-dimensional Euclidean distance between each point in *RS* and its closest point in S.

$$I_{iGD} = \frac{1}{|RF|} \sum_{y \in RF} \min_{x \in S} (d_{yx}) \quad (3.23)$$

d_{yx} is the distance between a reference solution *y* in *RF* and a solution *x* in *S,* in the *N*-dimensional objective space as defined in equation 3.24:

$$d_{yx} = \sqrt{(f_1(y) - f_1(x))^2 + \cdots + (f_N(y) - f_N(x))^2} \quad (3.24)$$

Where f_i (*x*) is the *i*th objective function value of a solution *x.* This metric indicates how well the obtained front covers the true Pareto front. A good algorithm should produce as many solutions closer to the reference front as possible and thus should have low I_{iGD}.

I_{HV}**:** This measure calculates the volume of the objective space dominated by the generated Pareto front from an algorithm with regard to a reference front [39]. It replaces the known bias of I_C towards the number of non-dominated solutions produced with the quality of non-dominated solutions generated by an algorithm making it a Pareto strict metric. I_{HV} reflects both the convergence and the diversity of a solution set.

$$I_{HV} = Volume\left(\bigcup_{s=1}^{S} \{x \in \mathbb{R}^m \mid r_i \leq x_i \leq s_i \ \forall\, i = 1, \ldots, m\}\right) \quad (3.25)$$

where *S* is the set of non-dominated solutions from the front to be evaluated, $r = (r_1, r_2, \ldots, r_m)$ is the reference point, m is the number of objectives and set$\{x\}$ is the hypercube area dominated by solution *s* with respect to a reference point *r* dominated by all solutions in RF. Thus, the function computes the occupied volume in the search space by the union of all the hypercubes. Thus, the higher the value of this metric, the closer the *S* is to the RF, and the better the algorithm under consideration.

Furthermore, the standard interactive metrics, Similarity Degree (SD), Similarity Factor (*SF*) and Price of Preference (*PP*) have been previously proposed in [48] to measure the performance of interactive optimization algorithms for a single objective optimization NRP problem. To adapt the metrics to multi-objective overtime planning optimization, the metrics are updated as defined below.

$Multi-objective\ Similarity\ Degree\ (MoSD)$: This metric measures the average percentage of similarity between a set of non-dominated solutions found by the proposed algorithm and a target solution in terms of the overtime plan produced. It directly measures the subjective satisfaction of the produced solutions because the more similar a candidate solution is to the target solution, the higher the subjective satisfaction. For a set of candidate solution S, each composed of overtime allocations $(x_1, x_2, \dots, x_N)$ where N corresponds to number of WPs in the problem, and a target solution P represented as $(p_1, p_2, \dots, p_N)$, the MoSD is computed as:

$$MoSD = \frac{\sum_{s=1}^{S}\left[\left(\frac{\sum_{i\,|\,1\le i\le N\ \wedge\ x_i = p_i}^{N} 1}{N}\right) \times 100\%\right]}{|S|} \tag{3.26}$$

$Multi-objective\ Similarity\ Factor\ (MoSF)$: this metric measures the average proportional gain in $MoSD$ achieved by an interactive algorithm compared to the one without interactive approach. It calculates how well the interactive algorithm satisfies the subjective preferences of the DM when compared to a non-interactive algorithm. A good interactive algorithm should have high ASF value. Given X as the set of solutions produced by an algorithm without interaction and Y as the solutions produced by an interactive algorithm, the $MoSF$ is calculated as:

$$MoSF = \left(\frac{MoSD\ (Y)}{MoSD\ (X)} - 1\right) \times 100\% \tag{3.27}$$

$Multi-objective\ \ Price\ of\ Preference\ (MoPP)$: This metric measures the percentage of loss in explicit objective values while incorporating implicit preferences from the DM through subjective evaluation. It estimates the price paid by the interactive algorithm in terms of explicit objective values to satisfy the DM's subjective preferences A good solution should have a low APP value. Given X and Yas defined in equation 3.27, the $MoPP$ is computed using the equation:

$$MoPP = \left(1 - \frac{\sum_{j=1}^{M}\left(\frac{\sum_{i}^{N_Y} Score_j(Y_i)}{N_Y}\right)}{\sum_{j=1}^{M}\left(\frac{\sum_{i}^{N_X} Score_j(X_i)}{N_X}\right)}\right) \times 100\% \tag{3.28}$$

$Score_j(Y_i)$ is the normalized j[th] objective value of the i[th] solution in Y, $Score_j(X_i)$ is the normalized j[th] objective value of the i[th] solution in X, N_Y and N_X are the number of solutions in Y and X respectively and M is the number of objectives.

Chapter 4 Machine Learning-Based Software Overtime Estimation

Existing studies in SOP have produced quality solutions for PMs using search-based metaheuristic MOO algorithms to plan software overtime allocations. However, the explicit solutions generated without the input knowledge of PMs may affect their industry acceptance and application by the PMs. This chapter addresses this problem by building and evaluating a machine learning-based overtime estimation model that learns from real-world PM annotations of publicly available software project datasets. To the best of the author's knowledge, this is the first time a machine-learning approach has been applied to SOP. The specific contributions of this chapter are as follows.

1. Production of software overtime estimation datasets from annotations collected from industry-based project management experts.
2. Evaluation of several heterogeneous regression models on the collected dataset for estimating PM's preference for overtime allocations
3. Evaluation of a greedy halving grid search-optimized Random Forest Regression (gHGS-RFR) as a scalable model for estimating the PM's preference for software overtime allocations.
4. Comparative analysis of the developed gHGS-RFR model against other hyperparameter optimization models with RFR.

4.1 Experimental Framework

The methodological approach for developing a machine learning model for software project overtime estimation is illustrated by the experimental framework shown in Figure 4.1. Since no dataset is publicly available for this specific application, the framework outlines how a labeled dataset was created using publicly accessible software project previously employed in SOP studies [22]. This process involves extracting data from these projects, generating multi-objective overtime planning solutions, collecting PM's annotations, and refining the dataset. Additionally, the framework features a pipeline for constructing and evaluating a greedy Halving Grid Search-optimized Random Forest Regression (gHGS-RFR) model with the dataset, enabling it to learn overtime estimation from expert annotations. A thorough comparative analysis with other regression models and hyperparameter optimization methods is also included to demonstrate the model's effectiveness and validate its suitability for this domain.

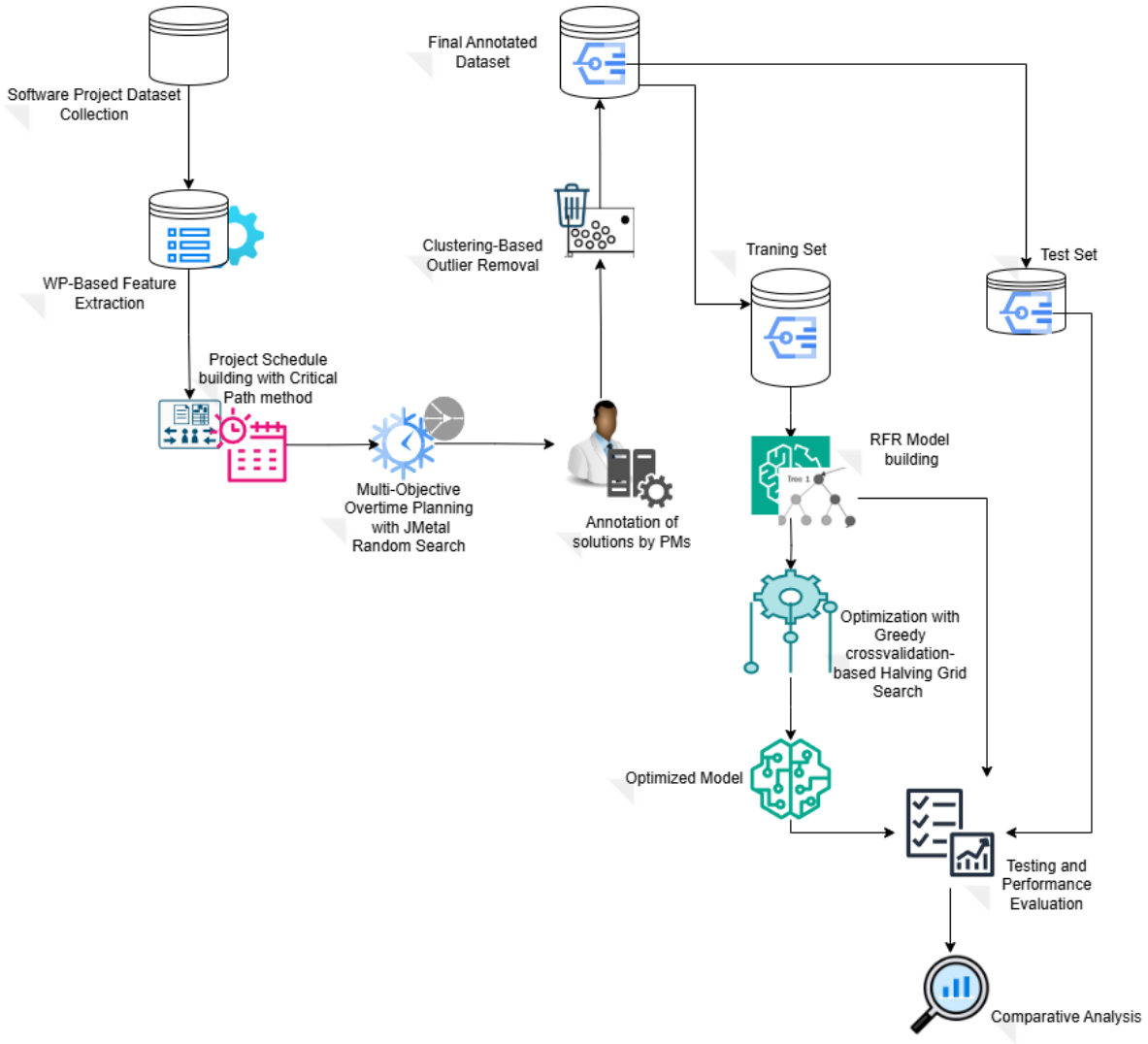


Figure 4.1. Experimental Framework for Machine Learning-Based Overtime Estimation in Software Projects

4.2 Software Overtime Estimation Dataset

Building a machine learning model for software development overtime estimation requires project managers to annotate real-world software project overtime data. This section discusses the method used for constructing the dataset from the original software projects described in Section 3.4.

4.2.1 Data Extraction from Projects

This first step toward building a dataset for training an overtime estimation model from the obtained software project is data extraction. In this step, WP-based features in the software project were extracted, as opposed to the features of individual activities, as in the original dataset. This measure is implemented to prevent the allocation of daily overtime to tasks that can be completed within a single day or less, as observed in the activity-based features. Specifically, external dependencies—where an activity depends on another activity in a different Work Package (WP)—were identified and extracted. The FPs of all activities within a WP were then aggregated to determine the overall FP of the WP. Internal dependencies among activities within a WP are not considered WP dependencies, as they

do not influence the complexity of schedule construction. Given the FP size of a WP, its expected duration can be estimated using a mean productivity value of 27.8 FP/developer-month suggested by [190] for IT projects and previously used in existing studies on SOP studies [22]. The duration in days for each WP is computed as described in Equation 4.1.

$$D_{duration(wp_i)} = 30\left(\frac{FP(wp_i)}{27.8}\right) \quad (4.1)$$

The CP method, which considers the longest precedence-respected path in the project schedule, was used to estimate the project's shortest possible duration (SPD). CP has been widely applied in previous research on overtime planning for software projects [21,22,29,30]. The information derived from WPs in the projects is summarized in Table 4.1. Dependencies, function points, and durations for each WP across all considered projects are also documented to assist annotators and feature extraction in model development. For example, Table 4.2 details WP-level properties for the software project ACAD (with all WP-based features for the projects listed in Appendix A). These features were provided to PMs along with the generated overtime plans for annotation.

Table 4.1. Extracted Properties of the pre-processed Software Projects Dataset

Project	WPs	WP-DEP	FP	SPD (days)
ACAD	10	9	185	181
WEBMET	11	17	225	243
PSOA	18	34	290	316
WEBAMHS	15	20	381	413
PARM	27	16	451	654
OPMET	21	27	635	473

Table 4.2. Detailed WP-level properties of the ACAD project

WP	No. Dependencies	Dependency list	FP	Duration (days)
Aluno	0	-	25	27
Trancamento	1	Aluno	9	10
Bolsa	1	Aluno	12	13
Professor	1	Area	22	24
Area	0	-	19	21
Linha	1	Area	12	13
Usuario	0	-	19	21
Disciplina	0	-	19	21
Turma	2	Professor, Disciplina	22	24
Inscricao	3	Aluno, Disciplina, Turma	26	28

4.2.2 Multi-Objective Overtime Allocation

For each project, multiple solutions for overtime planning that satisfy the objectives defined in Section 3.2 were generated utilizing the multi-objective random search within the JMetal MOEA framework library. This random search was employed to thoroughly explore the search space and prevent overreliance on a specific region, as the aim is to produce as many feasible solutions as possible to facilitate the training of a machine learning model. Specifically, a total of 10,000 solution instances, consisting of approximately equal numbers of randomly generated solutions and three well-

established overtime management strategies—CPM, MAR, and SH—were evaluated for each software project through the multi-objective random search, with the top 300 instances being selected. Samples of 20 solution instances, comprising overtime allocation plans along with their explicit objective values and the employed overtime planning strategies, are provided in Appendix B. The generated solutions are formatted in Excel CSV files and are accompanied by the extracted properties of the software projects for subjective assessment by software project managers.

4.2.3 Annotation of Solutions by PMs

Given that the ML approach required a labeled dataset for model training, the generated solutions were formatted in CSV and presented to PMs alongside the extracted features of the software projects for annotation. The input features (feature columns) of the dataset, associated with project tasks (Work Packages, WPs), were named using the convention *TaskName_Dependency_FP_Duration_[Critical Path]* to reveal the underlying properties of the tasks to the annotators. Subsequently, twenty experienced software project managers with expertise in overtime allocation were invited to evaluate the proposed overtime plans. The PMs provided numerical evaluation scores ranging from 1 to 100 for each solution instance, as adapted from [40], to reflect their subjective preferences regarding the overtime plans within the solutions. Data concerning the PMs' years of professional experience, their proficiency in software development (rated as high, moderate, or low), and their familiarity with overtime planning were collected. Table 4.3 presents the statistical details of the collected data. The PMs' experience spans from five to twelve years, with an average of 7.4 years. The majority of the PMs possess high expertise in software development and moderate experience in overtime planning.

Table 4.3. Statistics of data collected about project managers

Data	Values	Result
Years of Experience	Total	155
	Average	7.4
	Minimum	5
Software Development expertise	Low	4
	Moderate	7
	High	9
Overtime planning experience	Low	6
	Moderate	8
	High	6

4.2.4 Clustering-Based Outlier Removal

Owing to the unavoidable fatigue experienced by PMs when annotating solutions, as well as the apparent subjectivity inherent in their assessments, some inconsistencies in evaluation are expected, which may result in the presence of outliers in the collected annotations. To rectify these inconsistencies, the annotation scores obtained were further processed to eliminate outliers employing a clustering-based outlier removal method, as outlined in Algorithm 2. The algorithm calculates the mean and standard deviation (SD) of all annotation scores received from the PMs. If the deviation of scores from the mean exceeds a predetermined maximum deviation threshold (s), a K-Means clustering procedure is initiated to partition the distribution into k groups. The cluster with the smallest number of

data points is most likely composed of outliers [205], and it is removed from the set. The remaining clusters are merged, and new mean and SD are computed for the remaining set. This process is repeated until the SD falls below the specified threshold or the number of remaining data points in the set (annotations) is less than $\left\lceil 3\left(\frac{|N_0|}{2k}\right)\right\rceil$ (in this case, the solution instance is removed from the dataset due to significant disagreement in its evaluation by a considerable number of PMs). The algorithm outputs the annotation scores without outliers and their mean value.

Algorithm 2: Clustering-based outlier removal algorithm

Input: Annotation scores (N_0) received for an overtime plan, Number of clusters (*k*), maximum spread (s)

1. Initialize $N \leftarrow N_0$
2. Compute the $mean$ and SD of N
3. If $SD > s$:
4. Cluster N into k groups $(X_1, \dots X_k)$ with K-means
5. Remove the cluster with the minimum data points: $min\ (X_1, \dots X_k)$
6. Set N to the union of the remaining clusters: N $\leftarrow \cup(X_1, \dots X_k) - min\ (X_1, \dots X_k)$
7. If $|N| > \left\lceil 3\left(\frac{|N_0|}{2k}\right)\right\rceil$:
8. Go to step 2
9. Else:
10. Return 0
11. Return N and $mean$

Output: Annotation scores without outliers and the mean

The effectiveness of the algorithm in detecting outliers depends on the parameters *k* and *s*. In this study, these parameters were determined experimentally by testing various values on the ACAD project dataset, focusing on the average non-outlier scores removed by the algorithm. Values of 2, 3, and 4 were tested for *k*, while 2, 5, and 8 were tested for *s*. As shown in Table 4.4, the optimal settings were *s* = 5 and *k* = 3, which minimized the removal of non-outlier annotation scores. These parameter values were then applied to the remaining projects to identify outliers. A sample of the final PM evaluation scores for selected instances from the ACAD project, after applying the clustering-based outlier removal, is provided in Table 4.5.

Table 4.4. Average non-outlier scores removed by different parameters values of clustering-based outlier removal algorithm in ACAD Project

Maximum deviation (s)	Number of cluster (*k*)		
	2	**3**	**4**
2	6.7	4.3	4.8
5	4.6	1.5	3.7
8	3.5	2.2	2.0

The algorithm stops when roughly half of the PM evaluation scores for a solution are removed, indicating divergent opinions among the PMs on that instance, leading to their deletion from the dataset. In total, 1622 cleaned annotated solutions were generated across six software projects, forming the

final datasets for model training. Table 4.6 shows the number of instances in each project dataset and their distribution. The completed cleaned dataset is publicly available in [206] validation.

Table 4.5. Sample of Annotated Overtime Allocations for ACAD Project

Task1_0_25_27_C	Task2_1_9_10_C	Task3_1_12_13_C	Task4_1_22_24_C	Task5_0_19_21_C	Task6_1_12_13_C	Task7_0_19_21_C	Task8_0_19_21	Task9_2_22_24_C	Task10_3_26_28_C	PM's score
0	0	0	1	1	2	2	2	2	2	88.50
1	2	2	1	1	1	1	2	1	1	78.09
0	1	2	0	0	4	0	0	0	4	55.83
1	1	2	1	1	1	1	2	1	1	72.08
0	0	0	0	1	2	2	3	2	2	61.87
1	1	1	2	1	1	1	1	2	1	58.71
2	2	2	3	2	0	0	1	0	0	60.83
2	0	0	0	2	1	2	0	4	0	65.79

Table 4.6. Description of the final datasets from the software projects

Project	Instances (solutions)	Features (WPs)	Proportion of overtime management strategies			
			MAR	CPM	SH	Random
ACAD	282	10	68	71	73	70
WEBMET	269	11	63	68	71	67
PSOA	266	18	65	63	70	68
WEBAMHS	275	15	69	70	72	64
PARM	271	27	63	68	65	75
OPMET	259	21	64	59	67	69

4.3 Comparison of ML-Based Regression Models' Performance for Software Overtime Estimation

As this is the first time an ML approach is used in software overtime planning, various ML-based regression models selected from different categories were evaluated to determine the best fit for the proposed interactive optimization framework for SOP. The models applied and compared were: MLR, Ridge regression (Ridge), Least Absolute Shrinkage and Selection Operator (LASO) regressor, SVR, KNN regressor, MLP regressor, and GB. The models were implemented and trained on the annotated overtime plan solutions dataset of all the software projects. The dataset for each project was split into an 80% training set and a 20% test set. Validation on training sets was based on the K-fold cross-validation approach. 10-fold cross-validation has been proven experimentally to produce minimal and unbiased test error rates with low variance [207,208]. Hence, the choice of using 10-fold cross-validation for the comparative evaluation is to avoid overfitting. The results of these comparisons across the project dataset are presented in Tables 4.7 – 4.12. The results of the ML models in estimating PMs' satisfaction with overtime allocations, as evaluated across the six projects, highlight RFR's robustness while contextualizing its superior strengths relative to linear, kernel-based, and boosting ensemble methods.

Table 4.7. Results of ML-based regression models on ACAD project

	MAE↓	MSE↓	RMSE↓	MSLE↓	MMRE↓	R^2 ↑
MLR	9.54	226.20	15.04	0.34	0.269	0.65
Ridge	9.24	242.11	15.56	0.28	0.264	0.68
LASO	9.01	223.50	14.95	0.43	0.259	0.71
SVR	8.94	218.45	14.78	0.25	0.258	0.75
KNN	8.24	169.00	13.00	0.23	0.256	0.71
RFR	8.91	156.00	12.49	0.14	0.240	0.84
GB	8.22	162.56	12.75	0.15	0.245	0.88
MLP	8.65	190.16	13.79	0.28	0.253	0.81

Table 4.8. Results of ML-based regression models on WEBMET project

	MAE↓	MSE↓	RMSE↓	MSLE↓	MMRE↓	R^2 ↑
MLR	8.26	203.06	14.25	0.31	0.255	0.69
Ridge	9.56	199.94	14.14	0.24	0.253	0.62
LASO	9.34	209.96	14.49	0.38	0.250	0.73
SVR	8.14	173.71	13.18	0.20	0.248	0.79
KNN	9.43	157.75	12.56	0.18	0.245	0.74
RFR	7.85	133.63	11.56	0.11	0.229	0.89
GB	7.98	126.56	11.25	0.13	0.235	0.82
MLP	8.01	145.93	12.08	0.15	0.243	0.85

Table 4.9. Results of ML-based regression models on PSOA project

	MAE↓	MSE↓	RMSE↓	MSLE↓	MMRE↓	R^2 ↑
MLR	10.45	255.36	15.98	0.36	0.326	0.63
Ridge	10.32	245.55	15.67	0.39	0.319	0.67
LASO	10.24	233.78	15.29	0.45	0.308	0.69
SVR	9.98	219.34	14.81	0.30	0.278	0.72
KNN	10.08	195.44	13.98	0.32	0.281	0.70
RFR	9.64	165.38	12.86	0.18	0.251	0.82
GB	9.82	176.36	13.28	0.22	0.268	0.78
MLP	9.95	210.83	14.52	0.37	0.272	0.74

Table.4.10. Results of ML-based regression models on WEBAMHS project

	MAE↓	MSE↓	RMSE↓	MSLE↓	MMRE↓	R^2 ↑
MLR	11.87	263.74	16.24	0.42	0.320	0.63
Ridge	11.46	261.15	16.16	0.41	0.278	0.67
LASO	11.25	250.91	15.84	0.40	0.275	0.71
SVR	11.06	230.74	15.19	0.28	0.267	0.70
KNN	10.95	219.63	14.82	0.37	0.271	0.68
RFR	10.26	172.13	13.12	0.15	0.245	0.78
GB	10.46	224.10	14.97	0.31	0.258	0.75
MLP	10.73	236.54	15.38	0.38	0.263	0.76

Table 4.11. Results of ML-based regression models on PARM project

	MAE↓	MSE↓	RMSE↓	MSLE↓	MMRE↓	R^2 ↑
MLR	14.43	305.90	17.49	0.48	0.365	0.60
Ridge	12.78	267.32	16.35	0.45	0.339	0.69
LASO	13.25	281.57	16.78	0.38	0.342	0.63
SVR	10.85	248.38	15.76	0.31	0.273	0.72
KNN	12.17	261.15	16.16	0.45	0.321	0.67
RFR	10.65	211.99	14.56	0.20	0.256	0.79
GB	10.58	235.93	15.36	0.25	0.267	0.73
MLP	11.68	211.12	14.53	0.23	0.259	0.76

Table. 4.12. Results of ML-based regression models on OPMET project

	MAE↓	MSE↓	RMSE↓	MSLE↓	MMRE↓	R^2 ↑
MLR	14.81	342.99	18.52	0.49	0.365	0.61
Ridge	13.07	306.60	17.51	0.47	0.392	0.63
LASO	13.45	289.34	17.01	0.42	0.354	0.64
SVR	11.75	277.89	16.67	0.39	0.301	0.69
KNN	12.62	305.90	17.49	0.59	0.356	0.67
RFR	10.72	211.70	14.55	0.21	0.259	0.77
GB	10.85	239.94	15.49	0.32	0.255	0.72
MLP	12.13	213.74	14.62	0.23	0.257	0.79

Linear models (MLR, Ridge, and LASO) consistently underperformed across all projects, reflecting their inability to capture non-linear relationships inherent in PM satisfaction dynamics. For instance, In ACAD, MLR achieved an R^2 of 0.65, significantly lower than RFR's 0.84, while its MAE (9.54) and MSE (226.20) were considerably higher than RFR's (8.91, 156.00). Similarly, in OPMET, Ridge regression yielded an RMSE of 17.51 and an R^2 of 0.69, compared to RFR's 14.55 and 0.77, respectively. The poor performance of linear models highlights the limitations of assuming linear feature interactions in overtime satisfaction, which likely involve threshold effects (for example, dissatisfaction beyond a critical number of overtime hours). Kernel-Based Models: SVR and KNN exhibited project-specific variability, excelling in low-dimensional projects but underperforming in others. SVR outperformed RFR in the ACAD project (MAE: 8.94 vs. 8.91; R^2: 0.75 vs. 0.84), but lagged behind in the WEBAMHS project (RMSE: 15.19 vs. 13.12; R^2: 0.70 vs. 0.78). KNN, on the other hand, achieved competitive MAE in the ACAD project (8.24) but suffered high MSE in the PARM project (261.15 vs. RFR's 211.99), likely due to sensitivity to local noise and scaling issues. RFR's ensemble approach provided more consistent accuracy, as its bagging mechanism mitigated overfitting and stabilized predictions across diverse project conditions.

Neural networks-based MLP demonstrated strong performance in certain projects (for instance in WEBMET: R^2 = 0.85 and OPMET: R^2 = 0.79), but it often required careful tuning and suffered from computational complexity. While MLP marginally outperformed RFR in WEBMET (RMSE: 12.08 vs. 11.56), RFR maintained superiority in PSOA (R^2: 0.82 vs. 0.74) and PARM(MMRE: 0.256 vs. 0.259). The "black-box" nature of MLP further limits its interpretability compared to RFR's feature importance

outputs, which are critical for actionable insights in allocating suitable overtime slots for the WPs in the projects. Considering another ensemble method, GB emerged as RFR's closest competitor, often achieving marginally lower MAE and MSE; in ACAD, GB outperformed RFR in MAE (8.22 vs. 8.91) and MSE (162.56 vs. 156.00), though RFR retained a higher R^2 (0.84 vs. 0.88). Conversely, in WEBMET, RFR surpassed GB in R^2 (0.89 vs. 0.82) and MSLE (0.11 vs. 0.13). In most projects and metrics, RFR marginally outperformed GB, highlighting its advantage in explanatory power and relative error minimization. GB's sequential error-correction strategy may enhance point prediction accuracy, but RFR's parallel training makes it more stable for multiple project environments.

To compare the cross-project performance of the ML models and estimate how the models generalize across various project complexities, the Adjusted-R^2 metric is adopted as it removes the bias towards dataset contextual characteristics. Table 4.13 presents cross-project Adjusted-R^2 results for the ML models, and Figure 4.2 compares their average.

Table 4.13. Adjusted-R^2 results for cross-project performance analysis of ML-models

Model	ACAD	WEBMET	PSOA	WEBAMHS	PARM	OPMET
MLR	0.57	0.61	0.43	0.45	0.19	0.34
Ridge	0.61	0.52	0.50	0.54	0.36	0.38
LASO	0.65	0.66	0.53	0.60	0.24	0.39
SVR	0.70	0.74	0.57	0.58	0.43	0.47
KNN	0.65	0.67	0.54	0.56	0.33	0.44
RFR	0.81	0.86	0.69	0.70	0.57	0.61
GB	0.85	0.77	0.66	0.65	0.45	0.53
MLP	0.77	0.81	0.60	0.61	0.51	0.64

The comparative evaluation of the ML models across the six projects reveals distinct patterns in their average performance, as measured by Adjusted-R^2. RFR emerges as the most robust model, achieving the highest mean Adjusted-R^2 (0.71), followed by MLP(0.66) and GB (0.65). These ensemble and neural network methods consistently outperform linear and kernel-based models, underscoring their capacity to capture non-linear relationships inherent in project managers' satisfaction dynamics. In contrast, linear models (MLR, Ridge, LASSO) exhibit the lowest average performance (MLR: 0.43, Ridge: 0.49, LASSO: 0.51), reflecting their inadequacy in modeling complex interactions. Kernel-based models (SVR: 0.58, KNN: 0.53) occupy an intermediate position, demonstrating moderate adaptability but lagging in high-variance projects like PARM and OPMET.

The superiority of RFR can be attributed to its ensemble architecture, which balances bias-variance trade-offs through bootstrap aggregation and feature randomization. While GB and MLP show competitive results, their performance gaps in specific projects—such as GB's lead in ACAD (Adjusted-R^2: 0.85) and MLP's edge in OPMET (0.64)—highlight project-dependent strengths. GB's sequential error correction performs well in datasets with gradual trends (such as the ACAD project), whereas MLP's deep learning capability captures intricate hierarchies in complex data (for example, the OPMET project). However, both models exhibit higher computational costs compared to RFR. Linear models, despite regularization (Ridge, LASSO), remain constrained by their inherent linearity, struggling to

explain variance in projects. So, RFR's dominance in four of the six projects positions it as the best choice for estimating PM's satisfaction with overtime plans, prioritizing generalizability and interpretability.

