BenchCouncil Transactions on Benchmarks, Standards and Evaluations

TBench
Volume 4, Issue 2
2024

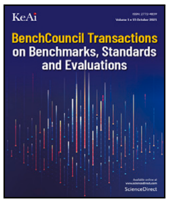
ISSN: 2772-4859

Contents

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/



Editorial

A short summary of evaluatology: The science and engineering of evaluation

Jianfeng Zhan [*]

The International Open Benchmark Council, DE, USA
ICT, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Keywords:
Evaluation
Benchmark
Evaluation system
Evaluation condition
Equivalent evaluation condition
Evaluatology

ABSTRACT

Evaluation is a crucial aspect of human existence and plays a vital role in each field. However, it is often approached in an empirical and ad-hoc manner, lacking consensus on universal concepts, terminologies, theories, and methodologies. This lack of agreement has significant consequences. This article aims to formally introduce the discipline of evaluatology, which encompasses the science and engineering of evaluation. The science of evaluation addresses the fundamental question: "Does any evaluation outcome possess a true value?" The engineering of evaluation tackles the challenge of minimizing costs while satisfying the evaluation requirements of stakeholders. To address the above challenges, we propose a universal framework for evaluation, encompassing concepts, terminologies, theories, and methodologies that can be applied across various disciplines, if not all disciplines.

This is a short summary of Evaluatology (Zhan et al., 2024). The objective of this revised version is to alleviate the readers' burden caused by the length of the original text. Compared to the original version (Zhan et al., 2024), this revised edition clarifies various concepts like evaluation systems and conditions and streamlines the concept system by eliminating the evaluation model concept. It rectifies errors, rephrases fundamental evaluation issues, and incorporates a case study on CPU evaluation (Wang et al., 2024). For a more comprehensive understanding, please refer to the original article (Zhan et al., 2024). If you wish to cite this work, kindly cite the original article.

Jianfeng Zhan, Lei Wang, Wanling Gao, Hongxiao Li, Chenxi Wang, Yunyou Huang, Yatao Li, Zhengxin Yang, Guoxin Kang, Chunjie Luo, Hainan Ye, Shaopeng Dai, Zhifei Zhang (2024). Evaluatology: The science and engineering of evaluation. BenchCouncil Transactions on Benchmarks, Standards and Evaluations, 4(1), 100162.

1. The motivation: why it is essential to establish the science and engineering of evaluation

Evaluation is a crucial aspect of human existence and plays a vital role in each field. However, it is often approached in an empirical and ad-hoc manner, lacking consensus on universal concepts, terminologies, theories, and methodologies. This lack of agreement has significant consequences. Even within computer sciences and engineering, it is not uncommon for evaluators to generate greatly divergent evaluation outcomes for the same individual or system[1] under scrutiny, which we refer to as the *subject*. These discrepancies can range from significant variations to the extent of yielding contradictory qualitative conclusions.

An example of this phenomenon can be observed when using the industry-standard CPU benchmark suite SPEC CPU2017 to assess the performance of the same processor [4]. Wang et al. [4] used SPEC CPU2017 to evaluate the same X86 processor, an Intel Xeon Gold 5120T, adhering to SPEC's procedures and rules. In the rest of this article, we use the same CPU evaluation experiment. Nonetheless, across various SPEC CPU2017 configurations with alterations in compiler flags and the number of copies/threads, the best and worst outcomes exhibit a notable difference of 86 times. Under the SPEC CPU2017 recommended configuration,[2] just 12 out of 43 SPEC CPU2017 workloads achieved the best performance among the varying configurations. From a measurement or metrology perspective, each procedure and

* Correspondence to: The International Open Benchmark Council, DE, USA.
E-mail address: jianfengzhan.benchcouncil@gmail.com.
URL: https://www.zhanjianfeng.org.

[1] This footnote is quoted from [1]. An individual can be defined as an object described by a given data set. A system is an interacting or interdependent group of individuals, whether of the same or different kinds, forming a unified whole [2,3].

[2] The SPEC CPU2017 recommended configuration sets the compiler flag to '-O3' and the number of threads/copies to the maximum number of hardware threads supported by the CPU.

https://doi.org/10.1016/j.tbench.2024.100175

Available online 12 September 2024

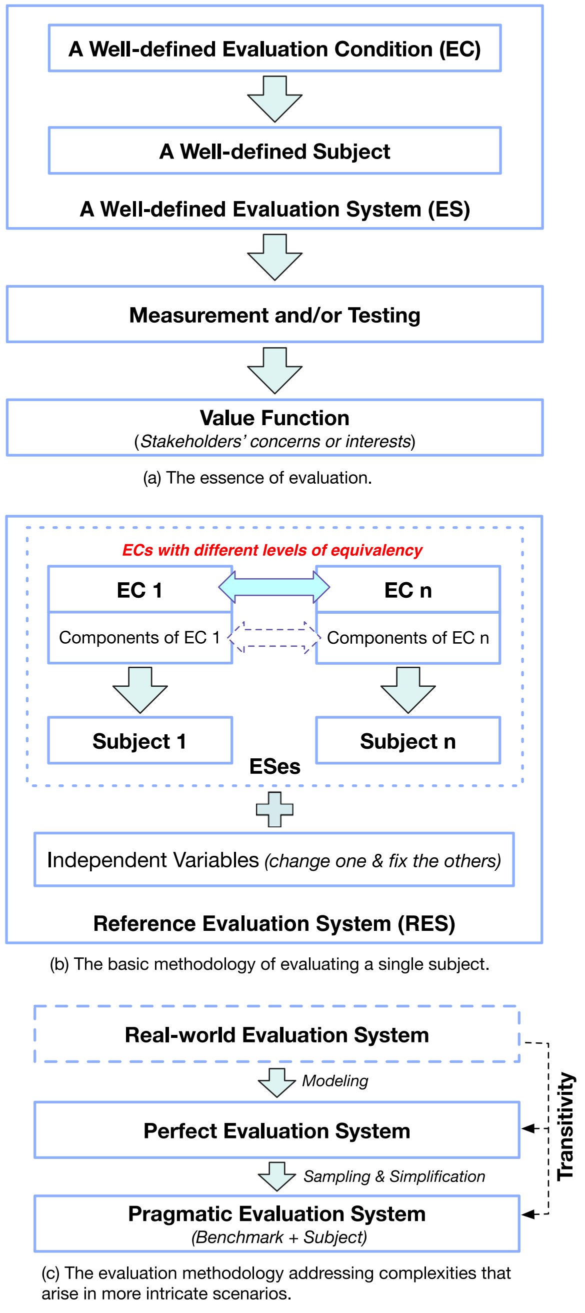

A Well-defined Evaluation Condition (EC)
A Well-defined Subject
A Well-defined Evaluation System (ES)
Measurement and/or Testing
Value Function
(Stakeholders' concerns or interests)
(a) The essence of evaluation.
ECs with different levels of equivalency
EC 1
EC n
Components of EC 1
Components of EC n
Subject 1
Subject n
ESes
Independent Variables (change one & fix the others)
Reference Evaluation System (RES)
(b) The basic methodology of evaluating a single subject.
Real-world Evaluation System
Modeling
Perfect Evaluation System
Sampling & Simplification
Pragmatic Evaluation System
(Benchmark + Subject)
Transitivity
(c) The evaluation methodology addressing complexities that arise in more intricate scenarios.

Fig. 1. The universal concepts, theories, and methodologies in evaluatology.

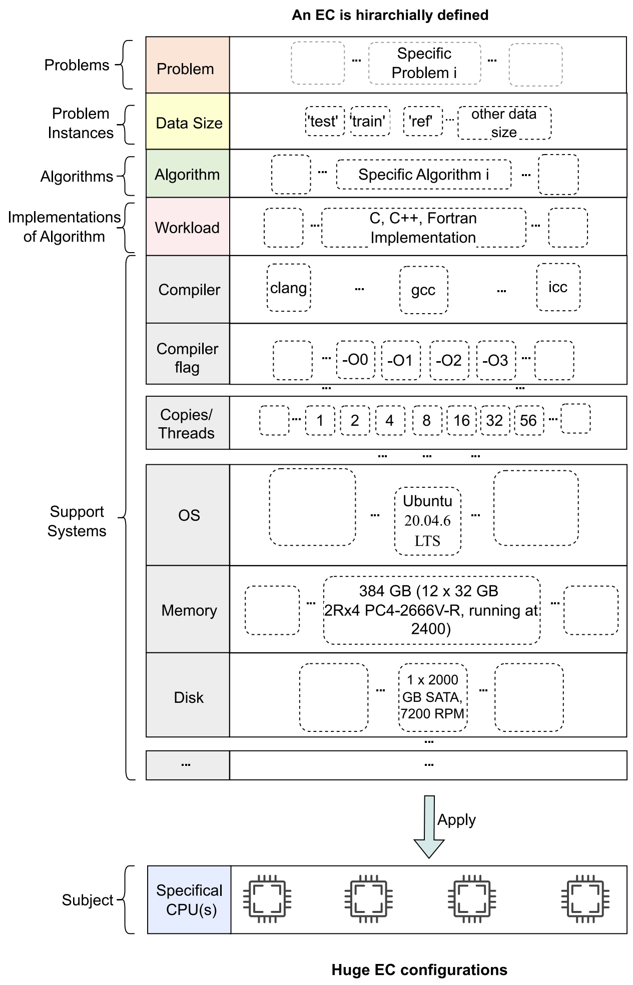


Fig. 2. In the context of CPU evaluation, a well-defined EC possesses huge configurations built on the basis of SPEC CPU2017, an industry-standard CPU benchmark suite. With the permissions of the authors of [4].

the obtained quantity is valid and correct. Nevertheless, the result under a particular configuration can be misleading if the distribution of outcomes across different configurations is not taken into account.

In this context, two fundamental questions naturally emerge: "What is the distinction between evaluation and measurement? Does any evaluation outcome possess a true value?" Such circumstances give rise to valid concerns surrounding these approaches' reliability, effectiveness, and efficiency when appraising the subject that is critical to safety, missions, and businesses.

For the first time, this article aims to formally introduce the discipline of evaluatology, which encompasses the science and engineering of evaluation. The science of evaluation addresses the fundamental question: "Does any evaluation outcome possess a true value?" The engineering of evaluation tackles the challenge of minimizing costs while satisfying the evaluation requirements of stakeholders. To address the above challenges, we propose a universal framework for evaluation, encompassing concepts, terminologies, theories, and methodologies that can be applied across various disciplines, if not all disciplines. Fig. 1 presents the universal concepts, theories, and methodologies in Evaluatology.

2. The science of evaluation

2.1. The essence of evaluation

The challenge in evaluation arises from the inherent fact that evaluating a subject in isolation falls short of meeting the expectations of stakeholders. Instead, it is crucial to create a minimal and well-defined evaluation system (ES) that satisfies the evaluation requirements of stakeholders. Providing the context to evaluate the subject, an ES is a minimum system consisting of the subject and other individuals or systems that are crucial in guaranteeing independent operation and addressing the concerns or interests of the subject's stakeholders.

In other words, evaluation can be seen as an experiment that deliberately applies a well-defined evaluation condition (EC) to a subject to create an ES. Building on the previous discussion, literally, an EC can be understood as the ES with the subject removed. We formally define EC as the minimal context that is crucial in guaranteeing independent operation and addressing the concerns or interests of the subject's stakeholders for evaluating the subject.

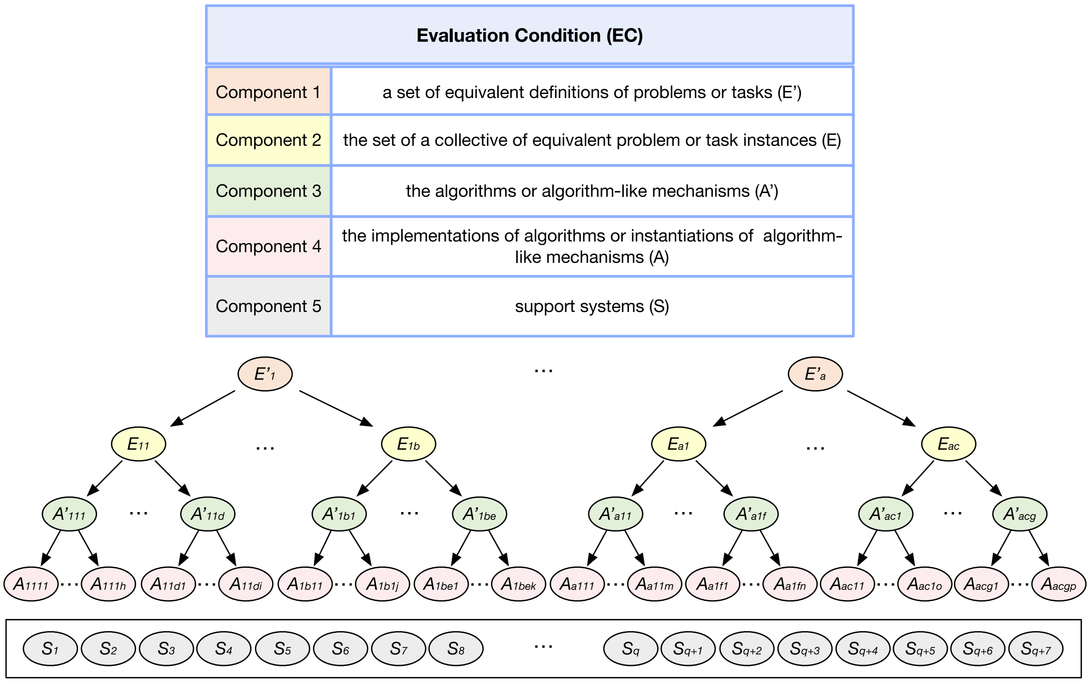

Evaluation Condition (EC)
Component 1 a set of equivalent definitions of problems or tasks (E')
Component 2 the set of a collective of equivalent problem or task instances (E)
Component 3 the algorithms or algorithm-like mechanisms (A')
Component 4 the implementations of algorithms or instantiations of algorithm-like mechanisms (A)
Component 5 support systems (S)
E'1 … E'a
E11 … E1b Ea1 … Eac
A'111 … A'11d A'1b1 … A'1be A'a11 … A'a1f A'ac1 … A'acg
A1111 … A111h A11d1 … A11di A1b11 … A1b1j A1be1 … A1bek Aa111 … Aa11m Aa1f1 … Aa1fn Aac11 … Aac1o Aacg1 … Aacgp
S1 S2 S3 S4 S5 S6 S7 S8 … Sq Sq+1 Sq+2 Sq+3 Sq+4 Sq+5 Sq+6 Sq+7

Fig. 3. The hierarchical definition of an EC.

A well-defined EC possesses huge EC configurations. For example, Fig. 2 presents an EC example built on the basis of SPEC CPU2017, an industry-standard CPU benchmark suite.

In one word, *evaluation is the process of inferring the impact of subjects indirectly within ES that cater to the requirements of stakeholders, relying on objective measurements and/or testing of the latter.*

2.2. Five evaluation axioms

Derived from the essence of evaluation, we propose five axioms focusing on key aspects of evaluation outcomes as the foundational evaluation theory. These axioms serve as the bedrock upon which we build universal evaluation theories and methodologies.

The Axiom of the Essence of Composite Metrics declares that the essence of the composite metric either carries inherent physical significance or is solely dictated by the value function.

The Axiom of True Evaluation Outcomes states that when a well-defined EC is applied to a well-defined subject, the evaluation outcomes possess true values, representing a distribution of value across different EC configurations.

The Axiom of Evaluation Traceability declares that for the same subject, the divergence in the evaluation outcomes can be attributed to disparities in ECs, thereby establishing evaluation traceability.

The Axiom of Comparable Evaluation Outcomes declares when each well-defined subject is equipped with equivalent ECs, their evaluation outcomes are comparable.

The Axiom of Consistent Evaluation Outcomes asserts that when a well-defined subject is evaluated under different configuration samples of a well-defined EC, their evaluation outcomes consistently converge towards the true evaluation outcomes.

2.3. Basic evaluation theory

Based on the five evaluation axioms, we present the universal evaluation theories.

2.3.1. The hierarchical definition of an EC

A well-defined EC serves as a prerequisite for meaningful comparisons and analyses of the subjects. As shown in Fig. 3, we propose a universal hierarchical definition of an EC and identify five primary components of an EC from the top to the bottom.

We start defining an EC from the problems or tasks that these stakeholders face and need to address with the following two reasons. First, the relevant stakeholders' concerns and interests are at the evaluation's core. These concerns and interests are best reflected through the problems or tasks they must face and resolve, which provide a reliable means to define an EC. Secondly, employing the same problem or task provides the necessary but not sufficient method to ensure the comparability of evaluation outcomes. While the problem or task serves as the foundation for the evaluation process, it cannot solely serve as the evaluation itself because it is often abstract and requires further instantiation to determine its specific parameters.

The second component is the set of problem or task instances, each of which is instantiated from a problem or task. Different from the first component, a problem or task instance is specific and could serve as the evaluation directly. After a problem or task is proposed, it is necessary to figure out a solution. The third component consists of the algorithms or algorithm-like mechanisms, each of which provides the solution to a problem or task. An algorithm-like mechanism refers to a process or abstract that operates in a manner similar to an algorithm. The fourth component encompasses the implementation of an algorithm or instantiation of an algorithm-like mechanism, which tackles problem or task instances. The fifth component is support systems that provide necessary resources and environments.

2.3.2. The establishment of EECs

In the process of evaluating subjects, it is of utmost importance to prioritize the use of the equivalent ECs (EECs) across diverse subjects. This means that in order to establish two EECs, it is crucial to ensure that the corresponding components within the same layer of the two ECs are equivalent. By maintaining equivalency at each layer, we can guarantee that evaluation results are not influenced by confounding variables in ECs, allowing for meaningful comparisons and assessments across different subjects.

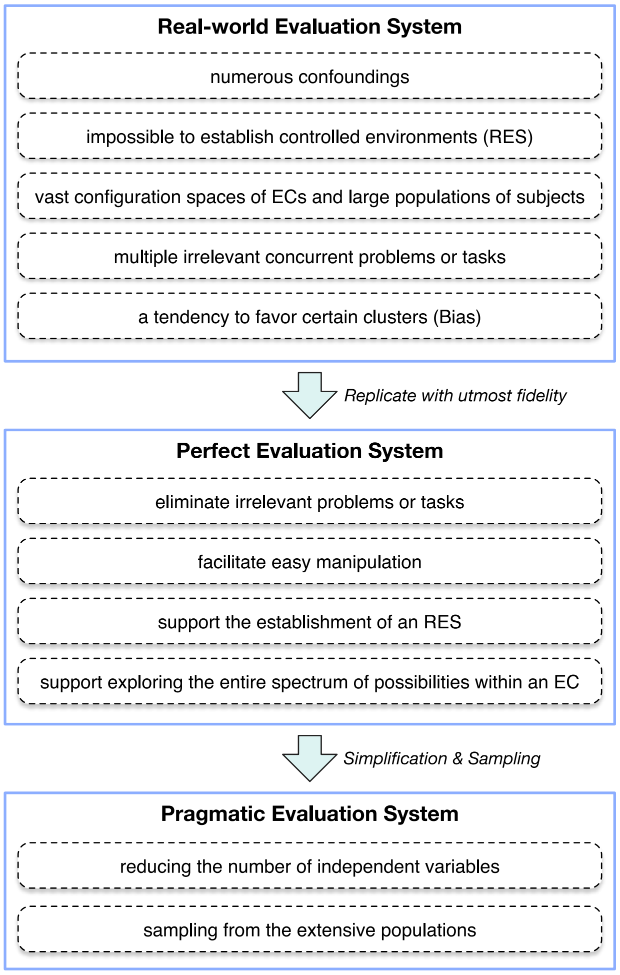

Real-world Evaluation System
numerous confoundings
impossible to establish controlled environments (RES)
vast configuration spaces of ECs and large populations of subjects
multiple irrelevant concurrent problems or tasks
a tendency to favor certain clusters (Bias)
Replicate with utmost fidelity
Perfect Evaluation System
eliminate irrelevant problems or tasks
facilitate easy manipulation
support the establishment of an RES
support exploring the entire spectrum of possibilities within an EC
Simplification & Sampling
Pragmatic Evaluation System
reducing the number of independent variables
sampling from the extensive populations

Fig. 4. Universal evaluation methodology in complex scenarios.

We take the same CPU evaluation example to demonstrate why it is essential to guarantee EECs. In the SPEC CPU2017 workloads, the outcomes vary significantly, ranging from 1.95 to 242.06, across different configurations of SPEC CPU2017. For instance, the workload named *541.leela_r* shows a 242.06-fold difference in outcomes for the Intel Xeon Gold 5120T between a configuration with the '-O3' compiler flag and 56 copies versus another configuration with the '-O0' compiler flag and 1 copy. These significant disparities in evaluation outcomes for the same processor highlight the importance and necessity of EECs.

2.3.3. The establishment of a reference ES

We apply ECs to diverse subjects to constitute ESes. An ES configuration refers to a specific point within the ES configuration space, and each ES configuration has many independent variables. There is a subtle difference between an EC configuration and an ES configuration, as the subject itself also may possess many independent variables.

We propose a new concept named a reference ES (RES) to address confounding variables. An RES mandates that each ES configuration changes only one independent variable at a time, maintaining the other variables as controls. Subsequently, we utilize the measurement and/or testing to gauge the functioning of the RES. Finally, from the amassed measurement and testing data of the ESes, we deduce the cause–effect impacts of the independent variable that we modify.

Similarly, we can define the concept of reference EC (REC). An REC mandates that each EC configuration changes only one independent variable at a time, maintaining the other variables as controls.

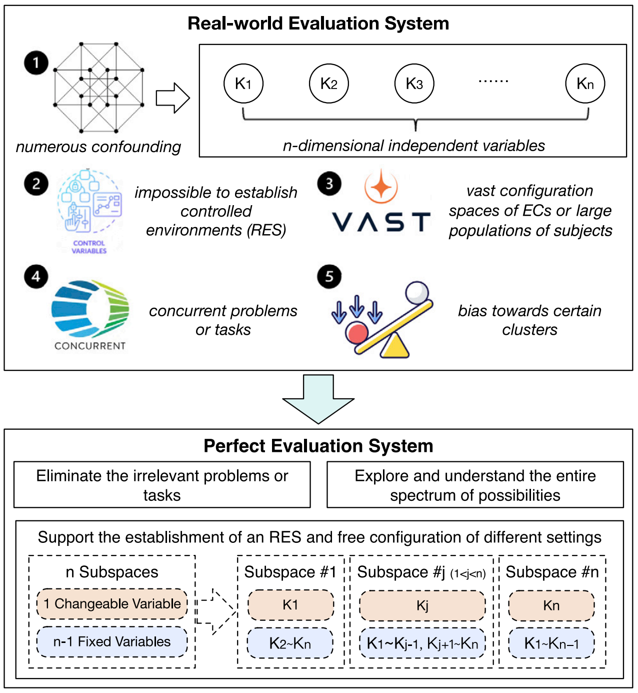

Real-world Evaluation System
1
numerous confounding
K1
K2
K3
······
Kn
n-dimensional independent variables
2
impossible to establish controlled environments (RES)
CONTROL VARIABLES
3
VAST
vast configuration spaces of ECs or large populations of subjects
4
CONCURRENT
concurrent problems or tasks
5
bias towards certain clusters
Perfect Evaluation System
Eliminate the irrelevant problems or tasks
Explore and understand the entire spectrum of possibilities
Support the establishment of an RES and free configuration of different settings
n Subspaces
1 Changeable Variable
n-1 Fixed Variables
Subspace #1
K1
K2~Kn
Subspace #j (1<j<n)
Kj
K1~Kj-1, Kj+1~Kn
Subspace #n
Kn
K1~Kn-1

Fig. 5. A perfect ES resembles a real-world ES.

2.4. Universal evaluation methodology in complex scenarios

Addressing the complexities that arise in more intricate scenarios, we reveal that the key to effective and efficient evaluations in various complex scenarios lies in the establishment of a series of ESes that maintain transitivity (see Fig. 4). In the full original version [1], we have formally defined what transitivity is in a mathematical form.

In real-world settings, we refer to the minimal real-world systems that are used to evaluate specific subjects as the real-world ES. Assuming no safety concerns are present, the real-world ES serves as a prime candidate for creating an optimal evaluation environment, enabling the assessment of diverse subjects. However, there are several significant obstacles to consider, i.e., the presence of numerous confounding variables, the challenges of establishing an RES, prohibitive evaluation costs resulting from the huge configuration spaces, multiple irrelevant concurrent problems or tasks taking place, and the inclination to exhibit bias towards certain clusters within the ES configuration space.

We posit the existence of a perfect ES that replicates the real-world ES with utmost fidelity (see Fig. 5). A perfect ES eliminates irrelevant problems or tasks, has the capability to thoroughly explore and comprehend the entire spectrum of possibilities of an ES, and facilitates the establishment of an RES. However, the perfect ES possesses huge configuration space, entails a vast number of independent variables, and hence results in prohibitive evaluation costs. To address this challenge, it is crucial to propose a pragmatic ES that simplifies the perfect ES in two ways: reducing the number of independent variables that have negligible effect and sampling from the extensive configuration space. A pragmatic ES provides a means to estimate the parameters of the real-world ES.

Literally, a real-world, perfect, or pragmatic EC can be understood as the corresponding ES without the subject included.

2.5. Four fundamental issues in evaluatology

We put forth four fundamental issues in the discipline of evaluatology.

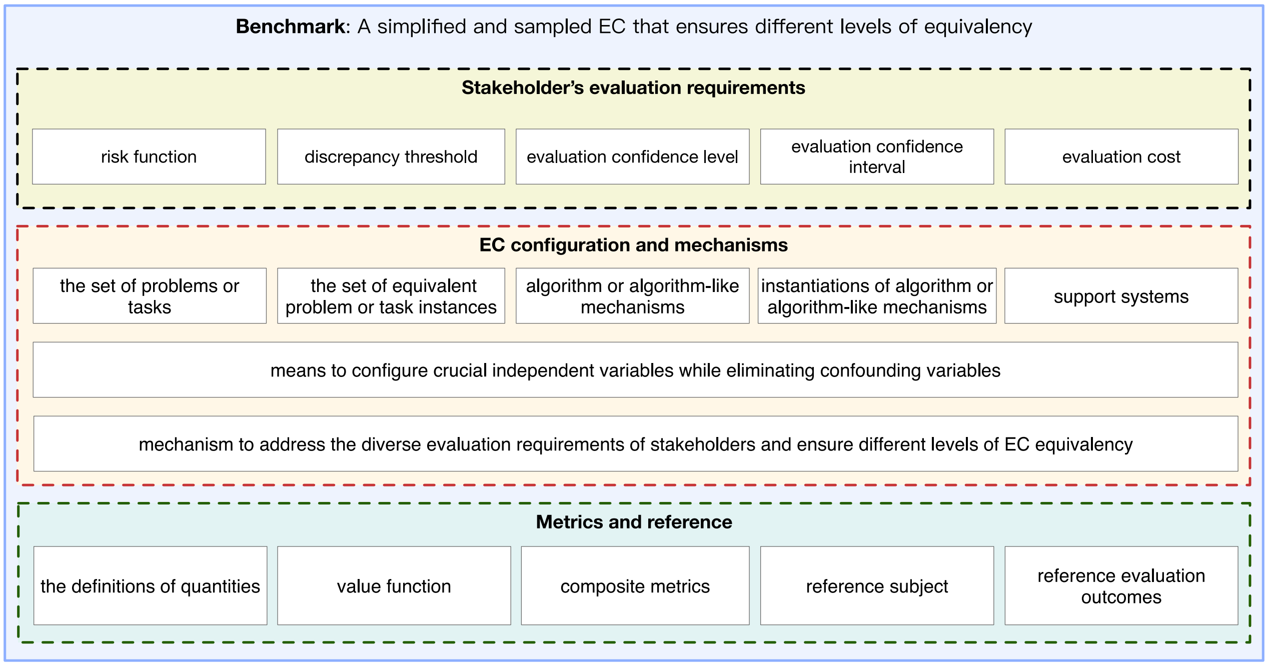

Benchmark: A simplified and sampled EC that ensures different levels of equivalency
Stakeholder's evaluation requirements
risk function
discrepancy threshold
evaluation confidence level
evaluation confidence interval
evaluation cost
EC configuration and mechanisms
the set of problems or tasks
the set of equivalent problem or task instances
algorithm or algorithm-like mechanisms
instantiations of algorithm or algorithm-like mechanisms
support systems
means to configure crucial independent variables while eliminating confounding variables
mechanism to address the diverse evaluation requirements of stakeholders and ensure different levels of EC equivalency
Metrics and reference
the definitions of quantities
value function
composite metrics
reference subject
reference evaluation outcomes

Fig. 6. A benchmark comprises three essential constituents.