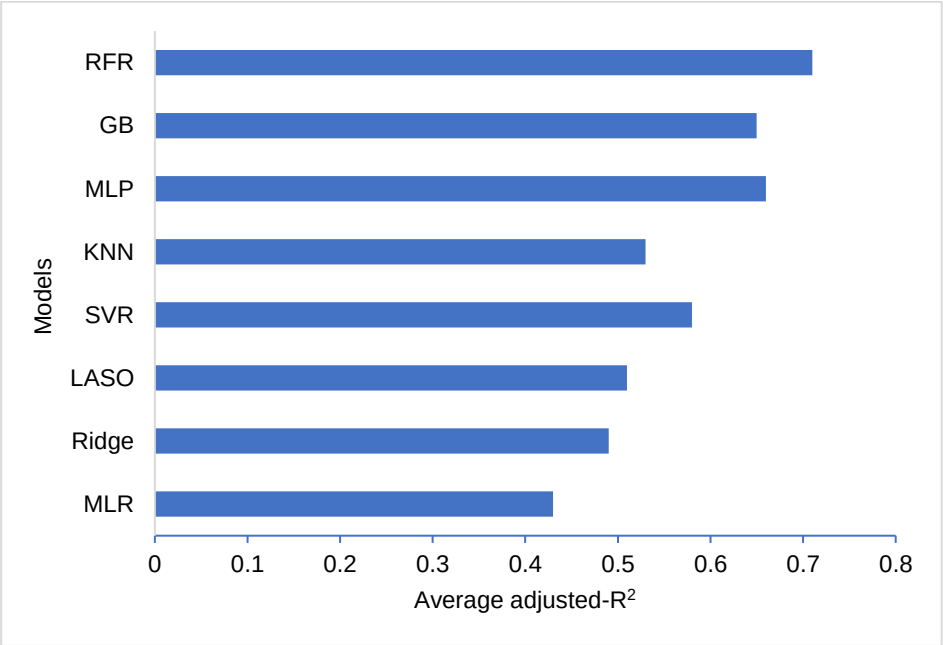


Figure 4.2. Average Adjusted-R^2 of ML models across project datasets

4.4 Further Analysis of RFR Model for Overtime Estimation

As the best-performing model among the tested, the RFR model's evaluation results, in terms of regression error and accuracy performance metrics, as well as its improvement over other model categories and scalability across the six software projects, were further analyzed. Detailed performance evaluation results of RFR in all datasets are presented in Table 4.14.

Table 4.14. RFR Performance Results

Project	MAE	MSE	RMSE	MSLE	MMRE	R^2
ACAD	8.91	156.00	12.49	0.14	0.240	0.84
WEBMET	7.85	133.63	11.56	0.10	0.229	0.89
PSOA	9.64	165.38	12.86	0.18	0.251	0.80
WEBAMHS	10.26	172.13	13.12	0.15	0.245	0.78
PARM	10.65	211.99	14.56	0.20	0.256	0.80
OPMET	10.72	211.70	14.55	0.21	0.259	0.77
Average	**9.67**	**175.14**	**13.19**	**0.16**	**0.247**	**0.81**

RFR achieved an average MAE of 9.67, indicating that, on average, the model's predictions deviated from actual PM satisfaction values by approximately 9.67 units. Specifically in the WEBMET project, RFR exhibited the lowest MAE (7.85), suggesting superior predictive precision in this project, likely due to data homogeneity or fewer outliers. Conversely, in OPMET, RFR recorded the highest

MAE (10.72), indicating challenges in modeling satisfaction for this project, which may be due to nonlinear relationships or domain-specific complexities. The narrow range of MAE values (7.85–10.72) across projects highlights RFR's general robustness, though project-specific tuning may further reduce outliers. With an average MSE of 175.14, the model demonstrates moderate error dispersion. RFR's lowest MSE (133.63) in WEBMET aligns with its superior MAE and RMSE, reinforcing WEBMET as the best-modeled project. In contrast, RFR had the highest MSE values (211.70–211.99) in OPMET and PARM projects, reflecting greater error variance. The squared nature of MSE amplifies the impact of outliers, as seen in the 35% increase from MAE to RMSE averages (9.67 → 13.19), which is critical for applications requiring strict error control. The average RMSE (13.19) reflects the model's sensitivity to larger errors, as RMSE penalizes outliers quadratically. Again, RFR performed best in the WEBMET project (RMSE = 11.56), while in the OPMET and PARM projects, it had the worst RMSE values (14.55–14.56), signaling pronounced error variability in these projects. The consistent alignment of RMSE trends with MAE suggests that extreme errors are not dominant, but they are still impactful. For instance, in OPMET, RMSE (14.55) exceeds MAE (10.72) by 35.7%, underscoring the presence of occasional high-magnitude prediction errors.

The average MSLE (0.16) suggests reasonable accuracy in predicting logarithmically scaled satisfaction values. As observed in RMSE, RFR produced the lowest MSLE in WEBMET (0.10), while in OPMET, it had the highest MSLE (0.21), indicating challenges in handling complex satisfaction patterns in more complex projects. MSLE's focus on relative error makes it particularly relevant for projects where proportional accuracy (for instance, a 10% error) is more critical than absolute deviations. The MMRE average (0.247) falls just below the 0.25 threshold, often deemed acceptable in estimation models for software engineering. WEBMET again led (0.229), while OPMET trailed (0.259). This metric's emphasis on relative error magnitude highlights RFR's consistency but also reveals opportunities for refinement in projects where even minor proportional errors could impact decision-making.

The average R^2 of 0.81 indicates that RFR explains 81% of the variance in PM satisfaction across projects, which is indicative of its estimation accuracy. It achieved the highest R^2 (0.89) in the WEBMET project, indicating near-optimal alignment between predicted and actual values. At the same time, in WEBAMHS, it had the lowest R^2 (0.78), suggesting residual unexplained variance, possibly due to unaccounted contextual factors (such as productivity dynamics). The strong overall R^2 performance validates RFR's utility as a reliable predictor in this domain.

RFR's performance is strongest in projects with lower data variability in terms of the number of features (for instance, WEBMET and ACAD), as evidenced by its dominance across MAE, RMSE, MSE, MSLE, and R^2. Conversely, higher errors in OPMET and PARM suggest domain-specific challenges, such as nonlinear satisfaction drivers or higher complexities. While its average performance is robust, project-specific disparities—particularly in OPMET and PARM—highlight the need for further improvement on the model's scalability and contextual adaptations through hyperparameter tuning or feature engineering.

The project-specific percentage improvements in performance achieved by RFR compared to other model categories are summarized in Tables 4.15 – 4.18.

Table. 4.15. Percentage improvement of RFR against linear models

Project	Metric	% Improvement (maximum)
ACAD	$R^{22\ 2}$	+29.2%
WEBMET	MAE	+5.0%
OPMET	RMSE	+21.4%
PARM	MSE	+30.7%

Table. 4.16. Percentage improvement of RFR against kernel-based models

Project	Model	Metric	Baseline	RFR	% Improvement
ACAD	SVR	MAE	8.94	8.91	+0.3%
WEBMET	KNN	R^{22}	0.74	0.89	+20.3%
WEBAMHS	SVR	RMSE	15.19	13.12	+13.6%
PARM	KNN	MSE	261.15	211.99	+18.8%

Table. 4.17. Percentage improvement of RFR against a neural network-based model

Project	Metric	MLP	RFR	% Improvement
WEBMET	R^2	0.85	0.89	+4.7%
PSOA	MAE	9.95	9.64	+3.1%
OPMET	RMSE	14.62	14.55	+0.5%
PARM	MMRE	0.259	0.256	+1.2%

Table. 4.18. Percentage improvement of RFR against the boosting ensemble model

Project	Metric	GB	RFR	% Improvement
ACAD	MAE	8.22	8.91	-8.4%
WEBMET	R^2	0.82	0.89	+8.5%
PSOA	MSE	176.36	165.38	+6.2%
OPMET	MMRE	0.255	0.259	-1.6%

RFR consistently outperformed linear models, significantly improving error reduction and prediction power. Its R^2 improvement of 15–30% over linear models, reflecting its ability to model non-linear interactions, and its MSE reductions of 20–30% highlight its robustness to outliers. Also, RFR demonstrated moderate to substantial improvements over kernel-based models, particularly in high-dimensional datasets. It outperformed KNN in R^2 (15–20%) and SVR in RMSE (10–15%), but showed minimal gains in MAE. In particular, KNN's sensitivity to local noise (e.g., PARM MSE: +18.8%) limited its reliability. RFR rivaled or exceeded MLP's performance without requiring extensive hyperparameter tuning. While MLP achieved marginally better RMSE in WEBMET, RFR's 3–5% gains in MAE and R^2, coupled with lower computational costs, make it more applicable in overtime allocation estimation. GB and RFR exhibited competitive trade-offs, with GB excelling in point predictions and RFR in stability: GB outperformed RFR in MAE (for instance, in ACAD: -8.4%), but RFR achieved 6–8% better R^2 and MSE in most projects. RFR's superior interpretability (feature importance) offsets minor accuracy trade-offs.

Additionally, the average performance improvements of RFR across all project datasets, compared to different categories of ML models, considering all metrics, are presented in Figure 4.3. It can be observed that RFR demonstrated dominance in MSLE, achieved better trade-offs in the MAE-RMSE ratio, showed higher performance in MMRE, and exhibited consistency in R^2. RFR's 23.8–57.9% improvements in MSLE over all models validate its utility for projects requiring proportional error minimization. While GB and MLP occasionally outperform RFR in MAE, RFR's 12–33% MSE/RMSE advantages justify its balanced handling of outliers without over-penalization. Additionally, RFR's 2.5–24.6%²gains in R^2 confirm its superiority in explaining variance, critical for PM's trust in the model and actionable insights. RFR demonstrated broad superiority across all metrics in most projects, making it the most reliable model among the tested models for predicting PM satisfaction in overtime allocations. While Gradient Boosting and Neural Networks achieve niche advantages (for instance, in MAE), RFR's holistic performance, stability, and interpretability solidify its application in modeling PM's subjective preferences for overtime allocations in software projects.

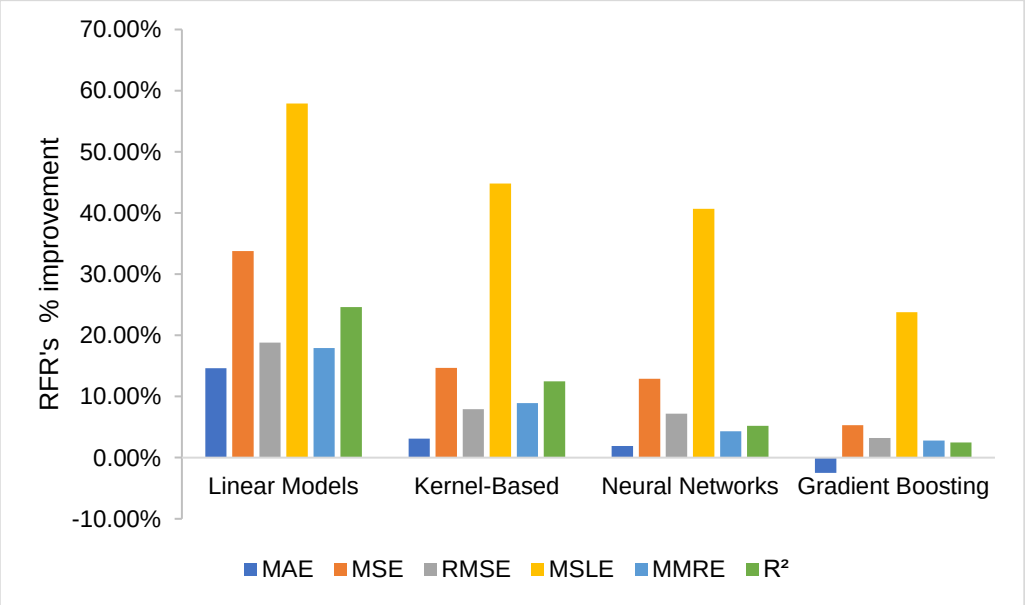


Figure 4.3. Average improvement of RFR against other models in all projects

The scalability of RFR across projects is analyzed based on Adjusted-R^2 using the number of WPs and FPs as measures of project size and complexity. The number of WPs in the project translates to the dimensionality of the generated datasets of overtime allocation plans. According to the feedback from the participating PMs in this work, the number of WPs determines the preferred patterns of overtime allocation. Additionally, the number of FPs determines the complexity of the projects and the effort required to complete them, directly affecting the partial and total overtime hours assigned to tasks and the entire project, respectively. The scalability patterns observed in the results are shown in Figures 4.4 and 4.5.

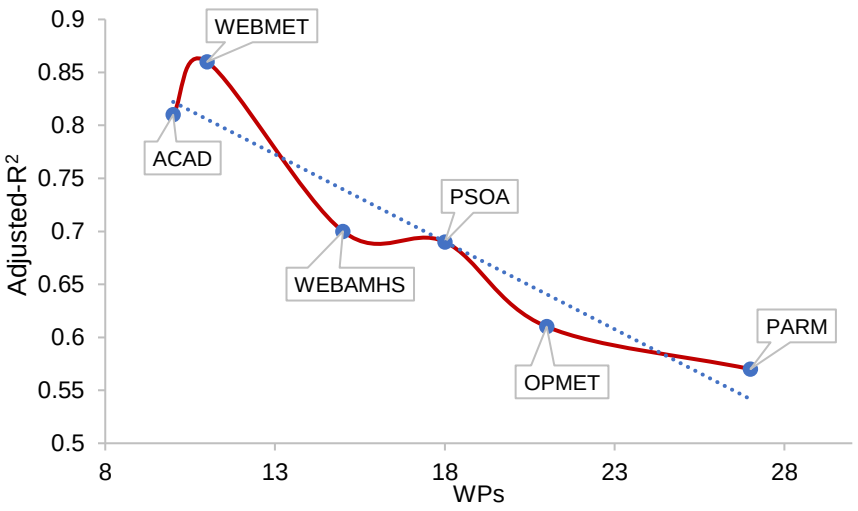


Figure 4.4. Adjusted-R^2 of RFR vs number of WPs in the projects

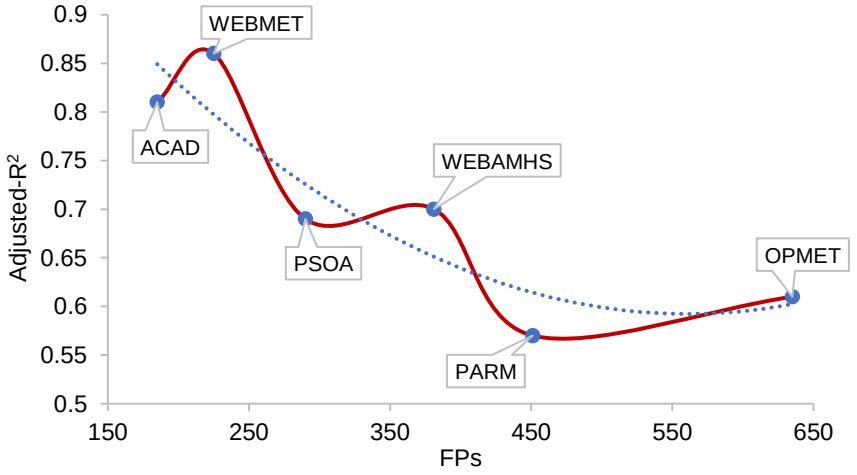


Figure 4.5. Adjusted-R^2 of RFR vs the number of FP in the projects

Both graphs exhibit a consistent inverse correlation between RFR model performance and project complexity, though the strength and nature of this relationship differ between the two metrics. In the FP, performance declines steadily from 0.86 (WEBMET, 225 FPs) to 0.61 (PARM, 635 FPs), reflecting a near-linear degradation as functional complexity increases. This trend suggests that the RFR model struggles to maintain accuracy in projects with higher functional granularity, likely due to increased interdependencies and non-linear relationships that affect model generalization. This may be due to the presence of tasks with high developer effort in large projects. These types of tasks are difficult to allocate overtime for, and they contribute significantly to the overall overtime of the project. An exception is otherwise observed between ACAD and WEBMET as the performance increases with increasing FP size. However, the difference in FP between these two projects is significantly lower than that between the PSOA and WEBAMHS projects, which might explain this exceptional result. By contrast, the adjusted-R^2 versus WP graph shows a sharper initial decline (e.g., 0.81 at 10 WPs to 0.69 at 18 WPs), followed by a decline (0.61–0.57) as WPs exceed 20. This nonlinearity implies that while initial increases in WPs significantly strain model performance, adaptive mechanisms may

mitigate losses at higher WP thresholds. The divergent trends in FP and WP highlight key differences in how project complexity affects model efficacy, revealing that distinct aspects of complexity differentially impact predictive performance. Notably, PARM (450 FPs, 27 WPs) and OPMET (635 FPs, 21 WPs) serve as outliers in both graphs, with performance metrics disproportionately lower than their complexity values would predict. This deviation may indicate unaccounted variables, such as WP dependencies, that may intensify complexity effects. These findings underscore the need to calibrate model selection to project complexity profiles.

4.5 Optimizing Hyperparameters of RFR with Greedy Halving Grid Search

Fine-tuning hyperparameters can lead to better model performance and higher accuracy. To find the best trade-off between RFR model complexity and performance, its hyperparameters are usually optimized [209–211]. Such parameters include *n_estimator*, *min_samples_split*, *max_features*, *max_depth*, *min_samples_leaf* and bootstrap. *n_estimator* defines the number of trees within the forest. Increasing this value generally enhances the model's performance by reducing variance and overfitting; however, the benefits may diminish beyond a certain threshold, leading to increased computational costs. *min_samples_split* represents the minimum number of samples necessary to split an internal node in the trees. Setting this parameter high prevents the model from overfitting specific patterns, thereby improving generalization. *max_features* specifies the number of features to consider when seeking the optimal split. A low value introduces randomness, which reduces overfitting and results in less correlated trees. Conversely, a high value may boost accuracy but also increase the risk of overfitting. *max_depth* controls the maximum depth of each tree in the ensemble. Deeper trees can model more complex data patterns, but may cause overfitting if excessively deep. Shallower trees tend to simplify the model but risk underfitting. *min_samples_leaf* denotes the minimum number of samples required at a leaf node. Higher values promote smoothing by ensuring leaves contain sufficient samples. Bootstrap determines whether bootstrap sampling is employed during tree construction; utilizing bootstrap samples introduces randomness into the sampling process.

Search-based optimization techniques, including Grid Search (GS), Random Search (RS), and Bayesian Optimization (BO), are employed to fine-tune hyperparameters [209–212]. Nevertheless, certain intrinsic challenges constrain the effectiveness and applicability of these approaches. As an exhaustive search strategy, grid search is hindered by high time complexity in large search spaces and underperforms with extensive datasets [213]. Random search, which depends on stochastic sampling, is inefficient in thoroughly exploring the search space, necessitating numerous iterations to identify the optimal parameter combination, thereby incurring substantial computational costs [212]. BO, functioning as a black-box method, encounters scalability issues in high-dimensional optimization scenarios, rendering it computationally demanding [214]. A more efficient alternative, Halving Grid Search (HGS), has recently been proposed to enhance the search efficiency of RS and to mitigate the performance limitations of GS [215,216]. HGS utilizes the successive halving approach introduced in [217] to traverse the hyperparameter space. This method allocates resources uniformly—such as training samples and time—among hyperparameter configurations, assesses their performance, and discards the lower-

performing half [218]. This process iterates, with additional resources applied in subsequent rounds, until only the top-performing configuration remains. The procedural outline of the original HGS is provided in Algorithm 3. It ensures that only models with the highest performance—those possessing the optimal hyperparameters—advance to subsequent iterations. However, when HGS is employed for hyperparameter optimization using k-fold cross-validation, the associated computational costs escalate exponentially [219]. Consequently, a greedy variant of HGS (gHGS) has been proposed to optimize the hyperparameters of RFR, thereby enhancing software over time estimation.

Algorithm 3: Successive Halving Algorithm

Input: set of hyperparameter configurations n, total budget B (e.g. training samples)

1: Initialize $S_0 \leftarrow [n]$
2: for $r = 0$ to $\lceil \log_2 n \rceil - 1$:ss
3: allocate $B_r = \sum_{j=0}^{r} \left\lfloor \frac{B}{|S_j| \lceil \log_2 n \rceil} \right\rfloor$ resources to each configuration $i \in S_r$
4: Train model on each i using cross-validation and compute the performance value p_i
5: set S_{r+1} to $\left\lceil \frac{|S_r|}{2} \right\rceil$ configurations in S_r with the highest p_i
6: end for

Output: the configuration in $S_{\lceil \log_2 n \rceil}$

gHGS combines the greedy cross-validation approach [220] with the successive halving algorithm to achieve rapid convergence. Unlike traditional cross-validation, which evaluates all folds of a model sequentially before moving to the next, greedy cross-validation uses a heuristic to quickly identify and focus on the most promising candidates. It starts by evaluating one fold per candidate, generating partial performance metrics, and then selectively evaluates the next fold of the top-performing model so far. This approach allows the best models to be fully assessed earlier than the less promising ones. Figure 4.6 shows the difference between greedy and standard cross-validation. A detailed explanation of the greedy cross-validation method is reported in [219].

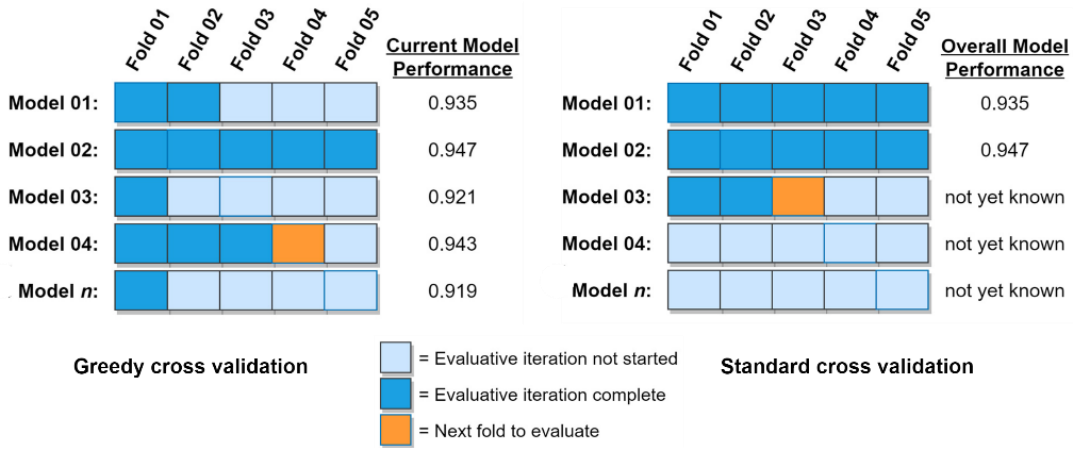


Figure 4.6. Illustrative representation of greedy and standard cross-validation. Adapted from [220]

In each iteration of gHGS's successive halving process, the greedy cross-validation method is used to select the best model configurations, enabling them to proceed as quickly as possible to the

following evaluation stage. Since the number of models required for each successive halving iteration is computed in advance with the formula $S_{r+1} = \left\lceil \frac{|S_r|}{2} \right\rceil$, the current halving iteration can be stopped once the number of ML models assessed through greedy cross-validation matches $\left\lceil \frac{|S_r|}{2} \right\rceil$. The computational cost of evaluating all candidate models in each iteration before choosing the best ones for the next is reduced by leveraging the inherent prioritization of promising ML models in greedy cross-validation. Consequently, any iteration of successive halving, except the final one, can be terminated early once enough models have been evaluated to meet the next iteration's requirements. The algorithm for gHGS used in this study is shown in Algorithm 4.

The gHGS was used to optimize the hyperparameters of the RFR model. To prevent allocating an excessive number of samples for training, the minimum number of training samples per iteration was set to *6k*, where *k* is the number of folds. With k set to 5, each iteration required at least 30 training samples. The hyperparameter search ranges are detailed in Table 4.19 and were based on the best RFR hyperparameters identified in prior studies across various software engineering applications. In total, 2048 models were evaluated to determine the optimal RFR hyperparameters for each software project.

Algorithm 4: Greedy Halving Grid Search (gHGS) Algorithm

Input: the set of all hyperparameter configurations n, total number of training samples B, number of folds *k*

Initialize $S_0 \leftarrow [n]$, $B_{min} \leftarrow 6 \times k$ (Minimum number of training samples iteration)
for $r = 0$ to $\lceil \log_2 n \rceil - 1$:
    allocate $B_r = max\left(\sum_{j=0}^{r} \left\lfloor \frac{B}{|S_j| \lceil \log_2 n \rceil} \right\rfloor, B_{min}\right)$ training samples to each configuration $i \in S_r$
    $I_{Best} = \emptyset$ (the set of best performing models in the current iteration)
    split B_r into k folds, such that $B_r = \{ b_1, b_2, \ldots, b_k \}$
    for each $i \in S_r$ do
        train i using folds $\{ b_2, \ldots, b_k \}$
        $P_i \leftarrow$ Performance of i evaluated for fold b_1
        $N_i \leftarrow 1$ (number of folds of i evaluated for fold b_1
    while $n(I_{Best}) < \left\lceil \frac{|S_r|}{2} \right\rceil$ do
        $i^* \leftarrow$ best incompletely evaluated $i \in S_r$ per *P*
        $N_{i^*} \leftarrow N_{i^*} + 1$
        train i^* with all folds $b_j \in B_r$, such that $j \neq N_{i^*}$
        evaluate the performance P_i of i^* using fold $b_{N_{i^*}}$
        P_{i^*} = is the average P_i of i^* for folds = $\{ b_1, \ldots, b_{N_{i^*}} \}$
        if $N_{i^*} = k$:
            add i^* to I_{Best}
    end while
    set S_{r+1} to I_{Best}
end for

Output: the hyperparameter configuration in $S_{\lceil \log_2 n \rceil}$

Table 4.19. RFR's hyperparameters searching configurations

Parameter	Range
n_estimator	{50, 100, 150, 200}
min_samples split	{2, 5, 10, 15}
max_features_	{Log_2, Sqrt, 0.2, 0.3}
max_depth	{5, 10, 15, "None"}
min_samples_leaf	{1, 2, 4, 6}
bootstrap	{ "Yes", "No"}

To evaluate the performance of the proposed gHGS-RFR based on enhancement validation, usefulness, and impact, its results were compared with RFR without tuning, RFR optimized with other search-based hyperparameter tuning methods (GS, RS, and BO), and RFR optimized with HGS, respectively. The regression errors (comprising mean errors: MAE and RMSE, and relative errors: MSLE and MMRE) and the accuracy (measured with R^2) of the optimized RFR models compared with the default RFR model are presented in Table 4.20, with the best results in bold. Analysis of the presented results reveals several key insights regarding the effectiveness of various hyperparameter optimization methods. All optimization techniques demonstrated clear improvements over the baseline RFR, with average reductions in MAE ranging from 14% to 15% and increases in R^2 between 4% and 8%. Among the methods evaluated, HGS and gHGS emerged as the top performers, achieving the lowest error metrics—MAE values between 8.27 and 8.28 and RMSE values around 11.02 to 11.03—alongside the highest R^2 score of 0.89. BO also showed strong performance, consistently outperforming traditional methods like GS and RS across all projects, with an R^2 of 0.87 compared to 0.84–0.85, underscoring its efficiency in navigating the hyperparameter space. Notably, HGS-based methods dominated in five out of six projects (approximately 83%) across all key metrics, highlighting their robustness and adaptability. The most substantial performance gains were observed in the ACAD project, where HGS and gHGS methods achieved an R^2 of 0.90, marking the most significant improvement among all evaluated scenarios.

Validating significant enhancement on the default RFR, RFR-gHGS demonstrated superior capability in minimizing absolute and square errors, outperforming RFR in all projects. The average MAE reduction of 14.5% (from 9.67 to 8.27) relative to baseline RFR reflects its efficacy in reducing estimation biases. Notably, MAE improvements reached 28% in the ACAD project, suggesting exceptional performance. The RMSE results further validate this advantage, with RFR-gHGS achieving a 16.4% average reduction. This dual improvement in both MAE and RMSE indicates that RFR-gHGS not only reduces average estimation deviations but also effectively controls outlier-induced errors. In addition, RFR-gHGS delivered marked improvements in relative error metrics over baseline RFR with an 18.8% reduction in average MSLE (0.16 to 0.13) and a 20% reduction in average MMR. This highlights its enhanced ability and improved accuracy in prioritizing the relative magnitude of errors. The stability of these improvements across datasets—particularly in OPMET, where it produced the lowest MMRE (0.191) and MSLE (0.169)—suggests that RFR-gHGS robustly adapts to heterogeneous project complexities. The R^2 improvements underscore the enhanced performance of RFR-gHGS, with an average 8% increase over the baseline RFR. This enhancement is especially significant in complex

datasets like WEBAMHS and OPMET, where R^2 reached 0.89, indicating near-optimal accuracy in estimating PM's preferences in overtime planning. These improvements underscore the crucial role of advanced hyperparameter optimization in overcoming the limitations of default RFR configurations, which frequently suffer from suboptimal parameter selection. The specific improvements achieved by RFR-gHGs over the default RFR in terms of average regression error and accuracy are shown in Figures 4.7 and 4.8, respectively.

Table 4.20. Results of RFR hyperparameters optimization methods

Dataset	Metrics	Models					
		RFR	RFR-GS	RFR-RS	RFR-BO	RFR-HGS	RFR-gHGS
ACAD	MAE	8.91	7.82	8.17	7.04	6.43	**6.42**
	RMSE	12.49	11.65	12.01	11.89	10.34	**10.32**
	MSLE	0.140	0.135	0.139	0.130	0.124	**0.122**
	MMRE	0.240	0.225	0.237	**0.204**	0.206	0.205
	R^2	0.84	0.87	0.85	0.88	0.88	**0.90**
WEBMET	MAE	7.85	7.68	7.70	7.50	7.45	**7.44**
	RMSE	11.56	10.78	10.07	9.55	**9.38**	9.47
	MSLE	0.104	0.100	0.098	0.087	**0.082**	0.083
	MMRE	0.229	0.220	0.200	**0.187**	0.188	**0.187**
	R^2	0.89	0.89	0.90	**0.91**	**0.91**	**0.91**
PSOA	MAE	9.64	8.71	9.04	8.39	**8.15**	8.18
	RMSE	12.86	11.94	11.67	10.01	**9.82**	9.84
	MSLE	0.181	0.152	0.175	0.135	**0.129**	**0.129**
	MMRE	0.251	0.221	0.246	0.201	**0.195**	**0.195**
	R^2	0.80	0.85	0.82	0.87	**0.88**	**0.89**
WEBAMHS	MAE	10.26	8.64	9.51	8.73	8.72	**8.71**
	RMSE	13.12	11.89	12.01	**10.87**	10.88	10.88
	MSLE	0.153	0.136	0.145	0.125	**0.118**	0.121
	MMRE	0.245	0.237	0.235	0.212	**0.203**	0.204
	R^2	0.78	0.83	0.81	0.86	**0.89**	**0.89**
PARM	MAE	10.65	10.12	10.02	**9.62**	9.64	**9.62**
	RMSE	14.56	13.94	13.05	**12.78**	**12.78**	12.79
	MSLE	0.202	0.182	0.185	0.176	**0.152**	**0.152**
	MMRE	0.256	0.231	0.225	0.208	0.204	**0.203**
	R^2	0.80	0.84	0.84	0.85	0.87	**0.88**
OPMET	MAE	10.72	9.97	9.84	9.27	9.27	**9.25**
	RMSE	14.55	13.86	13.67	12.91	12.88	**12.87**
	MSLE	0.214	0.195	0.187	0.174	0.171	**0.169**
	MMRE	0.259	0.206	0.195	0.190	**0.191**	**0.191**
	R^2	0.77	0.79	0.83	0.85	**0.89**	**0.89**

Compared to conventional hyperparameter optimization methods (GS, RS, BO), gHGS consistently outperformed all benchmarks. Average regression error and accuracy results across all datasets are depicted in Figures 4.9 and 4.10. It can be observed that RFR-gHGS maintained consistent superiority, surpassing RFR-BO—the best performing benchmark method—by 1.9% in average MAE (8.43 to 8.27) and 2.3% in average R^2 (0.87 to 0.89) across all projects. It also outperformed RFR-BO by 7.1% (0.14 to 0.13) in average MMRE. Similarly, RFR-gHGS outperformed RFR-GS and RFR-RS by 6.2% and 8.6% in MAE, respectively, underscoring the limitations of exhaustive and random search strategies in high-dimensional parameter spaces. While RFR-BO demonstrated strong results in datasets like WEBMET (R^2 = 0.91) due to its probabilistic efficiency, RFR-gHGS matched or exceeded its performance in the remaining projects while maintaining superior generalizability across all datasets.