First and foremost, establishing the EC that can yield true and consistent evaluation outcomes for the same subject stands as the cornerstone of evaluatology. Moreover, ensuring the transitivity of ECs is equally crucial in complex scenarios, transitioning from a real-world EC to perfect and pragmatic ECs.

Secondly, ensuring the evaluation outcomes are comparable for different subjects stands as another cornerstone of evaluatology. Given the formidable task of ensuring EECs and the potential multitude of independent variables within an EC and a subject, mitigating the adverse impacts of confounding variables poses a significant challenge.

Thirdly, the critical engineering challenge in implementing evaluation processes is determining how to conduct a cost-effective evaluation while maintaining controlled outcome discrepancies. That is how to strike a balance between ensuring the discrepancy threshold of the evaluation outcomes and managing the associated costs.

Fourthly, how to ensure evaluation traceability is a multifaceted issue that requires the application of both scientific and engineering principles. It involves attributing any discrepancy in evaluation outcomes to disparities in the underlying ECs and subjects, thereby establishing clear and transparent traceability.

3. The engineering of evaluation

Benchmarks are extensively employed across various disciplines, albeit lacking a formal definition. Based on the science of evaluation, we propose a precise delineation of a benchmark as *an EC. In reality, it could be a simplified and sampled EC, specifically a pragmatic EC, that ensures different levels of equivalency.* Based on this concept, we propose a benchmark-based universal engineering of evaluation across different disciplines.

Within the framework of this definition, a benchmark comprises three essential constituents. The first constituent is the *stakeholder's evaluation requirements*, which encompass various factors. These include the risk function, which evaluates the potential risks associated with the benchmark. Additionally, the discrepancy threshold, which determines the acceptable level of deviation from the true evaluation outcomes, is considered. The evaluation confidence level and evaluation confidence interval play a crucial role in predicting the parameter of a perfect ES. Lastly, the evaluation cost is taken into account, and the resources required for conducting the evaluation are assessed. By considering these elements, the benchmark can effectively address the evaluation requirements of stakeholders.

The second constituent of the benchmark framework is the *EC configuration and mechanisms*. This includes several elements crucial for the benchmark's effectiveness. Firstly, it involves defining the set of problems or tasks the stakeholders face when addressing. Additionally, it encompasses the set of problem or task instances, which helps ensure specificity in the evaluation process. The benchmark also considers algorithms or algorithm-like mechanisms, which play a significant role in solving the defined problems or tasks, and includes their instantiations. The support systems, which provide necessary resources and environments, are also taken into account.

Moreover, the benchmark provides the means to configure crucial independent variables while eliminating confounding variables that could potentially impact the evaluation outcomes. Also, the benchmark provides the mechanism to address the diverse evaluation requirements of stakeholders. For example, it ensures different levels of EC equivalency, determining the extent to which different benchmark instances can be considered equivalent.

By considering these EC configurations and mechanisms, the benchmark can provide a comprehensive and standardized approach to different evaluation issues.

The third constituent is the *metrics and reference*, including the definitions of quantities, the value function, composite metrics, the reference subject, and the reference evaluation outcomes.

In the subsequent sections of this article, we will refer to these three constituents as the complete constituents of a benchmark. Fig. 6 shows the three essential constituents of a benchmark.

4. A case study of CPU evaluation

Three fundamental steps are involved in our examination of CPU evaluation as a case study in evaluatology. The first step is to delineate the EC and the subject. The second step entails applying the well-defined EC to the subject to establish the ES. Finally, the third step focuses on attaining consistent and comparable evaluation outcomes.

In CPU evaluation, a specific CPU, a well-defined subject, includes components such as decoders, issue queues, arithmetic logic units (ALUs), branch predictors, reorder buffers (ROBs), and caches.

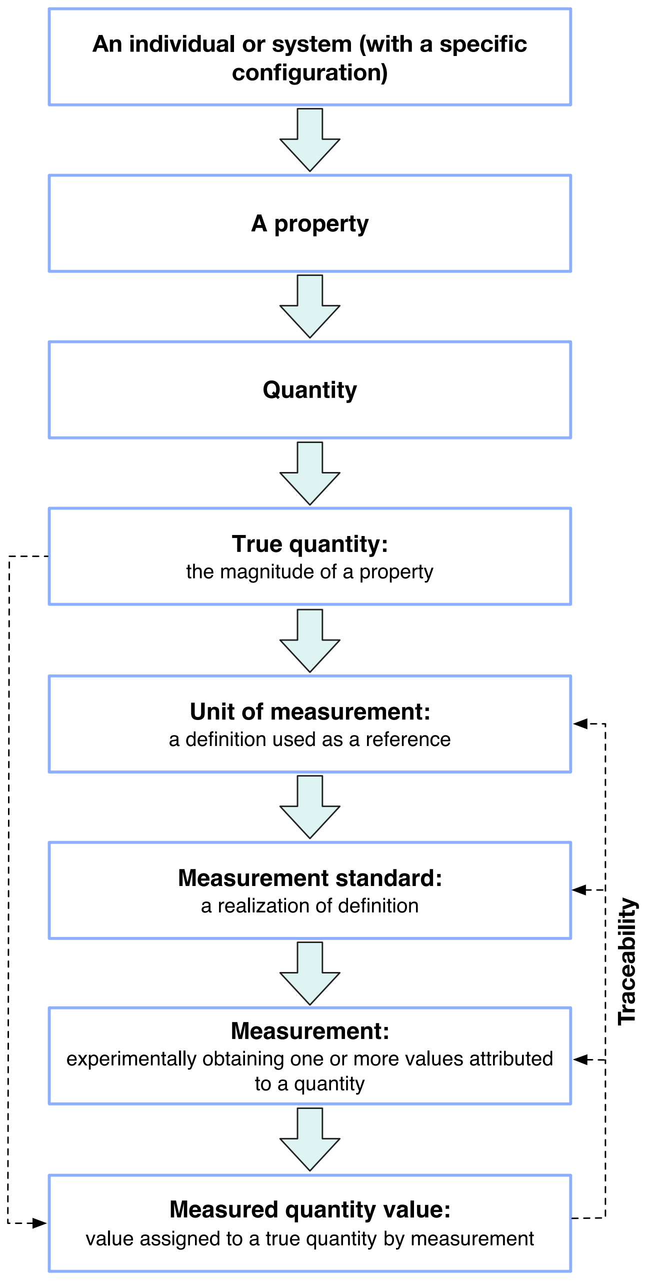

An individual or system (with a specific configuration)
A property
Quantity
True quantity: the magnitude of a property
Unit of measurement: a definition used as a reference
Measurement standard: a realization of definition
Measurement: experimentally obtaining one or more values attributed to a quantity
Measured quantity value: value assigned to a true quantity by measurement
Traceability

Fig. 7. A simplified yet systematic conceptual framework for metrology [5,6].

As shown in Fig. 2, a well-defined EC consists of five components. At the highest level are the problems that are of concern to stakeholders, such as arithmetic operations. The next level is the problem instances, which provide data variables and constraints of the problem, such as the arithmetic operations on two 32-bit unsigned integers n_1 and n_2. The third level is the algorithms, an abstract representation of the solution for the given problem, which can be described in natural language or pseudocode. The next level is algorithm implementations, which involve coding the algorithms in a specific programming language supported by the computer, such as a C or Java program. The final level is the support systems, which encompass all environments or configurations required to run a program on a specific CPU, including but not limited to the compilers, compiler flags, copies/threads, OSes, OS settings, memories, memory settings, disks, and disk settings.

When a well-defined EC is applied to a specific CPU, the true evaluation outcome emerges as a distribution of outcomes across various EC configurations. Under a specific EC configuration, the true outcome can

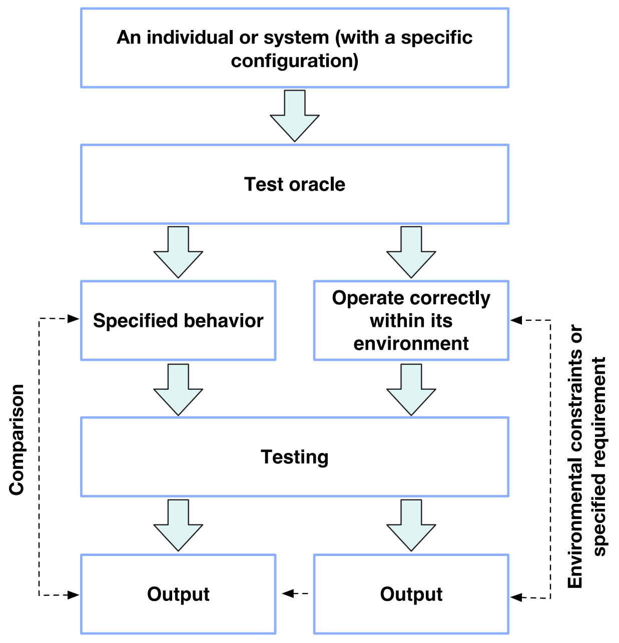

An individual or system (with a specific configuration)
Test oracle
Specified behavior
Operate correctly within its environment
Testing
Output
Output
Comparison
Environmental constraints or specified requirement

Fig. 8. A simplified yet systematic conceptual framework for testing [7,8].

be estimated by taking the mean or median of multiple experiments. Furthermore, under a sample of configurations of a well-defined EC, the true evaluation outcomes can be estimated using a confidence interval at a specified confidence level, with the sample mean serving as the estimate.

The evaluation outcomes of different CPUs are comparable under EECs. The EECs consist of identical problems, problem instances, algorithms, algorithm implementation, support systems containing the same compiler, compiler flag, copies/threads setting, OS, OS setting, memory, memory setting, disk, and disk setting in CPU evaluation. Any divergence in ECs will result in incomparable evaluation outcomes for different CPUs.

5. The differences between evaluation, measurement and testing

We elucidate the marked disparity between evaluation, measurement, and testing.

Metrology is the science of measurement and its applications. The essence of metrology lies in quantities and their corresponding measurements (see Fig. 7).

A test oracle is a method used to verify whether an individual or system being tested has performed correctly during a specific execution. Testing is the process of executing an individual or system to determine whether it (1) conforms to the specified behavior defined by the test oracles (the first category) and/or (2) operates correctly within its intended environment as defined by the test oracles (the second category) (see Fig. 8).

First and foremost, measurement, testing, and evaluation focus on different issues within the same scenario, e.g., the CPU evaluation experiment. The primary focus of evaluation lies in defining an EC that can yield true and consistent evaluation outcomes for the same subject. Eliminating confounding variables within the EC to ensure that evaluation outcomes remain comparable across different subjects is another crucial issue. In this context, measurement and testing is a micro-level activity that addresses the previously mentioned issue within a particular EC configuration.

Secondly, measurement, testing, and evaluation serve distinct purposes within the same scenario. Evaluation focuses on the subject, while measurement or testing targets ES. Measurement or testing is carried out directly on the ES, whereas evaluation is derived indirectly within the ES.

Thirdly, measurement, testing, and evaluation have different outcomes within the same scenario. The evaluation outcomes appear as the distribution of outcomes with respect to different EC configurations. However, testing and measurement outcomes appear as the distribution of outcomes with respect to the same EC configurations.

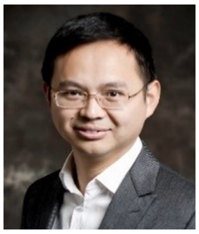
By virtue of the aforementioned reasons, we can assert that metrology or testing serves as but one foundational aspect in the realm of evaluations.

Acknowledgments

The author expresses gratitude to all the authors of "Evaluatology: The science and engineering of evaluation. BenchCouncil Transactions on Benchmarks, Standards, and Evaluations, 4(1), 100162". and acknowledges the contribution of Dr. Wanling Gao for refining all figures, and my Ph.D. student, Chenxi Wang, for providing the CPU evaluation example.

References

[1] J. Zhan, L. Wang, W. Gao, H. Li, C. Wang, Y. Huang, Y. Li, Z. Yang, G. Kang, C. Luo, H. Ye, S. Dai, Z. Zhang, Evaluatology: The science and engineering of evaluation, BenchCouncil Trans. Benchmarks Stand. Eval. 4 (1) (2024) 100162.
[2] System, 2024, https://www.merriam-webster.com/dictionary/system. (Accessed 6 February 2024).
[3] A. Backlund, The definition of system, Kybernetes 29 (4) (2000) 444–451.
[4] C. Wang, L. Wang, W. Gao, Y. Yang, Y. Zhou, J. Zhan, Achieving consistent and comparable CPU evaluation outcomes, 2024, Technical Report, International Open Benchmark Council.
[5] I. BiPM, I. IFCC, I. IUPAC, O. ISO, The international vocabulary of metrology—basic and general concepts and associated terms (VIM), JCGM 200 (2012) 2012.
[6] R.N. Kacker, On quantity, value, unit, and other terms in the JCGM international vocabulary of metrology, Meas. Sci. Technol. 32 (12) (2021) 125015.
[7] L. Baresi, M. Young, Test oracles, 2001.
[8] J.A. Whittaker, What is software testing? And why is it so hard? IEEE Softw. 17 (1) (2000) 70–79.



Dr. Jianfeng Zhan is a Full Professor at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), and University of Chinese Academy of Sciences (UCAS), the director of the Research Center for Distributed Systems, ICT, CAS. He received his B.E. in Civil Engineering and MSc in Solid Mechanics from Southwest Jiaotong University in 1996 and 1999 and his Ph.D. in Computer Science from the Institute of Software, CAS, and UCAS in 2002. His research areas focus on evaluatology, evaluatology-based design automation, and optimization automation. His exceptional expertise is exemplified by his introduction to the discipline of evaluatology, an endeavor that encompasses the science and engineering of evaluation; within this discipline, his proposition of a universal framework for evaluation encompasses essential concepts, terminologies, theories, and methodologies for application across various disciplines. He has made substantial and effective efforts to transfer his academic research into advanced technology to impact general-purpose production systems. Several technical innovations and research results, including 35 patents from his team, have been adopted in benchmarks, operating systems, and cluster and cloud system software with direct contributions to advancing parallel and distributed systems in China or worldwide. Over the past two decades, he has supervised over ninety graduate students, post-doctors, and engineers. Dr. Jianfeng Zhan is the founder and chairman of BenchCouncil. He also holds the role of Co-EIC of BenchCouncil Transactions on Benchmark, Standards and Evaluations, alongside Prof. Tony Hey. Dr. Zhan has served as an Associate Editor for IEEE TPDS (IEEE Transactions on Parallel and Distributed Systems) from 2018 to 2022. In recognition of his exceptional contributions, he has been honored with several prestigious awards. These include the second-class Chinese National Technology Promotion Prize in 2006, the Distinguished Achievement Award of the Chinese Academy of Sciences in 2005, the IISWC Best Paper Award in 2013, and the Test of Time Paper Award from the Journal of Frontier of Computer Science.

BenchCouncil Transactions on Benchmarks, Standards and Evaluations 4 (2024) 100163

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/

Research article

BinCodex: A comprehensive and multi-level dataset for evaluating binary code similarity detection techniques

Peihua Zhang [a,b], Chenggang Wu [a,b,c], Zhe Wang [a,c,*]

[a] SKLP, Institute of Computing Technology, China
[b] UCAS, China
[c] Zhongguancun Laboratory, China

ARTICLE INFO

Keywords:
Dataset
Binary code similarity detection
Compiler optimization
Code obfuscation

ABSTRACT

The binary code similarity detection (BCSD) technique can quantitatively measure the differences between two given binaries and give matching results at predefined granularity (e.g., function), and has been widely used in multiple scenarios including software vulnerability search, security patch analysis, malware detection, code clone detection, etc. With the help of deep learning, the BCSD techniques have achieved high accuracy in their evaluation. However, on the one hand, their high accuracy has become indistinguishable due to the lack of a standard dataset, thus being unable to reveal their abilities. On the other hand, since binary code can be easily changed, it is essential to gain a holistic understanding of the underlying transformations including default optimization options, non-default optimization options, and commonly used code obfuscations, thus assessing their impact on the accuracy and adaptability of the BCSD technique. This paper presents our observations regarding the diversity of BCSD datasets and proposes a comprehensive dataset for the BCSD technique. We employ and present detailed evaluation results of various BCSD works, applying different classifications for different types of BCSD tasks, including pure function pairing and vulnerable code detection. Our results show that most BCSD works are capable of adopting default compiler options but are unsatisfactory when facing non-default compiler options and code obfuscation. We take a layered perspective on the BCSD task and point to opportunities for future optimizations in the technologies we consider.

1. Introduction

The widespread presence of binary code across diverse domains, including traditional PC software, emerging IoT device firmware [1], and malicious software, highlights the criticality of conducting research exclusively focused on binary code to effectively address software security concerns. In recent years, the binary code similarity detection (BCSD) technique [2–28] has garnered substantial attention and proven its versatility across diverse fields, including vulnerability discovery, malware detection, software plagiarism detection, patch analysis, software supply chain analysis, etc.

With the continuous advancements in machine learning, especially deep learning, learning-based methods have emerged as a prominent approach in mainstream BCSD tools [9,10,12,14,18,26,28,29]. These methods harness the power of neural network architectures and techniques to extract intricate patterns and representations from binary code, resulting in significant improvements in accuracy. By leveraging large-scale training datasets and sophisticated neural network architectures, learning-based BCSD techniques have achieved state-of-the-art accuracy in various tasks.

However, recent BCSD works have faced a challenge in distinguishing their capabilities and accuracy due to the absence of a standard dataset. Our study (detailed in Section 2.3) reveals that 22 BCSD works, which were conducted in the past decade and published on top venues, utilized different datasets. Consequently, it has become difficult to assess the true effectiveness of these methods, hindering the ability to make informed decisions and impeding progress in the field. Given the broad range of applications and the growing number of works in the BCSD field, the establishment of a standardized dataset is crucial.

To address this issue, we propose a standardized BCSD dataset dubbed as BinCodex.[1] This dataset aims to evaluate BCSD works in diverse scenarios, enabling researchers to compare and assess different techniques more effectively. Additionally, a standardized dataset would facilitate the validation of new methods, foster collaboration among researchers, and contribute to the overall progress of BCSD.

* Corresponding author at: SKLP, Institute of Computing Technology, China.
E-mail address: wangzhe12@ict.ac.cn (Z. Wang).

[1] “Bin” signifies the focus on analyzing binary code, while “Codex” conveys the idea of a comprehensive collection of code samples.

https://doi.org/10.1016/j.tbench.2024.100163
Received 26 February 2024; Received in revised form 23 April 2024; Accepted 7 May 2024
Available online 21 May 2024

We first thoroughly inspected all possible change points of the binary code and divided them into 4 groups: different platforms (e.g., x86 and arm), different compilers (e.g., GCC and Clang), different compiler options (including default and non-default), and different code obfuscation techniques.

Among them, compiler optimization options play a vital role in shaping the structure of binary code, resulting in noticeable differences. Previous studies have identified default options (e.g., O0/1/2/3) as a critical challenge in binary diffing [28,30]. Additionally, apart from default optimization levels, non-default options can also change binary code significantly [31] but are challenging to exhaustively analyze due to their large combination space, which encourages BinCodex to explore. Lastly, code obfuscation techniques [32–38] can alter code to modify binary characteristics, posing potential challenges to BCSD techniques. Recent work [39] has identified the impact of inter-procedural obfuscation on BCSD techniques, but its comprehensive evaluation is yet to be fully explored, which also motivates the BinCodex.

It is non-trivial to construct a comprehensive dataset for the BCSD technique due to the challenges posed by program diversity, efficiency, and measurement diversity. For example, (1) To balance the dataset size with the program diversity, choosing which programs is a problem; (2) Since the binary can be easily changed from several aspects, the enumeration of all possible change point combinations is impossible because of the large searching space. (3) BCSD tools differ in the granularity of their features and the representation of those features. How to normalize these disparities without deducing their accuracy is challenging.

To address the first challenge, we create the dataset with careful consideration of several factors, including a large amount of code (over 10 million lines of code), the representation of different binary code types (e.g., system software, compiler, interpreters, commonly used libraries, firmware, and typical vulnerable code), the diversity of code samples, varying levels of similarity, and different granularity of similarity detection tasks (e.g., function level and basic block level).

To tackle the second challenge, we aim to reduce the searching space in several directions. Firstly, we select default options among different compilers, which helps eliminate unnecessary variations. Secondly, we utilize a search-based compiler tool to explore the non-default options within a single compiler, avoiding redundant evaluations. Lastly, we choose a commonly used optimization level as a baseline to evaluate code obfuscation techniques, which reduces the number of obfuscated binaries while still capturing their impact.

To overcome disparities in evaluation metrics, we addressed the third challenge by normalizing features and using a standardized distance measurement: *precision ratio*. By abandoning different metrics from various BCSD tools, we ensure consistency in the evaluation process, enabling fair comparisons and a more reliable assessment of effectiveness.

To achieve a more precise evaluation of BCSD tools, we developed BinCodex as a multi-level dataset incorporating various code transformation levels instead of merging all binaries into a single pool. This granular evaluation enables a more detailed understanding of the effectiveness of BCSD tools in different scenarios.

BinCodex is implemented and evaluated on the Linux system. Eight state-of-the-art binary diffing tools (Diemph [29], OPTango [40], jTrans [28], Asm2Vec [12], Safe [41], DeepBinDiff [10], VulSeeker [14], and BinDiff [42]) are evaluated. The results cover various BCSD tasks, including pure function pairing and vulnerable code detection, and employ different classifications for different types of tasks. The results highlight that most BCSD works perform well when default compiler options are used but face challenges with non-default options. Additionally, while many BCSD tools demonstrate adaptability to intra-procedural code obfuscation, they struggle with inter-procedural obfuscation techniques. The evaluation provides a deep understanding of the current state of BCSD works, identifies the necessity of a standardized BCSD dataset, and points to opportunities for future optimizations in the field.

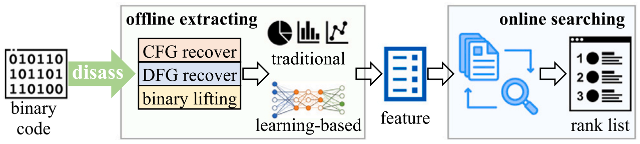


Fig. 1. The overall process of binary code similarity detection.

Our contributions can be summarized as follows:

- **A deep understanding of current BCSD works.** The paper is the first to provide a comprehensive summary of BCSD works based on their dataset characteristics. This understanding highlights the need for a standardized BCSD dataset.
- **A comprehensive BCSD dataset.** We present a BCSD dataset and propose three methods to enhance its comprehensiveness. These methods include dataset selection, searching space reduction, and metrics normalization. They ensure that the dataset includes representative programs, diverse features, and standardized metrics. We also perform multi-level evaluations to gain a detailed understanding of BCSD tools in different scenarios.
- **New insights from implementation and evaluation.** We evaluate BinCodex using eight state-of-the-art BCSD works, demonstrating its effectiveness in accurately assessing and comparing different methods. The insights gained from the evaluation contribute to fair evaluations, foster innovation, and advance the overall development of the BCSD field.

2. Background and motivation

2.1. Binary code similarity detection

Binary Code Similarity Detection (BCSD) is a technique used to analyze and compare binary code to identify similarities between binaries. It allows for quantitatively measuring differences and providing matching results at predefined levels of granularity, typically at the function level. As shown in Fig. 1, the process of BCSD typically begins with the disassembly of binaries, where the binary code is converted into assembly code, providing a representation that retains some semantic information of the program. This disassembly step serves as the foundation for most BCSD techniques. The workflow of BCSD techniques can be divided into two stages: offline feature extraction and online code search.

In the offline stage, tools extract features from binaries. Recent research focuses on determining which features should be extracted for effective BCSD. Based on the methods of the BCSD works, they can be classified into two categories [10]: traditional approaches and learning-based approaches.

- Traditional approaches extract low-level features from the binary code, such as opcode histograms. For example, Genius [43] and BinDiff [42] extract the number of string constants, numeric contacts, and different kinds of instructions as the identity of basic block and function, respectively. Besides, many works [6,7, 16,25,44–46] have tried to extract semantic-level features as the identity of binary code, such as using I/O syntax to describe a basic block [6].
- Learning-based approaches leverage machine learning techniques to automatically learn discriminative features from the binary code. Various models have been used to extract features and learn representations that capture the underlying patterns in the code. For example, Asm2Vec [12] regards the assembly language as a special language, abstracts each element (e.g., opcode and operands) in the instructions as tokens in the natural language, and generates the representation of each token through training and clustering.

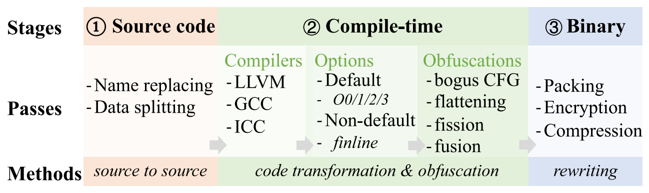


Fig. 2. Binary generation process, change points included.

Both traditional and learning-based approaches have their strengths and limitations. Traditional approaches often struggle with complex code transformations. Learning-based approaches, on the other hand, can adapt to diverse code representations and exhibit better robustness against code variations. However, they require large amounts of labeled data for training and can be computationally intensive.

In the online stage, tools calculate the similarity between the extracted features to identify matched pairs. This process can be seen as a search process, where the scale of the searching space is influenced by the granularity of the defined features. For instance, if the granularity is set at the function level, the searching space will be determined by the number of functions in the binaries. On the other hand, if the granularity is set at the basic block level, the searching space will be further expanded, considering the larger number of basic blocks in the code.

Additionally, the representation of the features impacts the method used for similarity calculation. When dealing with vector-related features, distance metrics such as Euclidean distance or cosine similarity are commonly employed. These metrics quantify the dissimilarity or similarity between feature vectors. On the other hand, when working with graph-related features, graph-matching methods come into play. These methods aim to find correspondences between nodes or subgraphs of the extracted features, taking into account both structural and semantic similarities.

2.2. Code transformation

The BCSD techniques meet challenges due to the ease with which binary code can be altered. Even instructions with identical semantics but different registers can have different binary representations. To this end, we first summarized the binary code generation process in Fig. 2, emphasizing the various points where changes can occur.

Source code transformations (Fig. 2 ①) primarily involve data obfuscation techniques that alter the format of data within a program. These transformations aim to prevent direct matching of data, often used to conceal sensitive information like private keys. However, in the BCSD scenario, these transformations are usually excluded as BCSD focuses on binary code rather than specific data values. Therefore, data obfuscation techniques targeting data format are not directly applicable to BCSD.

Dynamic code rewriting approaches (Fig. 2 ③), inspired by the concept of packing [47,48], focus on encoding or encrypting code as data. However, these techniques are also usually excluded in the context of BCSD due to they can be automatically unpacked [49–51] or be memory-dumped [52–55], and lose the transformation effect.

In contrast, compile-time transformations (Fig. 2 ②) focus on modifying the code during compilation without any further runtime modifications. These transformations have a significant impact on the structure of the resulting binary, making them a key area of interest in BCSD research. Compile-time transformations can manifest in several ways:

- Different compilers may implement the same optimization technique differently, leading to variations in the resulting binary code.
- Compiler options, including both non-default and default options, can influence the generated binary code (detailed in Section 2.2.1).
- Code obfuscations, which intentionally introduce complexity and disguise code, can result in substantial differences in binary code (detailed in Section 2.2.2).

2.2.1. Compiler optimization

Compiler optimization, which is originally used to improve the software performance (e.g., function inline) or reduce the binary size (e.g., dead-code elimination), has the potential to substantially modify the binary code, thereby exerting a significant impact on its differences. The binary code compiled from the same source code with different optimizations can exhibit a remarkably distinct code layout. Therefore, previous works have regarded compiler optimization as one of the key challenges to address in the BCSD task. For example, both Zeek [30] and jTrans [28] consider evaluating whether their methods can withstand the binary differences caused by different compiler optimization levels (including O0/1/2/3 and Os) as an important setup.

Except for the default compiler options, which integrate several optimization techniques, non-default optimization options also have a large effect on changing the binary code. Recent works have found they can expand the binary code difference significantly, which can be larger than the difference between O0 and O3 [31,40].