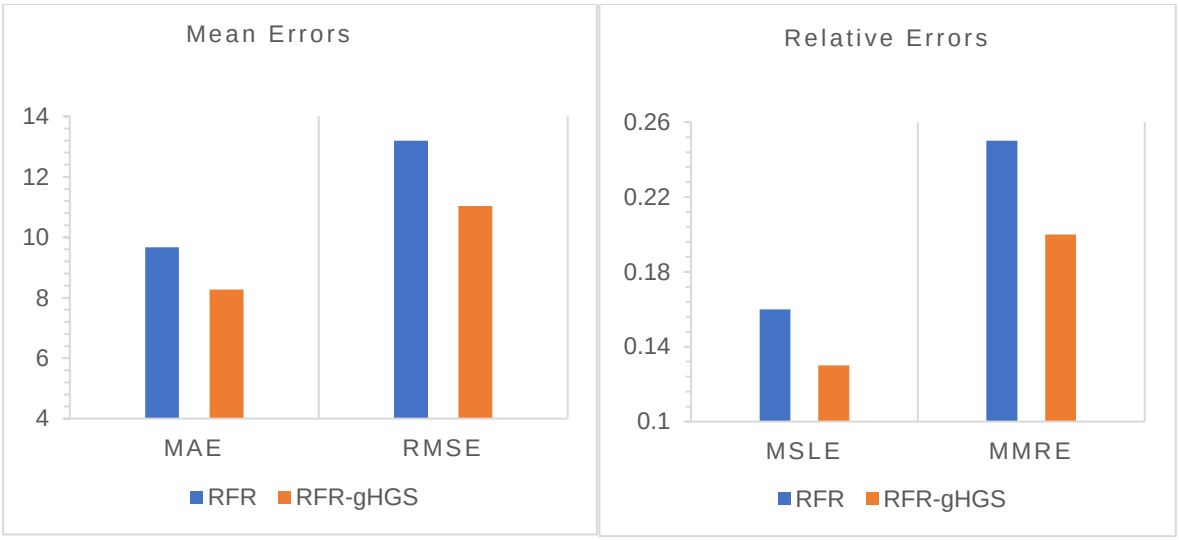


Figure 4.7. Average regression errors of RFR-gHGS compared with RFR across all projects

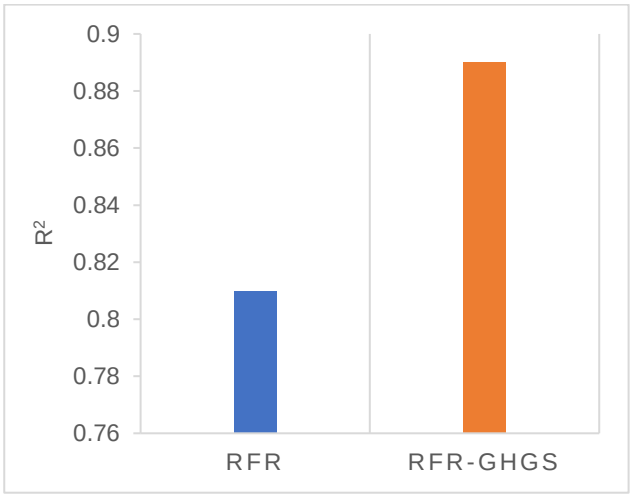


Figure 4.8. Average accuracy of RFR-gHGS compared with RFR across all projects

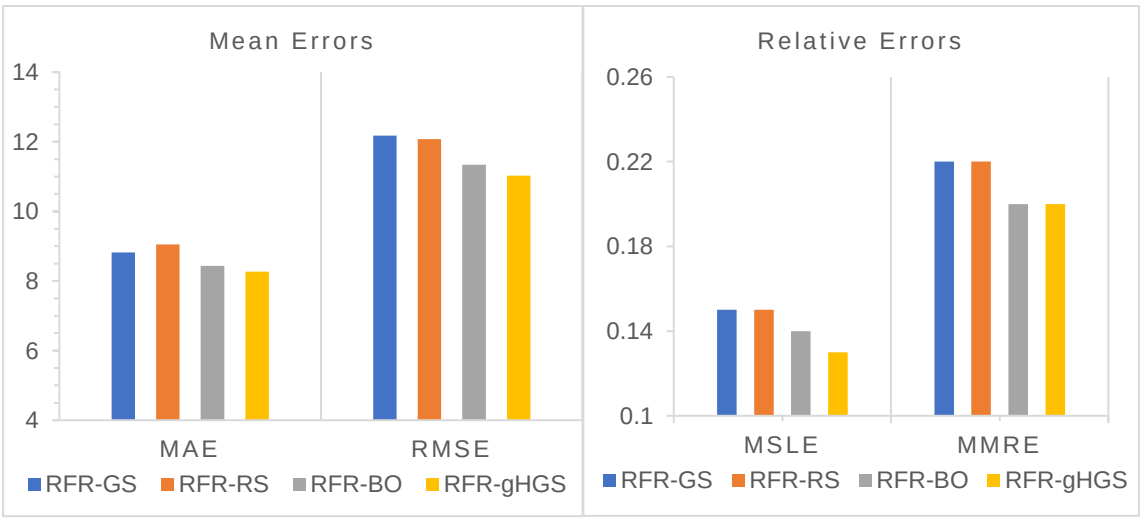


Figure 4.9. Average regression errors of RFR-gHGS compared with benchmark hyperparameters tuning methods across all projects

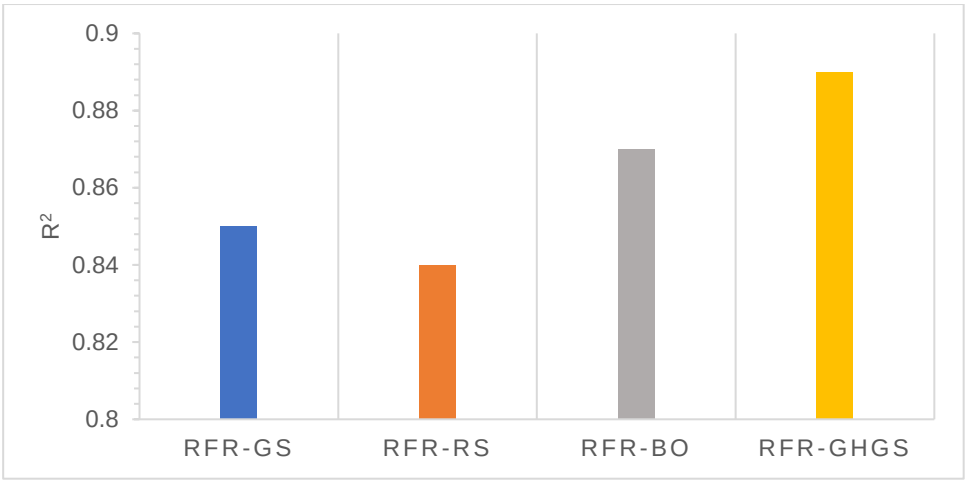


Figure 4.10. Average accuracy (measured as $R^{2)}$ of RFR-gHGS compared with benchmark hyperparameter tuning methods across all projects

The comparison with RFR-HGS revealed marginal but consistent advantages of RFR-gHGS. As depicted in Figures 4.11 and 4.12, both methods achieved the same average R^2 (0.89) and MMRE (0.20), but RFR-gHGS slightly outperformed RFR-HGS in average MAE (8.27 vs. 8.28) and RMSE (11.03 vs. 11.02). This marginal improvement can be attributed to the integration of the greedy cross-validation approach in the HGS framework, which enhances parameter exploration by prioritizing the evaluation of only promising regions. The stability of RFR-gHGS was particularly evident in datasets with high dimensionality (for instance, PARM, OPMET), where it achieved the lowest errors, suggesting superior adaptability to heterogeneous software projects overtime estimation datasets.

Figure 4.11. Average regression errors of RFR-gHGS compared with RFR-HGS across all projects

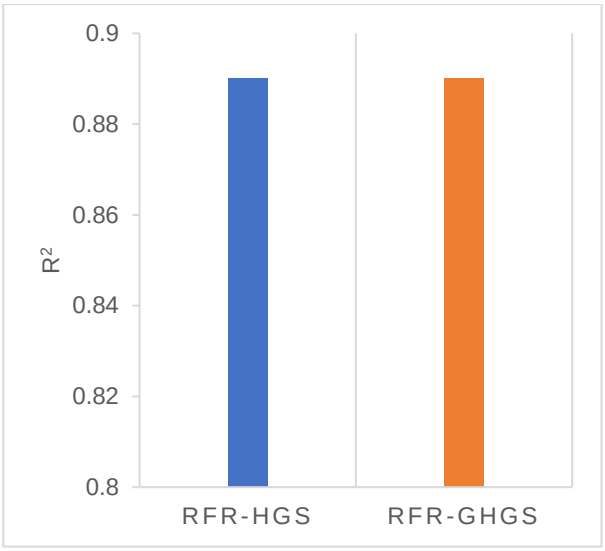


Figure 4.12. Average accuracy of RFR-gHGS compared with RFR-HGS across all projects

The evaluation of RFR-gHGS against benchmark methods reveals considerable enhancements across three critical dimensions of regression performance: general error minimization (MAE, RMSE), relative error reduction (MSLE, MMRE), and predictive accuracy (R^2). Although RFR-HGS showed a competitive performance, its computational cost in terms of running time almost doubles that of the proposed RFR-gHGS, which limits its computational efficiency in this domain. The wall-clock time of RFR-gHGS, as compared to other hyperparameter optimization methods, is presented in Figure 4.13. The evaluation of wall-clock time across the six project datasets reveals that RFR-gHGS is the most computationally efficient hyperparameter optimization method among those tested. It consistently achieved the lowest runtime, significantly outperforming traditional methods and even the more advanced HGS. On average, RFR-gHGS reduced computational cost by approximately 70% compared to RFR-GS, 50–60% compared to RFR-RS, and 40–45% compared to RFR-BO and RFR-HGS. These reductions are particularly valuable in large projects, such as PARM and OMET, where time is critical to model-building workflows. Dataset-specific improvements were most pronounced in WEBMET and ACAD, where RFR-gHGS achieved over 70% reduction in runtime compared to RFR-GS. Even in more computationally demanding datasets, such as PARM, RFR-gHGS maintained a substantial lead in efficiency. These findings underscore RFR-gHGS's strength not only in predictive performance but also in practical scenarios where computational resources are a constraint. Detailed efficiency gains (in percentage) achieved by RFR-gHGS against other methods in all project datasets are presented in Table 4.21.

Table 4.21. Percentage improvement of RFR-gHGS in wall-clock time over other models in all projects

Dataset	RFR-GS (%)	RFR-RS (%)	RFR-BO (%)	RFR-HGS (%)
ACAD	69.84	52.12	65.30	63.36
WEBMET	71.09	54.86	65.06	63.87
PSOA	69.26	66.76	46.24	47.84
WEBAMHS	66.07	59.60	40.45	43.25
PARM	63.09	56.64	45.71	41.06
OPMET	65.47	60.33	45.76	42.63

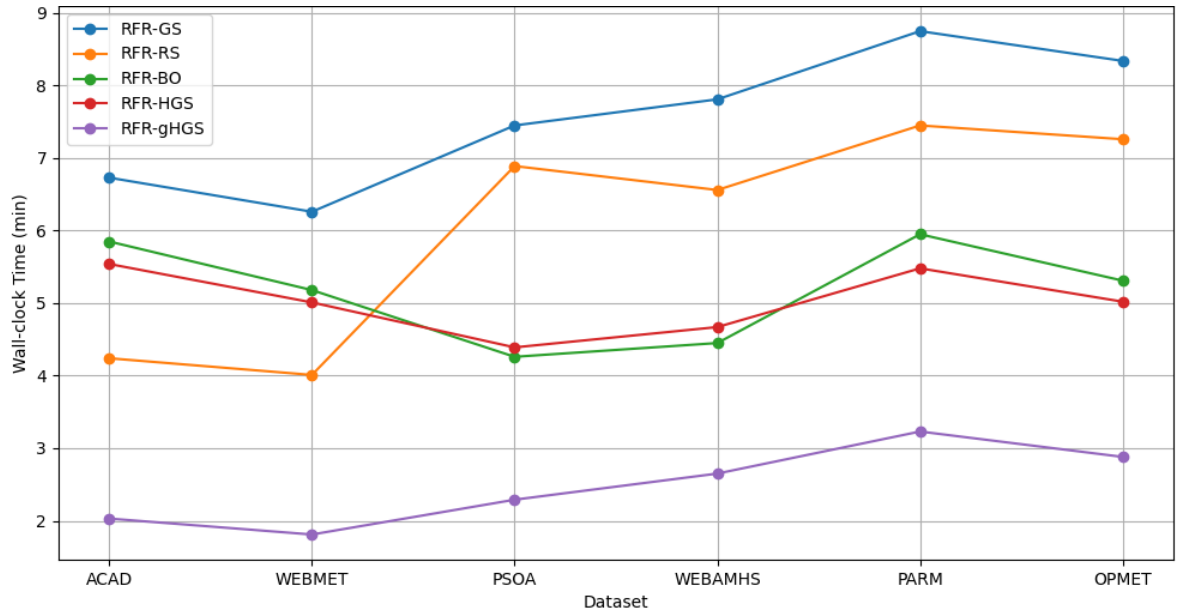


Figure 4.13. Wall-clock time of hyperparameter optimization methods across all projects

Furthermore, the scalability of RFR-gHGS is examined to assess whether optimizing RFR with gHGS enhances its stability across projects of varying complexity. As done for RFR without hyperparameter optimization, the Adjusted-R^2 measure is analyzed based on the number of WPs and FPs in the projects. As shown in Figures 4.14 and 4.15, the results indicate that RFR-gHGS clearly enhances model scalability and robustness. Unlike the baseline RFR, which exhibits a marked decline in Adjusted-R^2 as complexity increases, RFR-gHGS maintains nearly consistent levels of predictive accuracy. Specifically, the Adjusted-R^{22}values for RFR-gHGS remain within a narrow and elevated range (approximately 0.87 to 0.91), regardless of the growth in WPs or FPs. This stability suggests that the gHGS optimization effectively tunes the model to generalize well across varying scales of input complexity. The minimal performance degradation observed indicates that RFR-gHGS is not only more accurate but also more resilient to the challenges of more complex projects. Such scalability is crucial in real-world applications, where project size and complexity can vary significantly, and it underscores the practical value of incorporating advanced hyperparameter optimization techniques into machine learning workflows.

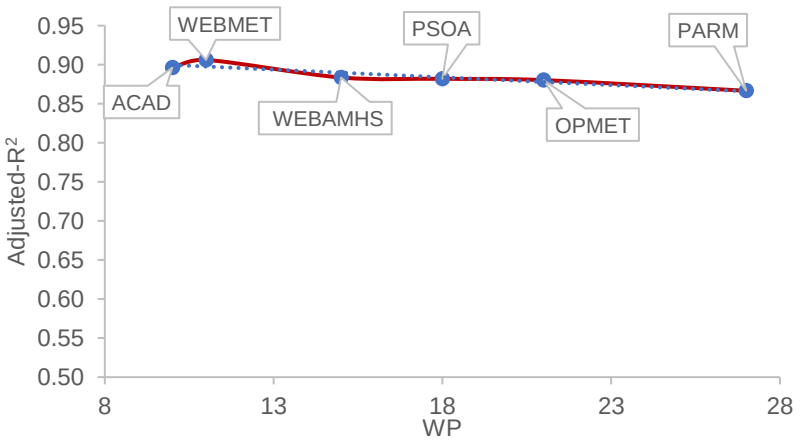


Figure 4.14. Adjusted-R^2 of RFR-gHGS vs number of WPs in the projects

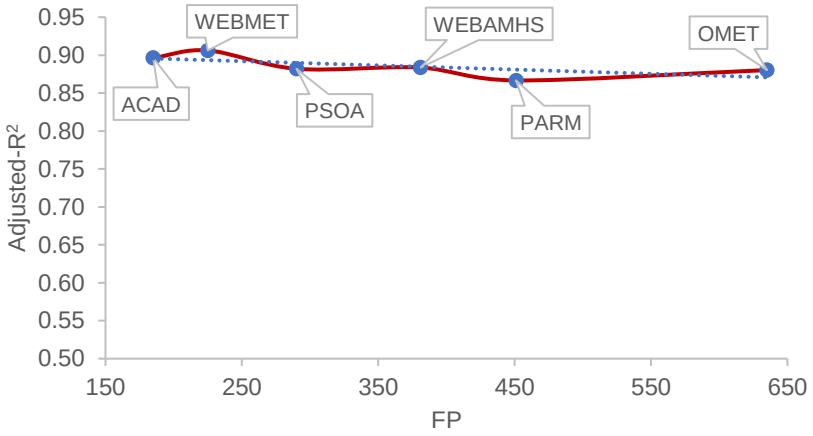


Figure 4.15. Adjusted-R^2 of RFR-gHGS vs number of FP in the projects

Lastly, the interpretability of the gHGS-RFR model was analyzed using SHapley Additive exPlanations (SHAP) method. This step is crucial for explaining the estimation output of the gHGS-RFR model across all six project datasets. SHAP works by estimating the contribution of each feature in a dataset to the model's final prediction [221]. The contribution of each feature to the final prediction of a single instance is calculated independently and then summed up for all instances to produce SHAP values [222]. The SHAP values represent the impact of each feature in the dataset on the model's prediction. Figures 4.16 to 4.21 present the SHAP summary plots for all six datasets; each point on the plots represents a SHAP value for a feature in a specific instance, with color indicating the feature value (red for high and blue for low).

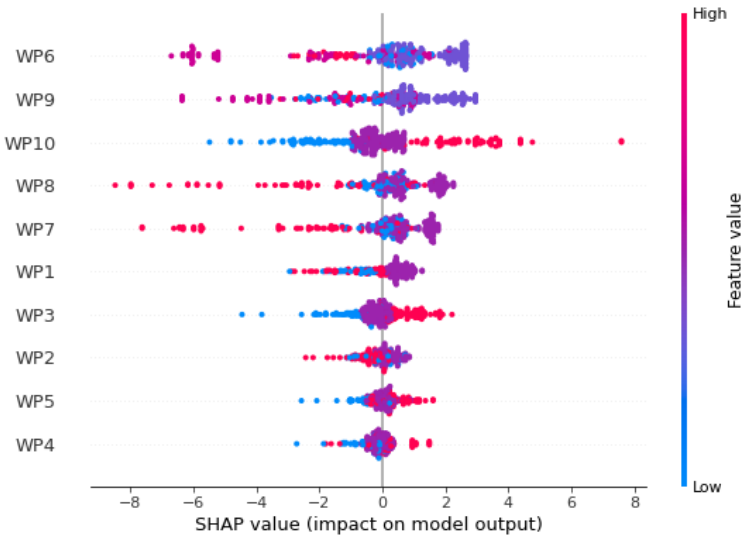


Figure 4.16. SHAP Summary Plot for ACAD dataset

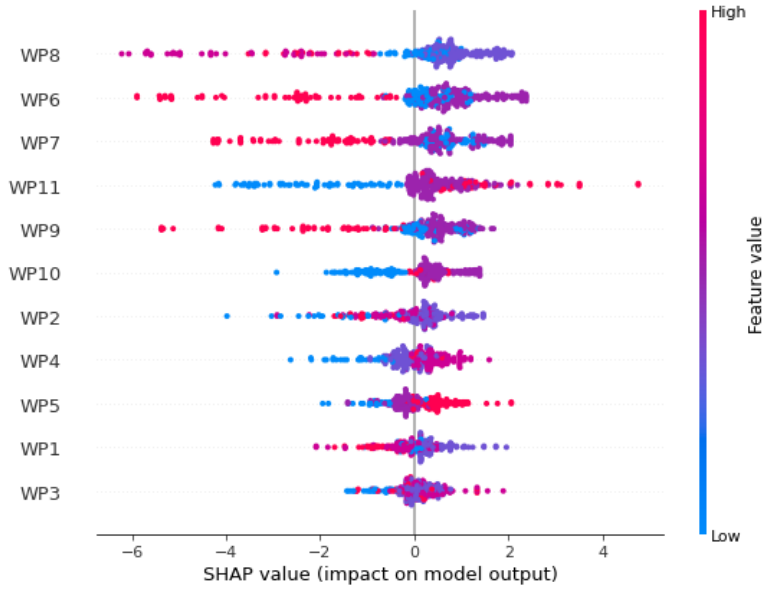


Figure 4.17. SHAP Summary Plot for WEBMET dataset

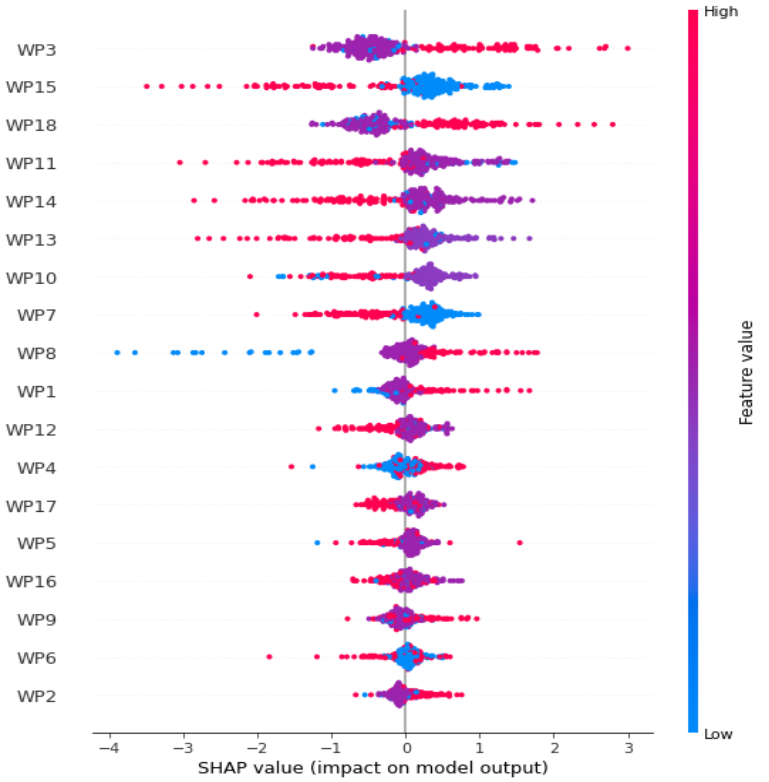


Figure 4.18. SHAP Summary Plot for PSOA dataset

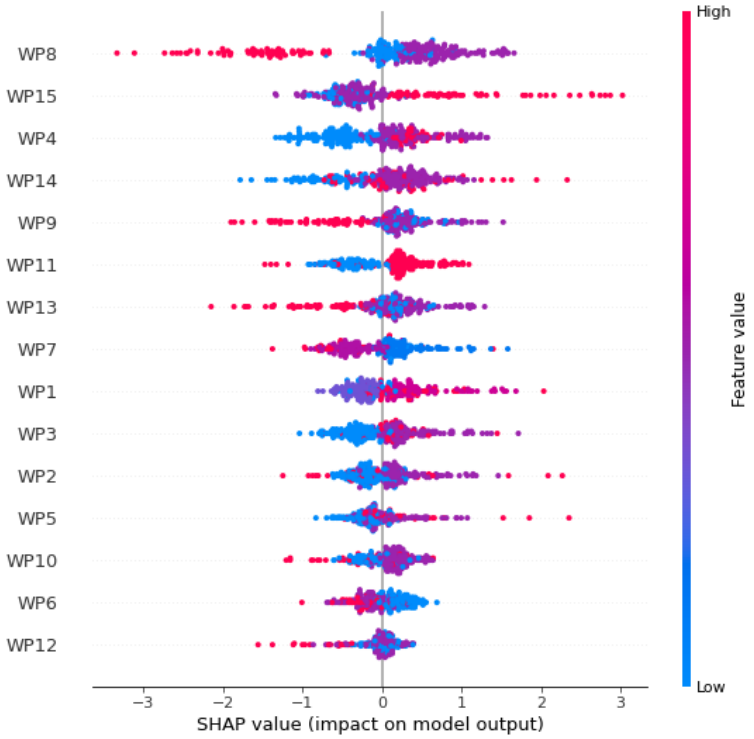


Figure 4.19. SHAP Summary Plot for WEBAMHS dataset

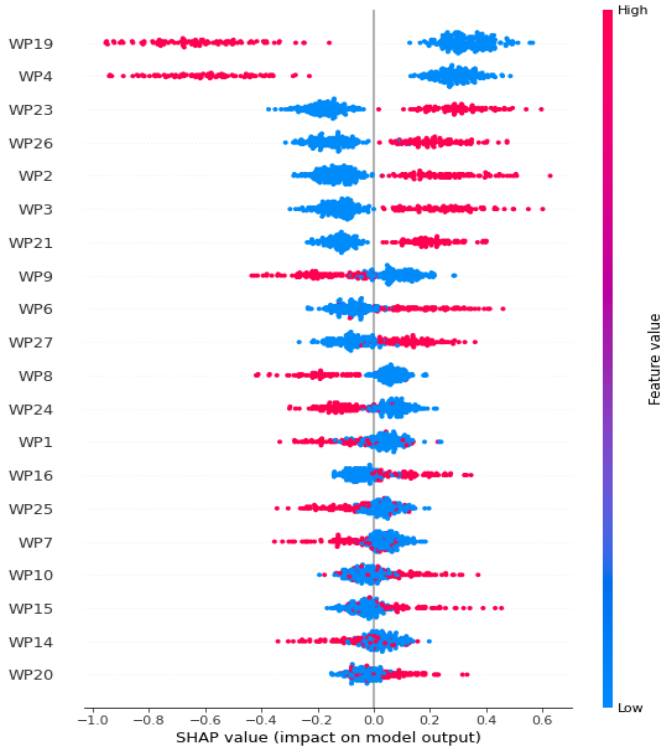


Figure 4.20. SHAP Summary Plot for WEBAMHS dataset

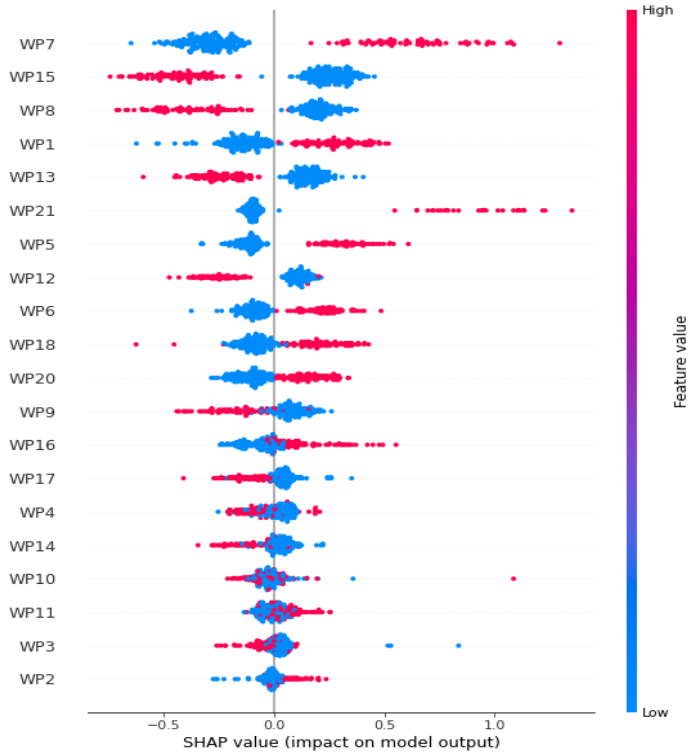


Figure 4.21. SHAP Summary Plot for WEBAMHS dataset

The key observations from the plots are that high FP or duration values for certain WPs often correspond to negative SHAP values, indicating that PMs tend to rate overtime plans lower when critical tasks are large or time-consuming. In contrast, low-dependency WPs with moderate FP values tend to have positive SHAP contributions, suggesting that PMs favor overtime allocations that target less interdependent tasks. Specifically, high values of WP6 and WP9 in the ACAD dataset were associated with negative SHAP values, indicating that PMs penalize overtime plans that heavily burden these tasks. In WEBMET, WP8 and WP6 showed positive SHAP values when their values were moderate, suggesting that PMs tend to favor allocating overtime to manageable, non-critical tasks. WP11 and WP14 in PSOA had high SHAP values when their dependency levels were low, indicating that PMs prefer overtime on isolated tasks to avoid cascading delays. In WEBAMHS, WP8 and WP15 had strong positive SHAP values when their durations were short, implying that PMs reward overtime plans that accelerate quick wins in critical modules. In the PARM dataset, WP19 and WP4 exhibited negative SHAP values when their overtime values were high, suggesting that PMs are cautious about allocating overtime to large and complex tasks. In OPMET, WP7 and WP15 had high SHAP values when overtime is high and low, respectively. These WPs have low dependency levels and their durations were moderate, reinforcing the preference for overtime on less entangled tasks.