2.2.2. Software obfuscation

Software obfuscation transforms the program without changing its functionality to make it hard to analyze. There is an arms race between software obfuscation and BCSD. Software obfuscation does not want BCSD techniques to match un-obfuscated with obfuscated code successfully, and vice versa. There have been various techniques proposed for software obfuscation. For ease of introduction, we categorize them by obfuscation granularity:

- **Instruction level:** Instruction substitution (SUB) [33,35] replaces the original instruction with equivalent instruction(s). O-LLVM [35] designed 10 different strategies for arithmetic and logical operations.
- **Basic block level:** Bogus control flow (BCF) [33,35,63] inserts dead code into the original control flow and often utilizes permanent true or false predicates to prevent these codes from being executed, thereby ensuring the original functionality of the program.
- **Function level:** Control flow flattening (FLA) [33,35] converts the control flow of the function into the "`switch-case`" form, which is hard to analyze, and maintains the original jump relationship by controlling the values of the cases.
- **Module level:** Function fission [39] splits a function into multiple sub-functions. Conversely, function fusion [39] combines two functions into a single function. These code obfuscation techniques have proved their potential to alter function semantics significantly.

2.3. Motivation

We first conducted a comprehensive analysis of 22 BCSD works published in top venues over the past decade. These works were summarized based on their dataset characteristics and their ability to handle different code transformations during evaluations.

Our findings, as summarized in Table 1, reveal that there is a lack of a standardized dataset for BCSD techniques. Instead, the evaluated datasets were scattered across 36 different datasets or programs. It is worth noting that none of the 22 BCSD works utilized an identical dataset for evaluation. Each work employed its own self-constructed dataset, which allowed for detailed design considerations but lacked persuasiveness in terms of dataset consistency.

Since code transformation is a common source contributing to binary code differences, testing the resilience against transformation has become a common evaluation step for BCSD. As shown in Table 1,

Table 1
BCSD works from top venues in the last decade, summarized by their dataset characteristics and code transformation adaptability in their evaluations.

	Approach	Year	Venue	Dataset[a]	Code transformation adaptability			Diffing characteristics		
					Compiler[b]	Options[c]	Obfuscation[d]	Gran[e]	Metric	Platform[f]
1	DiEmph [29]	2023	ISSTA	2, 7, 8, 18, 27, 29, 32	❶❷	◐	–	F	Precision	①
2	sem2vec [56]	2023	TOSEM	2, 7, 10, 11, 20, 24, 27, 36	❶❷	◐	SUB, BCF, FLA	F	Precision	①
3	VulHawk [57]	2023	NDSS	7-11, 22, 25, 27, 29, 32, 35	❶	◐	–	F	AUC	① ② ③
4	OPTango [40]	2023	ISSRE	9, 30, 31	❶	●	–	F	Recall	①
5	TIKNIB [58]	2022	TSE	9, 16	❶❷❸	◐	SUB, BCF, FLA	F	self-defined	① ② ③
6	jTrans [28]	2022	NDSS	1, 9	❶❷	◐	–	F	Recall	①
7	ISRD [59]	2021	ICSE	5, 28, 36	❶❷	◐	–	F, B, I	Precision, Recall	①
8	Asteria [60]	2021	DSN	3, 12, 27	❶	○	–	F	ROC, AUC	① ② ④
9	DeepBinDiff [10]	2020	NDSS	7, 10, 11	❶	◐	–	B	Recall, Precision	①
10	Asm2Vec [12]	2019	S&P	4, 7, 9, 18-20, 24, 27, 29, 32, 36	❶❷	◐	SUB, BCF, FLA	F	Precision	①
11	SAFE [41]	2019	DIMVA	2, 6-10, 15, 21, 26, 27, 34	❶❷	◐	–	F	ROC, AUC, Precision, Recall	① ②
12	InnerEye [13]	2019	NDSS	2, 7, 10, 11, 27	❷	◐	–	B	ROC, AUC	① ②
13	alphaDiff [27]	2018	ASE	9, 14	❶❷	◐	–	P	Recall	① ②
14	VulSeeker [14]	2018	ASE	4, 7, 9, 12, 27	❶	◐	–	F	Precision	① ② ③
15	FirmUp [61]	2018	ASPLOS	9, 12	–	○	–	F	Precision	① ② ③
16	BinSequence [62]	2017	ASIACCS	23, 36	❸	○	–	F	Precision	①
17	IMF-SIM [46]	2017	ASE	7	❶❷❹	◐	SUB, BCF, FLA	F	Precision	①
18	BinGo [6]	2016	FSE	9	❶❷	◐	–	F	Precision	① ②
19	Genius [43]	2016	CCS	4, 7, 9, 12, 27	❶❷	◐	–	F	Recall, FPR	① ② ③
20	Multi-k-MH [16]	2015	S&P	4, 7, 12, 27	❶❷	◐	–	B	TP, FP	① ② ③
21	CoP [44]	2014	FSE	13, 17, 27, 33	❶❹	◐	BCF, FLA	P	self-defined	①
22	BinSlayer [11]	2013	PPREW	7	❶	○	–	P	Precision	①

[a] Including 36 micro-datasets. 1: ArchLinux repositories, 2: Binutils, 3: Buildroot, 4: BusyBox, 5: Bzip2, 6: CCV, 7: Coreutils, 8: Curl, 9: CVEs, 10: Diffutils, 11: Findutils, 12: Firmwares, 13: Gecko, 14: GitHub repositories, 15: GNU Scientific Library, 16: GNU software packages, 17: Gzip, 18: ImageMagick, 19: Libcurl, 20: Libgmp, 21: Libhttpd, 22: Libmicrohttpd, 23: Libpng, 24: LibTomCrypt, 25: Mtools, 26: OpenMPI, 27: OpenSSL, 28: PreComp, 29: PuTTY, 30: SPEC CPU 2006, 31: SPEC CPU 2017, 32: SQLite, 33: Thttpd, 34: Valgrind, 35: Wget, 36: zlib.

[b] Including 4 compilers. ❶: GCC, ❷: Clang, ❸: MSVC, ❹: ICC.

[c] Consideration of compiler options. ○: None, ◐: Only default options, ●: Both default and non-default options.

[d] Consideration of code obfuscations. -: None, SUB: instruction substitution, BCF: bogus control flow, FLA: control flow flattening, IBV: Insert bogus variables, SSO: split structure object.

[e] Granularity for BCSD. P: program, F: function, B: basic block, I: instruction.

[f] The implementation platform of the BCSD works. ①: x86, ②: ARM, ③: MIPS, ④: PowerPC.

many BCSD works have considered binaries compiled with different mainstream compilers, and most of them concentrate on the GCC and Clang.

Under a specific compiler, most of them only considered the default optimization options such as O0 to O3, but failed to explore non-default optimization options (only 1 work). However, the trend of compiling binaries using non-default options settings other than the default options has been increasing in recent years [31], for example, the virus binaries used non-default options to hide their binary code features. To this end, it is essential to evaluate the adaptability of the BCSD tools under non-default compiler options.

In terms of code obfuscation, only a few works have considered it, and the evaluation is often limited to intra-procedural obfuscation techniques (e.g., at the statement, basic block, or function level), which do not fundamentally change the semantics of each function. Thus existing BCSD techniques can still capture the obfuscation effect. However, recent advancements in code obfuscation techniques have increasingly focused on inter-procedural obfuscations, which have shown the ability to alter function semantics [39], which is a crucial factor in defeating BCSD techniques.

As for the granularity, most works focus on function-level, while some also consider basic block and instruction-level differences for their specific detection purpose. Besides, the matrix that measures the accuracy of BCSD works also varies. Some of them use standard metrics like true positive (TP) or false positive (FP). Precision@k and Recall@k are also used to measure the proportion of relevant results among the top k retrieved items. Some works also define the metric based on their specific features, which does not apply to others.

In the BSCD tool landscape, most tools are primarily developed for the x86 platform, with some considering cross-platform compatibility. While it is theoretically possible to adapt x86-focused tools to other platforms, such adaptations would involve additional efforts such as integrating disassembler backends and platform-specific API construction.

As BCSD techniques continue to improve in feature extraction and binary code representation, particularly with the application of deep learning, their ability to capture semantics becomes more robust. Consequently, while many of these works claimed superiority over others, the lack of dataset standardization raises the possibility that dataset mismatch could be a contributing factor to their comparative results. Therefore, there is a need for *a unified dataset that covers a wide range of representative programs and datasets*. A standardized dataset would facilitate fair comparisons among different BCSD techniques, enabling researchers to make more reliable and meaningful claims about their effectiveness.

Table 2
The detail of BinCodex dataset, including the origin of the dataset and the detail of multi-level workloads.

Dataset		Multi-level setting					
Info	Statistics	Workloads	Task	#Binaries	Compiler	Metrics	Options/Tools
Coreutils SPEC CPU 2006	113 progs 2.0M LOC	BIN-Default	Adaptability to default compiler options.	whole dataset	GCC Clang	Precision@1	O0-O3, Os, Ofast O0-O3
SPEC CPU 2017 Libraries	7.8M LOC 522K LOC	BIN-Nondefault	Adaptability to non-default compiler options.	whole dataset	GCC	Precision@1	BinTuner [31]
Firmwares #CVE	418K LOC 69	BIN-Obfuscated	Adaptability to different obfuscation methods.	whole dataset	Clang	Precision@1	O-LLVM [35], Khaos [39]
#Binaries #Functions	3.1K 68.9M	BIN-Vulnerable	Vulnerable function searching.	CVEs only	Clang	Precision@1/10/50	O-LLVM [35], Khaos [39]

Besides, the dataset should be able to cover different kinds of transformations to *fully measure the adaptability of BCSD techniques against possible transformations, including non-default compiler options and inter-procedural code obfuscations*. This viewpoint aligns with the literature published from both offensive and defensive perspectives:

- Many BCSD works have acknowledged the impact of inter-procedural optimizations (e.g., function inlining) on diffing accuracy [7,43,59,61,62,64–68].
- Existing research has demonstrated that inter-procedural obfuscation can decrease the accuracy of BCSD tools (up to 60%) with minimal overhead (less than 5%) [39].
- Existing research [31,40] has also proved that non-default options can decrease the accuracy of BCSD tools.

Given that all current BCSD tools only have a common implementation on the x86 platform, this paper follows the same setting to evaluate as many BCSD tools as possible. By concentrating on this specific instruction set, the dataset can provide a more targeted assessment of the capabilities and limitations of BCSD tools.

3. Methodology

Considering the above factors, this paper proposes a unified dataset that incorporates a wide range of programs and possible transformations, particularly non-default compiler options, and inter-procedural code obfuscation techniques, and uses it to explore and evaluate existing BCSD techniques.

3.1. Challenges

Designing a dataset for BCSD poses several challenges, which can be summarized as follows:

- **Creating a representative dataset (C1):** On the one hand, the dataset's codebase should be extensive enough to encompass a wide range of code features. On the other hand, the dataset needs to encompass representative programs from various application scenarios commonly encountered in BCSD.
- **Generating diverse code variants efficiently (C2):** Enumerating all possible code variants is impractical due to the numerous change points discussed in Section 2.2. For instance, even a single change point, such as compiler options in GCC, includes over 200 options resulting in more than 2^{200} combinations. Exhaustively combining these options to compile the same source code would result in an infeasible number of binary variants. How to generate a dataset with diverse features without exhaustively enumerating all possible combinations is challenging.
- **Normalizing features and similarity calculation (C3):** BCSD tools differ in the granularity of their features and the representation of those features. Furthermore, the similarity calculation methods employed by each tool may vary. How to normalize these disparities without deducing their accuracy is challenging.

The following subsections detail the design of BinCodex and address the above challenges. By doing so, a meticulously designed BCSD dataset can be constructed, enabling accurate evaluation of BCSD tools.

3.2. The BinCodex dataset

The Origin of Datasets (C1). To overcome the C1 challenge, we conducted an extensive analysis of widely used software across different domains and carefully selected representative programs from various application scenarios commonly encountered in BCSD. As Table 2 shows, our dataset includes programs from diverse domains, such as utility programs (e.g., Coreutils), compilers (e.g., GCC in SPEC CPU), language interpreters (e.g., Perl interpreter in SPEC CPU), JavaScript engines (e.g., QuickJS), network protocols (e.g., LibCurl), web applications, libraries (OpenSSL), embedded firmware (e.g., BusyBox), and artificial intelligence applications (e.g., alpha-beta tree search and Monte Carlo tree search).

By including programs from such diverse domains, our dataset accurately reflects the challenges and complexities faced by BCSD techniques in real-world scenarios. It provides a comprehensive evaluation of the accuracy of BCSD tools, as it encompasses all the types of programs used in existing BCSD works in Table 1. This ensures that BinCodex serves as a valuable dataset for evaluating and comparing different BCSD techniques.

Searching Space Reduction (C2). The enumeration of all compiler options and obfuscation techniques for all compilers is impossible because of the large searching space. To this end, our reduction mainly contains the following 3 steps.

Inter-compiler reduction. In our dataset, we first chose to focus on the GCC and Clang compilers for alignment with existing BCSD research and their widespread usage with extensive optimization capabilities. Based on that, we reduce the inter-compiler searching space by *concentrating binary variants under their default compiler options*. For example, as shown in Table 2, GCC and Clang are both used when considering the default compiler options (*BIN-Default* workload), and only GCC is used when considering the non-default compiler options (*BIN-Nondefault* workload).

The exclusion of Clang in the *BIN-Nondefault* workload was a deliberate decision made for the purpose of introducing a specific challenge and evaluating the adaptability of BCSD tools to non-default compiler options using a single compiler. By concentrating on GCC for the non-default compiler options, we aimed to isolate and assess the challenges posed by non-default options in binary code analysis. This approach allowed us to investigate the specific impact of non-default options on BCSD without the additional variability introduced by using multiple compilers in this particular workload.

Intra-compiler reduction. Exhaustively combining non-default options to compile the same source code would result in an infeasible number of binary variants. Besides, different programs are likely to use different combinations of options since their code patterns are different. For example, the `funroll-loops` option can change binary for programs with loops but has no effect on those programs without loops. To acquire the program-specific combinations, we tailor *search-based* *iterative compilation for the auto-tuning of non-default compiler options*. Specifically, we utilize the BinTuner [31] tool, which uses the genetic algorithm to reveal the optimal effects on binary code differences. By adopting such an efficient approach, we can strategically select a subset of compiler flags, optimization levels, and code transformations to generate a diverse set of code variants without the need for exhaustive enumeration.

Obfuscation reduction. In our dataset, we have included obfuscation techniques as an important aspect of evaluation, considering the sensitivity of BCSD techniques to code obfuscation. To represent this, we selected two well-known obfuscation tools based on the LLVM infrastructure.

The first obfuscation tool is O-LLVM [35], a popular compiler-based obfuscation tool widely used in software engineering, systems security, and programming language research. O-LLVM offers three *intra-procedural obfuscation* methods of different granularity: `SUB`, `BCF`, and `Fla`.

Additionally, we incorporated the *inter-procedural* obfuscation tool called Khaos [39] in our dataset. Khaos focuses on changing function semantics, a key factor in defeating BCSD techniques. It provides two obfuscation primitives named `Fission` and `Fusion`.

When generating obfuscated binaries, we established a baseline by using the commonly used compiler option instead of generating obfuscated binaries for all options. Specifically, we chose the O2 optimization level as the baseline. This decision was made because O0 and O1 optimization levels are less commonly used in real-world applications, while the O3 optimization level may remove the obfuscation effect of O-LLVM (further discussed in Section 6).

Metrics (C3). To overcome the disparities of evaluation metrics, it is crucial to establish a standardized metric for similarity calculation. From our observation, these disparities come from the online searching stage (introduced in Section 2.1). Specifically, after the feature is extracted from the binary file and the distance is calculated, different tools use diversity methods to explain the accuracy. To this end, we abandon the different metrics of different BCSD tools and use a consistent measurement for the accuracy — *precision@k*, which is also commonly used in several BCSD works [12,28,29,31,56].

In the BCSD scenario, the search results are presented as a ranked list. Consider two binary files P and Q that are compiled from the same source code but with different options. Each of them has N functions, where $P = \{p_1, p_2, \dots, p_i, \dots, p_N\}$ and $Q = \{q_1, q_2, \dots, q_i, \dots, q_N\}$. The ground truth is p_i matches q_i, where $p_i \in P$ and $q_i \in Q$. For each function $p_i \in P$, the BCSD tools identify the top-k functions in Q that are most similar to p_i. These functions are ordered by a similarity score, indicating their rank $Rank_{q_i}$ in the list. Utilizing the definitions of precision, the precision ratio $precision@k$ is determined using the following metric:

$$precision@k = \frac{1}{N}\sum_{p_i \in P}(Rank_{q_i} \leq k) \times 100\% \tag{1}$$

For example, if the $precision@10$ of a specific BCSD tool exceeds 80%, it means over 80% functions in P are matched correctly in the top-10 candidate functions from Q. By normalizing the metrics of different BCSD tools and mapping the results to a common representation space, the dataset can facilitate fair and consistent evaluation.

Multi-level workloads. As shown in Table 2, BinCodex contains 4 different workloads. The *BIN-Default* workload aims to evaluate the adaptability to default compiler options, which is commonly used in real-world programs. The *BIN-Nondefault* workload is specifically designed for the emerging use of non-default options, which are confirmed by security analysts that these options can make reverse engineering analysis complicated [31,69]. Obfuscation often introduces challenges to existing BCSD tools, given that they extract syntactic- or graph-level information, which does not necessarily reflect the real functionality, thus the *BIN-Obfuscated* workload is used to evaluate if these tools are vulnerable to assembly codes with similar functionality but the differing appearance. Considering the diverse application scenarios of BCSD, BinCodex also adopts 69 CVEs to form the *BIN-Vulnerable* workload for the specific vulnerable code searching scenario and normalizes the measurement using *precision@k*.

The workloads are designed in increasing order of difficulty for BCSD tools, with the first three of them representing a progression from easier to harder transformations. For instance, most BCSD tools can handle transformations between O0 and O3, but only a few consider non-default options, and they all face challenges when dealing with inter-procedural obfuscation [39]. The *BIN-Obfuscated* workload presents similar difficulties to BCSD tools as the *BIN-Vulnerable* workload, as the vulnerable codes in the latter are also obfuscated. To precisely measure the vulnerability search result, it uses more comprehensive measurements — *precision@1/10/50*.

Algorithm 1: Dataset generation algorithm.

```
Input: Program source code set $progSet$, $N_{def}$, $N_{nondef}$, $N_r$, $N_{obf}$
Output: Binary variants dataset $binSet$
$binSet \leftarrow \{\}$
for each program $prog \in progSet$ do
    for $i = 1$ to $N_{def}$ do
        $def \leftarrow getDefaultOption(i)$;
        $bin_{def} \leftarrow$ Compile $prog$ with option $def$;
        $binSet[prog].append(bin_{def})$;
    end
    for $i = 1$ to $N_{nondef}$ do
        $def \leftarrow getDefaultOption(i)$;
        $bin_{def} \leftarrow$ Compile $prog$ with option $def$;
        $max_{diff} \leftarrow 0$;
        $bin_{max} \leftarrow bin_{def}$;
        for $i = 1$ to $N_r$ do
            $bin_{nondef} \leftarrow BinTuner(bin_{def})$;
            $diff \leftarrow BinDiff(bin_{def}, bin_{nondef})$;
            if $max_{diff} < diff$ then
                $max_{diff} \leftarrow diff$;
                $bin_{max} \leftarrow bin_{nondef}$;
            end
        end
        $binSet[prog].append(bin_{max})$;
    end
    for $i = 1$ to $N_{obf}$ do
        $obf \leftarrow getObfOption(i)$;
        $bin_{obf} \leftarrow$ Compile $prog$ with option $obf$;
        $binSet[prog].append(bin_{obf})$
    end
end
return $binSet$;
```

3.3. Workflow of BinCodex

For the dataset listed in Table 2, algorithm 1 outlines the workflow of BinCodex to generate their binary variants. For each program $prog$ in the $progSet$, three types of variants, namely default compiler options, non-default compiler options, and code obfuscation, are applied to $prog$ to produce a set of binary variants, which are then added to $binSet$. N_{def}, N_{nondef}, and N_{obf} denote the number of default compiler options, non-default compiler options, and code obfuscation techniques that need to be generated for each $prog$, respectively.

Firstly, when generating the binary variants with default compiler options, we choose all default compiler options (2 compilers, 10 options in all) in Table 2 to apply. Secondly, to ensure the diversity of the generated samples under non-default compiler options, we perform N_r rounds of generating binary variants by the BinTuner [31] tool and use

Table 3
Summary of the chosen diffing works.

BCSD works	Diffing granularity	Symbol relying	Time/memory consuming	Call-graph lacking
Asm2Vec [12]	function	N	N	Y
SAFE [41]	function	N	N	Y
DeepBinDiff [10]	basic block	N	Y	N
jTrans-0 [28]	function	N	Y	Y
jTrans [28]	function	N	Y	Y
BinDiff [42]	all	Y	N	N
DiEmph [29]	function	N	Y	Y
VulSeeker [14]	function	N	Y	Y

the BinDiff [42] to select the most different variant bin_{max}. Thirdly, for each selection of the obfuscation technique, we generate all obfuscated variants from them under the same baseline. After the above process, the BinCodex contains 68,930,974 binary functions (3,627,946 functions in the source code, with 19 variants for each function), which is the largest dataset to our best knowledge.

4. Implementation and evaluation

The BinCodex is implemented under the Linux operating system. The evaluation is conducted on Ubuntu 22.04 (Kernel v5.15.0) which runs on the x86_64 platform (Intel Xeon Gold 6148 CPU with 160 cores and 1.5TB memory) since all current BCSD tools only have common implementation in the x86 platform. This section aims to answer the following questions:

- (Q1) How do the state-of-the-art BCSD works perform on BinCodex?
- (Q2) What is the impact of the three levels of code transformation techniques, namely default compiler options, non-default compiler options, and code obfuscation, on the effectiveness of BCSD works?
- (Q3) What kinds of code transformation have the greatest impact on the BCSD works?

Corresponding to the four workloads in BinCodex, the evaluation consists of four parts, including the adaptability to the default compiler options using GCC and Clang (Section 4.1), the non-default compiler options using GCC (Section 4.2), the code obfuscation using Clang (Section 4.3), and a specific application scenario of BCSD — vulnerable function searching (Section 4.4).
Confrontation targets. We leverage 8 state-of-the-art BCSD tools to evaluate BinCodex. Their characteristics are summarized in Table 3. All learning-based tools among them are retrained on BinCodex. The column "*symbol relying*" means whether the un-stripped binaries have side-effects or not, for example, BinDiff [42] uses function names to reduce the searching space. The column "*time/memory consuming*" means the diffing process takes a long time (e.g., over one 1 month) or requires a lot of memory (e.g., more than 1 TB). The column "*call-graph lacking*" means whether the call-graph is used as the feature. Their detailed techniques are as follows:

- **Asm2Vec** [12] employs random walks on the function CFG to sample instruction sequences and then uses the PV-DM model to learn function and instruction token embedding jointly.
- **SAFE** [41] utilizes a word2vec model to generate instruction embeddings and proposes a recurrent neural network for function embedding generation.
- **DeepBinDiff** [10] is a learning-based work for diffing the semantic similarity in basic block granularity.
- **jTrans-0** [28] incorporates control flow information from binary code into transformer-based language models for function embedding. **jTrans** [28] fine-tunes the pre-trained model to generate function embedding for the supervised learning task of binary diffing.
- **BinDiff** [42] is an industry-standard binary diffing tool, which diffs the semantic similarity in different granularity (e.g., instruction, basic block, function, call graph).
- **DiEmph** [29] detects undesirable instruction distribution biases caused by specific compiler conventions and repairs them by removing them from the dataset and fine-tuning the models.
- **OPTango** [40] is a transformer-based multi-central representation learning approach, which purely explores the solution to build a compiler optimization-agnostic tool.
- **VulSeeker** [14] is a vulnerability seeker that integrates function semantic emulation with semantic learning.

The selection of the eight BCSD tools for evaluation was carefully considered to cover a diverse range of techniques and approaches in the field. The rationale behind their selection can be summarized as follows:

1) *Representation of Different Techniques.* The chosen tools represent a variety of techniques employed, which ensures a comprehensive evaluation of BinCodex's performance across different methodologies, allowing us to analyze its effectiveness in various scenarios.
2) *Learning-Based Approaches.* Given almost all state-of-the-art diffing tools are learning-based, we included 6 learning-based tools. By evaluating BinCodex with these tools, we can assess its compatibility with learning-based approaches and compare its performance against state-of-the-art models.
3) *Industry-Standard Tool.* BinDiff, known as an industry-standard binary diffing tool, is included to provide a benchmark for comparison. Its comprehensive analysis capabilities, including instruction-level, basic block-level, function-level, and call graph-level comparisons, make it a valuable tool for evaluating BinCodex.
4) *Compiler Optimization-Agnostic Tools.* Tools like OPTango were selected to evaluate BinCodex's robustness against different compiler optimization variants. These tools focus on building optimization-agnostic models to overcome the challenges posed by variations in compiler optimization levels.
5) *Vulnerability Detection.* VulSeeker, a vulnerability seeker tool, is included to assess BinCodex's effectiveness in detecting vulnerabilities in binary code. This tool incorporates semantic emulation and learning techniques to identify potential vulnerabilities, providing a specific use case for evaluation.

Overall, the selection of these eight BCSD tools ensures a comprehensive assessment of BinCodex. By including tools with diverse techniques, learning-based approaches, industry-standard benchmarks, optimization-agnostic models, and vulnerability detection capabilities, we can thoroughly evaluate BinCodex's performance, compatibility, and effectiveness across different dimensions of binary code similarity analysis.

Each BCSD tool has its own specific application scenario and capabilities. In the evaluation process, the tools were selected and used based on their suitability for the respective tasks. For instance, DeepBinDiff [10] focuses on diffing binaries at the basic block level. Therefore, it was specifically evaluated in the code obfuscation part, where obfuscation methods can alter the basic blocks of the code. DiEmph [29], on the other hand, relies on jTrans [28] as its underlying tool. As a result, DiEmph was solely used in the code obfuscation part of the experiment, where jTrans was evaluated comprehensively. OPTango [40] was evaluated in the compiler-option-relevant parts of the experiment, aligning with its claim of being specifically designed for compiler options.

4.1. Adaptability to default compiler options

To evaluate the adaptability of BCSD tools under default compiler options, the experiment used the *BIN-Default* workload in the BinCodex.

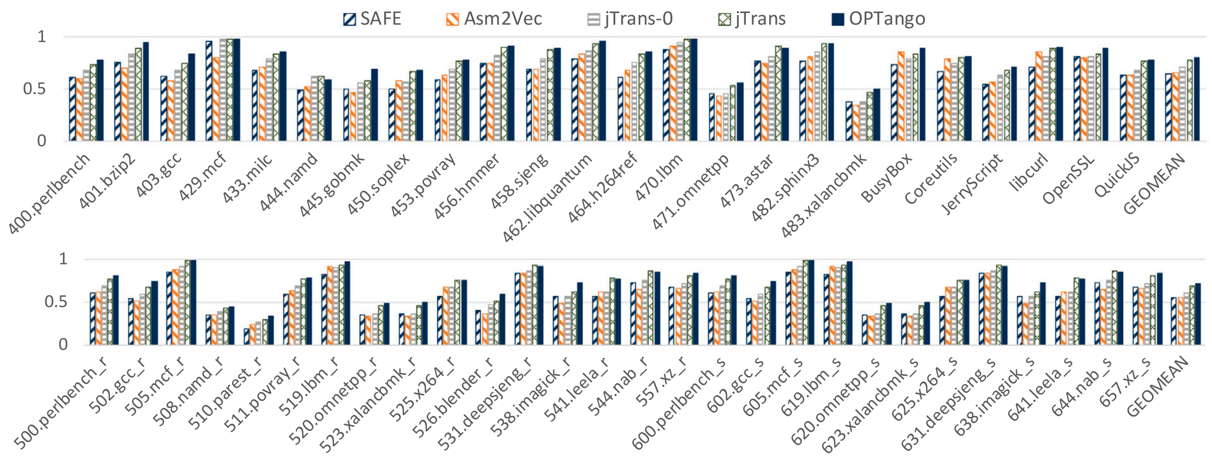


Fig. 3. The precision@1 results of chosen binary diffing works for binaries generated by default compiler options.