Additionally, it can be observed that WPs with high feature importance (ranking at the top of the plot) tend to fall in the second half of the schedule in most projects, confirming the SH strategy as the most preferred overtime allocation pattern by PMs, as reported in previous studies [21,22]. This analysis confirms that the model captures project-specific nuances, identifying which WPs PMs intuitively prioritize when evaluating overtime plans. These insights validate the model's ability to

internalize and replicate expert reasoning patterns. Moreover, the SHAP plots demonstrate that the model integrates multiple contextual feature characteristics, which is consistent with the multi-criteria nature of SOP. By identifying the most influential WPs and quantifying their impact in overtime estimation, the gHGS-RFR model offers actionable insights for PMs, supporting transparent, data-driven decision-making in software overtime planning.

4.6 Findings

The key findings from this chapter are:

1. The baseline RFR model demonstrated good predictive performance across six diverse software projects, achieving an average R^2 of 0.81 and an MAE of 9.67. However, its performance varied with project complexity, declining in larger and more intricate datasets. The comparative analysis with other regression models revealed that RFR consistently outperformed linear and kernel-based models, and rivaled or exceeded the performance of MLP and GB in most metrics.
2. The integration of the greedy halving grid search optimization strategy significantly enhanced RFR's performance. It achieved an average reduction of 14.5% and 16.4% in MAE and RMSE, respectively, over the baseline RFR. It also improved relative error metrics (MSLE and MMRE) by approximately 18–20% and increased average R^2 to 0.89. These improvements were consistent across all projects, with the most notable gains observed in datasets such as ACAD and OPMET.
3. When benchmarked against other hyperparameter optimization methods – grid search, random search, Bayesian optimization, and standard halving grid search – greedy halving grid search consistently delivered superior results while achieving substantial computational efficiency. In addition to improved performance, greedy halving grid search reduced wall-clock time by 70% compared to grid search, 50-60% compared to random search, and 40-45% compared to Bayesian optimization and standard halving grid search, making it the most efficient among the tested methods for optimizing RFR hyperparameters.
4. The scalability analysis demonstrated that RFR optimized with greedy halving grid search maintained high and stable predictive accuracy across increasing project complexity, measured by the number of work packages and function points. This contrasted sharply with the baseline RFR, whose performance declined significantly with complexity.

These findings underscore the effectiveness, efficiency, and applicability of RFR-gHGS in real-world SOP scenarios, highlighting its potential as a reliable tool for supporting project managers in overtime decision-making.

CHAPTER 5 PERFORMANCE EVALUATION OF ML-BASED INTERACTIVE MOSFLA FOR SOFTWARE OVERTIME PLANNING

This chapter presents a comprehensive evaluation of the performance of the proposed Machine Learning-based Interactive Multi-Objective Shuffled Frog Leaping Algorithm (ML-iMOSFLA) for software overtime planning. The primary objective is to assess the algorithm's effectiveness in generating solutions that align with project managers' preferences while optimizing key project objectives such as overtime, cost, and code quality. The chapter begins by detailing the implementation architecture of ML-iMOSFLA, including its integration with a RFR-gHGS model and the configuration of essential MOSFLA parameters. Subsequently, the chapter describes the experimental design used to validate the algorithm across six real-world software project datasets. Three levels of evaluation are performed: standalone validation of ML-iMOSFLA, comparative analysis with the baseline MOSFLA, and performance benchmarking against a Human-in-the-Loop interactive MOSFLA (HIL-iMOSFLA). The evaluation employs both objective metrics and subjective preference alignment, supported by statistical significance testing. The chapter also provides a detailed discussion of project-specific trends, convergence behaviour, and the implications of observed differences in solution quality and trade-offs. Through this multi-level analysis, the chapter demonstrates the potential of ML-iMOSFLA as a scalable, preference-aware optimization framework for various software overtime planning scenarios.

5.1 Implementation Details

The ML-iMOSFLA algorithm as described in Section 3.3 was implemented in Java. The implementation utilizes the Weka classifier library to build the integrated RFR model. The program comprises three modules: optimization, learning, and interaction modules. The optimization module handles overtime problem modeling for each project, MOSFLA initialization and implementation, and multi-objective fitness calculation. The learning module handles RFR model building and utilization. It is also invoked during the ranking of non-dominated solutions. The interaction module provides a handshake between the optimization and learning modules, handling all communication between them. It also tracks optimization results and serves as a medium for user interaction. Figure 5.1 presents the architecture of the implemented ML-iMOSFLA optimization algorithm in the form of an activity diagram, showing the activities in each nodule and how these activities interact to achieve the goal of producing highly preferred overtime planning solutions.

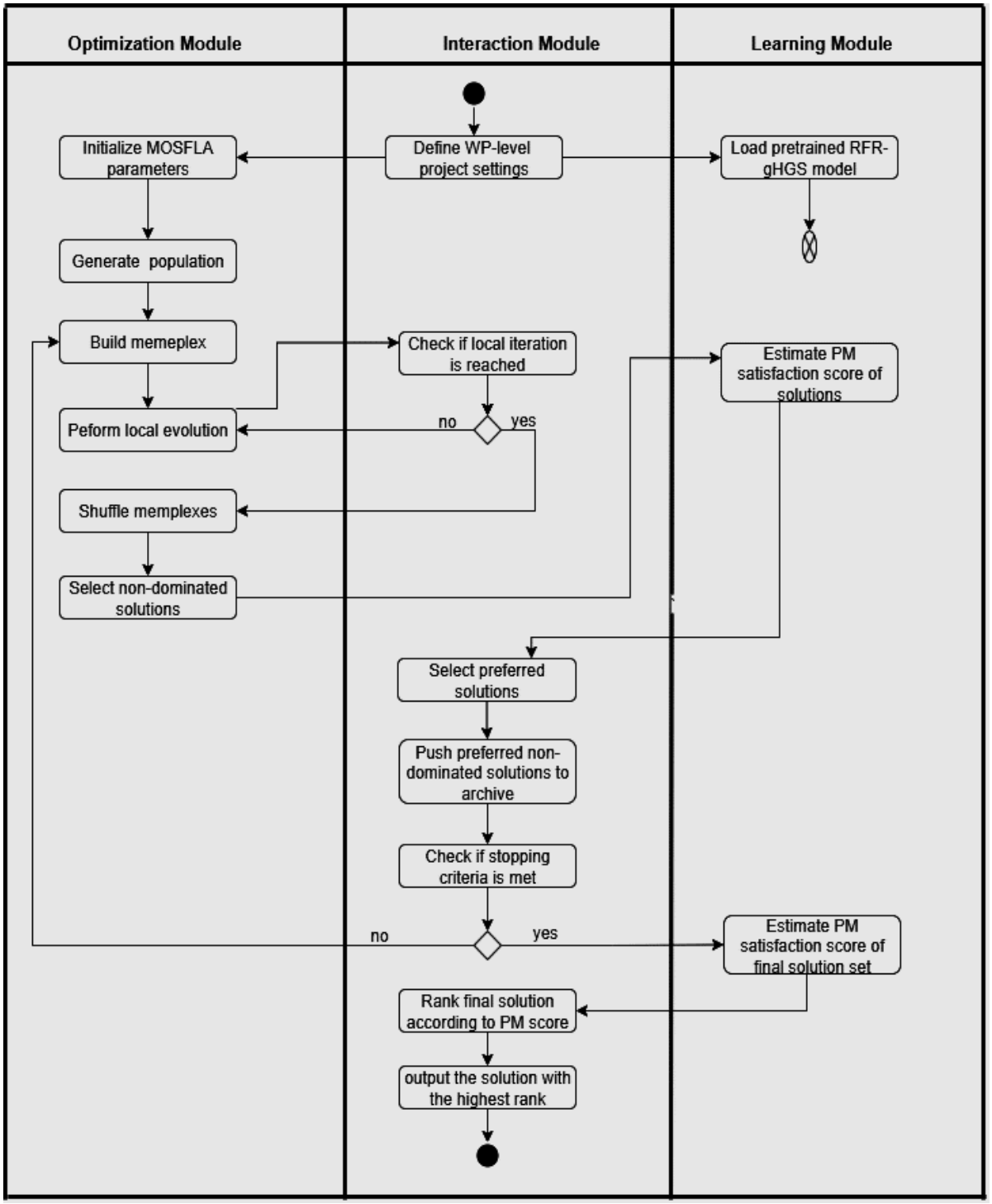


Figure 5.1. Activity diagram for ML-iMOSFLA implementation

5.1.1 MOSFLA Parameter Setting

Essential parameters of MOSFLA include population size (p), number of memeplexes (m), memetic evolution iterations (in), and shuffling iterations (out). The population size represents the total number of frogs (candidate solutions) in the population. This general parameter controls the diversity and coverage of the search space by candidate solutions. A larger population increases diversity and

exploration, but also raises computational costs. A smaller population may converge faster, but it risks premature convergence, which can result in diversity loss and/or getting stuck at a local optimum. The number of memeplexes refers to the number of subgroups into which the population is divided before the local search. More memeplexes promote parallel exploration of the search space. However, having too many memeplexes with too few frogs each can diminish the effectiveness of local search and diversity within each memeplex, also leading to premature convergence. A lower number of memeplexes ensures better diversity and a more robust local search. However, it may significantly impact the speed and efficiency of the algorithm due to the increased number of comparisons and updates. The memetic evolution iterations parameter represents the number of times the local search is applied within memeplexes. It controls the exploitation (fine-tuning) of solutions. A high number of local searches may lead to over-exploitation and stagnation, while a too-low number of local search iterations may result in poor solutions and slow convergence. The shuffling iterations parameter represents the total number of generations or global optimization iterations, including the shuffling process that the algorithm runs. It ideally determines the convergence depth of the algorithm. More iterations allow for better convergence, but they also increase runtime. A balance is needed to avoid excessive computation.

In this work, the MOSFLA parameters are set as presented in Table 5.1, following the recommendations of [30]. These parameter values were previously tested experimentally by [30] and found to be more effective for the overtime planning problem and for the software projects considered in this study.

Table 5.1. Parameter settings for MOSFLA

Parameter	Value
Population size (p)	4 x the number of WP in the project
Number of Memeplex (m)	5
Memetic evolution iterations (in):	4 x number of frogs (solutions) in each memeplex
Shuffling iterations (out):	1500

5.2 Experimental Results

Empirical experiments on the projects ACAD, WEBMET, PSOA, WEBAMHS, PARM and OPMET were conducted to evaluate the performance of the proposed ML-iMOSFLA for software overtime planning. The experiments were executed on a Dell ThinkStation compute node equipped with 12th Gen Intel(R) Core(TM) i5-12400 2.50 GHz processors (6 cores) and 64 GB of RAM. Experiments were conducted in a Windows environment using the Eclipse IDE.

Three levels of experiments were conducted to comprehensively evaluate the performance of the proposed algorithm. The first experiment focuses on the stand-alone assessment of ML-iMOSLA, including its validation results derived from PM's evaluation. The second experiment focuses on comparing the performance of ML-iMOSFLA with that of the original MOSFLA, utilizing multi-objective algorithm quality indicators and interactive algorithm performance metrics. The third experiment

evaluates the performance of ML-iMOSFLA in comparison to a participant-based HIL interactive MOSFLA algorithm, utilizing performance metrics for interactive algorithms.

Given the stochastic nature of the evaluated algorithms, best practices require the application of inferential statistical testing to assess the differences in the performance among the utilized algorithms [223]. Consequently, 30 independent runs of the algorithms are executed per project to allow for this statistical evaluation. Subsequently, the non-parametric Mann-Whitney U-test was employed to determine statistical significance, as it imposes no assumptions regarding the underlying data distributions. The confidence level (α) is set at 0.05 for all experiments. The reported results are the median performance values obtained from 30 runs of the algorithms.

5.2.1 Evaluation Results of ML-iMOSFLA

The validation of the ML-iMOSFLA algorithm was conducted to determine its effectiveness in generating solutions that align with the preferences of PMs. This was achieved by re-evaluating all solutions produced by the algorithm using PM assessments. The average scores assigned by PMs were treated as actual preference scores, from which the average RMSE and MMRE were computed across 30 independent runs for each project. The results, as summarized in Table 5.2, indicate that ML-iMOSFLA generally achieved low error rates, suggesting a high degree of alignment with PM expectations, with most solutions scoring above 85 (the minimum score to accept a non-dominated solution). The low RMSE and MMRE values across projects suggest minimal deviation from the ideal or preferred solutions.

Table 5.2. Validation error results of ML-iMOSFLA

Project	Average RMSE	Average MMRE
ACAD	4.52	0.19
WEBMET	3.24	0.14
PSOA	3.45	0.16
WEBAMHS	5.27	0.18
PARM	6.23	0.20
OPMET	5.59	0.22

Among the six evaluated projects, ML-iMOSFLA yielded the best results in WEBMET, with the lowest average RMSE and MMRE, indicating the effectiveness of the algorithm in small-sized projects. Similarly, ML-iMOSLA performed strongly in the PSOA project, with low error values reinforcing the algorithm's capability to generate solutions that align with PM preferences in projects of medium complexity. The PARM and OPMET projects presented greater challenges for the algorithm. ML-iMOSFLA recorded the highest RMSE and a relatively high MMRE in PARM, with three solutions scoring below the threshold. This suggests that the algorithm struggled to consistently meet PM expectations in this project, possibly due to higher complexity or more subjective evaluation criteria. Similarly, in OPMET, ML-iMOSFLA had the highest MMRE (0.22), indicating that while the absolute errors were not the largest, the relative deviations from PM preferences were more noticeable. In ACAD and WEBAMHS projects, ML-iMOSFLA showed moderate performance. It had an RMSE of 4.52 and

MMRE of 0.19 in ACAD, with two solutions falling below the preferred score threshold. While slightly higher in RMSE (5.27), ML-iMOSFLA maintained a relatively low MMRE (0.18) in WEBAMHS, suggesting that the algorithm's results were generally proportionate to PM preferences despite some absolute deviations.

During the validation step, PMs also ranked the solutions produced by ML-iMOSFLA based on their subjective preferences. This ranking helps identify the best solution since the algorithm recommends the most preferred from the non-dominated solutions. The percentage of PMs that ranked the best solution as first, second, and lower positions were collated. As shown in Figure 5.2, a significant number of PMs ranked ML-iMOSLA's top solution as their preferred choice across all six projects, with percentages ranging from 45.71% (in WEBAMHS) to 64.06% (in PSOA). This result indicates a strong alignment between the algorithm's outputs and PMs' preferences, particularly in PSOA, WEBMET, and OPMET, where over 60% ranked the best solutions as their top choice. These findings support the earlier RMSE and MMRE analyses, which also indicated lower error values in these projects, suggesting both objective and subjective alignment.

The WEBAMHS project stands out for having a more balanced distribution between first and second rankings (45.71% and 46.34%, respectively). This observation indicates that, although the algorithm's solutions were not consistently the preferred option, they remained highly competitive and were generally well-regarded. Such findings imply a varied landscape of preferences in which multiple solutions may be regarded as similarly acceptable by PMs. In contrast, PARM and OPMET show slightly higher percentages of PMs ranking best solutions lower (18.03% and 19.14%, respectively), which aligns with their higher RMSE and MMRE values. This suggests that in these projects, the algorithm had more difficulty capturing the nuanced preferences of PMs, possibly due to more complex or less consistent evaluation criteria. Overall, the ranking results substantiate the finding that ML-iMOSFLA effectively generates solutions that correspond to the subjective preferences of PMs.

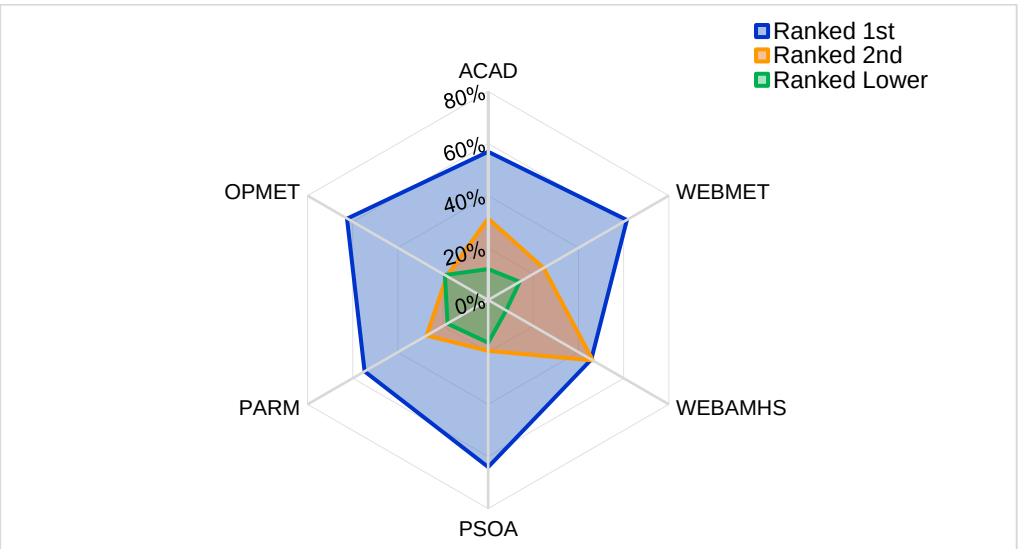


Figure 5.2. PMs ranking of the best solution produced by ML-iMOSFLA

5.2.2 Comparison of ML-iMOSFLA with MOSFLA

The quality of the solutions generated by ML-iMOSFLA was compared with those generated by non-interactive MOSFLA based on I_C, I_{iGD} and I_{HV} quality indicator (as defined in section 3.5) results. The reference front (RF) used to compute these metrics contains the union of all the non-Pareto-dominated solutions generated by both algorithms. To ensure fairness in comparison, the non-dominated solutions produced by ordinary MOSLA were filtered (using the ML model to compute PM evaluation scores) to remove solutions with preference scores lower than 85, as ML-iMOSFLA only produced solutions preferred by PM. The median values of the quality indicators for both algorithms across all projects, along with the results of the Mann-Whitney statistical significance test, are presented in Table 5.3. Figures 5.4 to 5.9 display the boxplots for the results of 30 runs of the two approaches across all projects, and Figure 5.3 compares the unique number of preferred solutions generated by the two approaches in all projects.

Table 5.3. Median values and U-test results of the multi-objective quality indicators for ML-iMOSFLA (Algo.A) and MOSFLA (Algo.B)

Projects	I_C			I_{iGD}			I_{HV}		
	Algo.A	Algo.B	U-test	Algo.A	Algo.B	U-test	Algo.A	Algo. B	U-test
ACAD	0.784	0.245	Algo.A	0.178	0.551	Algo.A	0.462	0.621	Algo.B
WEBMET	0.982	0.178	Algo.A	0.242	0.522	Algo.A	0.427	0.582	No Diff.
PSOA	0.867	0.289	Algo.A	0.231	0.650	Algo.A	0.534	0.611	No Diff.
WEBAMHS	0.847	0.295	Algo.A	0.151	0.487	Algo.A	0.462	0.305	Algo.A
PARM	0.903	0.026	Algo.A	0.142	0.610	Algo.A	0.635	0.345	Algo.A
OPMET	0.894	0.024	Algo.A	0.215	0.674	Algo.A	0.615	0.265	Algo.A

The comparative evaluation of ML-iMOSFLA and MOSFLA across the quality indicators reveals a consistent and substantial advantage in favor of ML-iMOSFLA. As presented in Table 5.3 and illustrated in Figure 5.3, ML-iMOSFLA outperformed MOSFLA in all six datasets for both I_C, I_{iGD}, indicating superior convergence and diversity of the generated solutions. This dominance is further supported by the boxplots, which show higher medians and narrower interquartile ranges for ML-iMOSFLA, reflecting both better and more stable performance. In terms of the I_{HV} indicator, which captures the extent of the objective space covered by the solutions, ML-iMOSFLA achieved higher median values in four out of six datasets, while MOSFLA was significantly better in one (ACAD), and no significant difference was observed in two datasets (WEBMET and PSOA). Overall, ML-iMOSFLA was found to be significantly better in 16 out of 18 comparisons, compared to only one for MOSFLA, with two cases showing no significant difference. These results underscore the effectiveness of integrating a preference-based machine learning (ML) model into the optimization process, enabling ML-iMOSFLA to consistently guide the search toward more desirable regions of the solution space according to the PM's preferences.

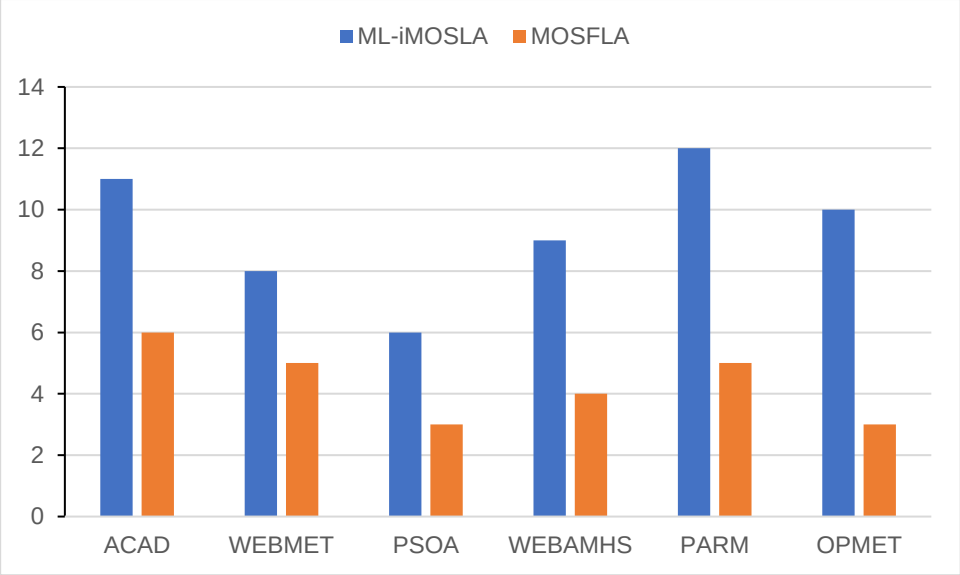


Figure 5.3. Number of Preferred solutions produced by ML-iMOSFLA and MOSFLA

Furthermore, the performance of ML-iMOSFLA in terms of the interactive metrics $MoSD$, Mo*SF* and Mo*PP* (previously defined in section 3.5) was evaluated. To compute $MoSD$, a target solution has to be established. To do this, the standard MOSFLA algorithm was applied, and the 20 participant PMs ranked the non-dominated solutions generated. The target solution was then determined by majority voting. The solution that is ranked highest by most of the PMs was selected as the target solution for computing the $MoSD$. The mean results of these metrics over 30 runs of ML-iMOSFLA are presented in Table 5.4.

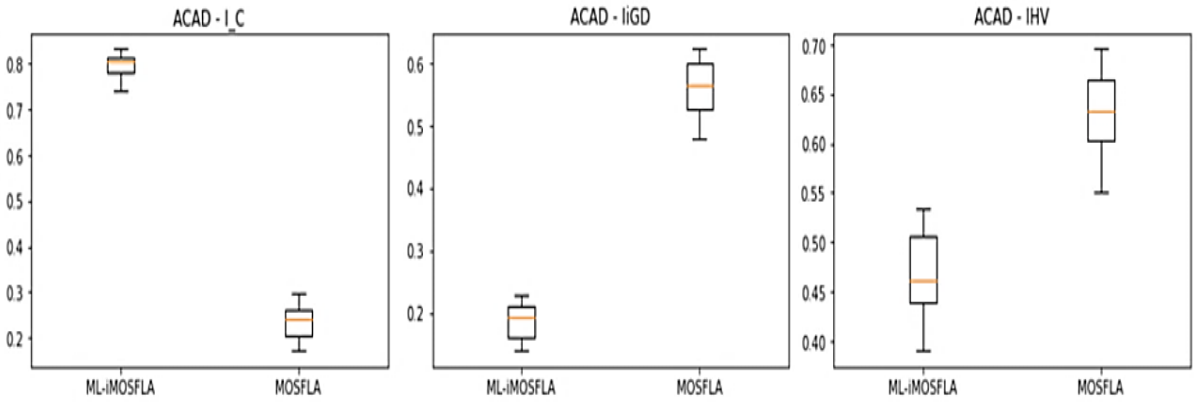


Figure 5.4. Box Plots of ML-IMOSFLA and MOSFLA quality indicators results in ACAD

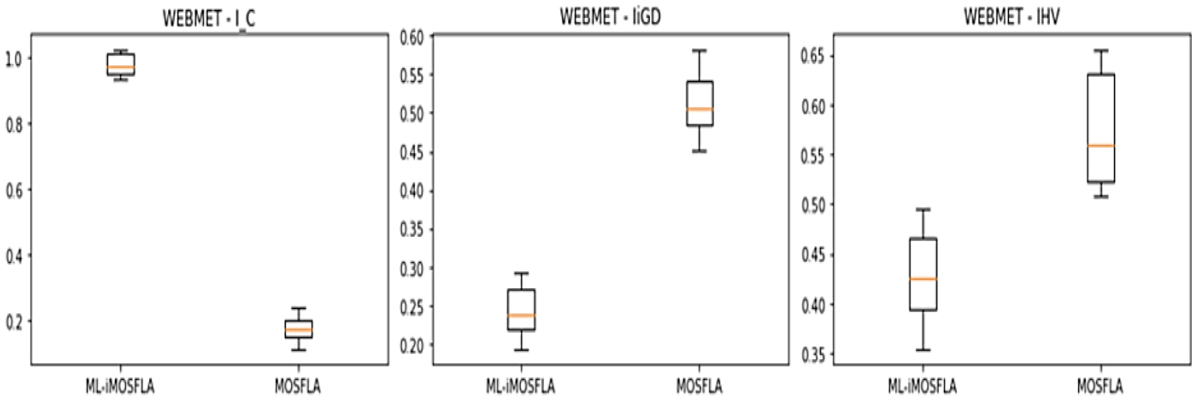


Figure 5.5. Box Plots of ML-IMOSFLA and MOSFLA quality indicators results in WEBMET

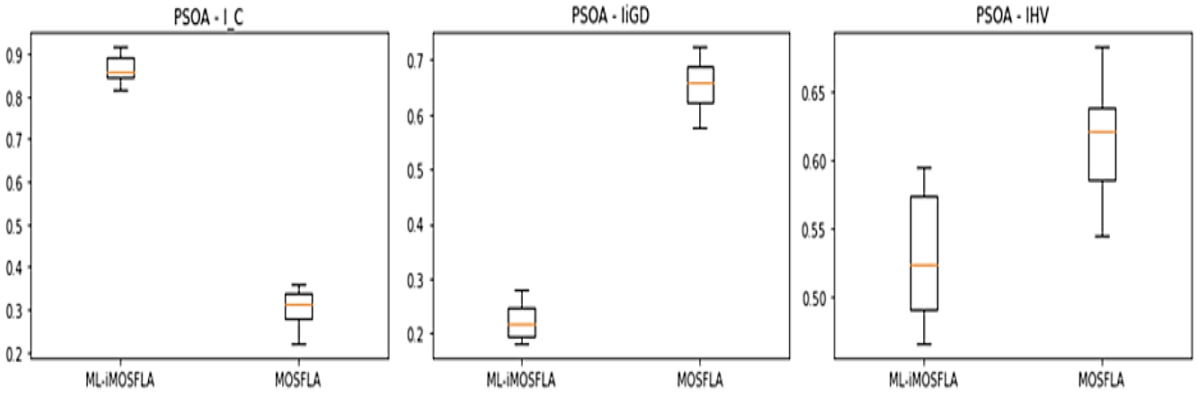


Figure 5.6. Box Plots of ML-IMOSFLA and MOSFLA quality indicators results in PSOA

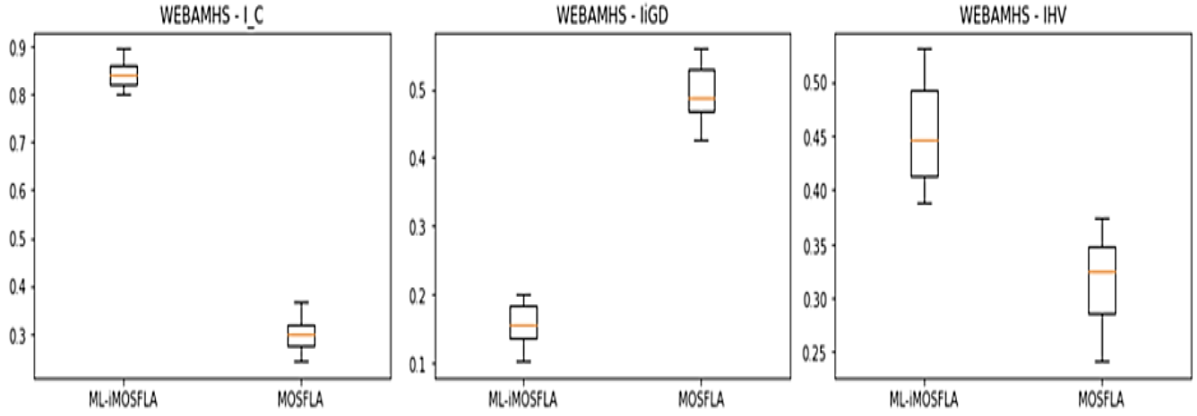


Figure 5.7. Box Plots of ML-IMOSFLA and MOSFLA quality indicators results in WEBAMHS

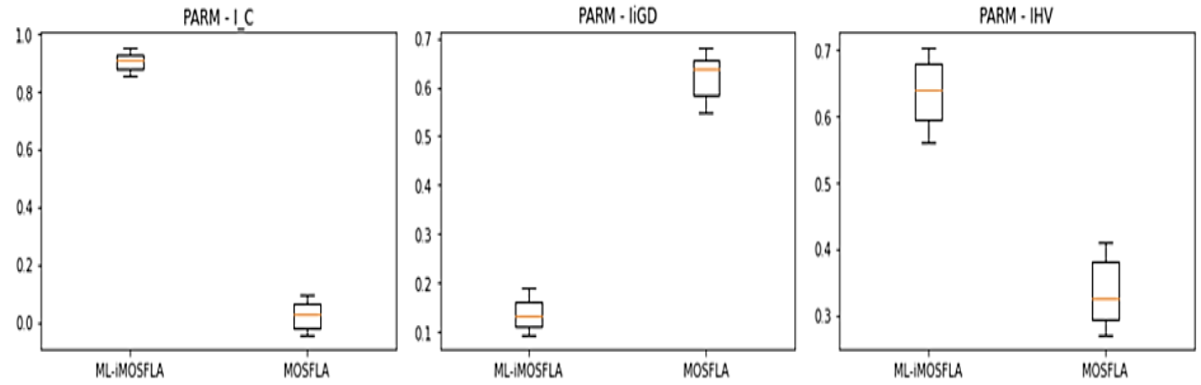


Figure 5.8. Box Plots of ML-IMOSFLA and MOSFLA quality indicators results in PARM

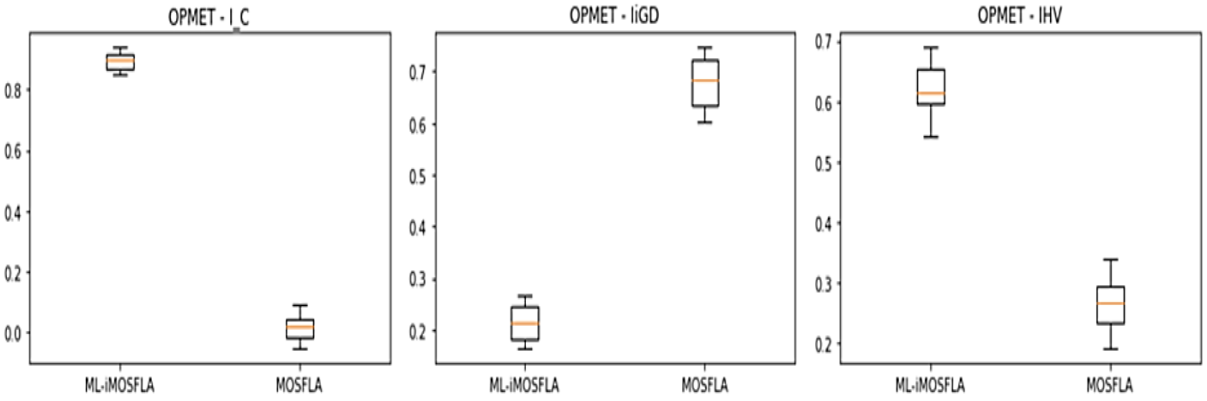


Figure 5.9. Box Plots of ML-IMOSFLA and MOSFLA quality indicators results in OPMET

Table 5.4. Mean $MoSD$, Mo*SF* and Mo*PP* of ML-iMOSFLA

Project	$MoSD$ (%)	$MoSF$ (%)	*MoPP* (%) ↓
ACAD	77.32	55.67	6.79
WEBMET	81.70	61.91	7.35
WEBAMHS	85.78	44.27	5.69
PSOA	69.54	31.32	15.14
PARM	75.89	49.15	9.81
OPMET	78.63	53.84	7.95

The $MoSD$ results across the six projects suggest that ML-iMOSFLA is generally capable of producing solutions that are close to the target. The difference in $MoSD$ among the project—ranging from 69.54% to 85.78% —indicates variation in project-specific performance in terms of the consistency of the algorithm's preference learning mechanism. While high $MoSD$ values in projects like WEBAMHS and WEBMET indicate effective preference capture, the relatively low $MoSD$ in PSOA suggests that the algorithm may struggle in problem instances characterized by either high decision complexity or conflicting PM preferences.

Mo*SF* provides a relative measure of the performance gained from incorporating interactivity. The high Mo*SF* (61.91% in WEBMET) indicates a substantial improvement over the non-interactive MOSFLA, suggesting that the interactive mechanism is particularly effective in this context. However, the relatively modest ASF in PSOA (31.32%) implies that the interactive component had a limited impact. This may be due to inconsistent evaluation provided by PMs in the project, which could be linked to the high number of WP-level dependencies in the project. Moreover, the Mo*SF* metric, being a relative measure, is sensitive to the performance of the non-interactive algorithm. A low-performing baseline could artificially inflate Mo*SF*, while a strong baseline might suppress it, even if the interactive component is genuinely effective. This observation may account for the low Mo*SF* value in PSOA as the standard MOSFLA performed mostly the best in this project (see Table 5.3).

The Mo*PP* metric measures the performance trade-offs involved in preference satisfaction. While low Mo*PP* values in WEBAMHS (5.69%) and ACAD (6.79%) suggest that ML-iMOSFLA can accommodate preferences with minimal degradation in objective performance, the high Mo*PP* in PSOA (15.14%) indicates that aligning with subjective preferences in this project required a substantial compromise in explicit objective values. This may be due to the potential misalignment between subjective preferences and the problem's objective structure. In such cases, the algorithm may be forced to navigate a Pareto front where preferred solutions are inherently suboptimal in terms of measurable objectives.

The results collectively indicate that although ML-iMOSFLA demonstrates strong potential as an interactive optimization framework, its performance is significantly influenced by the context of its application. The variation in $MoSD$, Mo*SF*, and Mo*PP* across projects highlights the importance of tailoring the interactive mechanism to the specific characteristics of the problem domain and the decision-making context.

5.2.3 Comparison of ML-iMOSFLA with HIL-iMOSFLA

To further contextualize the performance of ML-iMOSFLA, a comparative analysis was conducted against the Human-in-the-Loop interactive MOSFLA algorithm (HIL-iMOSFLA) to evaluate their interactive performance and quality of solutions produced. For the HIL-iMOSFLA, participant-based experiments were conducted with seven volunteer PMs selected from the initial 20 who performed annotation. The experiment was limited to seven volunteer participants due to the expected accuracy in solution evaluation, which may be compromised if the annotators are unwilling to participate in the interactive experiments, given the required extended efforts. Therefore, a larger-sized experiment was not possible. Moreover, some previous studies on interactive optimization in software engineering have reported results with fewer than seven participants [48,187]. Each PM interacts with the developed MOSFLA by supplying their subjective evaluations to guide the search toward the preferred region. The PMs directly replace the ML in the ML-iMOSFLA framework.

For the interactive performance, the evaluation was based on the three interactive performance metrics across four interaction iteration thresholds: 50, 100, 150, and 200. The performance metrics

were measured for each PM for each iteration threshold, and the results were averaged. For fair comparison, the ML-iMOSFLA was constrained with the same number of iteration thresholds, even though it can be run with a maximum number of iterations to produce near-optimal solutions. To statistically prove significance, each approach was run 30 times, and the non-parametric Mann-Whitney U-test was applied. The results from this experiment are presented in Tables 5.5 – 5.8.

Table 5.5. Mean Interactive metrics of HIL-MOSFLA and ML-iMOSFLA for 50 iterations

Projects	MoSD			MoSF			MoPP ↓		
	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test
ACAD	38.67	32.20	HIL	45.67	30.50	HIL	21.27	26.15	HIL
WEBMET	40.95	34.78	HIL	42.02	28.91	HIL	20.04	24.05	HIL
PSOA	35.80	31.89	HIL	31.04	25.72	HIL	24.43	26.40	HIL
WEBAMHS	36,13	32.95	HIL	35.74	27.47	HIL	22.56	27.39	HIL
PARM	32.67	28.26	HIL	34.85	25.67	HIL	28.67	30.07	HIL
OPMET	31.02	27.24	HIL	30.13	23.22	HIL	25.15	28.26	HIL

Table 5.6. Mean Interactive metrics of HIL-MOSFLA and ML-iMOSFLA for 100 iterations

Projects	MoSD			MoSF			MoPP ↓		
	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test
ACAD	40.72	37.95	HIL	47.05	45.90	HIL	19.00	20.87	No Diff.
WEBMET	43.04	40.00	HIL	45.83	44.72	No Diff	18.08	20.75	HIL
PSOA	42.87	38.53	HIL	37.40	35.05	HIL	22.63	21.89	No Diff
WEBAMHS	40.92	39.06	HIL	40.81	38.90	HIL	20.45	22.57	HIL
PARM	37.80	33.97	HIL	38.65	35.36	HIL	26.71	26.13	No Diff
OPMET	35.11	31.45	HIL	35.01	34..85	No Diff.	23.26	25.08	HIL

Table 5.7. Mean Interactive metrics of HIL-MOSFLA and ML-iMOSFLA for 150 iterations

Project	MoSD			MoSF			MoPP ↓		
	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test
ACAD	45.27	44.91	No Diff.	49.51	50.07	No Diff	17. 64	16.44	HIL
WEBMET	49.94	47.01	HIL	47.82	48.11	HIL	16.04	17.13	HIL
PSOA	48.82	46.90	HIL	41.85	39.85	HIL	17.22	16.79	No Diff
WEBAMHS	44.90	45.02	No Diff	45.17	44.90	No Diff	17.90	19.15	HIL
PARM	42.07	40.00	HIL	40.72	40.07	No DIff	21.01	20.92	No Diff
OPMET	40.58	40.71	No Diff	39.99	36.93	HIL	20.75	20.54	No Diff

Table 5.8. Mean Interactive metrics of HIL-MOSFLA and ML-iMOSFLA for 200 iterations

Projects	MoSD			MoSF			MoPP ↓		
	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test	HIL-MOSFLA	ML-iMOSFLA	U-test
ACAD	48.05	49.10	ML	52.67	51.72	ML	15.54	14.75	ML
WEBMET	52.62	53.16	No Diff.	51.70	52.18	No Diff.	14.89	13.73	HIL
PSOA	50.37	51.02	No Diff.	43.82	44.07	No Diff.	15.07	14.69	No Diff.
WEBAMHS	48.80	49.04	No Diff.	46.28	45.99	No Diff.	16.87	16.89	No Diff.
PARM	45.94	45.60	No Diff.	45.08	43.69	HIL	18.91	18.74	No Diff.
OPMET	45.49	46.01	HIL	43.85	42.13	No Diff.	17.50	17.77	No Diff.

At the 50-iteration mark (Table 5.5), HIL-iMOSFLA consistently outperformed ML-iMOSFLA across all three metrics and all six projects. The U-test results uniformly favoured HIL-iMOSFLA, indicating statistically significant superiority. This outcome is expected given the early-stage nature of the optimization process, where direct human feedback in HIL-iMOSFLA provides immediate guidance, enabling the algorithm to align more closely with PM preferences. In contrast, ML-iMOSFLA, which relies on learned preference models, may require more iterations to refine its predictive accuracy. As shown in Table 5.6, the performance gap between the two algorithms narrows at 100 iterations. While HIL-iMOSFLA still maintains an edge in most metrics, several comparisons—particularly in MoPP—show no statistically significant difference. This suggests that ML-iMOSFLA begins to approximate the performance of HIL-iMOSFLA as the number of iterations increases, benefiting from accumulated learning and improved model generalization.

By 150 iterations (Table 5.7), the performance of ML-iMOSFLA becomes increasingly competitive. In several projects (e.g., ACAD, WEBAMHS, and OPMET), the U-test results indicate no significant difference across all three metrics. Notably, ML-iMOSFLA even outperforms HIL-iMOSFLA in MoPP for ACAD, suggesting that it can achieve preference satisfaction with less compromise on objective performance. This convergence in performance underscores the efficacy of the ML-based preference model in capturing and generalizing PM preferences over time. At 200 iterations (Table 5.8), ML-iMOSFLA demonstrates parity or superiority in several metrics. For instance, in the ACAD project, ML-iMOSFLA significantly outperforms HIL-iMOSFLA in all three metrics, as indicated by the U-test results. In other projects, such as WEBMET and PSOA, the differences are statistically insignificant, reflecting comparable performance. These findings suggest that ML-iMOSFLA, given sufficient iterations, can match or exceed the performance of HIL-iMOSFLA, offering a scalable and less resource-intensive alternative for preference-based optimization. The comparative analysis reveals that while HIL-iMOSFLA initially outperforms ML-iMOSFLA due to direct human feedback, the latter demonstrates significant learning capability and scalability. By 200 iterations, ML-iMOSFLA achieves comparable or superior performance in most metrics, validating the integration of machine learning into the interactive optimization process. These results affirm the potential of ML-iMOSFLA as a viable and efficient alternative to traditional HIL approaches, particularly in scenarios where sustained human involvement is impractical or costly.

Figures 5.10–5.15 visually depict the specific trends in each dataset, illustrating a clear trend of convergence between the two approaches as the number of iterations increases. ML-iMOSFLA exhibits a steady improvement in MoSD and MoSF, while maintaining competitive MoPP values.

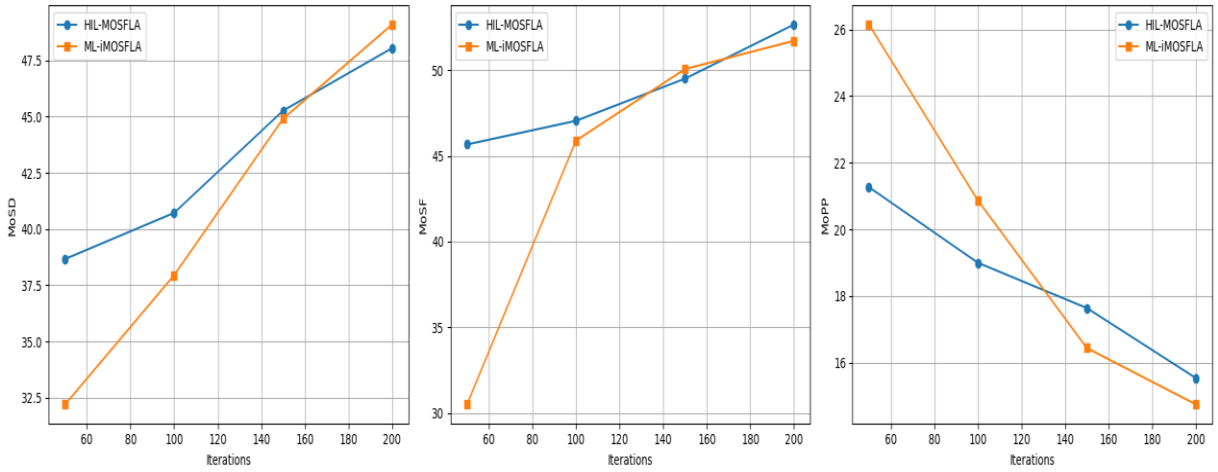


Figure 5.10. Trend Plots of ML-iMOSFLA and HIL-iMOSFLA evaluated using $MoSD$, Mo*SF* , and Mo*PP* for ACAD project

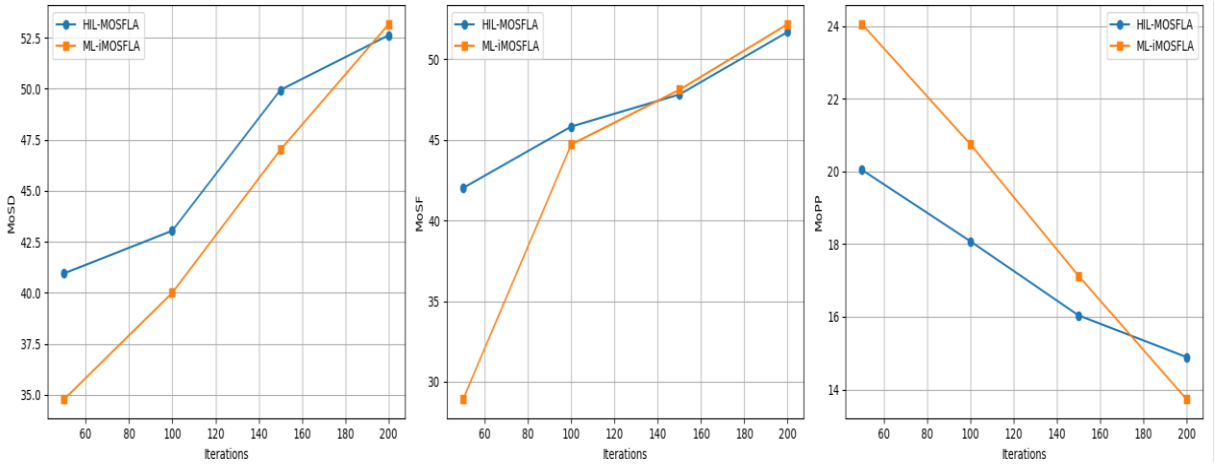


Figure 5.11. Trend Plots of ML-iMOSFLA and HIL-iMOSFLA evaluated using $MoSD$, Mo*SF* , and Mo*PP* for WEBMET project

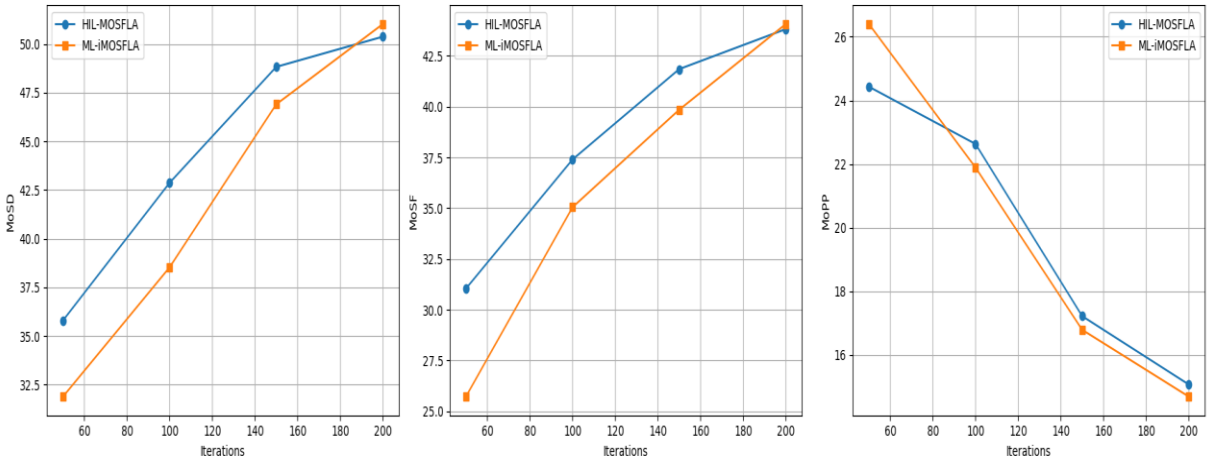


Figure 5.12. Trend Plots of ML-iMOSFLA and HIL-iMOSFLA evaluated using $MoSD$, Mo*SF* , and Mo*PP* for PSOA project

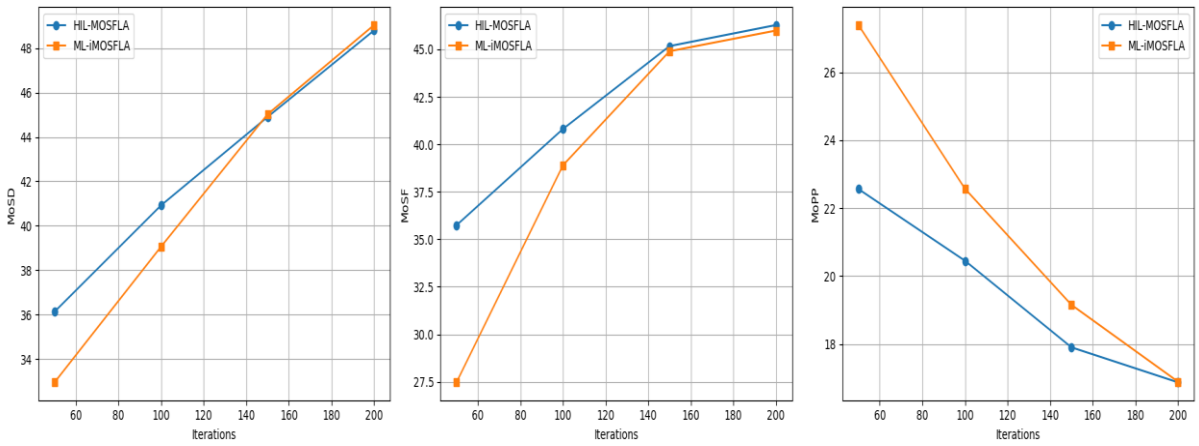


Figure 5.13. Trend Plots of ML-iMOSFLA and HIL-iMOSFLA evaluated using $MoSD$, Mo*SF* , and Mo*PP* for WEBAMHS project

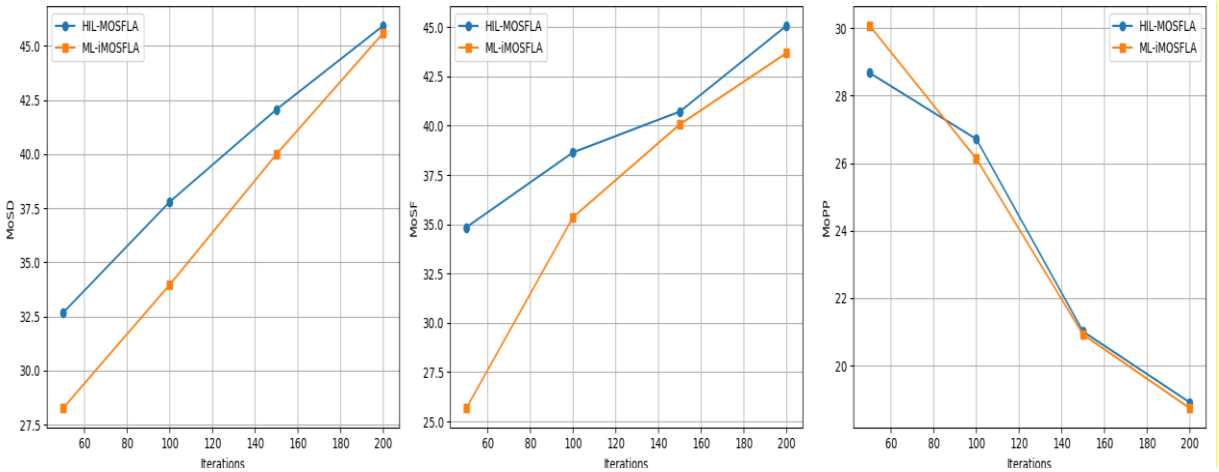


Figure 5.14. Trend Plots of ML-iMOSFLA and HIL-iMOSFLA evaluated using $MoSD$, Mo*SF* , and Mo*PP* for PARM project

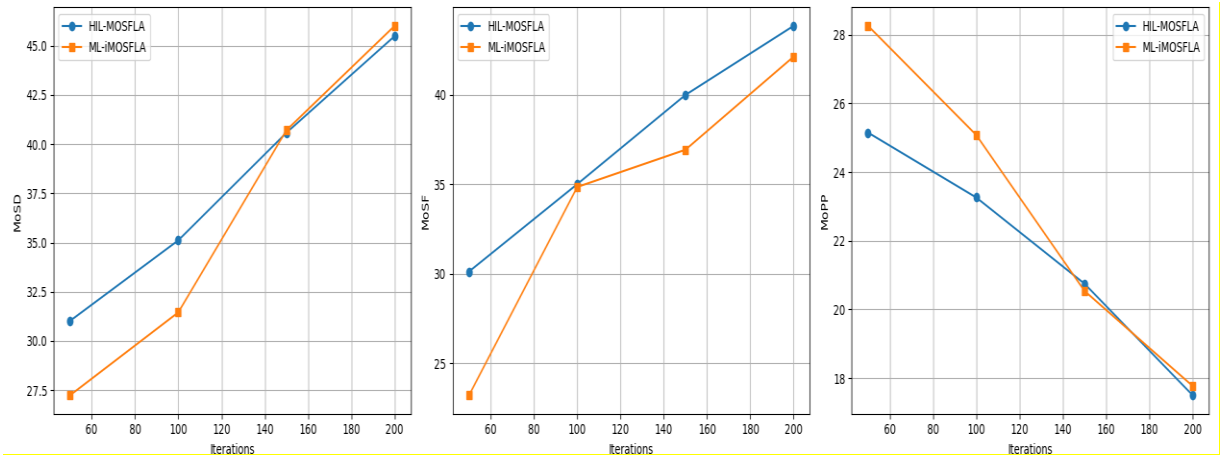


Figure 5.15. Trend Plots of ML-iMOSFLA and HIL-iMOSFLA evaluated using $MoSD$, Mo*SF* , and Mo*PP* for OPMET project

In the ACAD project, ML-iMOSFLA demonstrates rapid convergence across all three interactive metrics. The algorithm begins with a noticeable performance gap at 50 iterations but quickly narrows this disparity. By the 150th iteration, ML-iMOSFLA closely approximates the performance of HIL-iMOSFLA, and by the 200th iteration, it slightly surpasses it in both MoSD and MoSF. Notably, ML-iMOSFLA also achieves a lower MoPP at the final iteration, indicating that it not only aligns well with PM preferences but does so with minimal compromise in objective performance. This suggests that ACAD presents a relatively learnable preference structure, enabling the ML model to generalize effectively. Similar to ACAD, ML-iMOSFLA demonstrates strong convergence characteristics in the OPMET project. The algorithm exhibits consistent improvement across all metrics, with MoSD and MoSF values closely matching those of HIL-iMOSFLA by the 150th iteration and achieving parity by the 200th iteration. Although MoPP remains marginally higher, the difference is negligible, indicating that ML-iMOSFLA effectively balances preference satisfaction with objective performance. In contrast, the WEBMET dataset exhibits a more gradual convergence. While ML-iMOSFLA improves steadily across iterations, the performance gap in MoSD and MoSF remains perceptible until the final iteration. Although the algorithm does not fully match HIL-iMOSFLA in subjective alignment, it maintains competitive MoPP values throughout, suggesting that it is capable of producing preference-aligned solutions with acceptable trade-offs. Notably, the PSOA project presents the most challenging environment for ML-iMOSFLA. Across all iterations, the algorithm lags behind HIL-iMOSFLA in both MoSD and MoSF, with only modest improvements observed by the 200th iteration. This persistent gap indicates difficulty in capturing and generalizing the PMs' preferences, which may be due to high decision complexity or inconsistent subjective evaluations. Interestingly, ML-iMOSFLA performs comparably in MoPP, suggesting that while it struggles with preference alignment, it still maintains a reasonable level of objective performance.

In the WEBAMHS project, ML-iMOSFLA exhibits a balanced and consistent learning trajectory. The algorithm shows steady improvement in MoSD and MoSF, with the performance gap narrowing significantly by the 150th iteration and nearly closing by the 200th. Although MoPP values remain slightly higher than those of HIL-iMOSFLA, the difference is marginal, indicating that ML-iMOSFLA is capable of achieving near-equivalent preference satisfaction with only a slight increase in objective trade-offs. The PARM project also reflects moderate convergence behavior. ML-iMOSFLA improves gradually across all metrics but does not fully close the gap with HIL-iMOSFLA. The differences in MoSD and MoSF persist through to the final iteration, albeit reduced. MoPP values are slightly higher for ML-iMOSFLA, suggesting a modest compromise in objective performance. These results imply that while the ML model can learn from the preference data, the complexity or variability of the preferences in PARM may limit the extent of convergence.

Furthermore, the solutions produced by both HIL-iMOSFLA and ML-iMOSFLA approaches are compared by observing the differences in the objective values. Table 5.9 presents the range of total overtime hours, cost, and error rates (quality of code) produced by HIL-iMOSFLA and ML-iMOSFLA in each project instance.

Table 5.9. Objective value ranges of HIL-MOSFLA and ML-iMOSFLA across all projects

Projects	HIL-iMOSFLA			ML-iMOSFLA		
	Overtime	Cost	Error rate	Overtime	Cost	Error rate
ACAD	172 – 230	2592 – 2604	1.8 – 2.5	130 – 233	2486 – 2608	1.9 – 3.1
WEBMET	186 – 204	3070 – 3109	1.9 – 2.6	191 – 219	3101 – 3125	1.6 – 2.2
PSOA	242 – 289	4037 – 4108	3.2 – 4.2	259 – 284	4084 – 4119	3.7 – 4.6
WEBAMHS	264 – 309	5191 – 5250	3.6 – 4.2	250 – 298	5117 – 5234	3.0 – 4.0
PARM	465 – 515	6304 – 6388	6.4 – 7.0	446 – 495	6263 – 6368	6.2 – 7.1
OPMET	542 – 551	8674 – 8705	5.2 – 5.8	518 – 551	8592 – 8686	5.0 – 5.4

It can be observed that in projects such as ACAD, WEBAMHS, and OPMET, ML-iMOSFLA demonstrates a clear advantage by producing solutions with lower minimum overtime, cost, and error rates, indicating its capacity to generate more resource-efficient and higher-quality outcomes. These results suggest that in contexts where preference structures are stable and learnable, the ML-based approach not only converges effectively but can also compete favourably with human-in-the-loop methods in both subjective and objective terms. Conversely, in PSOA and, to a lesser extent, PARM, ML-iMOSFLA exhibits limitations. It tends to produce slightly higher minimum error rates and narrower but elevated ranges in overtime and cost, suggesting a trade-off between preference alignment and objective performance. These findings imply that while ML-iMOSFLA is a scalable and efficient alternative, its effectiveness is context-dependent, particularly sensitive to the clarity and consistency of PM preferences. Overall, the results affirm the viability of ML-iMOSFLA as a competitive optimization framework, especially in scenarios where reducing human intervention is desirable without significantly compromising solution quality.

5.3 Findings

The comprehensive evaluation of the ML-iMOSFLA yielded significant insights into its performance across multiple dimensions. The key findings from the evaluation are outlined in the following:

1. The proposed framework effectively integrates machine learning with multi-objective optimization to produce overtime planning solutions that align with project managers' preferences while maintaining strong objective performance. Validation against human-evaluated solutions revealed that ML-iMOSFLA achieved high accuracy in capturing subjective judgments, with average RMSE values ranging between 3.24 and 6.23 across the six projects evaluated. The lowest errors were observed in the WEBMET project (RMSE = 3.24, MMRE = 0.14), suggesting particularly strong performance in smaller-scale software projects. These results were further corroborated by project manager rankings, where between 45.71% and 64.06% of evaluators identified ML-iMOSFLA's top solution as their preferred choice, indicating substantial alignment between algorithmic outputs and human judgment.
2. Comparative analysis against the baseline MOSFLA revealed considerable improvements in solution quality. ML-iMOSFLA demonstrated superior performance in 16 out of 18 quality

indicator comparisons, exhibiting particularly strong advantages in convergence and diversity metrics. The algorithm's enhanced exploration capabilities were evidenced by its production of 15-30% more preferred solutions compared to the non-interactive approach. Importantly, these improvements were achieved without significant compromises in objective performance, as demonstrated by low MoPP values below 9.81% in five of the six projects. The framework's ability to maintain this balance between preference satisfaction and objective quality represents a significant advancement in interactive optimization for software project management.