Space reduction. To avoid the exhaustive enumeration of different optimization levels, we defined a baseline by selecting binaries generated with the O0 and O2 options (including Os and Ofast for GCC). These baseline binaries were used for the targets of diffing, where binaries generated with the O1 and O3 options were sequentially compared. For example, the binary *400.perlbench-O1* would be compared with both *400.perlbench-O0* and *400.perlbench-O2* binaries. Similarly, the *400.perlbench-O3* binary would also be compared with them. The results of these comparisons were then averaged.

This experimental setup allowed for efficient exploration of all default compiler options. From the optimization perspective, all the binaries can be regarded as two groups: unoptimized and optimized. The baseline contains the unoptimized and optimized binaries at the same time (e.g., O0 as the unoptimized, O2 as the optimized), as well as the binaries used for querying (e.g., O1 as the unoptimized, O3 as the optimized). In this way, the number of options is reduced from 2^{10} (10 options in all) to 2 (optimized and unoptimized).

Results. The experiment's results are shown in Fig. 3. The precision@1 means the BCSD tool can match the optimized function with the unoptimized function on the first candidate in its rank list, and higher accuracy means higher adaptability. For example, the results indicate that all the BCSD tools achieved a precision rate higher than 50%, which means all the BCSD tools can match over half of the functions as the first candidate. OPTango [40] demonstrated the best adaptability, followed by jTrans/jTrans-0 [28], Asm2Vec [12], and SAFE [41].

An interesting observation was made during the experiment: *larger binary sizes tended to yield lower accuracy*. This phenomenon can be attributed to the relationship between the searching space and the number of functions. As the number of functions increases, the searching space expands, and the likelihood of finding similar functions within the same binary also grows. Consequently, the false positive ratio increases. This observation was also noted in subsequent evaluations involving non-default compiler options and code obfuscations, which is discussed further in Section 6. Additionally, the OpenMP-related programs in SPEC CPU 2017 are slightly harder for the BCSD tools to achieve a high precision compared with non-OpenMp binary variants.

4.2. Adaptability to non-default compiler options

In this subsection, the adaptability of BCSD tools under non-default compiler options is evaluated using the *BIN-Nondefault* workload. BinTuner [31] is utilized to generate binaries with non-default options effectively.

The BinTuner tool. BinTuner [31] follows a specific procedure to generate binaries. Initially, it selects a baseline binary, such as the one generated with the O0 option. Then, it iteratively searches for the target binary by combining non-default options. During the search process, BinTuner leverages the differences between the baseline binary and the target binary to guide the selection of the next combination of options. This iterative approach allows BinTuner to effectively explore the vast space of non-default compiler options and generate binaries for evaluation purposes.

However, BinTuner is not a panacea, especially when the different optimization levels are considered. For example, we first set the O0 as the baseline for BinTuner, after it generated the binaries, we utilized BinDiff [42] to calculate the difference with the baseline. As depicted in Fig. 4, we observed that a significant portion of the generated binaries exhibited similarity to the O3 variants, indicating a lack of diversity. This outcome suggests that BinTuner may not effectively explore the entire optimization space when only one baseline is considered.

Multi-baseline setting. Firstly, to enhance the diversity of the evaluation, different default optimization levels (O0, O1, O2, O3) were chosen as the baselines to generate binaries using BinTuner. This approach resulted in the creation of four groups of binaries (Ot0, Ot1, Ot2, Ot3) where each group was based on a specific default optimization level. Secondly, in addition to the individual groups, a fifth group of binaries (Ot4) was generated by setting all four default optimization levels as the baseline simultaneously. This group's binaries were distinct from those generated by any of the default options alone. Lastly, to provide a comprehensive evaluation, all five groups of binaries (Ot0, Ot1, Ot2, Ot3, Ot4) were also merged into a single group (Ot). This merged group encompasses the entire range of binary variations generated by BinTuner.

Overlap in default and non-default options. There is an overlap between default and non-default compiler options, for example, both of them have `-funroll-loop` and `-finline` options. However, the default option binds options in the specific combination (for example, O3 enables `-funroll-loop` and `-finline` at the same time), while the non-default option does not bind the combination (for example, enable `-finline` option but disable `-funroll-loop` option at the same time), which can enlarge the binary difference. The detailed options used by BinTuner are shown in Table 6.

It is important to note that each binary was created by incorporating over 100 different non-default option configurations. The details of these configurations can be found in Table 7. By merging the different groups of binaries and conducting binary similarity detection on the combined dataset, the evaluation aimed to assess the adaptability and performance of the BCSD tools across a wide range of non-default compiler option configurations.

Result. The results are shown in Fig. 5, where every group shows the same trends of accuracy. Consider the Ot2 set as an example, where the target binaries are generated by searching for the maximum difference from the O2-based binary, Asm2Vec achieves the precision@1 scores of 0.537, SAFE and jTrans-0 demonstrate similar diffing accuracy,

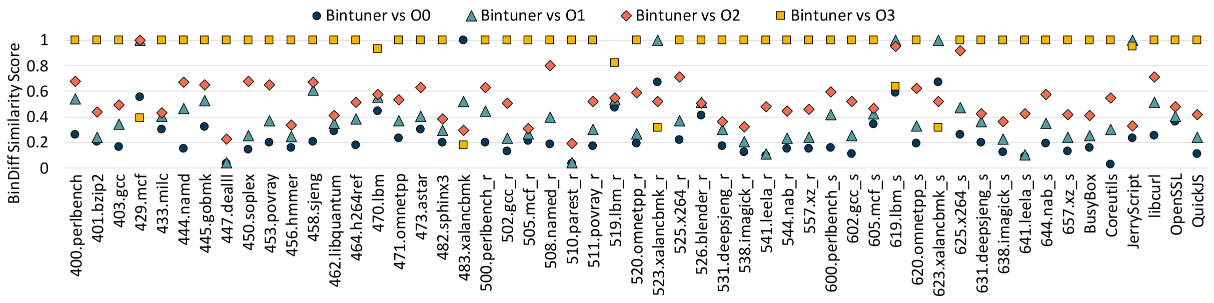


Fig. 4. The similarity score (normalized) of binaries generated by BinTuner [31] by only using the O0 as the baseline.

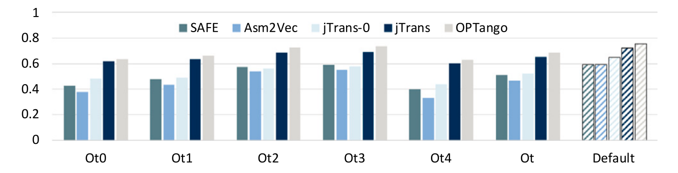


Fig. 5. The precision@1 results of chosen binary diffing works for binaries generated by non-default compiler options in GCC compiler. Higher similarity means higher adaptability.

achieving 0.571 and 0.563, respectively. jTrans outperforms jTrans-0 by 12.1% but falls 4.1% behind OPTango, which stands out with a precision of 0.726.

We also put the accuracy result of binaries generated by the default options in the rightmost group in Fig. 5(`Default`). By comparing it with the `Ot4` group, we can notice a significant decrease in accuracy under the non-default option settings compared to the default option settings for these binary diffing methods, which highlights the motivation of BinCodex.

4.3. Adaptability to code obfuscation

Obfuscators. As discussed in Section 3.2, we used the commonly used compiler option O2 as the baseline, and generated the obfuscated and un-obfuscated binaries in the *BIN-Obfuscated* workload. We keep all the binaries un-stripped to get the ground truth of paring. Besides, the Khaos [39] changes the number of functions because it is an inter-procedural obfuscation tool, thus we followed its evaluation setting to relax the requirements for *Precision@1*. The O-LLVM [35] remains unchanged since it does not change the function count. To ensure the consistency of the evaluation environment for the obfuscation tools, we upgrade the LLVM version of O-LLVM [35] to 9.0.1, which is the same as the Khaos [39]. All the existing BCSD tools adopted the old version of LLVM, which loses the obfuscation effect when facing a high optimization level (e.g., O3).

Histogram of Opcodes. We collected some internal information on the dataset to reveal the details of BinCodex. We used the *objdump* tool to disassemble all the binaries in BinCodex and calculated the histogram of opcodes. By comparing the vectors of opcodes between the original and obfuscated binaries, we can calculate the vector similarity. Since different programs may have varying numbers of instructions, we normalized the distances using the maximum similarity among all obfuscated programs. As depicted in Fig. 6, the distribution of opcodes varies in different obfuscation methods, in which `Fission` generates the binaries with the longest opcode distance and SUB generates the shortest. It demonstrates that BinCodex contains a diverse range of opcodes, and thus can fully cover the obfuscated binary scenario.

Results. After the binary is generated, we evaluated the accuracy of the selected BCSD tools by comparing obfuscated and un-obfuscated binaries. As depicted in Fig. 7, the result is divided into different groups by different obfuscation methods. The precision@1 means the BCSD tool can match the obfuscated function with the unobfuscated function on the first candidate in its rank list, and higher accuracy means higher adaptability. Among these obfuscation methods, compared with inter-procedural code obfuscation (`Fission` and `Fusion`), the BCSD tools are more adaptive to intra-procedural obfuscation (e.g., `SUB` and `BCF`). Besides, while the control flow flattening (`FLA`) archives a strong obfuscation effect, it also brings extremely high runtime overhead (2.8x slowdown), which is undesirable in real-world scenarios.

Binary variants that are obfuscated by the instruction substitution (`SUB`) are the easiest to pair, mainly because it does not change the function's control flow but only replace the opcodes of instructions, which are easy to adopt since these opcodes belong to the same opcode family, e.g., ALU. Although the bogus control flow (`BCF`) inserts dead code in the function, it has merely interfered with the original function control flow, thus also bringing limited obfuscated effect.

Inside each obfuscation method, different BCSD tools show different adaptability. We discuss them as follows:

- VulSeeker [14] takes more than 1 day to diff two large binaries and often gets killed due to memory limit. To speed up VulSeeker, we group the related functions into small groups to manually reduce the searching space.
- SAFE [41] and Asm2Vec [12] showed their advantages on intra-procedural obfuscation by capturing the semantics of functions.
- Because DeepBinDiff [10] uses the basic block as its granularity, its searching space is much larger than others and brings the time/memory consuming issue (e.g., requiring more than 10TB memory, waiting several weeks to compare two binaries). To this end, We reduced the dataset for DeepBinDiff [10] by only using programs with less than 40k lines of code. In this setting, it achieved higher accuracy in inter-procedural obfuscation methods (e.g., `Fusion`). This is because Khaos [39] uses original functions to obfuscate each other, lacking material reduces the obfuscation effect.
- Since we retrained the model of jTrans on BinCodex, it is more accurate than the pre-trained model jTrans-0.
- After DiEmph [29] detected the undesirable instruction distribution biases, it fine-tuned the models of jTrans [28], thus its adaptability is slightly higher than jTrans [28].
- Since BinDiff [42] takes advantage of function names of the symbols, its results are a little higher than others.

4.4. The ability to search vulnerable code

We use the *BIN-Vulnerable* workload in BinCodex to evaluate the ability to search real-world vulnerable code, each program contains at least one vulnerability (detailed in Table 5). In this experiment, we used Asm2Vec [12], VulSeeker [14] and SAFE [41] to calculate the *precision@n ratio* (the rank of truly matched pair in the matched result) of vulnerable functions. The reason why other tools were not used is that they only give top-1 matched results. We calculated *precision@1/10/50 ratio* of vulnerable functions.

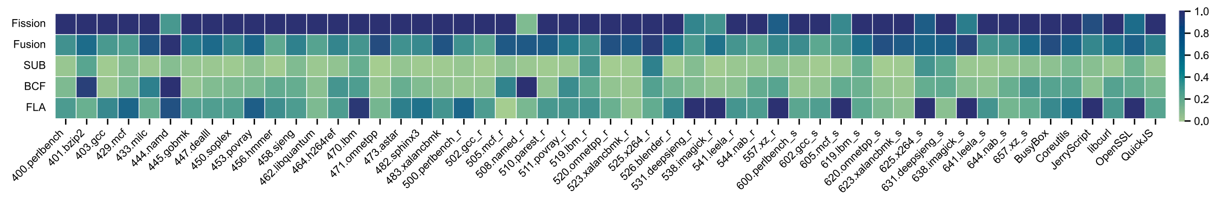


Fig. 6. The heat map of opcode histogram distance (normalized) of obfuscated binaries in BinCodex.

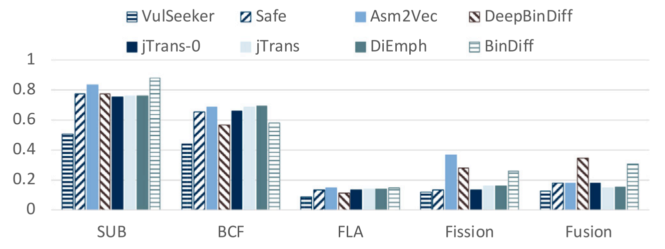


Fig. 7. Similarity results of chosen binary diffing works and tools. Diffing works use precision@1, and BinDiff uses its normalized scores. Higher similarity means a higher adaptability.

Table 4
Comparison with other BCSD datasets.

Dataset	Statistics		Transformations[a]		
	Source	#Funcs	Def	Non-Def	Obf
Esh [3]	8 CVE	1.5K	✔	✘	○
BINKIT [70]	GNU packages	36M	✔	✘	◐
BinaryCorp [28]	GNU packages	25M	✔	✘	○
BinCodex	GNU packages, libraries, SPEC CPU, firmware, 69 CVE	69M	✔	✔	●

[a] The possible transformations that the dataset considered, including default compiler options (Def), non-default compiler options (Non-Def), and code obfuscations (Obf, ○: none, ◐: only intra-procedural obfuscation methods, ●: both intra-procedural and inter-procedural obfuscation methods).

Fig. 8 gives the experimental results, which are divided into three groups as precision@1/10/50. The precision@1 means the BCSD tool can match the obfuscated function with the unobfuscated function on the first candidate in its rank list, while the precision@10 means it can match them in the top-10 candidates in its rank list, thus the accuracy is ascending in from precision@1 to precision@50. For example, the *precision@50 ratio* of `Fission` on Asm2Vec is around 0.5, which means about 50% of vulnerable functions can be found within the top-50 ranked functions using Asm2Vec.

Inside the same group, the *precision ratio* can reflect the vulnerable function searching ability of different BCSD tools. For example, for the `precision@1` group, Asm2Vec [12] is more accurate than SAFE [41], and both of them are better than VulSeeker [14]. Besides, it can also reveal the ability to hide the vulnerable function with different obfuscation methods. For example, under the same precision and binary diffing tool (e.g., `precision@50-Asm2Vec`), *Fission* and *Fusion* are better than *SUB*, *BCF*, and *FLA*.

5. Related works

Existing research often lacks transparency when it comes to disclosing their datasets. Among the few open datasets available, as shown in Table 4, Esh [3] proposed a dataset purely for vulnerability searching but with only 8 vulnerabilities. BinKit [70] claims as the largest binary dataset, however, it only consists of software from GNU packages like GNUtils and coreutils. Additionally, it only includes variants of compiler options for default optimization levels ranging from O0 to Ofast, which are fully covered in our proposed BinCodex dataset. Although BinKit considers code obfuscation, it only focuses on intra-procedural methods, which have been proven ineffective in both our evaluation and other works [39]. Another recent open-source dataset, Binarycorp [28], claims to offer diversity in terms of project size and application scenarios compared to BinKit. However, it only utilizes five compile options from O0 to Os, which are also included in our BinCodex dataset.

In contrast, our newly proposed BinCodex dataset provides a more comprehensive and realistic foundation. To the best of our knowledge, it encompasses the largest code base, including a diverse range of code sources such as GNU software packages, JS engines, firmware, and common libraries. Importantly, BinCodex incorporates 69 real-world vulnerable functions to cover more vulnerability patterns when facilitating the vulnerability searching scenario. Furthermore, it includes full-scale transformations such as non-default options and inter-procedural code obfuscation techniques, which have been neglected in other datasets but have been proven effective both in our evaluation and other works [39,40].

By incorporating these diverse factors, BinCodex offers valuable insights for BCSD techniques to learn from and evaluate the resilience of binary diffing methods against a wide range of compilation optimization variants and code obfuscation techniques. As mentioned in the introduction section, we are committed to contributing to the research community by making our dataset and trained models openly available. This will enable other researchers to utilize and build upon our work, fostering collaboration and further advancements in the field of binary code similarity and vulnerability analysis.

6. Discussion

Cross-platform consideration. As shown in Table 1, all current BCSD tools have their implementation in the x86 platform, thus the current dataset construction process primarily focuses the dataset specifically on evaluating BCSD techniques on the x86. Future work aims to incorporate cross-platform binaries to assess the accuracy of BCSD techniques across different architectures. This will require addressing the unique characteristics of each instruction set to ensure fair and meaningful comparisons. Including cross-platform binaries in the dataset is a valuable direction for future research.

Existing obfuscators. Aside from obfuscation techniques, we found that existing obfuscators have limitations in their implementation. In O-LLVM [35], In the process of implementing the SUB method, we found that the substituted instructions are usually optimized back to the original instructions, which would lose the obfuscation effect. To this end, we additional add strategies to prevent the de-obfuscation effect, including basic block splitting, adding reference or inline assemble `nop` instructions in the middle, etc. Besides, BCF and FLA skip the exception-relevant functions.

As for the inter-procedural obfuscation techniques, after it separates and aggregates these features, the searching difficulty increases, and the searching accuracy decreases. From our conclusion in table Table 3, the lack of call-graph consideration makes them unable to adopt inter-procedural obfuscation. We believe our study will raise awareness of inter-procedural obfuscation on binary diffing.

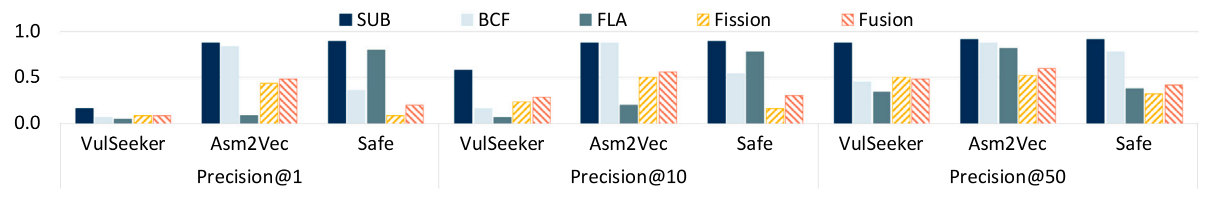


Fig. 8. Precision ratio for top@1/10/50 of vulnerable functions. Higher means stronger vulnerable function searching ability.

Table 5
Vulnerable code detail in BinCodex.

Program	CVE	Funtion	Program	CVE	funtion
JerryScript	CVE-2020–13991	opfunc_spread_arguments	libcurl	CVE-2016–5421	close_all_connections
QuickJS	CVE-2020–22876	compute_stack_size_rec		CVE-2016–7167	curl_easy_unescape
BusyBox1.33.1	CVE-2021–42378	getvar_i_int		CVE-2016–8615	get_line
	CVE-2021–42380	clrvar		CVE-2016–8616	ConnectionExists
	CVE-2021–42381	hash_init		CVE-2016–8617	Curl_base64_encode
	CVE-2021–42382	getvar_s		CVE-2016–8618	alloc_addbyter
	CVE-2021–42379	next_input_file		CVE-2016–8621	parsedate
	CVE-2021–42384	handle_special		CVE-2016–8622	unescape_word
	CVE-2021–42386	nvalloc		CVE-2016–8623	Curl_cookie_getlist
	CVE-2021–42383	evaluate		CVE-2016–8624	parseurlandfillconn
	CVE-2021–42385	evaluate		CVE-2016–8625	curl_version
OpenSSL 1.1.1	CVE-2022–0778	BN_mod_sqrt		CVE-2016–9586	dprintf_formatf
	CVE-2021–3712	EC_GROUP_new_from_ecparameters		CVE-2017–1000100	tftp_send_first
	CVE-2021–3711	sm2_plaintext_size		CVE-2017–1000254	ftp_statemach_act
	CVE-2021–3450	check_chain_extensions		CVE-2017–1000257	imap_state_fetch_resp
	CVE-2021–3449	init_sig_algs		CVE-2017–8817	setcharset
	CVE-2020–1971	GENERAL_NAME_dup		CVE-2018–1000007	Curl_http_output_auth
	CVE-2020–1967	tls1_check_sig_alg		CVE-2018–1000120	ftp_state_list
	CVE-2019–1563	cms_RecipientInfo_ktri_decrypt		CVE-2018–1000120	ftp_done
	CVE-2019–1547	EC_GROUP_set_generator		CVE-2018–1000120	ftp_parse_url_path
	CVE-2019–1543	chacha_init_key		CVE-2018–1000122	readwrite_data
	CVE-2018–0734	dsa_sign_setup		CVE-2018–1000301	Curl_http_readwrite_headers
	CVE-2018–0735	ec_scalar_mul_ladder		CVE-2019–5436	tftp_connect
libcurl	CVE-2014–0138	ConnectionExists		CVE-2019–5482	tftp_connect
	CVE-2014–3613	Curl_cookie_add		CVE-2020–8231	conn_is_conn
	CVE-2014–3620	Curl_cookie_add		CVE-2020–8231	curl_easy_duphandle
	CVE-2014–3707	FormAdd		CVE-2020–8231	curl_multi_add_handle
	CVE-2014–8150	parseurlandfillconn		CVE-2020–8231	Curl_open
	CVE-2015–3143	ConnectionExists		CVE-2020–8231	curl_multi_remove_handle
	CVE-2015–3145	sanitize_cookie_path		CVE-2020–8285	init_wc_data
	CVE-2015–3148	Curl_http_done		CVE-2021–22876	Curl_follow
	CVE-2015–3153	Curl_init_userdefined		CVE-2021–22898	suboption
	CVE-2016–0755	ConnectionExists		CVE-2021–22924	create_conn
	CVE-2016–5419	Curl_clone_ssl_config		CVE-2021–22925	suboption
	CVE-2016–5420	Curl_ssl_config_matches			

Advancing BCSD. Reducing the cost of diffing in binary code similarity detection is a challenge, particularly when dealing with smaller diffing granularity. Context information can be leveraged to narrow down the searching space and mitigate the associated costs. While previous works have predominantly focused on capturing and encoding control flow information, data flow has received less attention from the BCSD perspective. However, considering that data flow is harder to change during obfuscation, there is untapped potential in exploring data flow representation for improved BCSD techniques.

Furthermore, as observed in Section 4, larger binary sizes can lead to decreased diffing accuracy. To address this, one approach is to merge functions into groups, effectively reducing the searching space. An example of such an approach is FirmUp [61], which emphasizes optimizing the searching process rather than solely focusing on feature generation. This highlights the potential for further optimization in the searching process to enhance the accuracy of BCSD techniques.

7. Conclusion

We present a paper that addresses the challenges in the binary code similarity detection dataset and introduces the implementation and evaluation of a novel dataset called BinCodex. The primary objective of the dataset is to provide a standardized framework for evaluating BCSD techniques. To ensure the dataset's effectiveness, several factors are carefully considered during its construction, including diverse code types, a wide range of code samples, incorporating multiple aspects of binary change points, and different levels of workloads in similarity detection tasks. The implementation of BinCodex on a Linux system enables the evaluation of eight state-of-the-art BCSD tools. The results indicate that most BCSD works perform well when default compiler options or intra-procedural code obfuscation are used but face challenges with non-default options and inter-procedural obfuscation techniques. These findings provide valuable insights into the current state of BCSD works and highlight opportunities for future optimizations in the field.

CRediT authorship contribution statement

Peihua Zhang: Writing – original draft, Software, Methodology, Formal analysis, Conceptualization. **Chenggang Wu:** Supervision, Conceptualization. **Zhe Wang:** Writing – review & editing, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper

Appendix

See Tables 5–7.

Table 6
All compiler options in GCC, classified and incremental by the default optimization level. BinTuner uses all these options with all possible combinations.

-O1 (compared with O0)	-O2 (compared with -O1)	-O3 (compared with -O2)	Others
auto-inc-dec branch-count-reg compare-elim	align-functions align-jumps align-labels	gcse-after-reload	early-inlining-insns
combine-stack-adjustments cprop-registers	caller-saves code-hoisting crossjumping	loop-interchange	gcse-cost-distance-ratio
defer-pop delayed-branch forward-propagate	cse-follow-jumps cse-skip-blocks gcse-lm	loop-unroll-and-jam	iv-max-considered-uses
dce guess-branch-probability if-conversion	delete-null-pointer-checks devirtualize	predictive-commoning	reorder-blocks-algorithm=stc
if-conversion2 inline-functions-called-once	devirtualize-speculatively finite-loops	tree-partial-pre	prefetch-loop-arrays
tree-ter ipa-pure-const align-loops tree-ch ipa-reference-addressable merge-constants	gcse inline-functions indirect-inlining inline-small-functions ipa-bit-cp ipa-cp	tree-loop-distribution tree-loop-vectorize	
move-loop-invariants omit-frame-pointer	optimize-strlen partial-inlining ipa-ra	tree-slp-vectorize	
reorder-blocks dse shrink-wrap-separate	isolate-erroneous-paths-dereference	unswitch-loops	
split-wide-types ssa-backprop ssa-phiopt	ipa-icf reorder-blocks-algorithm=stc	vect-cost-model	
tree-bit-ccp tree-ccp tree-dce shrink-wrap	reorder-blocks-and-partition ipa-sra	vect-cost-model=dynamic	
tree-copy-prop tree-dominator-opts tree-dse	lra-remat rerun-cse-after-loop tree-vrp	version-loopsor-strides	
tree-forwprop tree-fre tree-phiprop tree-pta	sched-interblock sched-spec store-merging	split-loops split-paths	
tree-scev-cprop tree-sink tree-slsr tree-sra	thread-jumps tree-builtin-call-dce tree-pre	ipa-cp-clone	
ipa-profile unit-at-a-time ipa-reference tree-coalesce-vars	tree-switch-conversion tree-tail-merge hoist-adjacent-loads peephole2 ipa-vrp expensive-optimizations strict-aliasing schedule-insns2 optimize-sibling-calls reorder-functions schedule-insns	peel-loops	

Table 7
Top-5 non-default optimization settings generated from BinTuner [31].

-O0 -fauto-inc-dec -fforward-propagate -fcombine-stack-adjustments -fcompare-elim -fcprop-registers -fdce -fif-conversion2 -fno-delayed-branch -fdse -fno-defer-pop -fno-merge-constants -fno-guess-branch-probability -ftree-dominator-opts -finline-functions-called-once -fno-ipa-pure-const -fno-ipa-profile -fipa-reference -fbranch-count-reg -fno-move-loop-invariants -freorder-blocks -fno-shrink-wrap -fsplit-wide-types -ftree-copy-prop -ftree-bit-ccp -fno-tree-ter -fssa-backprop -fno-tree-coalesce-vars -fipa-cp-clone -ftree-forwprop -ftree-fre -fno-tree-sink -fno-tree-sra -fsplit-paths -ftree-pta -ftree-ccp -fno-ipa-cp -fno-unit-at-a-time -fno-omit-frame-pointer -ftree-phiprop -fno-tree-ch -ftree-slsr -fpeephole2 -fno-if-conversion -fno-ssa-phiopt -fno-shrink-wrap-separate -fthread-jumps -fno-align-functions -fno-align-labels -fno-align-loops -fstore-merging -fstrict-aliasing -fno-caller-saves -fno-crossjumping -fno-cse-follow-jumps -fcse-skip-blocks -fno-delete-null-pointer-checks -fno-devirtualize -fgcse -fno-gcse-lm -fno-devirtualize-speculatively -fno-expensive-optimizations -fno-hoist-adjacent-loads -finline-small-functions -findirect-inlining -ftree-vectorize -ftree-dce -fno-peel-loops -fno-isolate-erroneous-paths-dereference -fno-lra-remat -fno-optimize-sibling-calls -fno-optimize-strlen -fpartial-inlining -fno-ipa-icf -freorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fsched-interblock -fno-sched-spec -ftree-dse -fipa-sra -fno-schedule-insns -fno-tree-partial-pre -fstrict-overflow -ftree-builtin-call-dce -ftree-switch-conversion -ftree-tail-merge -fno-tree-slp-vectorize -fno-tree-pre -fno-tree-vrp -fno-ipa-ra -freorder-blocks -fno-schedule-insns2 -fno-code-hoisting -fvect-cost-model -fno-ipa-bit-cp -fno-ipa-vrp -freorder-blocks-algorithm=simple -finline-functions -fno-unswitch-loops -fno-predictive-commoning -fno-gcse-after-reload -ftree-loop-vectorize –param early-inlining-insns=295 –param gcse-cost-distance-ratio=52 –param iv-max-considered-uses=661 -fno-tree-loop-distribute-patterns
-O0 -fno-auto-inc-dec -fbranch-count-reg -fno-combine-stack-adjustments -fcompare-elim -fcprop-registers -fdce -fno-defer-pop -ftree-bit-ccp -fipa-profile -fforward-propagate -fguess-branch-probability -fif-conversion2 -fif-conversion -fno-inline-functions-called-once -fno-ipa-pure-const -fdse -fno-ipa-reference -fmerge-constants -fno-move-loop-invariants -fno-reorder-blocks -fno-shrink-wrap -fno-split-wide-types -fdelayed-branch -ftree-ccp -fno-tree-ch -fno-tree-coalesce-vars -ftree-copy-prop -ftree-dce -ftree-dse -ftree-forwprop -fno-tree-fre -ftree-sink -ftree-slsr -ftree-sra -ftree-pta -fno-tree-ter -fno-unit-at-a-time -fno-omit-frame-pointer -fcse-skip-blocks -ftree-dominator-opts -fssa-backprop -fno-align-functions -fno-tree-phiprop -fno-ssa-phiopt -fno-shrink-wrap-separate -fno-thread-jumps -fno-align-labels -fno-align-loops -fcrossjumping -fno-ipa-sra -fno-caller-saves -fno-cse-follow-jumps -fno-delete-null-pointer-checks -fdevirtualize -fno-devirtualize-speculatively -fno-expensive-optimizations -fno-gcse -fgcse-lm -fhoist-adjacent-loads -finline-small-functions -ftree-pre -fno-ipa-cp -fipa-icf -fno-reorder-blocks-and-partition -fpeel-loops -findirect-inlining -fno-isolate-erroneous-paths-dereference -flra-remat -fno-optimize-sibling-calls -foptimize-strlen -fpartial-inlining -fsched-spec -fno-tree-tail-merg -freorder-functions -frerun-cse-after-loop -fsched-interblock -fschedule-insns -fstrict-aliasing -fno-peephole2 -fstore-merging -fipa-ra -fno-tree-builtin-call-dce -fno-tree-switch-conversion -fno-strict-overflow -fno-tree-vrp -fno-reorder-blocks -fno-schedule-insns2 -fipa-vrp -fno-code-hoisting -freorder-blocks-algorithm=simple -fno-ipa-bit-cp -fno-inline-functions -funswitch-loops -fno-gcse-after-reload -fno-split-paths -fno-tree-partial-pre -ftree-slp-vectorize -ftree-loop-vectorize -ftree-loop-distribute-patterns -ftree-vectorize -fno-vect-cost-model -fipa-cp-clone -fno-predictive-commoning –param early-inlining-insns=56 –param gcse-cost-distance-ratio=85 –param iv-max-considered-uses=874
-O0 -fno-auto-inc-dec -fno-branch-count-reg -fno-combine-stack-adjustments -fcompare-elim -fno-cprop-registers -fno-dce -fno-defer-pop -fdse -fno-delayed-branch -fno-forward-propagate -fno-guess-branch-probability -fno-move-loop-invariants -fipa-profile -finline-functions-called-once -fipa-pure-const -fif-conversion -fno-ipa-reference -fmerge-constants -fno-if-conversion2 -fno-reorder-blocks -fshrink-wrap -fno-split-wide-types -fno-align-loops -fno-ssa-phiopt -ftree-pta -funit-at-a-time -fno-omit-frame-pointer -fno-tree-phiprop -fno-tree-dominator-opts -fno-ssa-backprop -fno-devirtualize -ftree-slsr -fshrink-wrap-separate -fno-ipa-cp -falign-functions -fno-align-labels -fno-caller-saves -fno-thread-jumps -fno-ipa-sra

(continued on next page)

Table 7 (*continued*).

-fcrossjumping -ftree-sra -fcse-follow-jumps -fno-cse-skip-blocks -fdelete-null-pointer-checks -fno-devirtualize-speculatively -fno-gcse -fgcse-lm -fexpensive-optimizations -fno-hoist-adjacent-loads -fno-inline-small-functions -fpartial-inlining -fno-ipa-icf -fisolate-erroneous-paths-dereference -fno-optimize-sibling-calls -ftree-pre -fno-sched-spec -fno-optimize-strlen -fno-indirect-inlining -fno-peephole2 -fno-reorder-blocks-and-partition -fno-code-hoisting -ftree-ccp -ftree-ch -ftree-coalesce-vars -ftree-copy-prop -ftree-dce -ftree-dse -ftree-forwprop -ftree-fre -ftree-sink -fno-tree-ter -fstore-merging -freorder-blocks-algorithm=simple -fno-ipa-bit-cp -fno-ipa-vrp -finline-functions -fno-tree-partial-pre -fno-predictive-commoning -fno-tree-vectorize -fpeel-loops -fsplit-paths -fgcse-after-reload -fno-tree-loop-vectorize -fno-tree-loop-distribute-patterns -fno-tree-slp-vectorize -fno-lra-remat -fno-rerun-cse-after-loop -freorder-functions -fno-strict-aliasing -fno-vect-cost-model -ftree-switch-conversion -fno-strict-overflow -ftree-builtin-call-dce -fno-tree-tail-merge -ftree-vrp -fno-ipa-ra -fno-schedule-insns -fno-schedule-insns2 -fsched-interblock -fno-reorder-blocks -fno-ipa-cp-clone -funswitch-loops -fno-tree-bit-ccp –param early-inlining-insns=846 –param gcse-cost-distance-ratio=14
-O3 -fno-auto-inc-dec -fno-branch-count-reg -fno-combine-stack-adjustments -fcompare-elim -fcprop-registers -fno-dce -fno-defer-pop -fno-dse -fdelayed-branch -fforward-propagate -fguess-branch-probability -fif-conversion2 -fif-conversion -fno-inline-functions-called-once -fipa-profile -fno-merge-constants -fno-ipa-reference -fno-tree-coalesce-vars -fmove-loop-invariants -fno-tree-copy-prop -fshrink-wrap -fno-split-wide-types -fno-tree-slsr -freorder-blocks -ftree-bit-ccp -ftree-ccp -fno-tree-pta -fno-tree-dce -ftree-dse -ftree-forwprop -ftree-fre -fno-tree-sink -fno-tree-ch -falign-labels -ftree-sra -fno-tree-ter -fomit-frame-pointer -ftree-phiprop -fno-tree-dominator-opts -fssa-backprop -fno-ssa-phiopt -fthread-jumps -funit-at-a-time -fno-ipa-pure-const -fshrink-wrap-separate -falign-functions -fno-align-loops -fcaller-saves -fno-crossjumping -fcse-follow-jumps -fcse-skip-blocks -fdelete-null-pointer-checks -fdevirtualize -fno-devirtualize-speculatively -fno-expensive-optimizations -fno-gcse -fno-gcse-lm -fhoist-adjacent-loads -finline-small-functions -findirect-inlining -fno-ipa-cp -fno-ipa-icf -fno-isolate-erroneous-paths-dereference -fno-lra-remat -fipa-sra -fno-optimize-sibling-calls -foptimize-strlen -fno-partial-inlining -fschedule-insns -freorder-blocks-and-partition -fno-reorder-functions -ftree-switch-conversion -frerun-cse-after-loop -fno-sched-interblock -fsched-spec -fno-strict-aliasing -fno-strict-overflow -ftree-builtin-call-dce -fno-peephole2 -ftree-tail-merge -fno-tree-pre -fipa-bit-cp -fno-ipa-ra -freorder-blocks -fschedule-insns2 -fno-code-hoisting -fno-store-merging -ftree-vrp -freorder-blocks-algorithm=simple -fipa-vrp -fno-inline-functions -fno-predictive-commoning -fgcse-after-reload -ftree-loop-vectorize -fipa-cp-clone -ftree-loop-distribute-patterns -fno-tree-slp-vectorize -fvect-cost-model -ftree-partial-pre -fpeel-loops -ftree-vectorize -fsplit-paths -funswitch-loops –param early-inlining-insns=482 –param gcse-cost-distance-ratio=59 –param iv-max-considered-uses=105
-O0 -fno-auto-inc-dec -fno-branch-count-reg -fno-combine-stack-adjustments -fcompare-elim -fcprop-registers -ftree-forwprop -fdelayed-branch -fipa-profile -fforward-propagate -fguess-branch-probability -fno-if-conversion2 -fif-conversion -fno-inline-functions-called-once -fipa-pure-const -fno-dce -fipa-reference -fmerge-constants -fmove-loop-invariants -fno-reorder-blocks -fshrink-wrap -fno-split-wide-types -ftree-bit-ccp -ftree-ccp -fno-tree-pta -fno-tree-coalesce-vars -fno-tree-copy-prop -fno-tree-dce -ftree-dse -ftree-slsr -fcrossjumping -fcaller-saves -fno-tree-sra -ftree-sink -fdse -fno-tree-ch -fthread-jumps -fno-unit-at-a-time -fomit-frame-pointer -ftree-phiprop -ftree-dominator-opts -fno-ssa-backprop -fno-ssa-phiopt -fshrink-wrap-separate -fno-gcse -fno-align-functions -falign-labels -fthread-jumps -fno-peephole2 -fno-tree-fre -fcse-follow-jumps -fno-defer-pop -fno-align-loops -fno-hoist-adjacent-loads -fdelete-null-pointer-checks -fno-devirtualize -fdevirtualize-speculatively -fno-expensive-optimizations -fno-cse-skip-blocks -fgcse-lm -finline-small-functions -fno-indirect-inlining -fipa-cp -fno-ipa-sra -fipa-icf -fisolate-erroneous-paths-dereference -fno-lra-remat -fno-optimize-sibling-calls -foptimize-strlen -fno-partial-inlining -fschedule-insns -freorder-blocks-and-partition -fipa-vrp -ftree-ter -fno-sched-interblock -fno-rerun-cse-after-loop -fno-reorder-functions -fno-sched-spec -fno-strict-aliasing -fno-tree-vrp -fno-tree-builtin-call-dce -ftree-switch-conversion -fstrict-overflow -ftree-tail-merge -ftree-pre -fno-reorder-blocks -fschedule-insns2 -fno-code-hoisting -fno-peel-loops -fipa-cp-clone -freorder-blocks-algorithm=simple -fno-ipa-bit-cp -finline-functions -funswitch-loops -fpredictive-commoning -fno-store-merging -fno-gcse-after-reload -fno-tree-loop-vectorize -fno-tree-loop-distribute-patterns -fno-tree-slp-vectorize -fno-vect-cost-model -ftree-partial-pre -fipa-ra -fno-split-paths -ftree-vectorize –param early-inlining-insns=798 –param gcse-cost-distance-ratio=46 –param iv-max-considered-uses=617

References

[1] statista, Number of connected IoT devices worldwide, 2020, https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

[2] S. Cesare, Y. Xiang, W. Zhou, Control flow-based malware variant detection, IEEE Trans. Dependable Secure Comput. 11 (4) (2013) 307–317, http://dx.doi.org/10.1109/TDSC.2013.40.

[3] Y. David, N. Partush, E. Yahav, Statistical similarity of binaries, ACM SIGPLAN Not. 51 (6) (2016) 266–280, http://dx.doi.org/10.1145/2908080.2908126.

[4] Y. Hu, Y. Zhang, J. Li, D. Gu, Cross-architecture binary semantics understanding via similar code comparison, in: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER, vol. 1, IEEE, 2016, pp. 57–67, http://dx.doi.org/10.1109/SANER.2016.50.

[5] T. Blazytko, M. Contag, C. Aschermann, T. Holz, Syntia: Synthesizing the semantics of obfuscated code, in: 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 643–659.

[6] M. Chandramohan, Y. Xue, Z. Xu, Y. Liu, C.Y. Cho, H.B.K. Tan, Bingo: Cross-architecture cross-os binary search, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 678–689, http://dx.doi.org/10.1145/2950290.2950350.

[7] Y. Hu, Y. Zhang, J. Li, D. Gu, Binary code clone detection across architectures and compiling configurations, in: 2017 IEEE/ACM 25th International Conference on Program Comprehension, ICPC, IEEE, 2017, pp. 88–98, http://dx.doi.org/10.1109/ICPC.2017.22.

[8] Y. David, N. Partush, E. Yahav, Similarity of binaries through re-optimization, in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017, pp. 79–94, http://dx.doi.org/10.1145/3062341.3062387.

[9] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, D. Song, Neural network-based graph embedding for cross-platform binary code similarity detection, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 363–376, http://dx.doi.org/10.1145/3133956.3134018.

[10] Y. Duan, X. Li, J. Wang, H. Yin, Deepbindiff: Learning program-wide code representations for binary diffing, in: Network and Distributed System Security Symposium, 2020, http://dx.doi.org/10.14722/ndss.2020.24311.

[11] M. Bourquin, A. King, E. Robbins, Binslayer: accurate comparison of binary executables, in: Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop, 2013, pp. 1–10, http://dx.doi.org/10.1145/2430553.2430557.

[12] S.H. Ding, B.C. Fung, P. Charland, Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization, in: 2019 IEEE Symposium on Security and Privacy, SP, IEEE, 2019, pp. 472–489, http://dx.doi.org/10.1109/SP.2019.00003.

[13] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, Z. Zhang, Neural machine translation inspired binary code similarity comparison beyond function pairs, in: NDSS, The Internet Society, 2019, http://dx.doi.org/10.14722/ndss.2019.23492.

[14] J. Gao, X. Yang, Y. Fu, Y. Jiang, J. Sun, VulSeeker: A semantic learning based vulnerability seeker for cross-platform binary, in: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2018, pp. 896–899, http://dx.doi.org/10.1145/3238147.3240480.

[15] J. Pewny, F. Schuster, L. Bernhard, T. Holz, C. Rossow, Leveraging semantic signatures for bug search in binary programs, in: Proceedings of the 30th Annual Computer Security Applications Conference, 2014, pp. 406–415, http://dx.doi.org/10.1145/2664243.2664269.

[16] J. Pewny, B. Garmany, R. Gawlik, C. Rossow, T. Holz, Cross-architecture bug search in binary executables, in: 2015 IEEE Symposium on Security and Privacy, IEEE, 2015, pp. 709–724, http://dx.doi.org/10.1109/SP.2015.49.

[17] Y. Xu, Z. Xu, B. Chen, F. Song, Y. Liu, T. Liu, Patch based vulnerability matching for binary programs, in: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2020, pp. 376–387, http://dx.doi.org/10.1145/3395363.3397361.

[18] L. Zhao, Y. Zhu, J. Ming, Y. Zhang, H. Zhang, H. Yin, Patchscope: Memory object centric patch diffing, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 149–165, http://dx.doi.org/10.1145/3372297.3423342.

[19] Z. Xu, B. Chen, M. Chandramohan, Y. Liu, F. Song, Spain: security patch analysis for binaries towards understanding the pain and pills, in: 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE, IEEE, 2017, pp. 462–472, http://dx.doi.org/10.1109/ICSE.2017.49.

[20] X. Hu, K.G. Shin, S. Bhatkar, K. Griffin, {MutantX-S}: Scalable malware clustering based on static features, in: 2013 USENIX Annual Technical Conference (USENIX ATC 13), 2013, pp. 187–198.

[21] J. Jang, D. Brumley, S. Venkataraman, Bitshred: feature hashing malware for scalable triage and semantic analysis, in: Proceedings of the 18th ACM Conference on Computer and Communications Security, 2011, pp. 309–320, http://dx.doi.org/10.1145/2046707.2046742.

[22] D. Xu, J. Ming, D. Wu, Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping, in: 2017 IEEE Symposium on Security and Privacy, SP, IEEE, 2017, pp. 921–937, http://dx.doi.org/10.1109/SP.2017.56.

[23] A. Caliskan, F. Yamaguchi, E. Dauber, R. Harang, K. Rieck, R. Greenstadt, A. Narayanan, When coding style survives compilation: De-anonymizing programmers from executable binaries, 2015, http://dx.doi.org/10.48550/arXiv.1512.08546, arXiv preprint arXiv:1512.08546.

[24] W. Jin, S. Chaki, C. Cohen, A. Gurfinkel, J. Havrilla, C. Hines, P. Narasimhan, Binary function clustering using semantic hashes, in: 2012 11th International Conference on Machine Learning and Applications, 1, IEEE, 2012, pp. 386–391, http://dx.doi.org/10.1109/ICMLA.2012.70.

[25] Y. Xue, Z. Xu, M. Chandramohan, Y. Liu, Accurate and scalable cross-architecture cross-os binary code search with emulation, IEEE Trans. Softw. Eng. 45 (11) (2018) 1125–1149, http://dx.doi.org/10.1109/TSE.2018.2827379.

[26] H. Wang, P. Ma, Y. Yuan, Z. Liu, S. Wang, Q. Tang, S. Nie, S. Wu, Enhancing DNN-based binary code function search with low-cost equivalence checking, IEEE Trans. Softw. Eng. (2022) http://dx.doi.org/10.1109/TSE.2022.3149240.

[27] B. Liu, W. Huo, C. Zhang, W. Li, F. Li, A. Piao, W. Zou, αDiff: cross-version binary code similarity detection with dnn, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 667–678, http://dx.doi.org/10.1145/3238147.3238199.

[28] H. Wang, W. Qu, G. Katz, W. Zhu, Z. Gao, H. Qiu, J. Zhuge, C. Zhang, Jtrans: Jump-aware transformer for binary code similarity detection, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, in: ISSTA 2022, Association for Computing Machinery, New York, NY, USA, 2022, pp. 1–13, http://dx.doi.org/10.1145/3533767.3534367.

[29] X. Xu, S. Feng, Y. Ye, G. Shen, Z. Su, S. Cheng, G. Tao, Q. Shi, Z. Zhang, X. Zhang, Improving binary code similarity transformer models by semantics-driven instruction deemphasis, in: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, in: ISSTA 2023, Association for Computing Machinery, New York, NY, USA, 2023, pp. 1106–1118, http://dx.doi.org/10.1145/3597926.3598121.

[30] N. Shalev, N. Partush, Binary similarity detection using machine learning, in: Proceedings of the 13th Workshop on Programming Languages and Analysis for Security, PLAS '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 42–47, http://dx.doi.org/10.1145/3264820.3264821.

[31] X. Ren, M. Ho, J. Ming, Y. Lei, L. Li, Unleashing the hidden power of compiler optimization on binary code difference: An empirical study, in: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021, pp. 142–157, http://dx.doi.org/10.1145/3453483.3454035.

[32] C. Linn, S. Debray, Obfuscation of executable code to improve resistance to static disassembly, in: Proceedings of the 10th ACM Conference on Computer and Communications Security, 2003, pp. 290–299, http://dx.doi.org/10.1145/948109.948149.

[33] C. Collberg, S. Martin, J. Myers, J. Nagra, Distributed application tamper detection via continuous software updates, in: Proceedings of the 28th Annual Computer Security Applications Conference, 2012, pp. 319–328, http://dx.doi.org/10.1145/2420950.2420997.

[34] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, A. Pretschner, Code obfuscation against symbolic execution attacks, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, 2016, pp. 189–200, http://dx.doi.org/10.1145/2991079.2991114.

[35] P. Junod, J. Rinaldini, J. Wehrli, J. Michielin, Obfuscator-LLVM–software protection for the masses, in: 2015 IEEE/ACM 1st International Workshop on Software Protection, IEEE, 2015, pp. 3–9, http://dx.doi.org/10.1109/SPRO.2015.10.

[36] H. Xu, Y. Zhou, Y. Kang, F. Tu, M. Lyu, Manufacturing resilient bi-opaque predicates against symbolic execution, in: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE, 2018, pp. 666–677, http://dx.doi.org/10.1109/DSN.2018.00073.

[37] M. Hammad, J. Garcia, S. Malek, A large-scale empirical study on the effects of code obfuscations on android apps and anti-malware products, in: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 421–431, http://dx.doi.org/10.1145/3180155.3180228.

[38] H. Wang, S. Wang, D. Xu, X. Zhang, X. Liu, Generating effective software obfuscation sequences with reinforcement learning, IEEE Trans. Dependable Secure Comput. (2020) http://dx.doi.org/10.1109/TDSC.2020.3041655.

[39] P. Zhang, C. Wu, M. Peng, K. Zeng, D. Yu, Y. Lai, Y. Kang, W. Wang, Z. Wang, Khaos: The impact of inter-procedural code obfuscation on binary diffing techniques, in: Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization, 2023, pp. 55–67.

[40] H. Geng, M. Zhong, P. Zhang, F. Lv, X. Feng, OPTango: Multi-central representation learning against innumerable compiler optimization for binary diffing, in: 2023 IEEE 34th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2023, pp. 774–785.

[41] L. Massarelli, G.A.D. Luna, F. Petroni, R. Baldoni, L. Querzoni, Safe: Self-attentive function embeddings for binary similarity, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2019, pp. 309–329, http://dx.doi.org/10.1007/978-3-030-22038-9_15.

[42] zynamics GmbH and Google LLC, BinDiff manual, 2022, http://www.zynamics.com/bindiff/manual/index.html.

[43] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, H. Yin, Scalable graph-based bug search for firmware images, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 480–491, http://dx.doi.org/10.1145/2976749.2978370.

[44] L. Luo, J. Ming, D. Wu, P. Liu, S. Zhu, Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 389–400, http://dx.doi.org/10.1145/2635868.2635900.

[45] J. Ming, D. Xu, Y. Jiang, D. Wu, {BinSim}: Trace-based semantic binary diffing via system call sliced segment equivalence checking, in: 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 253–270.

[46] S. Wang, D. Wu, In-memory fuzzing for binary code similarity analysis, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2017, pp. 319–330, http://dx.doi.org/10.1109/ASE.2017.8115645.

[47] C. Nachenberg, Computer virus-antivirus coevolution, Commun. ACM 40 (1) (1997) 46–51, http://dx.doi.org/10.1145/242857.242869.

[48] K.A. Roundy, B.P. Miller, Binary-code obfuscations in prevalent packer tools, ACM Comput. Surv. 46 (1) (2013) 1–32, http://dx.doi.org/10.1145/2522968.2522972.

[49] M.F.X.J. Oberhumer, L. Molnár, J.F. Reiser, The ultimate packer for eXecutables, 2022, https://upx.github.io/.

[50] Oreans Technologies, Themida overview, 2022, https://www.oreans.com/themida.php.

[51] Z. Tang, K. Kuang, L. Wang, C. Xue, X. Gong, X. Chen, D. Fang, J. Liu, Z. Wang, SEEAD: A semantic-based approach for automatic binary code de-obfuscation, in: 2017 IEEE Trustcom/BigDataSE/ICESS, 2017, pp. 261–268, http://dx.doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.246.

[52] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, ACM Comput. Surv. (CSUR) 44 (2) (2008) 1–42, http://dx.doi.org/10.1145/2089125.2089126.

[53] A. Dinaburg, P. Royal, M. Sharif, W. Lee, Ether: malware analysis via hardware virtualization extensions, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 51–62, http://dx.doi.org/10.1145/1455770.1455779.

[54] M. Sharif, A. Lanzi, J. Giffin, W. Lee, Automatic reverse engineering of malware emulators, in: 2009 30th IEEE Symposium on Security and Privacy, IEEE, 2009, pp. 94–109, http://dx.doi.org/10.1109/SP.2009.27.

[55] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, P.G. Bringas, SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers, in: 2015 IEEE Symposium on Security and Privacy, IEEE, 2015, pp. 659–673, http://dx.doi.org/10.1109/SP.2015.46.

[56] H. Wang, P. Ma, S. Wang, Q. Tang, S. Nie, S. Wu, Sem2vec: Semantics-aware assembly tracelet embedding, ACM Trans. Softw. Eng. Methodol. 32 (4) (2023) http://dx.doi.org/10.1145/3569933.

[57] Z. Luo, P. Wang, B. Wang, Y. Tang, W. Xie, X. Zhou, D. Liu, K. Lu, VulHawk: Cross-architecture vulnerability detection with entropy-based binary code search, in: NDSS, 2023.

[58] D. Kim, E. Kim, S.K. Cha, S. Son, Y. Kim, Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned, IEEE Trans. Softw. Eng. (2022) 1–23, http://dx.doi.org/10.1109/TSE.2022.3187689.

[59] X. Xu, Q. Zheng, Z. Yan, M. Fan, A. Jia, T. Liu, Interpretation-enabled software reuse detection based on a multi-level birthmark model, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering, ICSE, IEEE, 2021, pp. 873–884, http://dx.doi.org/10.1109/ICSE43902.2021.00084.

[60] S. Yang, L. Cheng, Y. Zeng, Z. Lang, H. Zhu, Z. Shi, Asteria: Deep learning-based AST-encoding for cross-platform binary code similarity detection, in: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE, 2021, pp. 224–236.

[61] Y. David, N. Partush, E. Yahav, Firmup: Precise static detection of common vulnerabilities in firmware, ACM SIGPLAN Not. 53 (2) (2018) 392–404, http://dx.doi.org/10.1145/3173162.3177157.

[62] H. Huang, A.M. Youssef, M. Debbabi, Binsequence: Fast, accurate and scalable binary code reuse detection, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 155–166, http://dx.doi.org/10.1145/3052973.3052974.

[63] M. Ollivier, S. Bardin, R. Bonichon, J.-Y. Marion, How to kill symbolic deobfuscation for free (or: unleashing the potential of path-oriented protections), in: Proceedings of the 35th Annual Computer Security Applications Conference, 2019, pp. 177–189, http://dx.doi.org/10.1145/3359789.3359812.

[64] S. Eschweiler, K. Yakdan, E. Gerhards-Padilla, discovRE: Efficient cross-architecture identification of bugs in binary code, in: NDSS, vol. 52, 2016, pp. 58–79, http://dx.doi.org/10.14722/ndss.2016.23185.

[65] S. Alrabaee, P. Shirani, L. Wang, M. Debbabi, Fossil: a resilient and efficient system for identifying foss functions in malware binaries, ACM Trans. Priv. Secur. 21 (2) (2018) 1–34, http://dx.doi.org/10.1145/3175492.

[66] Y. David, E. Yahav, Tracelet-based code search in executables, ACM SIGPLAN Not. 49 (6) (2014) 349–360, http://dx.doi.org/10.1145/2594291.2594343.

[67] Q. Feng, M. Wang, M. Zhang, R. Zhou, A. Henderson, H. Yin, Extracting conditional formulas for cross-platform bug search, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 346–359, http://dx.doi.org/10.1145/3052973.3052995.

[68] Y. Hu, Y. Zhang, J. Li, H. Wang, B. Li, D. Gu, Binmatch: A semantics-based hybrid approach on binary code clone analysis, in: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2018, pp. 104–114, http://dx.doi.org/10.1109/ICSME.2018.00019.

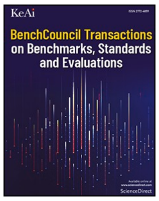
[69] Y. Liu, H. Wang, Tracking mirai variants, Virus Bull. (2018) 1–18.

[70] D. Kim, E. Kim, S.K. Cha, S. Son, Y. Kim, Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned, IEEE Trans. Softw. Eng. (2022).

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-onbenchmarks-standards-and-evaluations/



Full Length Article

Analyzing the impact of opportunistic maintenance optimization on manufacturing industries in Bangladesh: An empirical study

Md. Ariful Alam [a,b,*], Md. Rafiquzzaman [a], Md. Hasan Ali [c], Gazi Faysal Jubayer [d]

[a] Department of Industrial Engineering and Management, Khulna University of Engineering & Technology, Khulna 9203, Bangladesh
[b] Department of Industrial and Production Engineering, Bangladesh Army University of Science and Technology, Saidpur Cantonment, Saidpur 5310, Bangladesh
[c] Department of Industrial Engineering, Dalhousie University, 5269 Morris Street, Halifax NS B3H 4R2, Canada
[d] Department of Mechanical Engineering, Khulna University of Engineering & Technology, Khulna 9203, Bangladesh

ARTICLE INFO

Keywords:
Opportunistic maintenance
Multi-unit
Maintenance cost
Manufacturing
Simulation

ABSTRACT

The study investigates the impact of opportunistic maintenance (OM) optimization on manufacturing industries, especially in Bangladesh, to reduce maintenance costs. To that end, OM strategies have been proposed and optimized for multi-unit manufacturing systems, whereas most of the existing research is for single- or two-unit systems. OM strategies in this research cover one of the three policies: preventive replacement, preventive repair, and a two-level maintenance approach. The proposed two-level maintenance approach is a combination of lower-level maintenance, known as preventive repair, and higher-level maintenance, known as preventive replacement. Simulation optimization (SO) techniques using Python were utilized to evaluate the strategies. Historical data from two of Bangladesh's most promising and significant sectors, the footwear and railway industries, was used as the case study. Compared to the currently utilized corrective maintenance approach, the two-level maintenance approach is the most effective for both case studies, demonstrating cost savings of 16.9 % and 22.4 % for the footwear and railway industries, respectively. This study reveals that manufacturing industries can achieve significant cost savings by implementing the proposed OM strategies, a concept that has yet to be explored in developing countries like Bangladesh. However, the study considered the proposed approaches for major components of the system, and more significant benefits can be achieved if it is possible to apply them to all critical components of the system.