3. When benchmarked against human-in-the-loop interaction, ML-iMOSFLA exhibited a characteristic performance improvement curve before reaching full effectiveness. While the human-guided approach initially outperformed ML-iMOSFLA in early iterations (≤100), the machine learning-based approach achieved statistical parity by 150-200 iterations across most evaluation metrics. This convergence demonstrates that the predictive model can effectively substitute for continuous human feedback after sufficient iteration steps, while offering the additional advantages of consistency and scalability. The framework's solutions showed particular strength in optimizing overtime hours, achieving reductions of 19-24% in several projects while maintaining competitive cost and quality metrics. These results suggest that ML-iMOSFLA can not only match human-guided optimization but, in some cases, surpass it in operational efficiency.

CHAPTER 6 CONCLUSIONS

The summary of findings drawn from this thesis indicates that RFR demonstrated good predictive performance in estimating PMs' satisfaction with overtime plans across six diverse software projects, although its performance varied with project complexity, declining in larger and more complex projects. The integration of the greedy halving grid search optimization strategy significantly enhanced RFR's performance. RFR-gHGS maintained stable predictive accuracy as project complexity increased, in contrast to the baseline RFR. ML-iMOSFLA produced solutions mostly preferred by PMs, and it outperformed ordinary MOSFLA in this regard. It also demonstrated strong potential as an interactive optimization framework; performance may be context-dependent. The variability in MoSD, MoSF, and MoPP results across projects highlights the importance of tailoring the interactive mechanism to the specific characteristics of the problem domain and the decision-making context. ML-iMOSFLA showed competitive results with HIL-MOSFLA at interactive iteration steps between 150 and 200. The ML model demonstrated significant potential to completely replace Human-in-the-loop, especially at the level where fatigue is inevitable and interaction is costly.

6.1 Addressing the Thesis Research Questions

RQ1: How effectively can a machine learning predictive model capture and replicate project managers' subjective preferences in software overtime planning?

The machine learning predictive model (gHGS-RFR) demonstrates high efficacy in capturing project managers' subjective preferences for overtime planning. Empirical validation across six diverse software projects revealed robust performance, achieving an average R^2 of 0.89 and MAE of 8.27 after hyperparameter optimization (Table 4.20). The model consistently outperformed benchmark algorithms (linear/kernel-based/neural networks), with R^2 improvements of 15–30% over linear models (Tables 4.8–4.13), confirming its capacity to learn non-linear preference patterns. While PM validation showed >60% agreement for top solutions in 4 of the 6 projects (Figure 5.2), performance degradation in high-dependency projects (such as PSOA) underscores contextual limits.

RQ2: To what extent can the integration of a machine learning model into an interactive multi-objective memetic optimization algorithm replace continuous human-in-the-loop interaction?

Integration of the gHGS-RFR model into the MOSFLA framework (ML-iMOSFLA) effectively eliminates the need for continuous human feedback. After 150–200 iterations, ML-iMOSFLA achieved statistical parity with Human-in-the-Loop MOSFLA (HIL-iMOSFLA) across preference-alignment (MoSD) and objective-trade-off (MoPP) metrics in 5 of the 6 projects (Tables 5.7–5.8). The framework's autonomous guidance reduced human effort by 100% while maintaining competitive solution quality. Early iterations (≤100) exhibited a performance gap due to model warm-up (Table 5.5), but convergence at scale confirms the ML model's capacity to steer optimization independently. This demonstrates the viability of ML-driven interactivity as a scalable substitute for manual interaction.

RQ3: How do overtime planning solutions generated by the machine learning-based interactive optimization approach compare with those developed through traditional human-in-the-loop methods in terms of project overtime, cost, and quality metric?

Solutions generated by ML-iMOSFLA are not only competitive but often superior in terms of key project performance indicators—namely, total overtime hours, cost, and code quality. In 4/6 projects (such as ACAD, OPMET), ML-iMOSFLA achieved lower overtime minima (19–24% reduction), competitive cost ranges, and reduced error rates (Table 5.9). These outcomes suggest that the ML-based approach is capable of identifying more resource-efficient and higher-quality solutions. Even in more complex projects, such as PSOA and PARM, where ML-iMOSFLA showed slightly higher error rates due to high task dependencies, the differences were marginal and did not significantly compromise the overall solution quality. This reveals that machine learning can not only replicate human judgment but also enhance decision-making outcomes in software overtime planning.

RQ4: How scalable and adaptable is the proposed machine learning–driven optimization framework when applied to diverse software development project environments?

The scalability and adaptability of the proposed ML-driven optimization framework were demonstrated through its performance across six software projects of varying sizes and complexities. The RFR-gHGS model maintained high and stable predictive accuracy, with Adjusted R^2 values consistently ranging between 0.87 and 0.91, regardless of increases in the number of work packages or function points (Figures 4.14 and 4.15). The model not only improved accuracy but also enhanced resilience to variance induced by complexity. Similarly, ML-iMOSFLA demonstrated robust performance across all projects, adapting effectively to various preference structures and optimization landscapes (Tables 5.3 and 5.4). Even though performance degradation in dependency-rich projects, such as PSOA, may signal a boundary condition, the algorithm still maintains competitive performance. These findings affirm that the framework is not only technically sound but also, to some extent, scalable and adaptable, making it deployable in real-world software project management environments.

Table 6.1 summarizes the conclusions derived from addressing the research questions.

Table 6.1 Answers to research questions.

Research Question	Answer
RQ1	ML model (RFR-gHGS) effectively captures PM preferences with high accuracy.
RQ2	ML-iMOSFLA can replace human-in-the-loop interaction after a sufficient number of iterations.
RQ3	ML-iMOSFLA produces solutions comparable or superior to HIL in terms of overtime, cost, and quality of code.
RQ4	The framework is adaptable across projects with various complexities.

6.2 Achieved Research Objectives

The goal of this thesis is to develop and evaluate a machine learning-based interactive multi-objective algorithm for optimal software overtime planning. The details of how each of the five specific objectives of the thesis, as presented in Section 1.3, was met are outlined below.

Objective 1 was achieved through the systematic mapping study on the application of machine learning and multi-objective optimization algorithms in software project planning, as reported in Section 2.2 of the thesis.

Objective 2 was achieved in Section 3.1 of the thesis, where the conceptual framework for designing a machine learning-based interactive multi-objective optimization algorithm and a discussion on its implementation were presented.

Objective 3 was achieved in Chapter 4 with the development of a greedy cross-validation-based random forest regression machine learning model, which was trained and evaluated on newly produced datasets for estimating overtime in software projects.

Objective 4 was met through the design of the ML-iMOSFLA algorithm, as detailed in Section 3.3, and its implementation was discussed in Section 5.1 of the thesis.

Objective 5 was met in Chapter 5 of the thesis through the performance evaluation of the ML-iMOSFLA and its comparison with a non-interactive MOSFLA variant and a traditional Human-in-the-loop interactive MOSFLA approach.

Having answered all the research questions and accomplished all the defined objectives, the primary research problem of the thesis has been addressed, and thus, the overall dissertation goal has been achieved.

6.3 Research Significance and Thesis Contributions

Integrating a machine learning predictive model into an interactive multi-objective optimization algorithm can effectively replace continuous human-in-the-loop interaction, yielding overtime planning strategies that simultaneously optimize project overtime, cost, and quality metric while accurately reflecting project managers' subjective preferences. This approach is expected to reduce reliance on manual decision-making, thereby minimizing project overruns and enhancing overall software project performance compared to conventional methods. Specifically, the following significances are identified for the proposed framework:

1. *Elimination of Continuous Human Intervention:* The proposed framework enhances the overtime planning process by substituting the necessity for constant human-in-the-loop feedback with a machine learning model. This modification not only minimizes the workload on project managers but also mitigates the risk of decision fatigue and errors that are commonly associated with extended physical involvement.
2. *Enhanced Alignment with PM Preferences:* The machine learning model is developed to accurately capture and replicate the subjective assessments of software project managers, thereby ensuring that the optimized overtime plans are both objectively optimal and practically acceptable. This integration yields a more robust decision-making approach that reflects the inherent requirements of real-world projects**.**

3. *Improved Efficiency and Practicality:* The proposed framework provides a more efficient methodology for interactive multi-objective optimization within the domain of software project management. By automating the feedback loop through machine learning, this approach accelerates the decision-making process, enabling timely adjustments to project schedules and minimizing project delays and cost overruns**.**
4. *Innovation in Search-Based Software Project Management (SBSPM):* This study advances the state-of-the-art in SBSPM by integrating machine learning with multi-objective optimization. This novel approach makes a significant contribution to the domain of software project planning by offering an interactive solution that can be utilized in dynamic project environments.

Also, the major contributions of the thesis to scientific knowledge are outlined as follows:

1. Creation of new datasets to support machine learning-based overtime estimation in software engineering projects. Researchers can use the datasets to test the performance of various machine learning algorithms.
2. Introduction of a data-driven methodology for overtime estimation in software projects, using a machine learning approach. The greedy cross-validation-based halving grid search optimized random forest regression model represents a novel method in the software overtime planning domain.
3. Replacement of human-in-the-loop with a machine learning model in interactive multi-objective optimization algorithms. This provides researchers with a novel conceptual framework for implementing interactive optimization methods, particularly in software engineering.

6.4 Limitations

The machine learning–based interactive multi-objective optimization framework presented in this thesis shows significant promise in replicating subjective preferences of PMs and potentially replacing continuous Human-in-the-Loop interaction in software overtime planning. However, certain limitations have been identified, pertaining to data, construct, and practical implementation. These limitations are discussed as follows.

1. *Abstraction to WP level may oversimplify project complexity:* To manage complexity and reduce noise in the data, the proposed approach models software projects at the WP level rather than at the more granular task or activity level. While this abstraction improves model tractability and avoids issues like assigning overtime to trivial tasks, it may also obscure critical intra-WP dependencies and task-level dynamics that influence overtime planning decisions. Project managers often make careful and detailed decisions based on individual task characteristics and inter-task dependencies. Therefore, the model's behaviour and predictive accuracy might differ significantly if applied to a more comprehensive, task-based representation of the software projects.
2. *Limited Representativeness of Datasets:* The datasets used in this study are derived from publicly accessible software projects, which, while useful for reproducibility and benchmarking,

may not fully reflect the current realities of modern software development. These projects may differ in scale, tooling, structure, or domain-specific constraints compared to contemporary industrial projects. As a result, the external validity of the findings may be limited to projects similar in structure, scale, and domain to the ones used in this study. Collaborations with industry partners to obtain more recent and diverse datasets would enhance the generalizability and practical relevance of the proposed framework.

3. *Static Learning and Lack of Online Adaptation:* The current implementation of the RFR-gHGS model is trained offline and does not adapt during the optimization process. This static learning approach limits the model's ability to respond to evolving project conditions or shifting PM preferences. In contrast, human-in-the-loop systems can dynamically adjust in response to real-time feedback. Incorporating online learning or incremental model updates could enhance the responsiveness and adaptability of the system, especially in long-running or iterative project environments.
4. *Assumption of Stable and Consistent PM Preferences:* The framework assumes that PM preferences are stable and learnable from historical annotations. However, in practice, preferences may evolve due to changes in project scope or stakeholder priorities, which the current approach did not consider. This assumption may lead to suboptimal recommendations if the model is not regularly retrained or updated. Moreover, inconsistencies in PM evaluations—due to fatigue, ambiguity, or context-specific judgment—can introduce noise that affects model accuracy. This limitation can be addressed by incorporating mechanisms for preference drift detection and adaptive retraining to maintain alignment with evolving decision-making patterns.

6.5 Overall conclusion of the thesis

This thesis presents a significant advancement in the domain of software project planning by introducing a machine learning–driven interactive optimization framework tailored for software overtime planning. The research was motivated by the observed disconnect between the technically optimal solutions generated by existing search-based SOP methods and the subjective, context-specific preferences of project managers, which may hinder real-world applicability.

The study's primary contribution lies in the development of ML-iMOSFLA, an interactive memetic multi-objective optimization algorithm that incorporates a predictive model of PM preferences into the optimization loop. The RFR-gHGS model, trained on a uniquely constructed dataset of annotated overtime plans, demonstrated robust predictive performance across diverse project scenarios. The proposed greedy halving grid search method proved to be an efficient and scalable hyperparameter tuning strategy, enhancing the performance and generalizability of the RFR model. Its integration into the optimization process enabled the generation of overtime plans that not only minimized project cost, duration, and quality of code but also aligned closely with human decision-making patterns. Empirical evaluations across six real-world software projects confirmed the superiority of ML-iMOSFLA over both non-interactive and Human-In-the-Loop-based optimization methods. The

algorithm achieved higher solution quality, better preference alignment, and reduced interaction overhead, thereby addressing key limitations of existing SOP approaches.

The findings underscore the potential of combining machine learning with interactive optimization to bridge the gap between algorithmic efficiency and practical usability in software project management. By modeling and embedding human preferences into the optimization process, this research contributes a novel, scalable, and industry-relevant solution to the persistent challenge of software project overruns.

6.6 Future Works

To extend the current work in this thesis, the interactive optimization framework that integrates an ML model at the local evolution stage would be implemented and compared with the current model. Future work could also explore multi-level modelling that integrates both WP and task-level features to better capture the full complexity of software project planning. Additionally, testing the model's performance in modern real-world domains, such as Agile, automobile, or AI systems, would provide support for the generalizability of the approach. Finally, there is a need for an in-depth investigation into the structural trade-offs between subjective and objective criteria, potentially through post-hoc analysis of the Pareto front and stakeholder utility functions.

REFERENCES

[1] IEEE, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 (1990) 1–84. https://doi.org/10.1109/IEEESTD.1990.101064.

[2] F. Ferrucci, M. Harman, F. Sarro, Search-Based Software Project Management, in: G. Ruhe, C. Wohlin (Eds.), Softw. Proj. Manag. a Chang. World, Springer, 2014: pp. 373–399. https://doi.org/10.1007/978-3-642-55035-5.

[3] A.O. Ameen, H.A. Mojeed, A.T. Bolariwa, A.O. Balogun, M.A. Mabayoje, F.E. Usman-Hamzah, M. Abdulraheem, Application of Shuffled Frog-Leaping Algorithm for Optimal Software Project Scheduling and Staffing, Lect. Notes Data Eng. Commun. Technol. 72 (2021) 293–303. https://doi.org/10.1007/978-3-030-70713-2_28.

[4] A.V. Rezende, L. Silva, A. Britto, R. Amaral, Software project scheduling problem in the context of search-based software engineering: A systematic review, J. Syst. Softw. 155 (2019) 43–56. https://doi.org/https://doi.org/10.1016/j.jss.2019.05.024.

[5] T. Fatima, F. Azam, M.W. Anwar, Y. Rasheed, A Systematic Review on Software Project Scheduling and Task Assignment Approaches, ACM Int. Conf. Proceeding Ser. (2020) 369–373. https://doi.org/10.1145/3404555.3404588.

[6] R. Britto, V. Freitas, E. Mendes, M. Usman, Effort Estimation in Global Software Development: A Systematic Literature Review, in: 2014 IEEE 9th Int. Conf. Glob. Softw. Eng., 2014: pp. 135–144. https://doi.org/10.1109/ICGSE.2014.11.

[7] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, A. Egyed, TaskAllocator: A Recommendation Approach for Role-based Tasks Allocation in Agile Software Development, in: 2021 IEEE/ACM Jt. 15th Int. Conf. Softw. Syst. Process. 16th ACM/IEEE Int. Conf. Glob. Softw. Eng., 2021: pp. 39–49. https://doi.org/10.1109/ICSSP-ICGSE52873.2021.00014.

[8] M.Á. Vega-Velázquez, A. García-Nájera, H. Cervantes, A survey on the Software Project Scheduling Problem, Int. J. Prod. Econ. 202 (2018) 145–161. https://doi.org/https://doi.org/10.1016/j.ijpe.2018.04.020.

[9] M.Z.M. H, M.N. Mahdi, M.S. Mohd Azmi, L.K. Cheng, A. Yusof, A.R. Ahmad, Software Project Management Using Machine Learning Technique - A Review, in: 2020 8th Int. Conf. Inf. Technol. Multimed., 2020: pp. 363–370. https://doi.org/10.1109/ICIMU49871.2020.9243543.

[10] J. Ren, Search Based Software Project Management, University College Londom, 2013.

[11] M. Kuutila, M. Mäntylä, U. Farooq, M. Claes, Time pressure in software engineering: A systematic review, Inf. Softw. Technol. 121 (2020). https://doi.org/10.1016/j.infsof.2020.106257.

[12] K. Moløkken, M. Jørgensen, A Review of Surveys on Software Effort Estimation, in: Int. Symp. Empir. Softw. Eng. 2003. ISESE 2003., IEEE, 2003: pp. 223–230.

[13] E. Alba, J. Francisco Chicano, Software project management with GAs, Inf. Sci. (Ny). 177 (2007) 2380–2401. https://doi.org/10.1016/j.ins.2006.12.020.

[14] B. Crawford, R. Soto, F. Johnson, E. Monfroy, F. Paredes, A Max-Min Ant System algorithm to solve the Software Project Scheduling Problem, Expert Syst. Appl. 41 (2014) 6634–6645. https://doi.org/10.1016/j.eswa.2014.05.003.

[15] F. Luna, D.L. González-Álvarez, F. Chicano, M.A. Vega-Rodríguez, The software project scheduling problem: A scalability analysis of multi-objective metaheuristics, Appl. Soft Comput. J. 15 (2014) 136–148. https://doi.org/10.1016/j.asoc.2013.10.015.

[16] R.O. Oladele, H.A. Mojeed, A Shuffled Frog-Leaping Algorithm for Optimal Software Project Planning, African J. Comput. ICT 7 (2014) 147–152.

[17] V. Rachman, A.M. Ma'sum, Comparative analysis of ant colony extended and mix min ant system in sw project scheduling problem, Proc. - WBIS 2017 2017 Int. Work. Big Data Inf. Secur. 8 (2017) 85–91.

[18] Statista, Frequency of developers working overtime as of 2020, (n.d.).

[19] M. Van Der Hulst, S. Geurts, Associations between overtime and psychological health in high and low reward jobs, Work Stress 15 (2001) 227–240. https://doi.org/10.1080/026783701110.1080/02678370110066580.

[20] E. Kleppa, B. Sanne, G.S. Tell, Working Overtime is Associated With Anxiety and Depression: The Hordaland Health Study, J. Occup. Environ. Med. 50 (2008) 658–666. https://doi.org/10.1097/JOM.0b013e3181734330.

[21] F. Ferrucci, M. Harman, J. Ren, F. Sarro, Not going to take this anymore: Multi-objective overtime planning for software engineering projects, in: 2013 35th Int. Conf. Softw. Eng., 2013: pp. 462–471.

[22] M. DeO Barros, L.A.O. De Araujo, Learning overtime dynamics through multiobjective

optimization, in: GECCO 2016 - Proc. 2016 Genet. Evol. Comput. Conf., Association for Computing Machinery, Inc, 2016: pp. 1061–1068. https://doi.org/10.1145/2908812.2908824.
[23] B. Akula, J. Cusick, Impact of overtime and stress on software quality, in: WMSCI 2008 - 12th World Multi-Conference Syst. Cybern. Informatics, Jointly with 14th Int. Conf. Inf. Syst. Anal. Synth. ISAS 2008 - Proc., 2008: p. 214. https://doi.org/10.13140/RG.2.2.12815.59041.
[24] T. N., G. A.S., D. O., Effects of overtime on efficiency in software engineering projects; [Yazilim Mühendisliği Projelerinde Fazla Mesainin Verimlilik Üzerine Etkileri], in: CEUR Workshop Proc., 2017: pp. 135 – 146. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85035128505&partnerID=40&md5=0479d1549488ba73ae2ee99d3f33df1c.
[25] W. Zhang, Y. Yang, X. Liu, Reducing Delay Penalty of Multiple Concurrent Software Projects based on Overtime Planning, in: Proc. - 2020 35th IEEE/ACM Int. Conf. Autom. Softw. Eng. Work. ASEW 2020, 2020: pp. 47–52. https://doi.org/10.1145/3417113.3422152.
[26] D.X. Swenson, A Systems Model of Overtime Effects on Software Development Team Performance, The College of St. Scholastica, 2014.
[27] J. Jia, Y. Lu, System Dynamics Modeling for Overtime Management Strategy of Software Project, System (2007). https://doi.org/10.2495/CR080731.
[28] Javatpoint, Software Project Planning, (n.d.). https://www.javatpoint.com/software-project-planning (accessed December 2, 2023).
[29] F. Sarro, F. Ferrucci, M. Harman, A. Manna, J. Ren, Adaptive multi-objective evolutionary algorithms for overtime planning in software projects, IEEE Trans. Softw. Eng. 43 (2017) 898–917. https://doi.org/10.1109/TSE.2017.2650914.
[30] H.A. Mojeed, A.O. Bajeh, A.O. Balogun, H.O. Adeleke, Memetic approach for multi-objective overtime planning in software engineering projects, J. Eng. Sci. Technol. 14 (2019) 3213–3233.
[31] H. Hajjdiab, Al Shaima Taleb, Adopting Agile Software Development: Issues and Challenges, Int. J. Manag. Value Supply Chain. 2 (2011) 1–10. https://doi.org/10.5121/ijmvsc.2011.2301.
[32] A. Capodieci, L. Mainetti, L. Manco, A case study to enable and monitor real IT companies migrating from waterfall to agile, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2014: pp. 119–134. https://doi.org/10.1007/978-3-319-09156-3_9.
[33] A. Alashqur, Towards A Broader Adoption of Agile Software Development Methods, Int. J. Adv. Comput. Sci. Appl. 7 (2016) 94–98. https://doi.org/10.14569/ijacsa.2016.071212.
[34] M. Faisal Abrar, M. Sohail, S. Ali, M. Faran Majeed, I. Ali Shah, N. Rashid, N. Ullah, De-motivators for the adoption of agile methodologies for large-scale software development teams: An SLR from management perspective, J. Softw. Evol. Process 32 (2020) 1–20. https://doi.org/10.1002/smr.2268.
[35] S. Ali, L. Hongqi, M.F. Abrar, Systematic Literature Review of Critical Barriers to Software Outsourcing Partnership, in: 2018 5th Int. Multi-Topic ICT Conf., 2018: pp. 1–8. https://doi.org/10.1109/IMTIC.2018.8467254.
[36] M. DeO Barros, L.A.O. De Araujo, Learning overtime dynamics through multiobjective optimization, GECCO 2016 - Proc. 2016 Genet. Evol. Comput. Conf. (2016) 1061–1068. https://doi.org/10.1145/2908812.2908824.
[37] F. Ferrucci, M. Harman, J. Ren, F. Sarro, Not going to take this anymore: Multi-objective overtime planning for Software Engineering projects, in: Proc. - Int. Conf. Softw. Eng., 2013: pp. 462–471. https://doi.org/10.1109/ICSE.2013.6606592.
[38] F. Sarro, F. Ferrucci, M. Harman, A. Manna, J. Ren, Adaptive multi-objective evolutionary algorithms for overtime planning in software projects, IEEE Trans. Softw. Eng. 43 (2017) 898–917. https://doi.org/10.1109/TSE.2017.2650914.
[39] R. Saraiva, A.A. Araújo, A. Dantas, I. Yeltsin, J. Souza, Incorporating decision maker's preferences in a multi-objective approach for the software release planning, J. Brazilian Comput. Soc. 23 (2017). https://doi.org/10.1186/s13173-017-0060-0.
[40] C.L. Simons, J. Smith, P. White, Interactive ant colony optimization (iACO) for early lifecycle software design, Swarm Intell. 8 (2014) 139–157. https://doi.org/10.1007/s11721-014-0094-2.
[41] P. Tonella, A. Susi, F. Palma, Interactive requirements prioritization using a genetic algorithm, in: Inf. Softw. Technol., Elsevier B.V., 2013: pp. 173–187. https://doi.org/10.1016/j.infsof.2012.07.003.
[42] T. Wang, M. Zhou, A method for product form design of integrating interactive genetic algorithm with the interval hesitation time and user satisfaction, Int. J. Ind. Ergon. 76 (2020) 102901. https://doi.org/10.1016/j.ergon.2019.102901.
[43] G. Bavota, F. Carnevale, A. De Lucia, M. Di Penta, R. Oliveto, Putting the developer in-the-loop:

An interactive GA for software re-modularization, in: G. Fraser (Ed.), SSBSE '12 Lect. Notes Comput. Sci., Springer, 2012: pp. 75–89. https://doi.org/10.1007/978-3-642-33119-0_7.
[44] A. Dantas, I. Yeltsin, A.A. Araújo, J. Souza, Interactive software release planning with preferences base, in: M. Barros, Y. Labiche (Eds.), SSBSE '15, Lect. Notes Comput. Sci., Springer, 2015: pp. 341–346. https://doi.org/10.1007/978-3-319-22183-0_32.
[45] V. Nair, A. Agrawal, J. Chen, W. Fu, G. Mathew, T. Menzies, L. Minku, M. Wagner, Z. Yu, Data-driven search-based software engineering, in: Proc. - Int. Conf. Softw. Eng., 2018: pp. 341–352. https://doi.org/10.1145/3196398.3196442.
[46] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, A. Egyed, Machine Learning for Software Engineering: A Systematic Mapping, ArXiv Prepr. ArXiv2005.13299 (2020). http://arxiv.org/abs/2005.13299.
[47] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, A. Egyed, A Literature Review of Using Machine Learning in Software Development Life Cycle Stages, IEEE Access 9 (2021) 140896–140920. https://doi.org/10.1109/ACCESS.2021.3119746.
[48] A.A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, J. Souza, An Architecture based on interactive optimization and machine learning applied to the next release problem, Autom. Softw. Eng. 24 (2017) 623–671. https://doi.org/10.1007/s10515-016-0200-3.
[49] Standish-group, CHAOS report 2020: Beyond Infinity, (2020). https://www.standishgroup.com/news/45 (accessed July 8, 2023).
[50] M.N. Mahdi, M.H.M. Zabil, A.R. Ahmad, R. Ismail, Y. Yusoff, L.K. Cheng, M.S. Bin Mohd Azmi, H. Natiq, H.H. Naidu, Software project management using machine learning technique-a review, Appl. Sci. 11 (2021) 363–370. https://doi.org/10.3390/app11115183.
[51] H. Chennouk, B. El Bhiri, S. Bara, E.H. Ziyati, The machine learning in project management: A systematic mapping study, J. Theor. Appl. Inf. Technol. 98 (2020) 1550–1563. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086267533&partnerID=40&md5=8d4ac493c2b1e6b52696a8fe0deff8a3.
[52] M.N. Mahdi, M.H.M. Zabil, A.R. Ahmad, R. Ismail, Y. Yusoff, L.K. Cheng, M.S.B. Mohd Azmi, H. Natiq, H.H. Naidu, Software project management using machine learning technique-a review, Appl. Sci. 11 (2021). https://doi.org/10.3390/app11115183.
[53] C. Stylianou, A.S. Andreou, Human Resource Allocation and Scheduling for Software Project Management, in: Softw. Proj. Manag. a Chang. World, 2014: pp. 1–477. https://doi.org/10.1007/978-3-642-55035-5.
[54] M. Harman, S.A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (2012) 1–61.
[55] A.V. Rezende, L. Silva, A. Britto, R. Amaral, Software project scheduling problem in the context of search-based software engineering: A systematic review, J. Syst. Softw. 155 (2019) 43–56. https://doi.org/10.1016/J.JSS.2019.05.024.
[56] T. Chen, M. Li, The weights can be harmful: Pareto search versus weighted search in multi-objective search-based software engineering, ACM Trans. Softw. Eng. Methodol. 32 (2023) 1–40.
[57] M. Shameem, M. Nadeem, A.T. Zamani, Genetic algorithm based probabilistic model for agile project success in global software development, Appl. Soft Comput. 135 (2023) 109998.
[58] A. Zeb, F. Din, M. Fayaz, G. Mehmood, K.Z. Zamli, others, A Systematic Literature Review on Robust Swarm Intelligence Algorithms in Search-Based Software Engineering, Complexity 2023 (2023).
[59] F. Formica, T. Fan, C. Menghi, Search-based software testing driven by automatically generated and manually defined fitness functions, ACM Trans. Softw. Eng. Methodol. 33 (2023) 1–37.
[60] D. Sharma, G. Sharma, Systematic Literature Review of Search-Based Software Engineering Techniques for Code Modularization/Remodularization, Comput. Intell. Appl. Softw. Eng. Probl. (2023) 241–266.
[61] A.S. Habib, S.U.R. Khan, E.A. Felix, A systematic review on search-based test suite reduction: State-of-the-art, taxonomy, and future directions, IET Softw. 17 (2023) 93–136.
[62] A.V. Rezende, L. Silva, A. Britto, R. Amaral, Software project scheduling problem in the context of search-based software engineering: A systematic review, J. Syst. Softw. 155 (2019) 43–56. https://doi.org/10.1016/j.jss.2019.05.024.
[63] R. Malhotra, M. Khanna, R.R. Raje, On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions, Swarm Evol. Comput. 32 (2017) 85–109. https://doi.org/https://doi.org/10.1016/j.swevo.2016.10.002.
[64] X.-N. Shen, L.L. Minku, N. Marturi, Y.-N. Guo, Y. Han, A Q-learning-based memetic algorithm for multi-objective dynamic software project scheduling, Inf. Sci. (Ny). 428 (2018) 1–29.

https://doi.org/https://doi.org/10.1016/j.ins.2017.10.041.
[65] X. Shen, L.L. Minku, R. Bahsoon, X. Yao, Dynamic Software Project Scheduling through a Proactive-Rescheduling Method, IEEE Trans. Softw. Eng. 42 (2016). https://doi.org/10.1109/TSE.2015.2512266.
[66] J. Xiao, X.T. Ao, Y. Tang, Solving software project scheduling problems with ant colony optimization, Comput. Oper. Res. 40 (2013) 33–46. https://doi.org/10.1016/j.cor.2012.05.007.
[67] A.L.I. Abreu, M.A. Ampuero, A.R. Suárez, L. Rampersaud, Formation of project teams applying software multiobjective metaheuristics algorithms path, Intel. Artif. 17 (2014) 1–16. https://doi.org/10.4114/ia.v17n54.1101.
[68] N. Bibi, A. Ahsan, Z. Anwar, Project resource allocation optimization using search based software engineering — A framework, in: Ninth Int. Conf. Digit. Inf. Manag. (ICDIM 2014), 2014: pp. 226–229. https://doi.org/10.1109/ICDIM.2014.6991431.
[69] M. Knyazeva, A. Bozhenyuk, I. Rozenberg, Resource-constrained Project Scheduling Approach under Fuzzy Conditions, Procedia Comput. Sci. 77 (2015) 56–64. https://doi.org/10.1016/j.procs.2015.12.359.
[70] C. Stylianou, A.S. Andreou, A multi-objective genetic algorithm for intelligent software project scheduling and team staffing, Intell. Decis. Technol. 7 (2013) 59–80. https://doi.org/10.3233/IDT-120151.
[71] J. Xiao, M.L. Gao, M.M. Huang, Empirical study of multi-objective ant colony optimization to software project scheduling problems, GECCO 2015 - Proc. 2015 Genet. Evol. Comput. Conf. (2015) 759–766. https://doi.org/10.1145/2739480.2754702.
[72] Z.T. Kosztyán, E. Bogdány, I. Szalkai, M.T. Kurbucz, Impacts of synergies on software project scheduling, Ann. Oper. Res. 312 (2022) 883–908. https://doi.org/10.1007/s10479-021-04467-5.
[73] A. García-Nájera, M. del C. Gómez-Fuentes, A multi-objective genetic algorithm for the software project scheduling problem, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 8857 (2014) 13–24. https://doi.org/10.1007/978-3-319-13650-9_2.
[74] O. Bozorg-Haddad, M. Solgi, H.A. Loáiciga, Meta-heuristic and evolutionary algorithms for engineering optimization, John Wiley & Sons, 2017.
[75] C.A.C. Coello, G.T. Pulido, M.S. Lechuga, Handling multiple objectives with particle swarm optimization, IEEE Trans. Evol. Comput. 8 (2004) 256–279.
[76] E.B. Tirkolaee, A. Goli, M. Hematian, A.K. Sangaiah, T. Han, Multi-objective multi-mode resource constrained project scheduling problem using Pareto-based algorithms, Computing 101 (2019) 547–570. https://doi.org/10.1007/s00607-018-00693-1.
[77] A.L. Jaimes, S.Z. Martínez, An introduction to multiobjective optimization techniques, 2011.
[78] X. Shen, L.L. Minku, R. Bahsoon, X. Yao, Dynamic Software Project Scheduling through a Proactive-Rescheduling Method, IEEE Trans. Softw. Eng. 42 (2016) 658–686. https://doi.org/10.1109/TSE.2015.2512266.
[79] G. Guizzo, S.R. Vergilio, A.T.R. Pozo, G.M. Fritsche, A multi-objective and evolutionary hyper-heuristic applied to the Integration and Test Order Problem, Appl. Soft Comput. J. 56 (2017). https://doi.org/10.1016/j.asoc.2017.03.012.
[80] J. Zeng, W. Nie, Novel multi-objective optimization algorithm, J. Syst. Eng. Electron. 25 (2014) 697–710. https://doi.org/10.1109/JSEE.2014.00080.
[81] N. Saini, S. Saha, P. Bhattacharyya, Automatic Scientific Document Clustering Using Self-organized Multi-objective Differential Evolution, Cognit. Comput. (2019). https://doi.org/10.1007/s12559-018-9611-8.
[82] K. Langsari, R. Sarno, Sholiq, Optimizing time and effort parameters of COCOMO II using fuzzy Multi-objective Particle Swarm Optimization, Telkomnika (Telecommunication Comput. Electron. Control. 16 (2018) 2199–2207. https://doi.org/10.12928/TELKOMNIKA.v16i5.9698.
[83] Y. Zhang, B. Li, W. Hong, A. Zhou, MOCPSO: A multi-objective cooperative particle swarm optimization algorithm with dual search strategies, Neurocomputing 562 (2023) 126892. https://doi.org/10.1016/j.neucom.2023.126892.
[84] S.K. Pemmada, J. Nayak, B. Naik, A deep intelligent framework for software risk prediction using improved firefly optimization, Neural Comput. Appl. 35 (2023) 19523–19539. https://doi.org/10.1007/s00521-023-08756-x.
[85] J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, Swarm Evol. Comput. 32 (2017) 121–131. https://doi.org/10.1016/j.swevo.2016.06.002.
[86] B. Cao, W. Zhang, X. Wang, J. Zhao, Y. Gu, Y. Zhang, A memetic algorithm based on two_Arch2 for multi-depot heterogeneous-vehicle capacitated arc routing problem, Swarm Evol. Comput.