1. Introduction

Maintenance is the set of decisions and actions required by management, technical experts, and administrators to keep a system or asset functioning properly or bring it back to its previous state [1]. Maintenance has been treated as a neglected business function for many years, especially in manufacturing and engineering [2]. Two main reasons may have initiated this situation about maintenance: the first reason is thinking of the maintenance department as a support activity that has no direct relation with production processes; the second reason relates to the complexities of measuring maintenance contribution to firm profits. Thus, maintenance is usually considered a cost rather than an investment. Therefore, it can be concluded that such earlier views of negligence regarding maintenance are among the leading causes of low maintenance efficiency in industries [3]. Adequate and proper maintenance not only improves the company's essential performance, including product quality, reliability, and productivity, but also reduces monetary losses by increasing manufacturing system availability, keeping the standard of products, and maintaining workplace safety [4]. Also, a proper maintenance approach enhances system reliability, improves overall effectiveness, and minimizes long-term costs [5].

Over several decades, researchers have gathered a wealth of data on maintenance. According to them, maintenance can be broadly classified into two primary categories: preventative maintenance (PM) and corrective maintenance (CM). Preventive maintenance (PM) can be described as planned actions to enhance the anticipated lifespan of operational systems. In contrast, corrective maintenance (CM) addresses unforeseen component or system failures to restore operational functionality [6]. Researchers have emphasized on PM more than CM [7]. However, the existing maintenance practices in the manufacturing industries, especially in Bangladesh, are mainly failure-based, i.e., corrective maintenance. Sudden failures in a manufacturing system results in a great deal of production loss for the company here. However,

* Corresponding author at: BAUST, Saidpur Cantonment, Saidpur 5310, Bangladesh.
E-mail address: arif.ipe@baust.edu.bd (Md.A. Alam).

https://doi.org/10.1016/j.tbench.2024.100172
Received 17 May 2024; Received in revised form 26 July 2024; Accepted 8 August 2024
Available online 9 August 2024

Notations

θ	component scale parameter for the Weibull distribution
β	component shape parameter for the Weibull distribution
N	number of machines in the system
Z	number of parts in a machine
C_A	estimated daily average cost of maintenance of a machine
p	component age percentage
r	age restoration factor
C_{PV}	cost of preventative replacement
C_{PF}	fixed cost of preventive maintenance
C_P	cost of preventive maintenance
C_R	replacement cost of failed component
C_{PD}	predetermined labor cost rate for maintenance action (in Tk./hour)
C_{TR}	total replacement cost of failed component
$C_{support}$	supportive cost for preventive maintenance in a machine in the system
R_L	cost of lost production (per unit)
TC	total cost of maintenance
$Tk.$	Tk. is a Bangladeshi currency. At the time of the study, the exchange rate was 1 USD = 117.09 Tk.
$MTTCA$	mean time to corrective action
$MTTPM$	mean time to preventive maintenance
$MTTPRA$	mean time to preventive replacement action
$MTTPrA$	mean time to preventive repair action
P_R	production rate (units per hour)
$LF_{z,n}$	lifetime that is generated for component z in machine n.
$FA_{z,n}$	actual age of failure of part z in machine n
$IR_{z,n}$	indicator value for failure replacement
$IP_{z,n}$	preventive maintenance action value
IS_n	an indicator value that signifies whether the system has undergone any maintenance.
t_I	the total length of the iteration

other kinds of maintenance, like scheduled preventive maintenance (PM) and condition-based maintenance (CBM), are practiced in some industries to avoid such failures. Due to time and resource constraints, this kind of maintenance is not always possible to fully implement in practical situations. Moreover, the cost of preventive maintenance (PM) activities in industries is rising due to the growing intricacy of industrial systems and the broader implementation of PM [8]. So, most researchers are now attempting to optimize PM actions to save expenses. Sudden breakdowns of machines can be taken as an opportunity to implement maintenance on other parts or machines in the system. This kind of maintenance is known as opportunistic maintenance (OM). OM does not occur at a predetermined time or condition. Instead, it occurs when an "opportunity" arises [9]. Thus, OM saves time and money. The repairs of failures in different system components are widely acknowledged as potential opportunities to perform preventive maintenance on specific components within the systems under consideration [10,11].

OM has mainly been followed in the industries of developed countries, especially in the power sectors [12–15], gas [16], and railway sectors [17,18]. In South Asia, Bangladesh is one of the few emerging countries that has made such an exciting promise of economic growth [19]. Like all other developing countries, Bangladesh needs to industrialize more and more for its economy and society to grow quickly and steadily. However, the large and significant investment in the industrial and manufacturing sectors exposes many difficulties in achieving the lowest maintenance and operational costs. To lower these expenses to the greatest extent, exploring and implementing more intelligent and affordable maintenance strategies is essential for ensuring the country's sustained development. So, the industries of developing countries like Bangladesh can benefit significantly from adopting such OM strategies. The subsequent steps advance by answering the following research questions (RQ):

RQ1: Can opportunistic maintenance strategies reduce costs in manufacturing industries compared to corrective maintenance?

RQ2: Which OM strategy (replacement, repair, two-level) is the most cost-effective for manufacturing industries, especially Bangladesh's footwear and railway industries?

This paper offers a novel contribution by proposing and evaluating opportunistic maintenance strategies for multi-unit manufacturing industries, whereas most of the existing research is for single or two-unit systems. Moreover, the concept has yet to be explored for manufacturing industries in developing countries like Bangladesh and OM has many challenges here due to resource constraints and lack of advanced predictive maintenance technologies. Three OM strategies (preventive replacement, preventive repair, and a two-level maintenance approach) are proposed as solutions to optimize maintenance practices and reduce costs. The proposed two-level approach is a unique maintenance approach that distinguishes between a lower-level approach called preventive repair and a higher-level approach called preventive replacement. These two maintenance levels are intended to be carried out at distinct stages of the equipment's lifespan. Two case studies, (i) the footwear and (ii) the railway industries, are employed to verify the proposed OM strategies and demonstrate the potential impact of OM on cost savings. The feasibility of implementing OM strategies has been checked by comparing them to the currently used corrective maintenance approach and doing a sensitivity analysis. The study identifies the two-level maintenance approach as the most cost-effective option in the footwear and railway industries, providing valuable insights for decision-makers.

The rest of the paper is divided into subsequent sections. Section 2 comprehensively analyzes the contemporary literature on maintenance and opportunistic maintenance. Problem description and proposed opportunistic maintenance policy formulation are covered in Sections 3 and 4, respectively. Solution methodologies to assess the costs associated with suggested OM policies are presented in Section 5. Section 6 validates the proposed strategies through two different Bangladeshi manufacturing industry case studies, a comparative study, and a sensitivity analysis. Section 7 encompasses a summary of the research's findings, an evaluation of its limitations, and recommendations for potential areas of future research.

2. Literature review

Opportunistic maintenance offers numerous benefits that significantly boost the efficiency and cost-effectiveness of maintenance operations. Simultaneously maintaining several components, companies can reduce labor and material costs while minimizing total downtime, thus enhancing operational availability. This method ensures the optimal use of resources such as tools, spare parts, and personnel and facilitates better planning and scheduling. Regular maintenance based on opportunity helps to prevent unexpected failures, extending the lifespan of equipment and improving reliability, thereby decreasing the chances of sudden breakdowns [20]. Additionally, routine checks and maintenance ensure equipment meets safety standards, enhancing worker protection. Predictable maintenance activities also enable better financial planning and budget allocation, and well-maintained equipment operates more efficiently, leading to energy savings. Regular maintenance helps ensure compliance with industry regulations and standards, and data-driven insights from maintenance activities can inform future strategies [21]. Reliable operations resulting from consistent maintenance led to better service delivery and higher customer satisfaction. Maintenance intervals provide opportunities to integrate new technologies or improvements into existing systems, fostering innovation. Properly maintained equipment is less likely to leak or emit harmful substances, benefiting the environment. Finally, regular maintenance builds a more resilient operation capable of handling unexpected disruptions [20,21].

Manufacturing-based industrial sectors in Bangladesh typically struggle to make a proper profit due to higher maintenance and operational costs. Consequently, most industries view maintenance as a cost rather than an investment. However, maintenance significantly impacts multiple operational aspects such as production quantity, expenses, asset reliability, equipment availability, quality of the finished products, environmental sustainability, worker and end-user health and safety, and social welfare [22]. Due to limited time and resources, many industries in developing countries like Bangladesh cannot fully realize these benefits. A more adaptable and less expensive maintenance system is needed to address these issues. Opportunistic maintenance (OM) is one approach to achieve this, particularly for multi-component systems [23]. Implementing opportunistic maintenance in multi-stage manufacturing systems can yield substantial benefits [24].

The paper [12] presented an opportunistic maintenance strategy for wind turbines, addressing challenges posed by stochastic weather conditions and spare parts management. Using a Markov chain model to simulate wind speed time series, the study calculated maintenance wait times due to weather constraints. The economic benefits are demonstrated through numerical examples, showing a reduction in life cycle operation and maintenance costs by 10.92 % and 18.30 % compared to static-opportunistic and non-opportunistic maintenance strategies, respectively. According to Wang et al. [25], the maintenance policy that includes OM activities is the most effective, resulting in significantly lower average costs compared to other policies. Adopting a policy that incorporates condition-based and age-based OM measures can decrease maintenance expenses and enhance efficiency of a two-unit system. The model has potential for expansion to systems with more than two units [25]. Another study showed that employing a condition-based opportunistic maintenance strategy in offshore wind farms saves 32.46 % of costs compared to corrective and preventive maintenance strategies (CPM). However, this study did not consider maintenance time, which significantly impacts average maintenance costs [26]. In another study, Wang et al. [27] showed that a strategy utilizing the downtime of a unit to opportunistically repair other components resulted in savings of £130.0 per month and £25.2 per month compared to two policies where OM is not followed. In a comparative analysis, Li et al. [28] demonstrated that age-based OM can reduce the annual cost by 2.6 % and 1.5 %, respectively, when compared to two traditional opportunistic maintenance strategies. Bakhtiary et al. [17] introduced a novel approach to scheduling tamping interventions to minimize total maintenance costs. The proposed policy establishes an Opportunistic Maintenance Threshold (OMT) for preventive tamping on railway segments, leveraging a steady-state genetic algorithm to find the optimal OMT and tamping schedule. Using data from a railway line in Sweden, the study demonstrates that implementing an OMT can reduce machine preparation costs by approximately 46 %.

The paper [29] presented an advanced OM strategy tailored for multi-component systems in wind turbines, considering seasonal variations. It analyzed the impact of minimal repair, imperfect repair, and replacement on a component's effective age. A dynamic maintenance threshold is established to minimize the life-cycle maintenance cost of wind turbines. The strategy used a genetic algorithm for optimization and was validated through a case study, suggesting that higher component reliability and maintainability reduce the frequency of repairs and replacements. Research [30] concluded that opportunistic production-maintenance synchronization offers a viable solution for optimizing PM scheduling. Companies can achieve substantial cost savings and operational improvements by using production breaks for maintenance activities. The proposed model and algorithm provided a robust framework for future research and practical applications in various industrial settings. A study highlights the trade-offs between different maintenance approaches, noting the fixed costs of planned maintenance, the benefits of OM, and the expenses related to premature part replacements. When the cost of replacing premature parts is very high, OM may not be advantageous [31]. The timeframe of OM is often set to a constant value [28,32]. However, since different components are maintained in different OM windows, the total time required for maintenance varies. Thus, a static maintenance window is unrealistic [28,32]. Table 1 displays the literature review summary and a comparison with the current work.

2.1. Research gaps, novelty, and contributions

Table 1 provides a summary of the most recent literature that has investigated OM. Only a handful of studies dealt with manufacturing industries; only one dealt with multi-component and multi-unit manufacturing systems. To fill this research gap, our paper makes the following contributions:

- Our study incorporated three strategies to determine the best cost optimization method. The application of OM in Bangladesh presents unique challenges and opportunities compared to other countries. Bangladesh's manufacturing industries often face resource constraints and a lack of advanced maintenance infrastructure, hindering the implementation of sophisticated OM strategies. However, the potential for cost savings is significant due to the high failure rates and maintenance costs in these industries. Unlike in developed countries where OM strategies are supported by advanced predictive maintenance technologies, in Bangladesh, the focus is on optimizing existing resources and processes to achieve similar benefits. Our study shows that, with appropriate adaptation, OM strategies can be highly effective even in resource-limited settings.
- Most existing studies have proposed and implemented OM for single- or two-unit systems. Our study deals with more than two units as well as multi-component systems.
- Unlike most previous studies that avoided or ignored the consideration of maintenance duration [25,26], our study considers maintenance duration. In real life, the average maintenance cost is primarily impacted by maintenance duration, regardless of the maintenance crew's experience and expertise, which also indirectly reflects production loss.
- Our simulation-optimization approach has been implemented for two different types of manufacturing industries with uncertain OM time windows, whereas most literature studies considered deterministic OM time windows.

3. Problem description

Techniques for maintenance based on opportunity are distinguished by the component age percentage values where a decision on the component's maintenance can be made. Typically, the proposed policy is displayed in Fig. 1. Imagine that there has been a failure in a manufacturing system. The maintenance team is sent out to replace the malfunctioning component, and they take advantage of the situation to do preventative maintenance on the remaining components of the machine that will meet the condition. Assume that component *i* failed and

Table 1
A comparison of the present work with the most relevant papers in the literature.

Refs.	Types of System			Types of components			Types of Industries			Objective function		OM time Window		Maintenance duration		Solution Approach			
	Single unit	Two unit	More than two unit	Single	Two	More than two	Manufacturing	Power	Other	Cost	Reliability	Deterministic	Stochastic	Considered	Avoided	SO	Exact Methods	Metaheuristic	Approximate method
[12]	✓					✓		✓		✓	✓			✓					✓
[25]		✓			✓			✓		✓		✓			✓				✓
[26]	✓					✓		✓		✓		✓			✓	✓			
[29]	✓					✓		✓		✓								✓	
[17]			✓						✓	✓		✓			✓			✓	
[30]			✓				✓							✓				✓	
[27]		✓			✓			✓		✓		✓			✓				✓
[28]		✓				✓		✓		✓					✓	✓			
[32]			✓			✓	✓			✓		✓			✓	✓			
[31]	✓					✓		✓			✓				✓	✓			
Present work			✓			✓	✓			✓			✓	✓		✓			

component x will have to go through a preventative maintenance operation since its age is greater than the limit, which is now $p \times MTTF_x$. Where p is the component age percentage and $MTTF$ stands for the mean time to failure.

On the other hand, because the age of the component y is smaller than the $p \times MTTF_y$, there will be no maintenance work done on it, and it will continue to function either until the next opportunity arises or it may break down first.

The following presumptions provide the foundation for the suggested policies:

- The rate of failure for each individual component in a manufacturing system follows a Weibull distribution.
- Each component in a machine degrades independently.
- Failure of the machine results from any component failure.
- Assume that there are N number of machines in the system and each machine has Z important components.
- Any machine downtime due to a failure or maintenance action will result in production losses for the manufacturing plant.

4. Proposed Opportunistic Maintenance (OM) optimization strategies

The following section presents the methods followed in this study and details of the proposed OM optimization strategies.

4.1. Preventive repair and age restoration factor

The preventive repair approach is a novel maintenance methodology that has gained popularity in recent decades as an alternative to the conventional categorization of maintenance [33]. It will not make components as good as new. It will reduce the starting age of the component with a probability represented by *r,* where *r* is the age restoration factor. The age restoration factor represents the degree of improvement a component will experience following a repair action, specifically a decrease in age. It is used to find the component's age following a preventive repair. Here, $0 \leq r \leq 1$.

$$Component'\,sage\,after\,maintenance,\ Age_{new} = Age_{old} - Age_{old} \times r \tag{1}$$

$$Failure\,age\,after\,maintenance = r \times LF_{new} + (1 - r) \times LF_{Old} \tag{2}$$

Assume, for instance, that the age restoration factor, r=*0.6,* a hydraulic pipe's lifetime is 10 months (LF_{Old}) and that it has an age of 4 months (Age_{old}) before maintenance. And if it is replaced by a new one with a lifespan of 12 months (LF_{new}). As a result, according to Eq. (1), this hydraulic pipe will have a new age of 4-4 × 0.6=1.6 months after preventive repair, whereas, according to Eq. (2), maintenance activity with a preventive repair approach renews its failure age to 12 × 0.6+ 10 × 0.4=11.2 years.

4.2. Maintenance cost functions

In this study, the cost models include the various maintenance costs along with production losses during downtime. The detailed cost functions used in this paper are given in the following subsections:

4.2.1. Failure replacement and cost of corrective maintenance

When a failure occurs in the manufacturing system, a failure replacement action must be performed. If an industry follows only a failure replacement strategy, it will be regarded as a corrective maintenance action. The costs of Corrective maintenance are as follows:

$$TC = C_R \times IR_{z,n} + (C_{PD} + P_R \times R_L) \times MTTCA_z \tag{3}$$

Here, $IR_{z,n}$ is the indicator of failure. Due to failure, other costs, such as labor costs and production loss, also occured, which will be described in the next sections.

$$\text{Similarly, cost of lost production } C_{LP} = P_R \times R_L \times \sum_{Z=1}^{Z} MTTCA \text{ when } \sum_{Z=1}^{Z} MTTCA \geq MTTPM \tag{9}$$

4.2.2. Opportunistic preventive maintenance costs

If preventive maintenance is performed at an opportunity, the cost will be updated as follows:

$$C_{PM} = \sum\nolimits_{z=1}^{Z} C_{P,Z} \times IP_z + C_{Support} \times IS_N \tag{4}$$

For the preventive replacement approach, the cost of opportunistic preventive maintenance will be updated as follows:

$$TC = C_R \times IR_{z,n} + \sum_{n=1}^{N} \left(\sum\nolimits_{z=1}^{Z} C_{P,Z} \times IP_z + C_{Support} \times IS_N \right) + C_{PD} \times \left(MTTCA_z + \sum_{z=1}^{Z} MTTPM \right) + P_R \times R_L \times \sum_{Z=1}^{Z} MTTPM \tag{11}$$

$$C_{P,Z} = C_{PV} + C_{PF} \tag{5}$$

For preventive repair approach, the total cost of opportunistic preventive maintenance will be updated by assuming that the cost of preventive repair is dependent on the variable r, as in the following equation:

$$C_{P,Z} = r^2 C_{PV} + C_{PF} \quad where \quad 0 \leq r \leq 1 \tag{6}$$

In this context, C_{PV} represents the preventive replacement cost, while C_{PF} stands for the fixed cost for preventive maintenance. The entire preventative cost is $C_{PV+}C_{PF}$ will occur when the age restoration factor is 1 i.e., 100 % age restoration which is practically impossible.

4.2.3. Labor cost

Total labor costs depend on the total maintenance time and the predetermined labor cost rate i.e.,

$$C_L = C_{PD} \times \left(MTTCA_z + \sum_{z=1}^{Z} MTTPM \right) \tag{7}$$

Here, *MTTPM* will be *MTTPRA* for preventive replacement and *MTTPrA* for preventive repair action.

4.2.4. Cost of lost production

Production losses may occur for several reasons. It will occur in two ways at a time. Production loss will be calculated due to failure replacement action and opportunistic preventive maintenance action. As opportunistic maintenance in the model will be implemented at the same time of corrective maintenance, calculating production loss has some considerations in this case. When, the total mean time to preventive maintenance *(MTTPM)* will be larger than *MTTCA*, considering *MTTPM* is enough to calculate the cost of lost production and vice versa.

So, the cost of lost production,

$$C_{LP} = P_R \times R_L \times \sum_{Z=1}^{Z} MTTPM \text{ when } \sum_{Z=1}^{Z} MTTPM \geq MTTCA \tag{8}$$

Here, *MTTPM* will be *MTTPRA* for preventive replacement and *MTTPrA* for preventive repair action.

4.2.5. Total maintenance cost

The total cost due to failure replacement and opportunistic maintenance will be the sum of all maintenance-related costs, which is given as:

$$TC = C_R + C_{PM} + C_L + C_{LP} \tag{10}$$

So, the total cost of maintenance will be as follows:

Here, *MTTPM* is used to calculate the cost of lost production as in most of the cases, total *MTTPM* is larger than *MTTCA*. However, *MTTCA* can also be used if it becomes larger according to Section 4.2.4. To simplify the model representation, this approach of using *MTTPM* is followed in the whole study but real scenario has been considered in the simulation process.

The average maintenance cost per machine per day will be:

$$C_A = \frac{TC}{t_I \times N} \tag{12}$$

4.3. Opportunistic maintenance strategies

Opportunistic maintenance procedures are covered in three ways, as described in the following subsections.

4.3.1. Strategy 1: maintenance based on opportunities with a preventive replacement approach

The maintenance policy includes the implementation of corrective replacements as necessary. Additionally, this provides an opportunity to perform preventative replacements on various components within the same machine and other machines in the system. Replacement action will bring the component as good as new. Maintenance selection is dependent upon the component's age at the moment of failure. As soon as a failure occurs in machine n (n=1,..., N), do preventive replacement of component z (z=1,..., Z) if $agek_{z,n} \geq MTTF_z \times p$. Without preventative replacement, the component will continue to function until the machine fails again.

The objective function is

$$MinC_A(p) = \frac{\left[\begin{array}{l} C_R \times IR_{z,n} + \sum_{n=1}^{N} \left(\sum\nolimits_{z=1}^{Z} C_{P,Z} \times IP_z + C_{Support} \times IS_N \right) + \\ C_{PD} \times \left(MTTCA_z + \sum_{z=1}^{Z} MTTPRA \right) + P_R \times R_L \times \sum_{Z=1}^{Z} MTTPRA \end{array} \right]}{t_I \times N} \tag{13}$$

subject to

$0 \leq p \leq 1$

Where variable p is the component age percentage, C_A represents the estimated daily average cost of machine maintenance. Other notations are described earlier. The goal is to obtain the optimum age percentage, p in order to lower the anticipated daily average cost of maintenance.

4.3.2. *Strategy 2: maintenance based on opportunities with a preventive repair approach*

In this action, corrective replacements are carried out when needed, and we use the occasion as an opportunity to carry out preventive repair on other parts of the same machine and other machines in the system. The age of the machine part at the time of failure dictates the maintenance strategy used. If $age_{z,\,n} \geq MTTF_z \times p$, then preventive repair should be conducted at the moment of failure by decreasing the age of the component by r on component z (z = 1,...,Z) in machine n (n = 1,...,N). The preventive repair techniques described in section (4.1) are put into reality. In the event that the component does not meet the condition, it will be utilized until a subsequent breakdown occurs.

The objective function in this regard is

$$MinC_A(p,r) = \frac{\left[\begin{array}{l} C_R \times IR_{z,n} + \sum_{n=1}^{N}\left(\sum_{z=1}^{Z} C_{P,Z} \times IP_z + C_{Support} \times IS_N\right) + \\ C_{PD} \times \left(MTTCA_z + \sum_{z=1}^{Z} MTTPrA\right) + P_R \times R_L \times \sum_{Z=1}^{Z} MTTPrA \end{array}\right]}{t_I \times N} \tag{14}$$

subject to

$0 \leq p \leq 1$
$0 \leq r \leq 1$

Where p and r are design variables. p represents the age percentage for a component, and r represents the age restoration factor and other notations are the same as before. It is crucial to identify the optimum p and r to achieve the minimum daily maintenance cost per machine.

4.3.3. *Strategy 3: maintenance based on opportunities with a two-level approach*

In this model, corrective replacements are carried out as needed under this maintenance strategy, and we use the occasion to carry out two-level preventive maintenance procedures on other parts of the same machine and other machines of the system. Two age thresholds, $MTTF \times p_L$ and $MTTF \times p_H$ where $p_H > p_L$, define preventive repair and preventive replacement actions, respectively. The choice of maintenance is based on the component's age at the time of failure. If $MTTF_z \times p_H \geq age_{z,n} \geq MTTF_z \times p_L$, then preventive repair should be conducted at the moment of failure by decreasing the age of the component by r on component z (z = 1,...,Z) in machine n (n = 1,...,N). And if $age_{z,n} \geq MTTF_z \times p_H$, replace this component preventively. It is assumed here that older components are replaced more frequently than younger ones. If the component is not subjected to preventative maintenance, it will continue to function until the system experiences its subsequent breakdown.

The objective function in this regard is

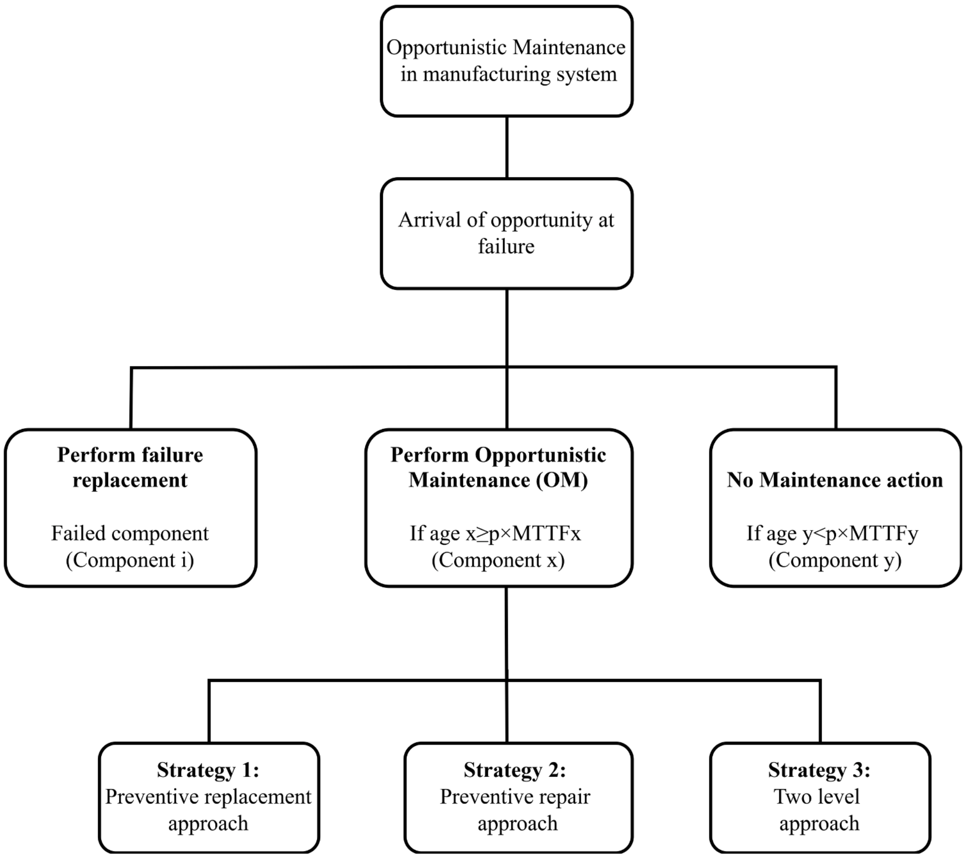

Opportunistic Maintenance in manufacturing system
Arrival of opportunity at failure
Perform failure replacement
Failed component (Component i)
Perform Opportunistic Maintenance (OM)
If age x≥p×MTTFx (Component x)
No Maintenance action
If age y<p×MTTFy (Component y)
Strategy 1: Preventive replacement approach
Strategy 2: Preventive repair approach
Strategy 3: Two level approach

Fig. 1. The idea of opportunistic maintenance.