63 (2021). https://doi.org/10.1016/j.swevo.2021.100864.
[87] N. Nigar, M.K. Shahzad, S. Islam, S. Kumar, A. Jaleel, Modeling Human Resource Experience Evolution for Multiobjective Project Scheduling in Large Scale Software Projects, IEEE Access 10 (2022) 44677–44690. https://doi.org/10.1109/ACCESS.2022.3169596.
[88] M.N. Mahdi, M.H.M. Zabil, A.R. Ahmad, R. Ismail, Y. Yusoff, L.K. Cheng, M.S.B. Mohd Azmi, H. Natiq, H.H. Naidu, Software project management using machine learning technique-a review, Appl. Sci. 11 (2021). https://doi.org/10.3390/app11115183.
[89] M. Jørgensen, A review of studies on expert estimation of software development effort, J. Syst. Softw. 70 (2004) 37–60. https://doi.org/10.1016/S0164-1212(02)00156-5.
[90] Z. Kotti, R. Galanopoulou, D. Spinellis, Machine Learning for Software Engineering: A Tertiary Study, ACM Comput. Surv. 55 (2023). https://doi.org/10.1145/3572905.
[91] O. Cico, B. Cico, A. Cico, AI-assisted Software Engineering: a tertiary study, in: 2023 12th Mediterr. Conf. Embed. Comput., 2023: pp. 1–6.
[92] M.I. Jordan, T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, Science (80-.). 349 (2015) 255–260. https://doi.org/10.1126/science.aaa8415.
[93] A. Alhaddad, I. Al-Baltah, A. Abualkishik, M. Abdellatief, A. Ali Al Kharusi, A systematic mapping study on software effort estimation, J. Theor. Appl. Inf. Technol. 98 (2020) 3619–3643. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85091034494&partnerID=40&md5=9725d68389773ad7c3827b7127a49b84.
[94] A. Najm, A. Zakrani, A. Marzak, Decision Trees Based Software Development Effort Estimation: A Systematic Mapping Study, in: 2019 Int. Conf. Comput. Sci. Renew. Energies, 2019: pp. 1–6. https://doi.org/10.1109/ICCSRE.2019.8807544.
[95] A. Idri, M. Hosni, A. Abran, Systematic literature review of ensemble effort estimation, J. Syst. Softw. 118 (2016) 151–175. https://doi.org/10.1016/j.jss.2016.05.016.
[96] S.K. Sehra, Y.S. Brar, N. Kaur, S.S. Sehra, Research patterns and trends in software effort estimation, Inf. Softw. Technol. 91 (2017) 1–21. https://doi.org/https://doi.org/10.1016/j.infsof.2017.06.002.
[97] H. Liu, M. Qiao, D. Greenia, R. Akkiraju, S. Dill, T. Nakamura, Y. Song, H.M. Nezhad, A machine learning approach to combining individual strength and team features for team recommendation, in: 2014 13th Int. Conf. Mach. Learn. Appl., 2014: pp. 213–218.
[98] E. Bisong, E. Tran, O. Baysal, Built to Last or Built Too Fast? Evaluating Prediction Models for Build Times, in: 2017 IEEE/ACM 14th Int. Conf. Min. Softw. Repos., 2017: pp. 487–490. https://doi.org/10.1109/MSR.2017.36.
[99] C. López-Martín, A. Abran, Neural networks for predicting the duration of new software projects, J. Syst. Softw. 101 (2015) 127–135. https://doi.org/10.1016/j.jss.2014.12.002.
[100] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, The Impact of Automated Parameter Optimization on Defect Prediction Models, IEEE Trans. Softw. Eng. 45 (2019) 683–711. https://doi.org/10.1109/TSE.2018.2794977.
[101] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Inf. Softw. Technol. 64 (2015) 1–18. https://doi.org/10.1016/j.infsof.2015.03.007.
[102] B. Kitchenham, P. Brereton, A systematic review of systematic review process research in software engineering, Inf. Softw. Technol. 55 (2013) 2049–2075. https://doi.org/10.1016/j.infsof.2013.07.010.
[103] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, 12th Int. Conf. Eval. Assess. Softw. Eng. EASE 2008 (2008) 1–10. https://doi.org/10.14236/ewic/ease2008.8.
[104] S.B. Kitchenham, Charters, Guidelines for performing systematic literature reviews in software engineering, Tech. Report, Ver. 2.3 EBSE Tech. Report. EBSE (2007).
[105] K.B.& C. S., Guidelines for performing systematic literature reviews in software engineering, Tech. Report, Ver. 2.3 EBSE Tech. Report. EBSE (2007).
[106] M. Perkusich, L. e Silva, A. Costa, F. Ramos, R. Saraiva, A. Freire, E. Dilorenzo, E. Dantas, D. Santos, K. Gorgônio, H. Almeida, A. Perkusich, Intelligent software engineering in the context of agile software development: A systematic literature review, Inf. Softw. Technol. 119 (2020). https://doi.org/10.1016/j.infsof.2019.106241.
[107] E. Mourão, J.F. Pimentel, L. Murta, M. Kalinowski, E. Mendes, C. Wohlin, On the performance of hybrid search strategies for systematic literature reviews in software engineering, Inf. Softw. Technol. 123 (2020). https://doi.org/10.1016/j.infsof.2020.106294.
[108] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the

systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (2007) 571–583. https://doi.org/10.1016/j.jss.2006.07.009.
[109] J. Masso, F.J. Pino, C. Pardo, F. García, M. Piattini, Risk management in the software life cycle: A systematic literature review, Comput. Stand. Interfaces 71 (2020) 103431. https://doi.org/10.1016/j.csi.2020.103431.
[110] T. Dybå, T. Dingsøyr, Strength of Evidence in Systematic Reviews in Software Engineering, ESEM'08 Proc. 2008 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. (2008) 178–187. https://doi.org/10.1145/1414004.1414034.
[111] S.Y. Chadli, A. Idri, Identifying and mitigating risks of software project management in global software development, ACM Int. Conf. Proceeding Ser. Part F1319 (2017) 12–22. https://doi.org/10.1145/3143434.3143453.
[112] SJR, Scimago Journal Ranking, (2021).
[113] CORE, Computing research and Education, (2021). http://portal.core.edu.au/conf-ranks/?search=&by=all&source=CORE2021&sort=atitle&page=1.
[114] M. Azzeh, A.B. Nassif, S. Banitaan, A better case adaptation method for case-based effort estimation using multi-objective optimization, in: Proc. - 2014 13th Int. Conf. Mach. Learn. Appl. ICMLA 2014, 2014: pp. 409–414. https://doi.org/10.1109/ICMLA.2014.73.
[115] E. Papatheocharous, A.S. Andreou, A hybrid software cost estimation approach utilizing decision trees and fuzzy logic, Int. J. Softw. Eng. Knowl. Eng. 22 (2012) 435–465. https://doi.org/10.1142/S0218194012500106.
[116] N. Rankovic, D. Rankovic, M. Ivanovic, L. Lazic, A New Approach to Software Effort Estimation Using Different Artificial Neural Network Architectures and Taguchi Orthogonal Arrays, IEEE Access 9 (2021) 26926–26936. https://doi.org/10.1109/ACCESS.2021.3057807.
[117] A. Ullah, B. Wang, J. Sheng, J. Long, M. Asim, F. Riaz, A novel technique of software cost estimation using flower pollination algorithm, in: Proc. - 2019 Int. Conf. Intell. Comput. Autom. Syst. ICICAS 2019, 2019: pp. 654–658. https://doi.org/10.1109/ICICAS48597.2019.00142.
[118] P. Suresh Kumar, H.S. Behera, J. Nayak, B. Naik, A pragmatic ensemble learning approach for effective software effort estimation, Innov. Syst. Softw. Eng. (2021). https://doi.org/10.1007/s11334-020-00379-y.
[119] M.A. Ramessur, S.D. Nagowah, A predictive model to estimate effort in a sprint using machine learning techniques, Int. J. Inf. Technol. 13 (2021) 1101–1110. https://doi.org/10.1007/s41870-021-00669-z.
[120] S.P. Singh, G. Dhiman, P. Tiwari, R.H. Jhaveri, A soft computing based multi-objective optimization approach for automatic prediction of software cost models, Appl. Soft Comput. 113 (2021). https://doi.org/10.1016/j.asoc.2021.107981.
[121] M.M. Öztürk, A tuned feed-forward deep neural network algorithm for effort estimation, J. Exp. Theor. Artif. Intell. 0 (2021) 1–25. https://doi.org/10.1080/0952813X.2021.1871664.
[122] V.S. Ionescu, An approach to software development effort estimation using machine learning, in: Proc. - 2017 IEEE 13th Int. Conf. Intell. Comput. Commun. Process. ICCP 2017, 2017: pp. 197–203. https://doi.org/10.1109/ICCP.2017.8117004.
[123] P. Pospieszny, B. Czarnacka-Chrobot, A. Kobylinski, An effective approach for software project effort and duration estimation with machine learning algorithms, J. Syst. Softw. 137 (2018) 184–196. https://doi.org/10.1016/j.jss.2017.11.066.
[124] O. Malgonde, K. Chari, An ensemble-based model for predicting agile software development effort, Empir. Softw. Eng. 24 (2019) 1017–1055. https://doi.org/10.1007/s10664-018-9647-0.
[125] S. Sharma, S. Vijayvargiya, An Optimized Neuro-Fuzzy Network for Software Project Effort Estimation, IETE J. Res. (2022). https://doi.org/10.1080/03772063.2022.2027282.
[126] V. Resmi, S. Vijayalakshmi, Analogy-based approaches to improve software project effort estimation accuracy, J. Intell. Syst. 29 (2020) 1468–1479. https://doi.org/10.1515/jisys-2019-0023.
[127] J. Popovic, D. Bojic, N. Korolija, Analysis of task effort estimation accuracy based on use case point size, IET Softw. 9 (2015) 166–173. https://doi.org/10.1049/iet-sen.2014.0254.
[128] H. Karna, L. Vicković, S. Gotovac, Application of data mining methods for effort estimation of software projects, Softw. - Pract. Exp. 49 (2019) 171–191. https://doi.org/10.1002/spe.2651.
[129] K. Tanaka, A. Monden, Z. Yücel, Applying auto-sklearn to Software Development Effort Estimation. | ソフトウェア開発工数予測における auto-sklearn の適用, Comput. Softw. 38 (2021) 46–52. https://doi.org/10.11309/jssst.38.4_46.
[130] S. Tuarob, N. Assavakamhaenghan, W. Tanaphantaruk, P. Suwanworaboon, S.U. Hassan, M. Choetkiertikul, Automatic team recommendation for collaborative software development, Empir.

Softw. Eng. 26 (2021). https://doi.org/10.1007/s10664-021-09966-4.
[131] L.L. Minku, X. Yao, Can cross-company data improve performance in software effort estimation?, ACM Int. Conf. Proceeding Ser. (2012) 69–78. https://doi.org/10.1145/2365324.2365334.
[132] J. Moeyersoms, E. Junqué De Fortuny, K. Dejaeger, B. Baesens, D. Martens, Comprehensible software fault and effort prediction: A data mining approach, J. Syst. Softw. 100 (2015) 80–90. https://doi.org/10.1016/j.jss.2014.10.032.
[133] X. Shen, Y. Guo, A. Li, Cooperative coevolution with an improved resource allocation for large-scale multi-objective software project scheduling, Appl. Soft Comput. J. 88 (2020). https://doi.org/10.1016/j.asoc.2019.106059.
[134] K. Moharreri, A.V. Sapre, J. Ramanathan, R. Ramnath, Cost-Effective Supervised Learning Models for Software Effort Estimation in Agile Environments, in: Proc. - Int. Comput. Softw. Appl. Conf., 2016: pp. 135–140. https://doi.org/10.1109/COMPSAC.2016.85.
[135] H. Karna, S. Gotovac, L. Vicković, Data mining approach to effort modeling on agile software projects, Inform. 44 (2020) 231–239. https://doi.org/10.31449/inf.v44i2.2759.
[136] S.M. Satapathy, B.P. Acharya, S.K. Rath, Early stage software effort estimation using random forest technique based on use case points, IET Softw. 10 (2016) 10–17. https://doi.org/10.1049/iet-sen.2014.0122.
[137] C. López-Martín, Effort prediction for the software project construction phase, J. Softw. Evol. Process 33 (2021). https://doi.org/10.1002/smr.2365.
[138] S.M. Satapathy, S.K. Rath, Empirical assessment of machine learning models for agile software development effort estimation using story points, Innov. Syst. Softw. Eng. 13 (2017) 191–200. https://doi.org/10.1007/s11334-017-0288-z.
[139] A. Hussain, M. Raja, P. Vellaisamy, S. Krishnan, L. Rajendran, Enhanced framework for ensemble effort estimation by using recursive-based classification, IET Softw. 15 (2021) 230–238. https://doi.org/10.1049/sfw2.12020.
[140] J.T.H. de A. Cabral, A.L.I. Oliveira, Ensemble Effort Estimation using dynamic selection, J. Syst. Softw. 175 (2021). https://doi.org/10.1016/j.jss.2021.110904.
[141] A.G. Priya Varshini, K. Anitha Kumari, V. Varadarajan, Estimating software development efforts using a random forest-based stacked ensemble approach, Electron. 10 (2021). https://doi.org/10.3390/electronics10101195.
[142] A.B. Nassif, L.F. Capretz, D. Ho, Estimating software effort using an ANN model based on use case points, in: Proc. - 2012 11th Int. Conf. Mach. Learn. Appl. ICMLA 2012, 2012: pp. 42–47. https://doi.org/10.1109/ICMLA.2012.138.
[143] H.D.P. De Carvalho, R. Fagundes, W. Santos, Extreme Learning Machine Applied to Software Development Effort Estimation, IEEE Access 9 (2021) 92676–92687. https://doi.org/10.1109/ACCESS.2021.3091313.
[144] S. Tong, Q. He, Y. Chen, Y. Yang, B. Shen, Heterogeneous Cross-Company Effort Estimation through Transfer Learning, in: Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, 2016: pp. 169–176. https://doi.org/10.1109/APSEC.2016.033.
[145] J.T.H.D.A. Cabral, R.D.A. Araujo, J.P. Nobrega, A.L.I. De Oliveira, Heterogeneous ensemble dynamic selection for software development effort estimation, in: Proc. - Int. Conf. Tools with Artif. Intell. ICTAI, 2018: pp. 210–217. https://doi.org/10.1109/ICTAI.2017.00042.
[146] R.D.A. Araújo, S. Soares, A.L.I. Oliveira, Hybrid morphological methodology for software development cost estimation, Expert Syst. Appl. 39 (2012) 6129–6139. https://doi.org/10.1016/j.eswa.2011.11.077.
[147] S.K. Palaniswamy, R. Venkatesan, Hyperparameters tuning of ensemble model for software effort estimation, J. Ambient Intell. Humaniz. Comput. 12 (2021) 6579–6589. https://doi.org/10.1007/s12652-020-02277-4.
[148] Y. Mahmood, N. Kama, A. Azmi, M. Ali, Improving Estimation Accuracy Prediction of Software Development Effort: A Proposed Ensemble Model, in: 2nd Int. Conf. Electr. Commun. Comput. Eng. ICECCE 2020, 2020: pp. 1–6. https://doi.org/10.1109/ICECCE49384.2020.9179279.
[149] N. Mittas, E. Papatheocharous, L. Angelis, A.S. Andreou, Integrating non-parametric models with linear components for producing software cost estimations, J. Syst. Softw. 99 (2015) 120–134. https://doi.org/10.1016/j.jss.2014.09.025.
[150] C. Stylianou, A.S. Andreou, Investigating the impact of developer productivity, task interdependence type and communication overhead in a multi-objective optimization approach for software project planning, Adv. Eng. Softw. 98 (2016) 79–96. https://doi.org/10.1016/j.advengsoft.2016.04.001.

[151] E. Kocaguneli, T. Menzies, J.W. Keung, Kernel methods for software effort estimation: Effects of different kernel functions and bandwidths on estimation accuracy, Empir. Softw. Eng. 18 (2013) 1–24. https://doi.org/10.1007/s10664-011-9189-1.
[152] F. Sarro, A. Petrozziello, M. Harman, Multi-objective software effort estimation, Proc. - Int. Conf. Softw. Eng. 14-22-May- (2016) 619–630. https://doi.org/10.1145/2884781.2884830.
[153] R. Fuentetaja, D. Borrajo, C.L. López, J. Ocón, Multi-step Generation of Bayesian Networks Models for Software Projects Estimations, Int. J. Comput. Intell. Syst. 6 (2013) 796–821. https://doi.org/10.1080/18756891.2013.805583.
[154] C. López-Martín, A. Abran, Neural networks for predicting the duration of new software projects, J. Syst. Softw. 101 (2015) 127–135. https://doi.org/10.1016/j.jss.2014.12.002.
[155] E. Kocaguneli, T. Menzies, J.W. Keung, On the value of ensemble effort estimation, IEEE Trans. Softw. Eng. 38 (2012) 1403–1416. https://doi.org/10.1109/TSE.2011.111.
[156] M. Vyas, N. Hemrajani, Predicting effort of agile software projects using linear regression, ridge regression and logistic regression, Int. J. Tech. Phys. Probl. Eng. 13 (2021) 14–19. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85111133407&partnerID=40&md5=cf9627f6a4b855f1ea3f38fde0df4c7c.
[157] P. Rai, S. Kumar, D.K. Verma, Prediction of Software Effort in the Early Stage of Software Development: A Hybrid Model, IEEE Can. J. Electr. Comput. Eng. 44 (2021) 376–383. https://doi.org/10.1109/icjece.2021.3084850.
[158] I. Attarzadeh, S.H. Ow, Proposing a novel artificial neural network prediction model to improve the precision of software effort estimation, Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. 87 LNICST (2012) 334–342. https://doi.org/10.1007/978-3-642-32615-8_33.
[159] S.H. Samareh Moosavi, V. Khatibi Bardsiri, Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation, Eng. Appl. Artif. Intell. 60 (2017) 1–15. https://doi.org/10.1016/j.engappai.2017.01.006.
[160] E.M. De Bortoli Fávero, D. Casanova, A.R. Pimentel, SE3M: A model for software effort estimation using pre-trained embedding models, Inf. Softw. Technol. 147 (2022). https://doi.org/10.1016/j.infsof.2022.106886.
[161] I. González-Carrasco, R. Colomo-Palacios, J.L. López-Cuadrado, F.J.G. Peñalvo, SEffEst: Effort estimation in software projects using fuzzy logic and neural networks, Int. J. Comput. Intell. Syst. 5 (2012) 679–699. https://doi.org/10.1080/18756891.2012.718118.
[162] A.B. Nassif, M. Azzeh, A. Idri, A. Abran, Software development effort estimation using regression fuzzy models, Comput. Intell. Neurosci. 2019 (2019). https://doi.org/10.1155/2019/8367214.
[163] L.L. Minku, X. Yao, Software Effort Estimation as a Multiobjective Learning Problem, ACM Trans. Softw. Eng. Methodol. 22 (2013). https://doi.org/10.1145/2522920.2522928.
[164] D.R. Pai, K.S. McFall, G.H. Subramanian, Software effort estimation using a neural network ensemble, J. Comput. Inf. Syst. 53 (2013) 49–58. https://doi.org/10.1080/08874417.2013.11645650.
[165] T.R. Benala, R. Mall, S. Dehuri, P. Swetha, Software effort estimation using functional link neural networks tuned with active learning and optimized with particle swarm optimization, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 8947 (2015) 223–238. https://doi.org/10.1007/978-3-319-20294-5_20.
[166] L. Song, L.L. Minku, Y.A.O. Xin, Software effort interval prediction via Bayesian inference and synthetic bootstrap resampling, ACM Trans. Softw. Eng. Methodol. 28 (2019). https://doi.org/10.1145/3295700.
[167] M. Gultekin, O. Kalipsiz, Story Point-Based Effort Estimation Model with Machine Learning Techniques, Int. J. Softw. Eng. Knowl. Eng. 30 (2020) 43–66. https://doi.org/10.1142/S0218194020500035.
[168] A. Idri, I. Abnane, A. Abran, Support vector regression-based imputation in analogy-based software development effort estimation, J. Softw. Evol. Process 30 (2018). https://doi.org/10.1002/smr.2114.
[169] I. Al-Taharwa, Teamwork Distribution: Local vs. Global Software Engineering Project Development Teamwork, Int. J. Emerg. Technol. Learn. 15 (2020) 183–201. https://doi.org/10.3991/ijet.v15i18.15489.
[170] N. Assavakamhaenghan, P. Suwanworaboon, W. Tanaphantaruk, S. Tuarob, M. Choetkiertikul, Towards Team Formation in Software Development: A Case Study of Moodle, in: 2020 17th Int. Conf. Electr. Eng. Comput. Telecommun. Inf. Technol., 2020: pp. 157–160. https://doi.org/10.1109/ECTI-CON49241.2020.9158078.
[171] L.L. Minku, X. Yao, Which models of the past are relevant to the present? A software effort

estimation approach to exploiting useful past models, Autom. Softw. Eng. 24 (2017) 499–542. https://doi.org/10.1007/s10515-016-0209-7.
[172] Z.H. Wani, K.J. Giri, R. Bashir, A Generic Data Mining Model for Software Cost Estimation Based on Novel Input Selection Procedure, Int. J. Inf. Retr. Res. 9 (2019) 16–32. https://doi.org/10.4018/ijirr.2019010102.
[173] N. Rankovic, D. Rankovic, M. Ivanovic, L. Lazic, A New Approach to Software Effort Estimation Using Different Artificial Neural Network Architectures and Taguchi Orthogonal Arrays, IEEE Access 9 (2021) 26926–26936. https://doi.org/10.1109/ACCESS.2021.3057807.
[174] A. Zakrani, M. Hain, A. Namir, Software development effort estimation using random forests: An empirical study and evaluation, Int. J. Intell. Eng. Syst. 11 (2018) 300–311. https://doi.org/10.22266/IJIES2018.1231.30.
[175] V. Tawosi, F. Sarro, A. Petrozziello, M. Harman, Multi-Objective Software Effort Estimation: A Replication Study, IEEE Trans. Softw. Eng. 48 (2022) 3185–3205. https://doi.org/10.1109/TSE.2021.3083360.
[176] A. Idri, I. Abnane, A. Abran, Support vector regression-based imputation in analogy-based software development effort estimation, J. Softw. Evol. Process 30 (2018). https://doi.org/10.1002/smr.2114.
[177] A.J. Nebro, J.J. Durillo, M. Machin, C.A. Coello Coello, B. Dorronsoro, A study of the combination of variation operators in the NSGA-II algorithm, in: Adv. Artif. Intell. 15th Conf. Spanish Assoc. Artif. Intell. CAEPIA 2013, Madrid, Spain, Sept. 17-20, 2013. Proc. 15, 2013: pp. 269–278.
[178] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, Automated parameter optimization of classification techniques for defect prediction models, Proc. - Int. Conf. Softw. Eng. 14-22-May- (2016) 321–332. https://doi.org/10.1145/2884781.2884857.
[179] P. Myszkowski, M. Laszczyk, I. Nikulin, M. Skowroński, iMOPSE: a library for bicriteria optimization in multi-skill resource-constrained project scheduling problem, Soft Comput. 23 (2019) 3397–3410.
[180] R. Szlapczynski, J. Szlapczynska, W-dominance: Tradeoff-inspired dominance relation for preference-based evolutionary multi-objective optimization, Swarm Evol. Comput. 63 (2021) 100866. https://doi.org/10.1016/j.swevo.2021.100866.
[181] M. Kadziński, M.K. Tomczyk, R. Słowiński, Preference-based cone contraction algorithms for interactive evolutionary multiple objective optimization, Swarm Evol. Comput. 52 (2020). https://doi.org/10.1016/j.swevo.2019.100602.
[182] G. Misitano, B. Afsar, G. Lárraga, K. Miettinen, Towards explainable interactive multiobjective optimization: R-XIMO, Auton. Agent. Multi. Agent. Syst. 36 (2022) 43.
[183] A. Ramírez, J.R. Romero, C.L. Simons, A Systematic Review of Interaction in Search-Based Software Engineering, IEEE Trans. Softw. Eng. 45 (2019) 760–781. https://doi.org/10.1109/TSE.2018.2803055.
[184] J. Hao, R. Deng, L. Jia, Z. Li, R. Alizadeh, L. Soltanisehat, B. Liu, Z. Sun, Y. Shao, Human-in-the-loop optimization for vehicle body lightweight design, Adv. Eng. Informatics 62 (2024) 102887. https://doi.org/10.1016/j.aei.2024.102887.
[185] T. do Nascimento Ferreira, A.A. Araújo, A.D.B. Neto, J.T. de Souza, Incorporating user preferences in ant colony optimization for the next release problem, Appl. Soft Comput. 49 (2016) 1283–1296.
[186] B. Marculescu, S. Poulding, R. Feldt, K. Petersen, R. Torkar, Tester interactivity makes a difference in search-based software testing: A controlled experiment, Inf. Softw. Technol. 78 (2016) 66–82. https://doi.org/10.1016/j.infsof.2016.05.009.
[187] A. Ghannem, G. El Boussaidi, M. Kessentini, Model refactoring using interactive genetic algorithm, in: G. Ruhe, Y. Zhang (Eds.), SSBSE 2013, Lect. Notes Comput. Sci., Springer, 2013: pp. 96–110. https://doi.org/10.1007/978-3-642-39742-4_9.
[188] B. Amal, M. Kessentini, S. Bechikh, J. Dea, L. Ben Said, On the use of machine learning and search-based software engineering for ill-defined fitness function: a case study on software refactoring, in: Search-Based Softw. Eng. 6th Int. Symp. SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proc. 6, 2014: pp. 31–45.
[189] Y. Li, J. Zhou, Y. Zhang, H. Qin, L. Liu, Novel Multiobjective Shuffled Frog Leaping Algorithm with Application to Reservoir Flood Control Operation, J. Water Resour. Plan. Manag. 136 (2010) 217–226. https://doi.org/10.1061/(ASCE)WR.1943-5452.0000027.
[190] C. Jones, Software assessments, benchmarks, and best practices, Addison-Wesley Longman Publishing Co., Inc., 2000.
[191] Cyaniclab team, Hourly Rates of Software Developers by Region: A Global Comparison, (2024).