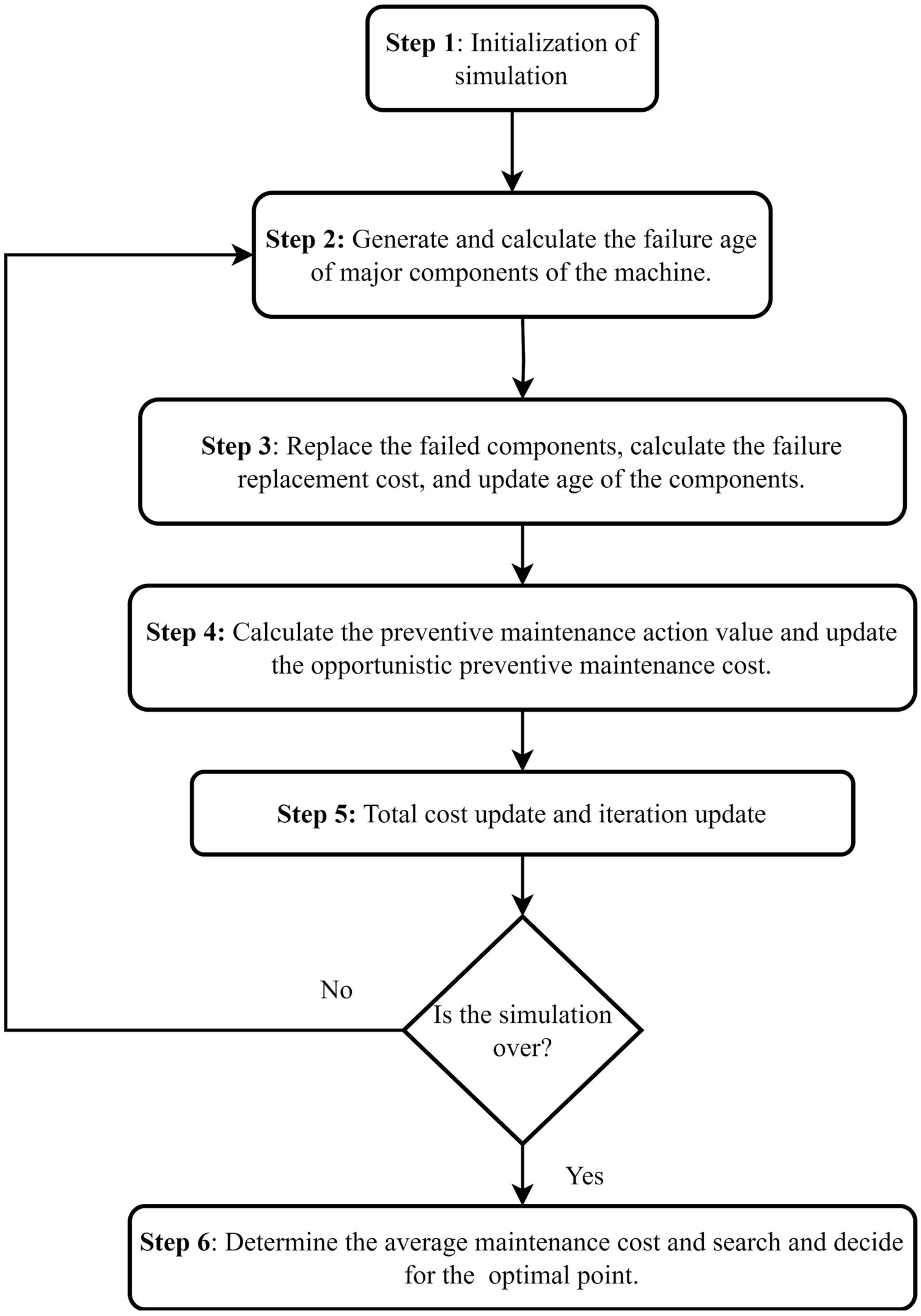

Step 1: Initialization of simulation
Step 2: Generate and calculate the failure age of major components of the machine.
Step 3: Replace the failed components, calculate the failure replacement cost, and update age of the components.
Step 4: Calculate the preventive maintenance action value and update the opportunistic preventive maintenance cost.
Step 5: Total cost update and iteration update
No
Is the simulation over?
Yes
Step 6: Determine the average maintenance cost and search and decide for the optimal point.

Fig. 2. Flow chart of the simulation process

$$MinC_A(p_L, p_H) = \frac{\left[\begin{array}{l} C_R \times IR_{z,n} + \sum_{n=1}^{N}\left(\sum_{z=1}^{Z} C_{P,Z} \times IP_z + C_{Support} \times IS_N\right) + \\ C_{PD} \times \left(MTTCA_z + \sum_{z=1}^{Z} MTTPM\right) + P_R \times R_L \times \sum_{Z=1}^{Z} MTTPM \end{array}\right]}{t_I \times N} \tag{15}$$

subject to

$$0 < p_L < p_H < 1$$

where the design parameters are p_L, p_H, correspond to component age percentage at two levels. The goal is to identify optimal age percentage to reduce the overall anticipated maintenance expense per machine per day. In case of preventive replacement and preventive repair, *MTTPRA and MTTPrA* will be used, respectively, in place of *MTTPM* in Eq. (15). The optimal value of *r,* found in strategy 2, will be used in this action to simplify the simulation process.

5. Solution methodologies

Simulation approaches have been developed to determine the average cost for all three models to evaluate the OM model. Assume that failure distributions of components are known, and it is possible to determine the ages of each component at the time of failure. The following steps have been performed for the simulation process, as shown in Fig. 2.

Step 1: Initialize the simulation. Define the total number of iterations, *I*. The total number of machines in the system (*N)* and the number of parts in the machine (*Z)* also have to be specified. Set the limits and values of the design variables *p, r,* p_L *and* p_H. Define the cost values for each component, encompassing failure replacement cost (C_R), variable replacement cost due to preventive maintenance (C_{PV}), fixed cost due to preventive maintenance (C_{PF}), predetermined labor cost rate (C_{PD}), and supportive cost for preventive maintenance in a machine in the system ($C_{Support}$). The total cost (*TC*), initially set to zero and will be revised throughout the simulation. Define the θ_z and β_z values for the Weibull distribution for each part. Determine *MTTF* from Eq. (16), stated by Ebeling, which is regulated by the Weibull distribution [34].

$$MTTF_z = \theta_z \Gamma\left(1 + \frac{1}{\beta_z}\right) \tag{16}$$

Generate component lifetimes (LF_z) by sampling the Weibull distribution parameters, θ_z and β_z. Initially, set the age values (Age_z) for all components to zero as all are new. Set initially the iteration time interval value $t_i = 0$. After i^{th} iteration, the time interval $t_i = min\ (LF_z)$. At that time, the other active component's age will also be equal to *min* (LF_z) because first failure will be occurred then.

Step 2: Calculating the failure age of the components, *FA*. At the beginning, the failure age of component *z,* FA_z is set to the value of LF_z. This means that initially, the failure age, i.e., real lifetime of the component (FA_z) is assumed to be equal to the generated lifetime (LF_z).

- For failure replacement:

If failure replacement occurs to component *z*, the new failure age of this component after replacement is updated as:

$$FA_z = LF_z \tag{17}$$

For strategy 1, preventive replacement approach:

If preventive replacement is applied to component *z*, the new failure age of this component after maintenance is updated in the same way as Eq. (17).

For strategy 2, preventive repair approach:

If preventive repair is applied to component z, the failure age of this component is updated as described in Section 4.1 as follows:

$$FA_z = r \times LF_z + (1 - r) \times FA_z \tag{18}$$

Eq. (18) is similar to Eq. (2). Here, FA_z in the right side is equal to the old lifetime of the component and component's new lifetime, LF_z will be generated at the time of preventive repair action.

For strategy 3, two-level approach:

Failure age will be updated according to Eqs. (17) and (18) depending on maintenance action. However, age restoration factor *r* is an optimal fixed value obtained in strategy 2.

Step 3: Updating the component age and iteration length, t_I. Firstly, the minimum lifetime will be searched from the list of LF_z. A component with a minimum lifetime will fail first. Then, the failure replacement cost, C_R will be calculated. The indicator value, IR_z indicates failure.

$$TC = TC + C_R \times IR_{z,n} \tag{19}$$

The minimum LF_z will be the updated age for the other components. These ages will be used to check the condition of opportunistic preventive maintenance.

$$Age_z = min(LF_z) \tag{20}$$

$$t_i = t_i + Age_z \tag{21}$$

Age after opportunistic maintenance will be updated as follows:

For strategy 1, preventive replacement approach:

As in preventive replacement, the component will be replaced, and its age after maintenance will be zero.

$$Age_z = 0 \tag{22}$$

The component that will not fulfill the condition will have no preventive maintenance and will wait for the next failure. The lifetime will be updated as follows. It is applicable to all those components that will not undergo preventive maintenance for all strategies.

$$LF = LF_{old} - Age_z \tag{23}$$

For strategy 2, preventive repair approach:

The age of the components and failure age after maintenance will be updated as described in Section 4.1, which is actually Eqs. (1) and (2), respectively.

For strategy 3, two-level approach:

Depending on maintenance action, component age will also be updated according to Eqs. (1) and (22) for preventive repair and preventive replacement approaches.

Step 4: The preventive maintenance action value, *IPz*, will be determined by checking the condition described in section 4.3. This value determines whether preventive maintenance should be performed on a machine component or not. Set IP_z to 1 if the conditions are fulfilled, indicating that preventive maintenance is required for this component. Otherwise, set IP_z to 0 (indicating no preventive maintenance is required).

After finding the preventive maintenance action value, IP_z, the cost of opportunistic preventive maintenance (C_{PM}), labor cost (C_L), and cost of lost production (C_{LP}) have to be determined as described in Section 4.2. These costs are added to the total cost (*TC*), and the overall total cost (*TC*) is found in the next step.

Step 5: Calculating total cost, TC. The total cost due to opportunistic maintenance can be found by summing failure replacement costs and all kind of maintenance costs as described in Section 4.2.5. These total costs will be calculated until Ith iteration occurs, and the updated costs will be as follows.

$$TC_I = \sum_{i=1}^{I} \left[\begin{array}{l} C_R \times IR_{z,n} + \sum_{n=1}^{N} \left(\sum_{z=1}^{Z} C_{P,Z} \times IP_z + C_{Support} \times IS_N \right) \\ + C_{PD} \times \left(MTTCA_z + \sum_{z=1}^{Z} MTTPM \right) + P_R \times R_L \times \sum_{Z=1}^{Z} MTTPM \end{array} \right] \tag{24}$$

After completing opportunistic preventative maintenance on all of the parts in each iteration, set I=i+1. Repeat steps 2 through 5 if the value of *I* is less than the highest number of iterations allowed by the simulation. In our case studies, I=100000 was used.

Step 6: Calculating the average maintenance cost, C_A. Maintenance

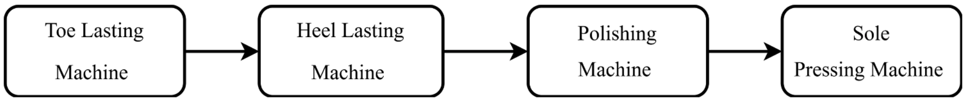


Fig. 3. Series system of machines in a FWLF in Bangladesh.

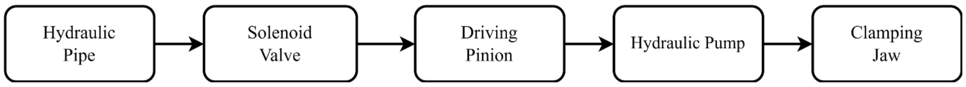


Fig. 4. Series system of components of wheel lathe in a railway industry.

cost per machine per day (C_A) is calculated as follows.

$$C_A = \frac{TC_I}{t_I \times N} \tag{25}$$

In case study I, N=4, and in case study II, N=1. In the case of study I, all similar machines are from the same brand and connected in series. In case study II, all machines are similar of the same brand and connected in parallel. Look for the optimum variables at which the related estimated maintenance cost per machine per day, $C_{A,}$ can be reduced to its lowest possible value. The optimal maintenance techniques are found when the ideal values of the variables p, r, p_L, and p_H have been discovered.

6. Case study

This section provides two different case studies to validate the proposed OM strategies for Bangladeshi manufacturing industries.

6.1. Data source

Case Study I: A Footwear and Leather Industry in Bangladesh

A renowned state-owned footwear and leather factory (FWLF) in Bangladesh has been selected for the first case study. As a developing country, Bangladesh has a huge opportunity to benefit from this sector, as raw leather is cheap here. So, if it is possible to lower the maintenance cost of this sector, it will be more profitable for the country. In this case study, a production line consisting of four distinct machines is considered, which makes a series system, as shown in Fig. 3. Only the major components of each machine are considered to simplify the simulation procedure. There are five (05) similar production lines. Similar machines are from the same brand. Maintenance costs can vary slightly depending on brand variations. But it is very negligible. Historical data from the last three years (2021–2023) was used in the case study.

Case Study II: A railway industry in Bangladesh

A state-owned railway company, which is also the largest railway industry in Bangladesh, has been selected as the second study area. The railway industry mainly manufactures and repairs railway parts. The industry's busiest C&W shop is filled with 10 similar wheel lathe machines from the same brand. Historical data from the last five years (2019–2023) was used in the case study. Maintenance costs vary very slightly for brand variations. It has a very low impact on average maintenance costs. Only five important components named hydraulic pipe, solenoid valve, driving pinion, hydraulic pump, and clamping jaw are considered for the simulation process.

In this case study, all machines are connected in parallel, and the failure of any machine will not affect the production of other machines. Thus, the value of N is one in this case. As failure of any component will stop the whole machine, the components can be considered a series system, as shown in Fig. 4.

6.1.1. Data collection of the case study

The data was collected from a footwear and leather factory (FWLF) and a railway industry in Bangladesh. The Weibull parameters of failure distribution were set based on the historical failure data of the machines with the consultancy of the industry's specialists. The data tables include information regarding the costs of the individual components. The costs include the replacement cost of the failed component (C_R), the variable cost of preventative replacement (C_{PV}), the fixed cost of preventive maintenance (C_{PF}), the predetermined cost rate for a maintenance team (C_{PD}), and the supportive cost for preventive maintenance for each component in a machine in the system ($C_{Support}$). Data on lost revenue due to downtime (R_L), production rate (P_R), and mean time of corrective and opportunistic maintenance action has also been collected to measure the cost of lost production and total labor costs.

6.1.2. Data for case study I

Data for various costs of major components of the machines of a FWLF in Bangladesh are presented in Table 2, Table 3, Table 4, Table 5.

6.1.3. Data for case study II

Data for various costs of major components of the wheel lathe in the railway industry are presented in Table 6.

6.2. Results of applying opportunistic maintenance in the manufacturing industry in Bangladesh

In this part, numerical examples are used to represent the benefit and comparison of the suggested maintenance optimization model incorporating opportunistic maintenance. A Python 3.9 code was written for the simulation procedure and was run on a Windows machine with an AMD Ryzen 7 5700U processor having a frequency of 2.20 GHz and 16GB of RAM. Optimization results are presented graphically, and comparisons among suggested maintenance optimization models are discussed.

6.2.1. Case study I

Strategy 1. With preventive replacement approach

As Fig. 5 shows, the ideal age percentage for a component in strategy

Table 2
Costs and parameters for failure distributions of major components of the Toe Lasting Machine (Tk.).

Machine Part	θ (Day)	β (Day)	C_R	C_{PV}	C_{PF}	C_{PD}	$C_{Support}$	MTTCA (hrs)	MTTPRA (hrs)	MTTPrA (hrs)	P_R (/hr)	R_L
Toe Band	120	2	8000	4000	1000	500	1000	0.5	0.5	0.3	100	150
Pincher	150	2	65000	32500	4000			0.5	0.5	0.16		
Timer	400	3	4000	2000	500			0.2	0.20	0.20		
Wiper Plate	220	3	16000	8000	800			0.5	0.5	0.33		

Table 3
Costs and parameters for failure distributions of major components of the **Heel Lasting** Machine (Tk.).

Machine Part	θ (Day)	β (Day)	C_R	C_{PV}	C_{PF}	C_{PD}	$C_{Support}$	MTTCA (hrs)	MTTPRA (hrs)	MTTPrA (hrs)	P_R (/hr)	R_L
Wiper Plate	220	2	8000	4000	1000	500	1000	0.5	0.5	0.20	100	150
Heel Band	220	2	3500	1750	500			0.25	0.25	0.33		
Relay	220	3	300	150	100			0.2	0.2	0.33		
Hydraulic Solenoid Bulb	400	3	6000	3000	1000			1	1	0.16		

Table 4
Costs and parameters for failure distributions of major components of the **Polishing** Machine (Tk.).

Machine Part	θ (Day)	β (Day)	C_R	C_{PV}	C_{PF}	C_{PD}	$C_{Support}$	MTTCA (hrs)	MTTPRA (hrs)	MTTPrA (hrs)	P_R (/hr)	R_L
Electrical Motor	220	2	16000	8000	1000	500	1000	1	1	0.5	100	150
Brush	52	3	1050	525	200			0.25	0.25	0.16		
Switch	220	2	1200	600	200			0.33	0.33	0.20		
Magnetic Conductor	220	3	2000	1000	400			0.33	0.33	0.33		

Table 5
Costs and parameters for failure distributions of major components of the **Sole Pressing** Machine (Tk.).

Machine Part	θ (Day)	β (Day)	C_R	C_{PV}	C_{PF}	C_{PD}	$C_{Support}$	MTTCA (hrs)	MTTPRA (hrs)	MTTPrA (hrs)	P_R (/hr)	R_L
Rubber Pad	66	3	1300	750	200	500	1000	0.5	0.5	0.20	100	150
Sensor	85	3	1200	600	200			0.33	0.33	0.33		
Selector Switch	130	3	650	325	100			0.5	0.5	0.33		
Air Regulator	400	2	5000	2500	500			1	1	0.2		
Limit Switch	220	2	600	300	50			0.33	0.33	0.33		

Table 6
Costs and parameters for failure distributions of major components of a wheel lathe in a railway industry in Bangladesh (Tk.).

Machine Part	θ (Day)	β (Day)	C_R	C_{PV}	C_{PF}	C_{PD}	$C_{Support}$	MTTCA (hrs)	MTTPRA (hrs)	MTTPrA (hrs)	P_R (/hr)	R_L
Hydraulic Pipe	400	2	10000	5000	1000		2000	1	1	0.5	0.625	30000
Solenoid Valve	280	3	10000	5000	1000			1	1	0.5		
Driving Pinion	950	3	10000	5000	1000	1000		2	2	1		
Hydraulic Pump	1500	2	137000	68500	5000			4	4	2		
Clamping Jaw	180	3	30000	15000	1000			1	1	0.5		

1 is 30 %, and the minimum cost is Tk. 411.4 per day per machine. So, according to this strategy, preventive replacement will be performed at 30 % and above of the age of the components. It is noted that OM will not be beneficial if the pre-mature equipment replacement cost is significantly high. But, in this study, the replacement costs of various machine parts are relatively low, which is usual for such kinds of manufacturing industries. Rather, the best strategy is the two-level maintenance approach for both of the case studies, where preventive replacements have been done from 110 % and 100 % or above of the component's age for case studies I and II, respectively, which will be discussed in the upcoming sections.

Strategy 2. With **a** preventive repair approach

As represented in Figs. 6, 7, and 8, the optimal point for maintenance entails conducting maintenance with a preventive repair approach on the part when its age becomes p=20 % of its lifetime, while the age restoration factor, r = 35 %. So, according to this strategy, preventive repair will be performed from 20 % and above of the components' lifespan or age. It is noted that opportunistic preventive repair at the early ages of the components is not impractical. For example, preventive repairs are usually done after buying a machine to increase the lifetime of the components regularly or at an opportunity.

Using a preventive repair approach, the study finds the minimum daily maintenance cost to be Tk. 376 per machine, according to these graphs.

Strategy 3. With two-level approach

Figs. 9 and 10 demonstrate that implementing the two-level approach results in a minimal cost of Tk. 366.5 per machine per day. This is the most cost-effective strategy among the three OM strategies.

Given the requirement $p_H > p_L$ in this proposed policy, it is essential to highlight that the costs associated with an area where $p_H < p_L$ in Figs. 9 and 10 are set to zero and need not require any consideration in this study. The age restoration factor of 35 % is used in this strategy as a fixed value, which was found to be the optimal value in strategy 2. The proposed two-level approach in the footwear and leather industry implements preventative repair for components aged from 20 % to 110 % of their lifespan, replacing them if they are aged from 110 % and above.

6.2.2. *Case study II*

Strategy 1. With a preventive replacement approach

Fig. 11 illustrates that the optimal age percentage for a component is 40 %, and the minimal cost is Tk. 610.1 per day per machine for preventive replacement approach. So, according to this strategy, preventive replacement will be performed from 40 % and above of the lifespan or age of the components.

Strategy 2. With a preventive repair approach

As depicted in Figs. 12, 13, and 14, optimal maintenance occurs at p=20 % of the part's lifespan with an age restoration factor of r=55 % in the preventive repair approach. So, according to this strategy, preventive repair will be performed from 20 % and above of the components' lifespan.

Using a preventive repair approach, the study finds the minimum daily maintenance cost to be Tk. 572.8 per machine as illustrated in the plots.

Strategy 3. With two-level approach

As illustrated in Figs. 15, 16 and 17 that employing the two-level approach achieves a minimal daily cost of Tk. 569.6 per machine, which is the minimum among the three strategies.

In this proposed policy, as $p_H > p_L$ is the optimization constraint, areas with $p_H < p_L$ in Figs. 15–17 are set to null. A fixed age restoration factor of 55 % from strategy 2 is used here. The two-level approach

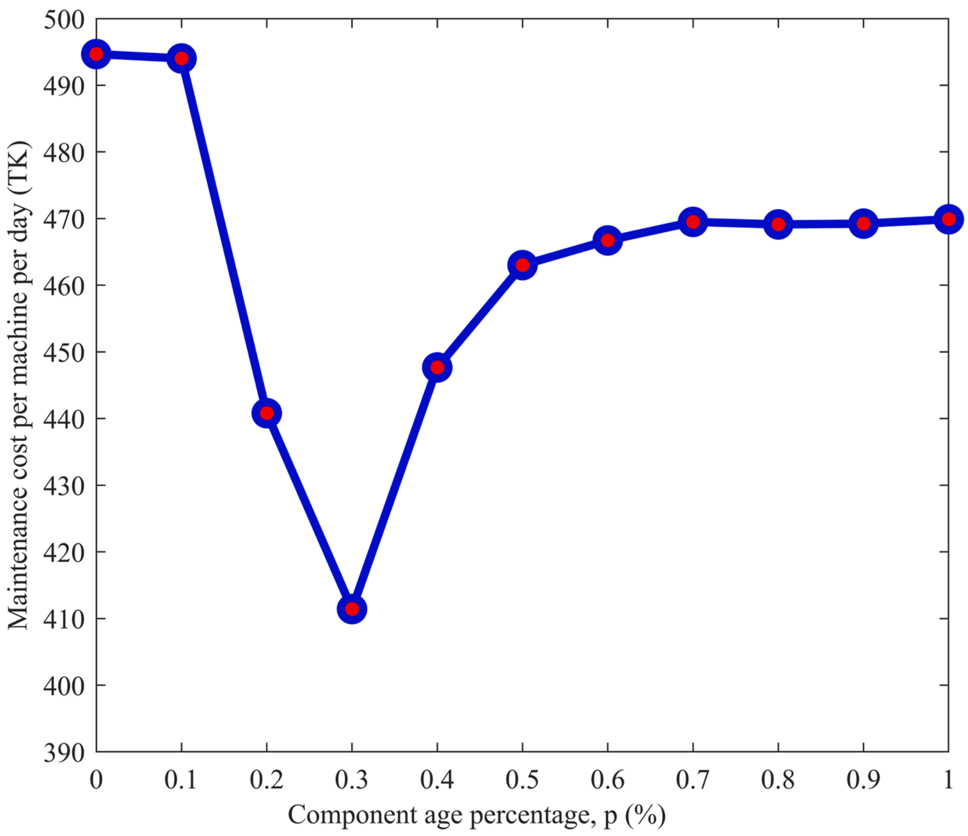


Fig. 5. Maintenance cost versus component age threshold value, p (%).

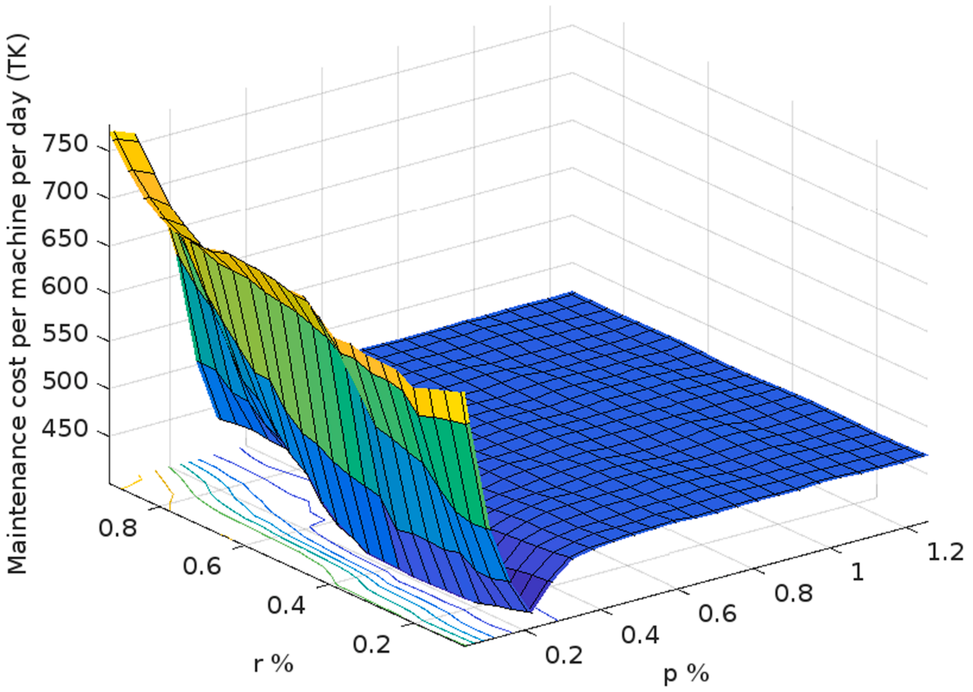


Fig. 6. Maintenance cost versus component age percentage (p) and age restoration factor (r).

employs preventive repair for components aged between 30 % and 100 % of their lifespan, with replacement from and beyond 100 %.

6.3. *Comparative study*

The proposed approaches are also examined in comparison to the currently followed corrective maintenance policy, which involves replacing a component only when it fails.

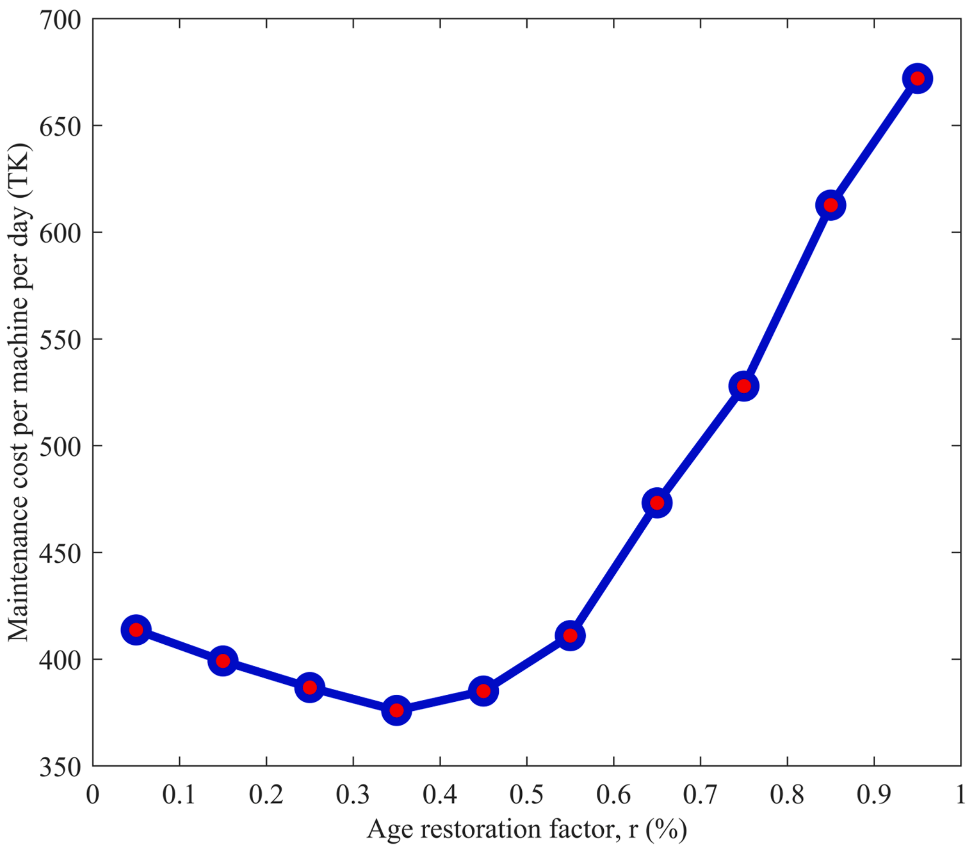


Fig. 7. Cost versus age restoration factor (r) plot at $p = 20$ %.