https://cyaniclab.com/blogs/hourly-rates-of-software-developers-by-region-a-global-comparison (accessed April 20, 2011).
[192] MoldStud Research team, An In-Depth Analysis of Global Software Development Rates and Their Comparative Breakdown Across Regions, (2025). https://moldstud.com/articles/p-an-in-depth-analysis-of-global-software-development-rates-and-their-comparative-breakdown-across-regions.
[193] M.M. Eusuff, K.E. Lansey, Optimization of water distribution network design using the shuffled frog leaping algorithm, J. Water Resour. Plan. Manag. 129 (2003) 210–225.
[194] X. Cui, Multi-objective evolutionary algorithms and their applications, Natl. Def. Ind. Press. Beijing (2006) 88–92.
[195] A. Hidalgo-Paniagua, M.A. Vega-Rodríguez, J. Ferruz, N. Pavón, MOSFLA-MRPP: Multi-Objective Shuffled Frog-Leaping Algorithm applied to Mobile Robot Path Planning, Eng. Appl. Artif. Intell. (2015). https://doi.org/10.1016/j.engappai.2015.05.011.
[196] F.E. Usman-Hamza, A.O. Balogun, S.K. Nasiru, L.F. Capretz, H.A. Mojeed, S.A. Salihu, A.G. Akintola, M.A. Mabayoje, J.B. Awotunde, Empirical analysis of tree-based classification models for customer churn prediction, Sci. African 23 (2024) e02054. https://doi.org/10.1016/j.sciaf.2023.e02054.
[197] B.J. Odejide, A.O. Bajeh, A.O. Balogun, Z.O. Alanamu, K.S. Adewole, A.G. Akintola, S.A. Salihu, F.E. Usman-Hamza, H.A. Mojeed, An Empirical Study on Data Sampling Methods in Addressing Class Imbalance Problem in Software Defect Prediction, Lect. Notes Networks Syst. 501 LNNS (2022) 594–610. https://doi.org/10.1007/978-3-031-09070-7_49.
[198] Y.A. Alsariera, A.O. Balogun, V.E. Adeyemo, O.H. Tarawneh, H.A. Mojeed, Intelligent Tree-Based Ensemble Approaches for Phishing Website Detection, J. Eng. Sci. Technol. 17 (2022) 563–582.
[199] T.E. Dagogo-George, H.A. Mojeed, A.O. Balogun, M.A. Mabayoje, S.A. Salihu, Tree-based homogeneous ensemble model with feature selection for diabetic retinopathy prediction, J. Teknol. Dan Sist. Komput. 8 (2020) 297–303. https://doi.org/10.14710/jtsiskom.2020.13669.
[200] A.O. BAJEH, H.O. ADELEKE, H.A. MOJEED, A.O. BALOGUN, O.C. ABIKOYE, F.E. USMAN-HAMZA, Ensemble models for predicting warts treatment methods, J. Eng. Sci. Technol. 16 (2021) 1030–1052.
[201] Chaya, Random Forest Regression, (2020). https://levelup.gitconnected.com/random-forest-regression-209c0f354c84 (accessed June 8, 2023).
[202] Y. Park, Concise Logarithmic Loss Function for Robust Training of Anomaly Detection Model, ArXiv Prepr. arXiv:2201 (2022). http://arxiv.org/abs/2201.05748.
[203] T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit, A Simulation Study of the Model Evaluation Criterion MMRE, IEEE Trans. Softw. Eng. 29 (2003) 985–995. https://doi.org/10.1109/TSE.2003.1245300.
[204] J. Karch, Improving on adjusted R-squared, Collabra Psychol. 6 (2020) 1–11. https://doi.org/10.1525/collabra.343.
[205] V. Kumar, S. Kumar, A.K. Singh, Outlier detection: a clustering-based approach, Int. J. Sci. Mod. Eng. 1 (2013) 16–19.
[206] H.A. Mojeed, Dataset for estimating overtime allocation in Software Development Projects, (2025). https://doi.org/10.21227/ksjx-t330.
[207] G. James, D. Witten, T. Hastie, R. Tibshirani, others, An introduction to statistical learning, Springer, 2013.
[208] M. Kuhn, K. Johnson, others, Applied predictive modeling, Springer, 2013.
[209] B.F.F. Huang, P.C. Boutros, The parameter sensitivity of random forests, BMC Bioinformatics 17 (2016) 1–13. https://doi.org/10.1186/s12859-016-1228-x.
[210] P. Probst, M.N. Wright, A.L. Boulesteix, Hyperparameters and tuning strategies for random forest, Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 9 (2019) 1–15. https://doi.org/10.1002/widm.1301.
[211] S.K. Jain, A.K. Gupta, Application of Random Forest Regression with Hyper-parameters Tuning to Estimate Reference Evapotranspiration, Int. J. Adv. Comput. Sci. Appl. 13 (2022) 742–750. https://doi.org/10.14569/IJACSA.2022.0130585.
[212] D. Navon, A.M. Bronstein, Random Search Hyper-Parameter Tuning: Expected Improvement Estimation and the Corresponding Lower Bound, (2022). http://arxiv.org/abs/2208.08170.
[213] G. Youness, N. Uyen, T. Phan, B.C. Boulakia, G. Youness, N. Uyen, T. Phan, B. Cohen, B. Bootbogs, G. Youness, BootBOGS : Hands-on optimizing Grid Search in hyperparameter tuning of MLP To cite this version : HAL Id : hal-04396690 BootBOGS : Hands-on optimizing Grid

Search in hyperparameter tuning of MLP, in: 20th ACS/IEEE Int. Con- Ference Comput. Syst. Appl. ACS/IEEE Int. Conf. Comput. Syst. Appl., Giza, Egypt, 2023.
[214] E. Merrill, A. Fern, X. Fern, N. Dolatnia, An empirical study of bayesian optimization: Acquisition versus partition, J. Mach. Learn. Res. 22 (2021) 1–25.
[215] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. 18 (2018) 1–52.
[216] A.H. Yazdani-Abyaneh, M. Krunz, Automatic Machine Learning for Multi-Receiver CNN Technology Classifiers, WiseML 2022 - Proc. 2022 ACM Work. Wirel. Secur. Mach. Learn. (2022) 39–44. https://doi.org/10.1145/3522783.3529524.
[217] Z. Karnin, T. Koren, O. Somekh, Almost optimal exploration in multi-armed bandits, 30th Int. Conf. Mach. Learn. ICML 2013 28 (2013) 2275–2283.
[218] K. Jamieson, A. Talwalkar, Non-stochastic best arm identification and hyperparameter optimization, Proc. 19th Int. Conf. Artif. Intell. Stat. AISTATS 2016 (2016) 240–248.
[219] D.S. Soper, Hyperparameter Optimization Using Successive Halving with Greedy Cross Validation, Algorithms 16 (2023). https://doi.org/10.3390/a16010017.
[220] D.S. Soper, Greed is good: Rapid hyperparameter optimization and model selection using greedy k-fold cross validation, Electron. 10 (2021). https://doi.org/10.3390/electronics10161973.
[221] M. Louhichi, R. Nesmaoui, M. Mbarek, M. Lazaar, Shapley values for explaining the black box nature of machine learning model clustering, Procedia Comput. Sci. 220 (2023) 806–811.
[222] R. Rodriguez-Pérez, J. Bajorath, Interpretation of machine learning models using shapley values: application to compound potency and multi-target activity predictions, J. Comput. Aided. Mol. Des. 34 (2020) 1013–1026.
[223] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, Proc. - Int. Conf. Softw. Eng. (2011) 1–10. https://doi.org/10.1145/1985793.1985795.

APPENDICES

Appendix A: Detailed WP-Level Features Extracted from all Projects

Table A.1. Detailed WP-based features extracted from ACAD project

WP	DEP	Transactions	FP	Duration (days)
Aluno	0	4	25	27
Trancamento	1	3	9	10
Bolsa	1	4	12	13
Professor	1	4	22	24
Area	0	4	19	21
Linha	1	4	12	13
Usuario	0	4	19	21
Disciplina	0	4	19	21
Turma	2	4	22	24
Inscricao	3	4	26	28

Table A.2. WP-based features extracted from WEBMET project

WP	DEP	Transactions	FP	Duration (days)
USUARIO	0	1	10	11
ESTACOES (EMS)	1	4	25	27
ESTACOES (EMA)	1	4	12	13
Centros	1	5	15	17
REGIONAIS	1	5	15	17
CONDICAO DE TEMPO PRESENTE E RECENTE	1	2	6	7
MAQUINAS	1	1	8	9
OBSERVACOES EMA (PTU, WIND)	2	4	13	15
OBSERVACOES EMS	2	2	25	27
RELATORIO	3	16	60	65
PARAMETRO	4	5	36	39

Table A.3. Detailed WP-based features extracted from PSOA project

WP	DEP	Transactions	FP	Duration (days)
Empregado	4	4	33	36
Afastamento	3	4	15	17
Gerencia	3	4	15	17
Lotacao	3	3	10	11
Falta	3	5	25	27
Horaextra	1	4	16	18
AssociaoCargoaEmpregado	4	3	11	12
Estabelecimento	0	3	19	21
CentroDeCusto	1	3	12	13
MotivoDeAfastamento	0	3	19	21
MotivodeFalta	0	3	19	21
MotivoDeTransferencia	0	3	19	21
Verba	0	3	19	21

Cargo	0	3	19	21
SalarioDoCargo	1	3	12	13
ProcessarFolha	4	1	7	8
VerbaParaEmpregado	3	3	12	13
ContraCheques	4	2	8	9

Table A.4. Detailed WP-based features extracted from WEBAMHS project

WP	DEP	Transactions	FP	Duration (days)
PERFIL	0	3	16	18
TERMINAL	0	2	13	15
TIPO DE AERONAVE	1	3	12	13
AERONAVE	0	1	10	11
LOCALIDADE	0	3	19	21
ENDERECOS	1	1	3	4
IDENTIFICADOR GEOGRAFICO	0	3	19	21
CONFIGURACAO	0	3	19	21
AMHS FORMULARIO	2	1	6	7
AIS FORMULARIO	4	6	46	50
ATS FORMULARIO	5	6	101	109
MET FORMULARIO	4	6	89	97
PARAMENTROS DO SISTEMA DE CONFIGURACAO	1	1	10	11
ENDERECOS CAAS	1	1	8	9
EVENTOS DE LOGIN DO SISTEMA	1	1	10	11

Table A.5. Detailed WP-based features extracted from PARM project

WP	DEP	Transactions	FP	Duration (days)
TipoFundamentacaoLegal	0	3	20	22
ReferenciaFundamentacaoLegal	0	3	20	22
AutoridadeEmissora	0	3	20	22
FundamentacaoLegal	3	3	22	24
ImpostoDeRenda	2	5	26	29
AliquotaImpostoDeRenda	3	4	16	18
ImpostoDeRendaExterior	1	5	24	26
SalarioFamilia	1	5	25	27
FaixaSalarioFamilia	1	4	13	15
ParametroLegal	0	3	15	17
PrevalenciaConsignacao	1	6	27	30
IndiceReajusteRGPS	1	5	24	26
Teto	0	3	22	24
TipoTeto	1	3	15	17
SalarioContribuicao	1	4	27	30
SalarioMinimoRegionalizado	1	2	13	15

LocaisdeTrabalho	0	2	6	7
EspecieBeneficio	0	2	11	12
Tratamento	0	2	11	12
FatorAtualizacao	0	2	11	12
Rubrica	0	2	11	12
TaxaConversaoMoeda	0	2	11	12
CotacaoFatorAtualizacao	0	2	11	12
GrupoBeneficio	0	2	11	12
CoeficienteReajustamentoBaseSM	0	2	13	15
IndiceReajusteBuracoNegro	0	2	13	15
SalarioMinimoGrupoBeneficio	0	2	13	15

Table A.6. Detailed WP-based features extracted from OPMET project

WP	DEP	Transactions	FP	Duration (days)
USUARIO	0	5	23	25
MENSAGENS	5	10	71	77
LOCALIDADE	1	5	37	40
FIR	0	5	25	27
SINOTICA	0	5	32	35
AREA	0	5	22	24
LOCALIDADE_GRUPO	0	5	22	24
GRUPO FIR	0	5	22	24
GRUPO DE ESTACOES SINOTICAS	0	5	22	24
GRUPO DE ENDEREÇOS	0	5	22	24
REGIÃO	0	5	22	24
PAÍS	0	4	19	21
REMETENTE	0	4	19	21
AVISO_AUTOMATICO	3	6	28	31
ASSOCIACAO_REMETENTES	1	4	12	13
ASSOCIACAO_LOCALIDADES	1	4	12	13
ASSOCIACAO_ESTACOES_SINOTICAS	1	4	12	13
REMETENTE AFTN	1	5	15	17
ADMINISTRAÇÃO	3	4	30	33
WAFS	0	4	12	13
RELATÓRIO	11	29	156	169

Appendix B: Sample of solutions generated for all software projects

Table B.1. Sample of selected solutions instances produced by Multi-objective random search for ACAD project

WP1, ….., WP10	Overtime	Cost	Error
3, 0, 0, 1, 0, 0, 1, 0, 0, 0	126	2490.885417	1.3
2, 3, 2, 2, 2, 0, 0, 0, 0, 0	200	2584.635417	2.5
0, 4, 0, 1, 0, 3, 0, 2, 0, 1	173	2592.447917	3.1
3, 4, 1, 0, 0, 0, 1, 0, 2, 0	203	2592.447917	3.1
4, 1, 3, 1, 0, 1, 0, 1, 0, 0	215	2592.447917	3.1
0, 3, 0, 1, 2, 0, 0, 0, 3, 2	224	2588.541667	2.8
1, 1, 1, 0, 2, 1, 3, 0, 1, 1	220	2584.635417	2.5
2, 4, 2, 2, 2, 0, 0, 0, 0, 0	210	2604.166667	3
2, 2, 2, 2, 2, 0, 0, 1, 1, 0	235	2596.354167	2.4
2, 2, 2, 2, 3, 0, 0, 0, 1, 0	235	2600.260417	2.7
2, 2, 4, 2, 2, 0, 0, 0, 0, 0	216	2604.166667	3
2, 4, 2, 2, 2, 0, 0, 0, 0, 0	210	2604.166667	3
2, 2, 2, 2, 2, 0, 0, 1, 1, 0	235	2596.354167	2.4
2, 2, 2, 3, 2, 0, 0, 1, 0, 0	235	2600.260417	2.7
3, 2, 3, 2, 2, 0, 0, 0, 0, 0	230	2604.166667	3
2, 2, 2, 2, 2, 0, 1, 0, 0, 1	239	2596.354167	2.4
2, 2, 2, 2, 2, 1, 1, 0, 0, 0	224	2596.354167	2.4
0, 1, 2, 1, 3, 4, 0, 1, 0, 0	196	2608.072917	3.3
1, 0, 2, 2, 0, 4, 0, 1, 0, 2	230	2604.166667	3
2, 2, 2, 2, 2, 1, 1, 0, 1, 0	248	2611.979167	2.6

Table B.2. Sample of selected solutions instances produced by Multi-objective random search for WEBMET project

WP1, ….., WP11	Overtime	Cost	Error
1, 2, 1, 1, 1, 1, 1, 1, 1, 2	313	3132.813	2.6
1, 1, 1, 1, 1, 1, 2, 1, 1, 2	295	3132.813	2.6
1, 1, 1, 1, 1, 1, 2, 1, 1, 1	256	3117.188	2.4
2, 1, 1, 2, 1, 1, 1, 1, 1, 1	275	3132.813	2.6
2, 1, 1, 1, 1, 1, 1, 1, 1, 2	297	3132.813	2.6
2, 1, 1, 1, 1, 1, 1, 2, 2, 1	300	3148.438	2.8
1, 1, 1, 1, 1, 1, 1, 2, 1, 2	327	3132.813	2.6
2, 1, 1, 2, 2, 1, 1, 1, 1, 1	292	3148.438	2.8
1, 2, 1, 1, 1, 2, 1, 1, 1, 1	281	3132.813	2.6
1, 1, 2, 1, 1, 1, 2, 1, 2, 1	296	3148.438	2.8
1, 1, 1, 1, 1, 1, 1, 1, 2, 1	312	3117.188	2.4
1, 2, 1, 1, 1, 1, 1, 1, 2, 1	301	3132.813	2.6
1, 1, 1, 2, 1, 1, 1, 1, 2, 1	291	3132.813	2.6
1, 1, 2, 1, 1, 1, 1, 1, 1, 2	299	3132.813	2.6
2, 1, 2, 1, 1, 1, 1, 1, 1, 1	271	3132.813	2.6
2, 1, 1, 2, 0, 2, 0, 3, 0, 1, 1	259	3136.719	2.9

2, 0, 2, 1, 1, 0, 2, 2, 2, 1, 0	249	3132.813	2.6
0, 1, 4, 0, 0, 1, 0, 3, 0, 1, 1	235	3113.281	3.1
2, 2, 4, 0, 1, 0, 0, 1, 3, 0, 0	241	3144.531	3.5
1, 4, 0, 3, 0, 3, 0, 0, 1, 0, 0	218	3132.813	3.6

Table B.3: Sample selected solutions instances produced by Multi-objective random search for PSOA project

WP1, ….., WP18	Overtime	Cost	Error
2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1	285	3140.625	3.2
1, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2	292	3148.438	2.8
1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1	226	3156.25	3.4
1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1	296	3148.438	2.8
1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1	282	3148.438	2.8
2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1	209	3105.469	2.5
2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 1	291	3132.813	2.6
1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2	265	3132.813	2.6
1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1	263	3132.813	2.6
1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1	304	3148.438	2.8
1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1	239	3136.719	2.9
1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2	300	3148.438	2.8
1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1	287	3132.813	2.6
0, 3, 0, 3, 1, 1, 2, 3, 0, 0, 0, 2, 4, 0, 1, 0, 0	259	3140.625	3.2
0, 1, 3, 0, 0, 1, 2, 1, 0, 1, 2, 0, 1, 3, 3, 1, 1	272	3089.844	2.3
0, 1, 3, 2, 1, 4, 2, 0, 2, 1, 1, 0, 1, 2, 0, 1, 1	283	3132.813	2.6
0, 1, 0, 0, 0, 1, 0, 0, 1, 2, 2, 2, 2, 4, 2, 2, 2	304	3148.438	2.8
0, 0, 0, 1, 0, 1, 0, 0, 0, 2, 2, 2, 2, 3, 3, 2, 2	298	3148.438	2.8
1, 0, 0, 1, 0, 1, 0, 0, 0, 2, 2, 2, 2, 2, 2, 3, 2	285	3132.813	2.6
0, 0, 0, 0, 0, 1, 1, 0, 0, 2, 2, 2, 3, 3, 2, 3, 2	263	3132.813	2.6

Table B.4. Sample selected solutions instances produced by Multi-objective random search for WEBAMHS project

WP1, ….., WP15	Overtime	Cost	Error
2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1	479	5273.438	4
1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2	477	5273.438	4
1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1	437	5226.563	3.4
1, 1, 2, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2	481	5273.438	4
1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1	488	5273.438	4
1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2	500	5242.188	3.6
1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1	545	5242.188	3.6
1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1	495	5273.438	4
2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1	479	5257.813	3.8

2, 3, 2, 2, 2, 2, 2, 0, 1, 0, 0, 0, 0, 1, 1	248	5246.094	3.9
2, 2, 2, 2, 3, 3, 3, 0, 0, 0, 1, 0, 1, 0, 0	372	5269.531	4.7
0, 1, 2, 2, 0, 2, 2, 2, 1, 0, 1, 0, 2, 2, 4	355	5296.875	4.8
1, 2, 0, 3, 0, 1, 0, 4, 1, 0, 1, 0, 0, 1, 2	316	5222.656	4.1
3, 0, 3, 1, 0, 0, 1, 2, 2, 1, 1, 1, 0, 0, 1	448	5218.75	3.8
3, 0, 3, 0, 1, 0, 2, 2, 1, 0, 0, 2, 1, 2, 1	439	5250	4.2
0, 1, 1, 3, 4, 0, 1, 1, 1, 2, 0, 0, 1, 0, 0	305	5207.031	3.9
1, 3, 1, 2, 2, 3, 1, 0, 0, 0, 0, 0, 3, 0, 2	228	5253.906	4.5
0, 0, 4, 1, 0, 2, 3, 0, 0, 3, 0, 1, 0, 1, 1	401	5226.563	4.4
0, 2, 0, 2, 3, 4, 2, 1, 2, 0, 1, 0, 1, 2, 0	346	5285.156	4.9
2, 3, 3, 2, 3, 3, 2, 0, 1, 1, 0, 0, 0, 0, 0	316	5289.063	5.2

Table B.5. Sample selected solutions instances produced by Multi-objective random search for PARM project

WP1, ….., WP27	Overtime	Cost	Error
1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2	691	6450.521	7.4
2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1	671	6434.896	7.2
3, 4, 0, 0, 0, 1, 3, 0, 3, 0, 1, 0, 2, 1, 0, 0, 0, 2, 2, 0, 2, 0, 2, 3, 0, 0, 4	582	6419.271	9
0, 1, 2, 1, 1, 1, 0, 1, 0, 3, 2, 0, 3, 0, 0, 2, 4, 4, 0, 1, 3, 1, 1, 0, 0, 0, 0	525	6384.115	8.3
1, 3, 1, 2, 0, 3, 0, 1, 0, 4, 1, 0, 1, 0, 0, 1, 2, 1, 3, 3, 2, 0, 1, 0, 0, 3, 1	570	6430.99	8.9
1, 0, 1, 0, 0, 2, 1, 4, 4, 1, 1, 1, 1, 2, 0, 1, 1, 0, 0, 0, 3, 4, 1, 2, 2, 0, 2	607	6446.615	9.1
0, 2, 4, 1, 0, 0, 1, 1, 3, 4, 0, 3, 0, 1, 0, 0, 4, 1, 0, 2, 3, 0, 0, 3, 0, 1, 0	568	6442.708	9.8
1, 0, 0, 2, 0, 1, 2, 0, 2, 3, 0, 0, 1, 2, 2, 0, 0, 1, 0, 3, 1, 2, 0, 3, 2, 4, 1	564	6407.552	8.1
1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 2, 3, 2, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2	610	6477.865	8.5
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 2, 3, 2, 2, 4, 2, 3, 2, 2, 2, 3, 3, 2, 2	563	6473.958	9.2
1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 1, 1	654	6434.896	7.2
1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2	624	6388.021	6.6
1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 1, 2	627	6419.271	7
1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1	671	6434.896	7.2
1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1	692	6450.521	7.4
1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1	647	6434.896	7.2
2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 2, 2	689	6450.521	7.4
1, 1, 2, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1	660	6419.271	7
1, 1, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 1, 1, 2	673	6450.521	7.4
1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1	692	6450.521	7.4

Table B.6. Sample selected solutions instances produced by Multi-objective random search for OPMET project

WP1, ….., WP21	Overtime	Cost	Error
2, 1, 1, 0, 0, 0, 1, 2, 1, 1, 1, 1, 2, 0, 4, 0, 0, 4, 1, 2	891	8690.104	6.4
1, 2, 3, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 1, 0, 2	688	8584.635	4.3
0, 0, 4, 0, 0, 2, 1, 0, 1, 0, 1, 3, 1, 1, 3, 3, 0, 0, 1, 4	727	8701.823	7.3
2, 2, 2, 2, 3, 2, 2, 3, 2, 2, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0	794	8682.292	5.8
2, 2, 2, 2, 2, 2, 3, 4, 3, 2, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0	806	8705.729	6.6
3, 2, 2, 2, 2, 2, 3, 2, 3, 2, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0	803	8701.823	6.3
3, 3, 2, 2, 3, 2, 2, 2, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0	866	8690.104	6.4
2, 2, 2, 3, 2, 2, 4, 2, 2, 2, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0	824	8701.823	6.3
1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1	918	8705.729	5.6
1, 1, 2, 1, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1	872	8690.104	5.4
1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2	848	8705.729	5.6
1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1	880	8721.354	5.8
2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1	773	8674.479	5.2
2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2	857	8721.354	5.8
1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1	844	8674.479	5.2
1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2	868	8721.354	5.8
2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 2	851	8721.354	5.8
2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2	912	8705.729	5.6
2, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2	870	8721.354	5.8
2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1	928	8705.729	5.6

Appendix C: Regression plots for ML models

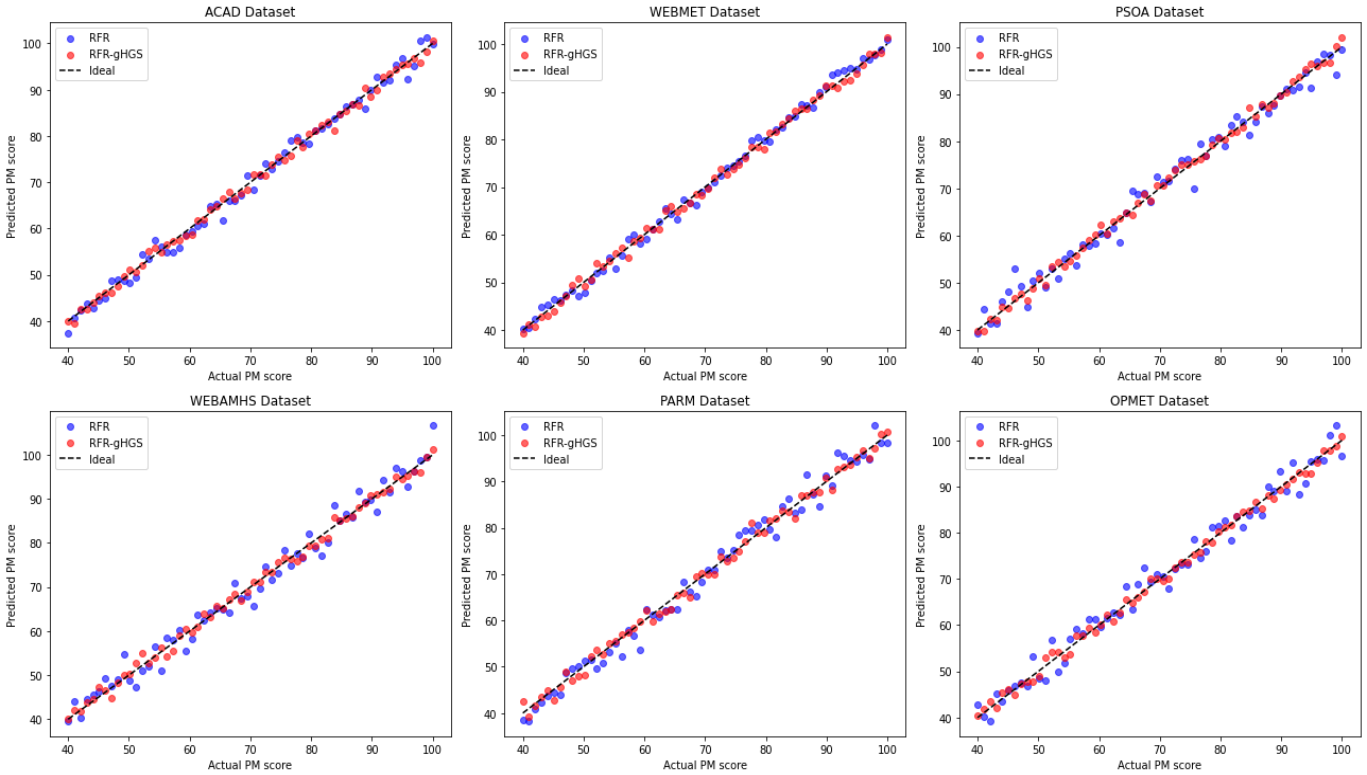


Figure C.1. Scatter plots showing regression results of RFR and RFR-gHGS in all projects

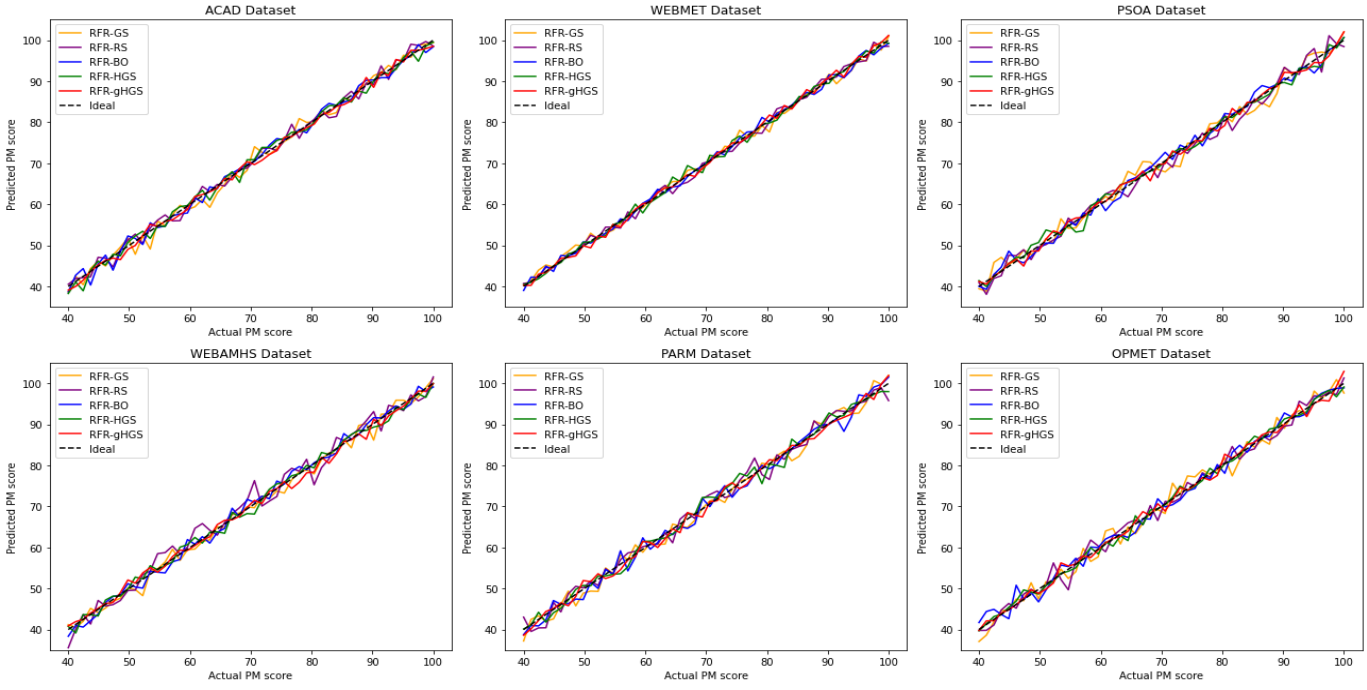


Figure C.2. Regression plots for RFR-gHGS and other hyperparameter optimization methods across all projects

LIST OF FIGURES

LIST OF TABLES