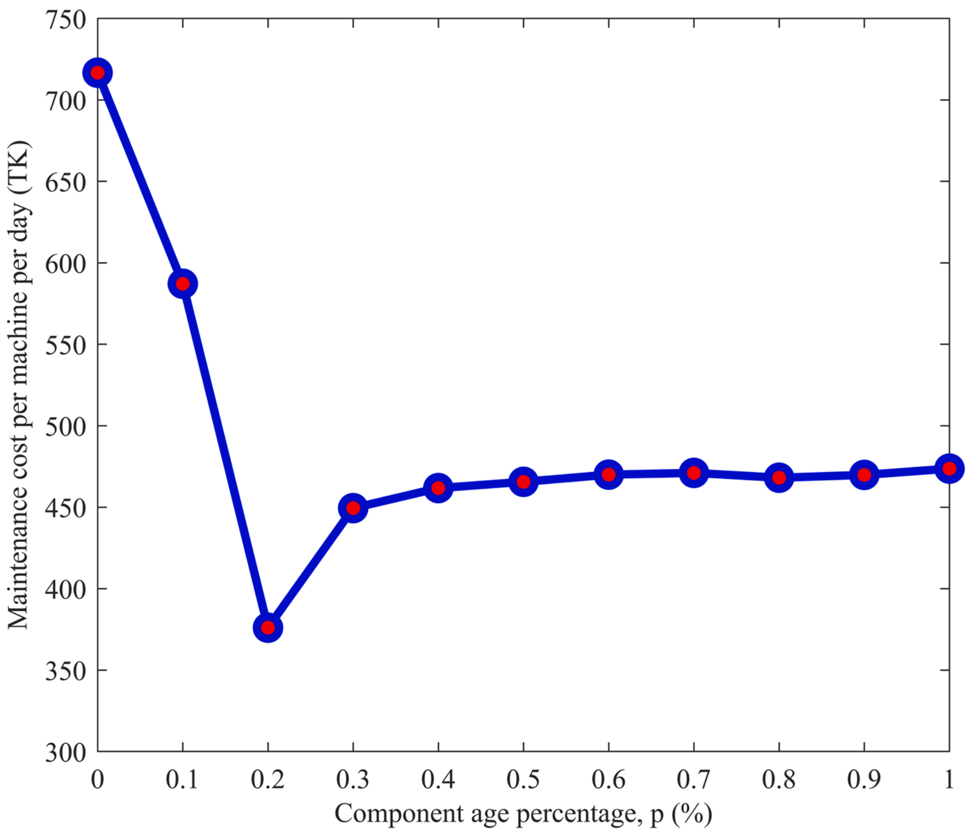


Fig. 8. Maintenance cost versus component age percentage (p) plot at $r = 35$ %.

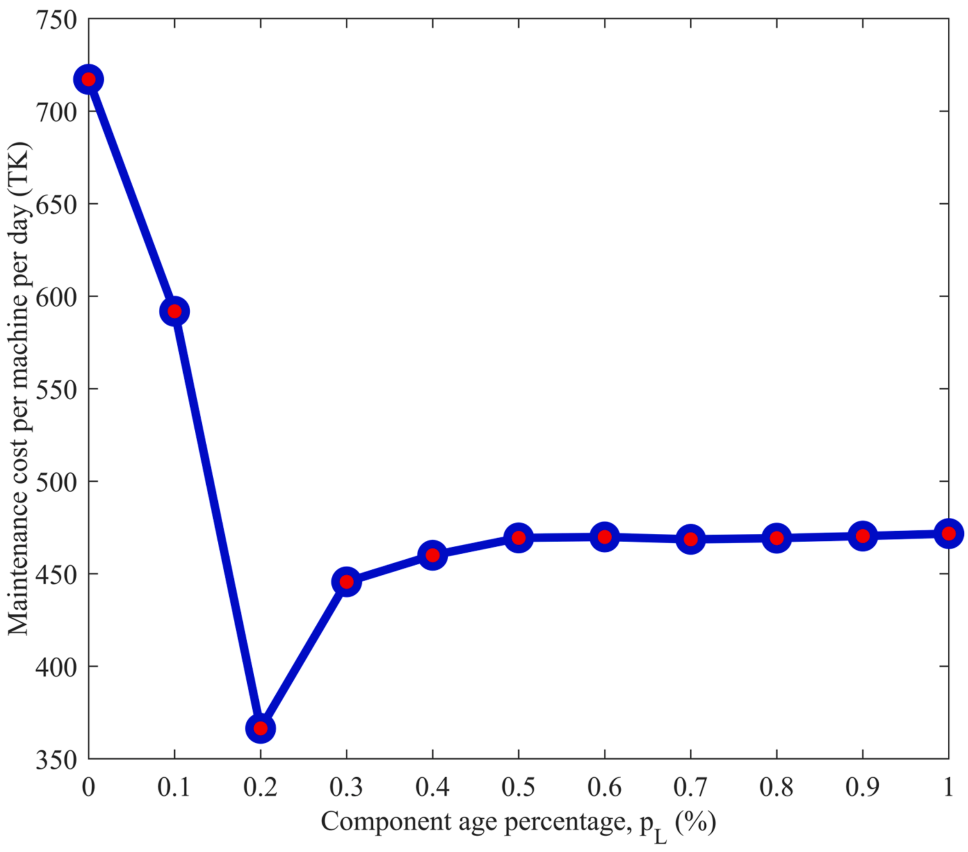


Fig. 9. Maintenance cost versus component age percentage, p_L (p_H = 110 %).

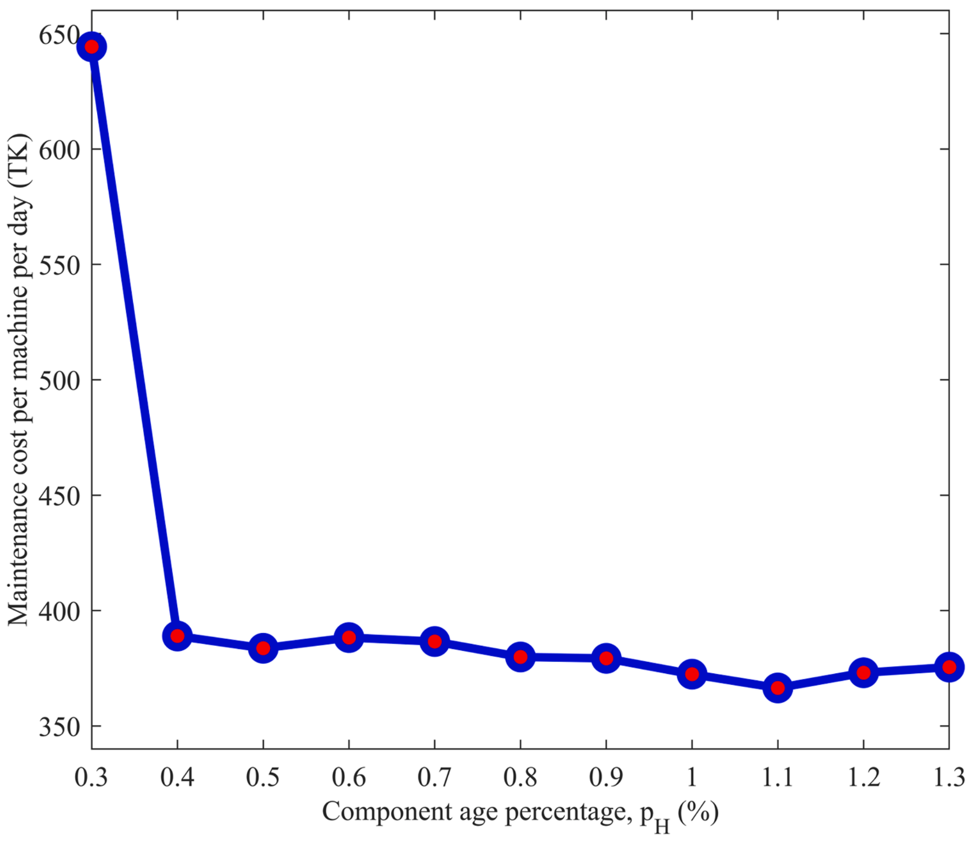


Fig. 10. Maintenance cost versus component age percentage, p_H (p_L=20 %).

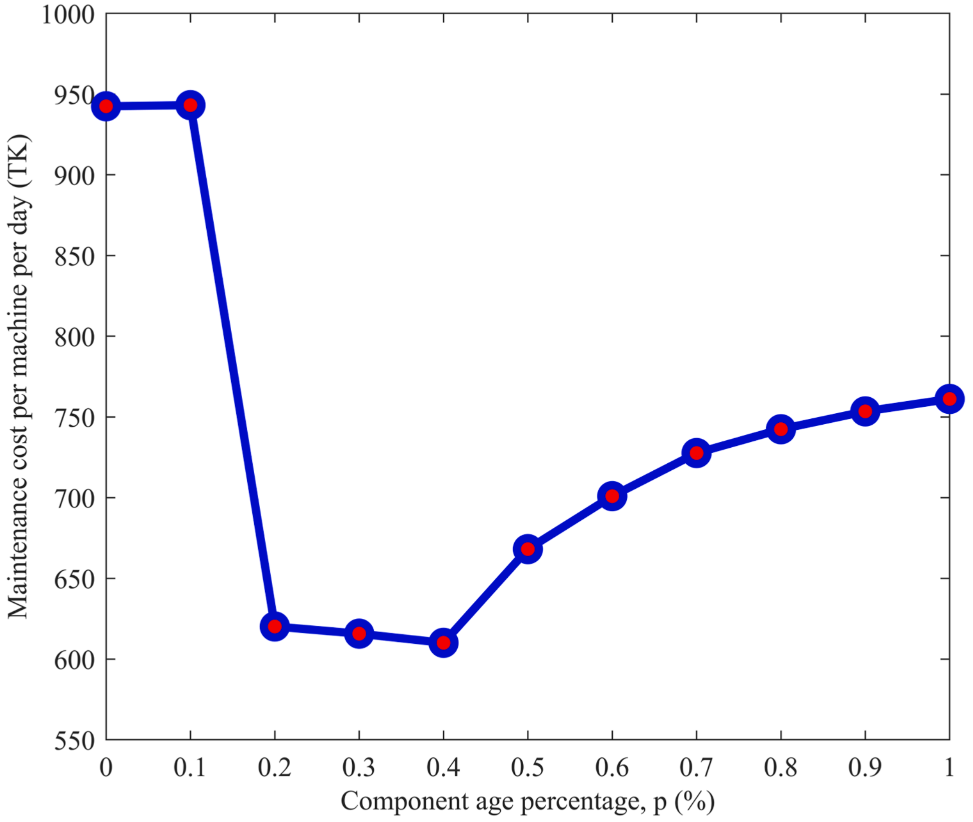


Fig. 11. Maintenance cost versus component age threshold value, p (%).

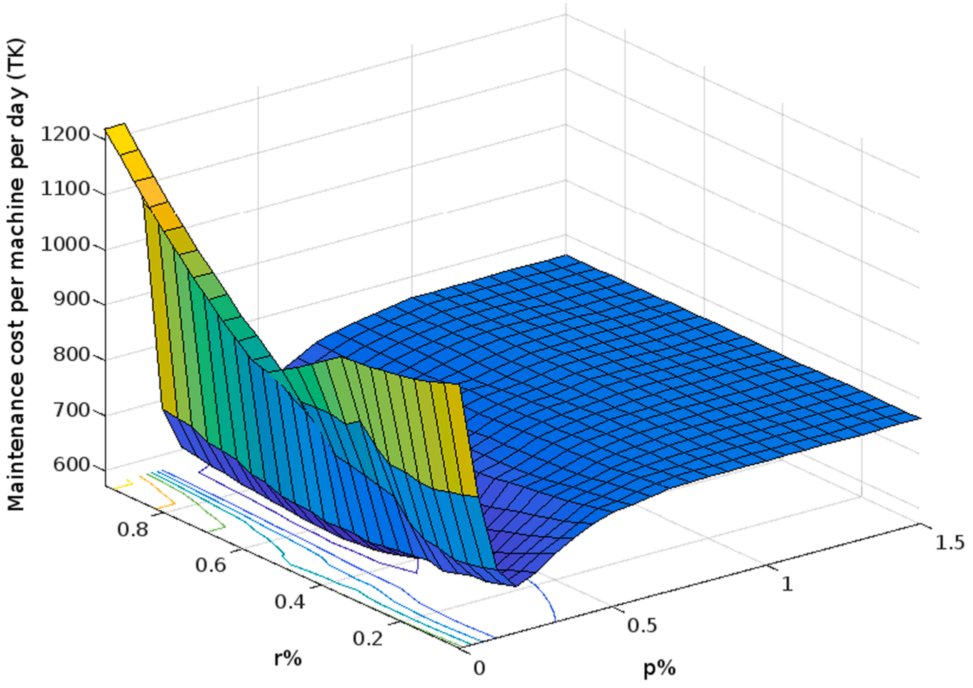


Fig. 12. Maintenance cost versus component age percentage (p) and age restoration factor (r).

6.3.1. *Cost of corrective maintenance*

The overall mean cost of implementing corrective action only for both case studies is calculated using the data from Tables 2 through 6. Employing corrective maintenance only, the maintenance cost per machine per day was found to be Tk. 440.92 and Tk. 733.97 for case studies I and II, respectively.

6.3.2. *Comparison results*

Table 7 summarizes the results and compares them with the industries' existing maintenance approaches (corrective maintenance).

In Section 6.2.1, the optimization findings demonstrate that the proposed techniques yield an ideal average cost of Tk. 411.4 per day, Tk. 376 per day, and Tk. 366.5 per day for preventive replacement,

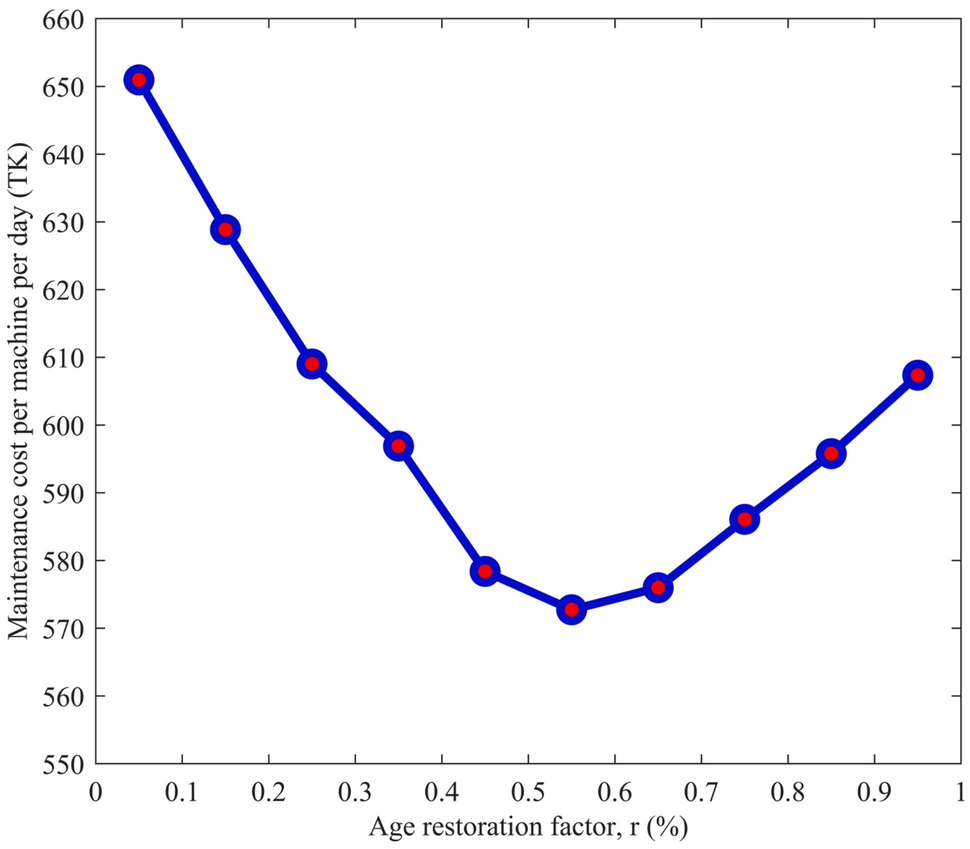


Fig. 13. Cost versus age restoration factor (r) plot at p=20 %.

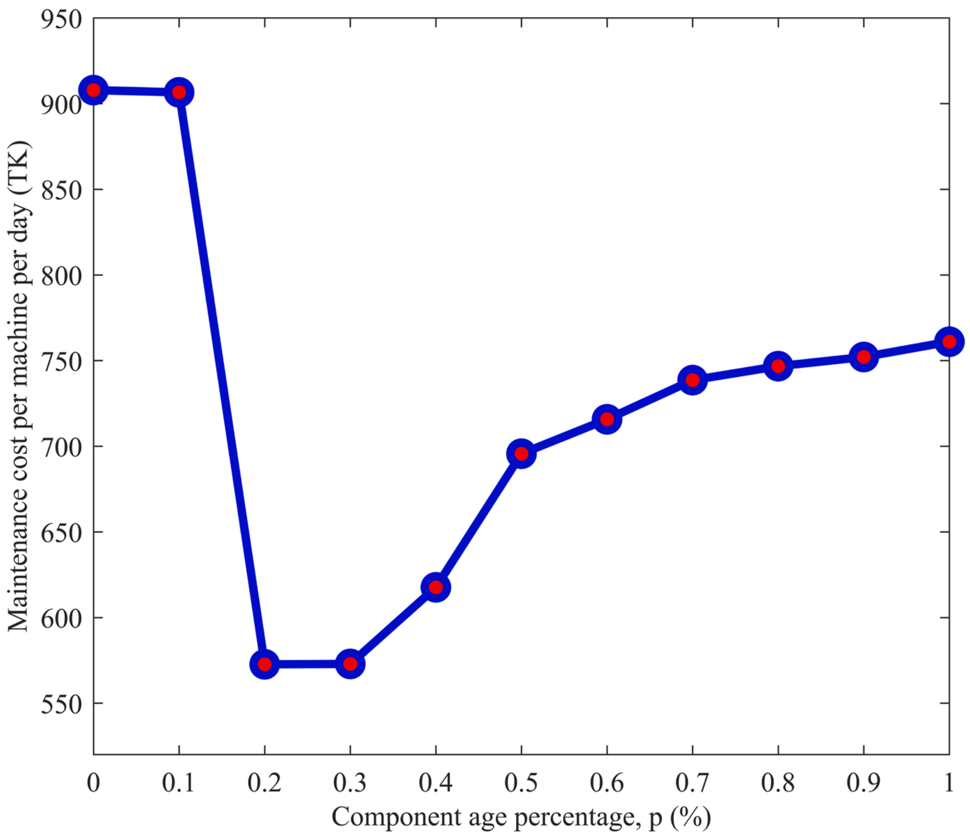


Fig. 14. Maintenance cost versus component age percentage (p) plot at r=55 %.

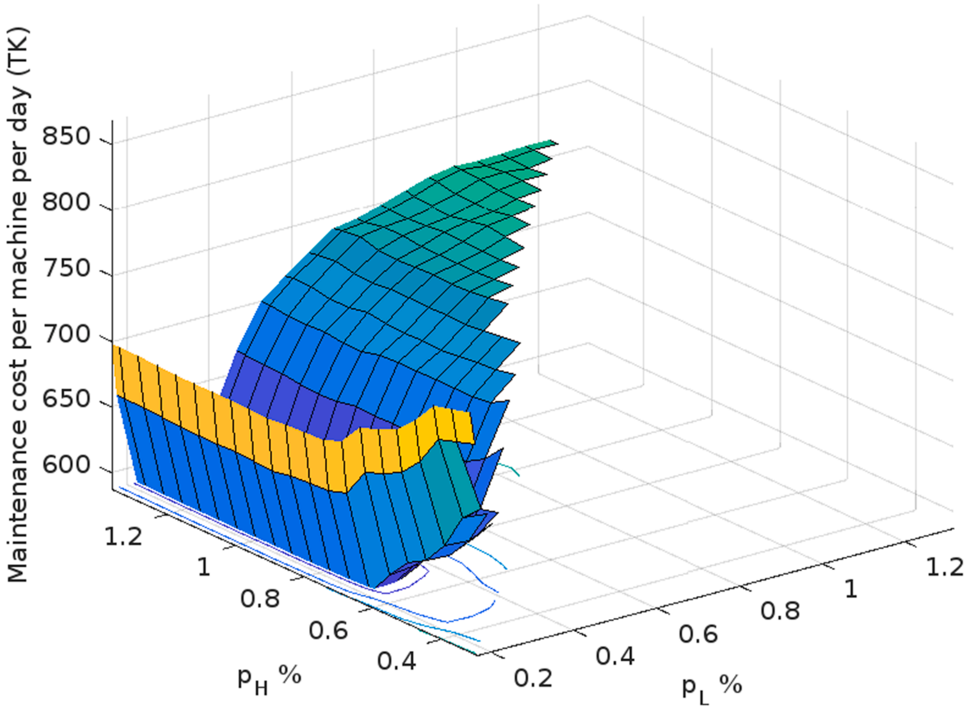


Fig. 15. Maintenance cost versus component age percentage, p_L and p_H.

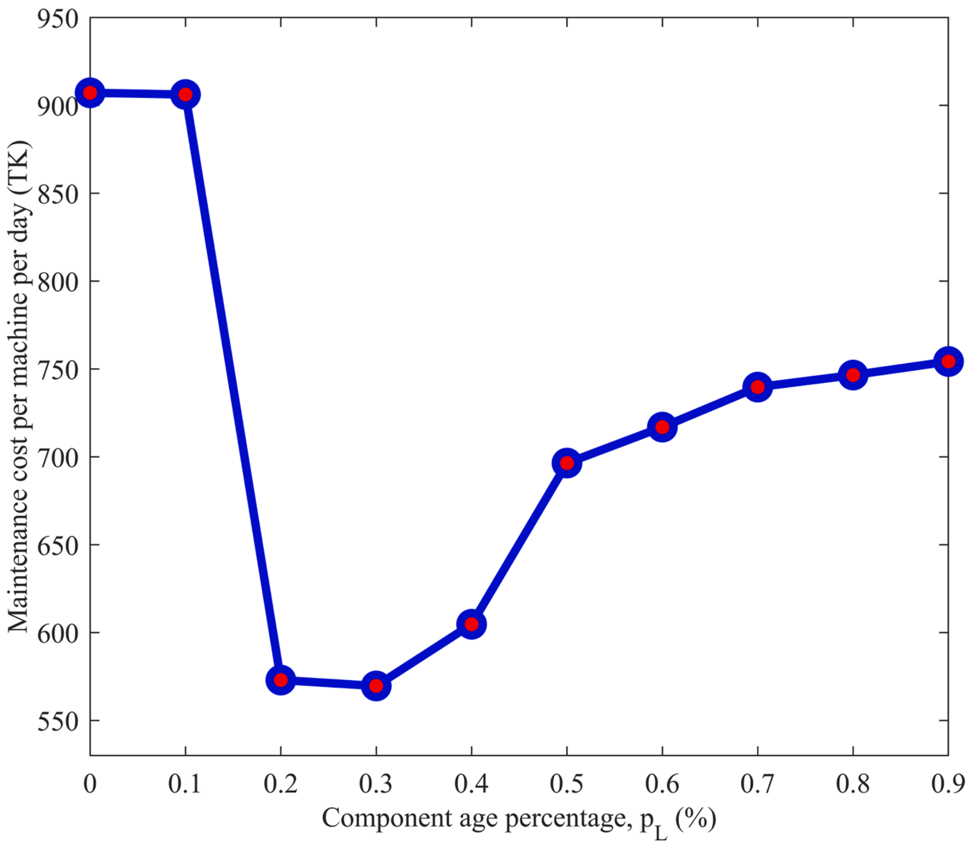


Fig. 16. Maintenance cost versus component age percentage, p_L (p_H=100 %)

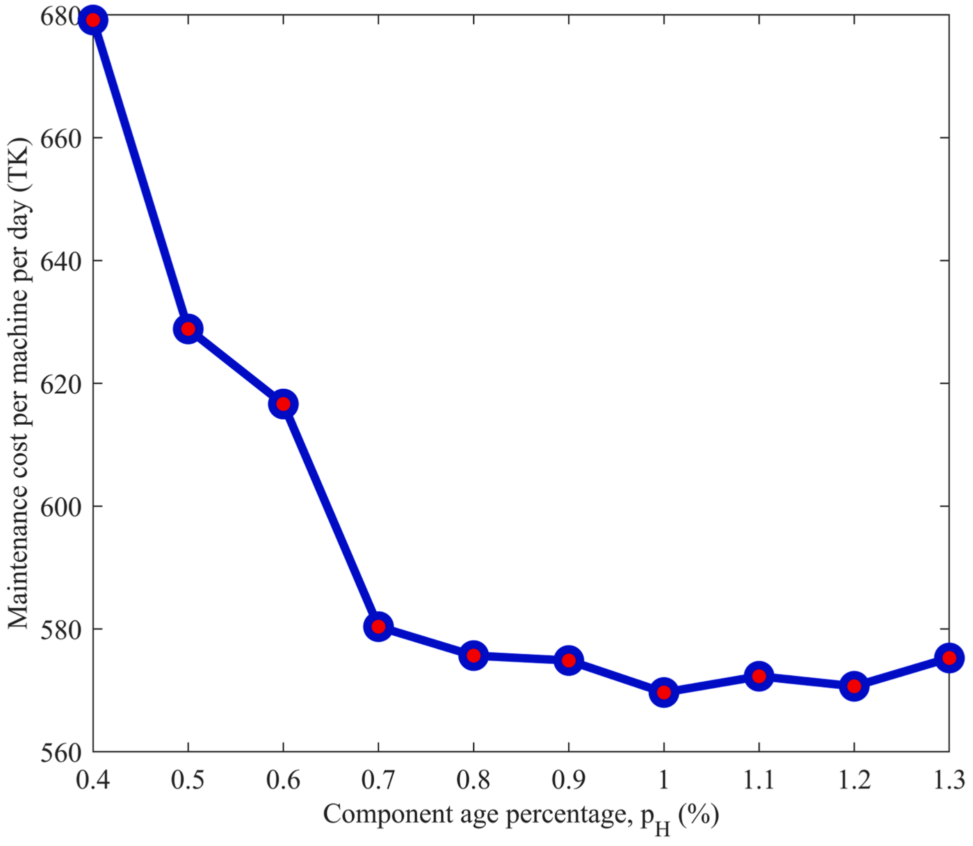


Fig. 17. Maintenance cost versus component age percentage, p_H (p_L=30 %)

Table 7
Comparison of optimal maintenance costs with corrective maintenance strategy.

Proposed opportunistic Maintenance strategy	Case study I: A FWLF in BD		Case study II: Railway industry of BD	
	Optimal cost	Cost savings	Optimal cost	Cost savings
Preventive replacement approach	Tk. 411.4	6.7 %	Tk.610.1	16.9 %
Preventive repair approach	Tk. 376	14.7 %	Tk. 572.8	21.9 %
Two level approach	Tk. 366.5	16.9 %	Tk. 569.6	22.4 %

preventive repair, and two-level approach, respectively, for case study I. The two-level approach seems the best, showing 16.9 % cost savings compared to the current corrective maintenance approach. However, the preventive repair approach, which is simpler than two-level action, also showed a good result, with 14.7 % cost savings. In contrast, the preventive replacement approach indicates only 6.7 % cost savings compared to corrective action.

However, Case Study II showed better performance than Case Study I. The optimization results in Section 6.2.2 show that the proposed strategies achieve an optimal average cost of Tk. 610.1 per day, Tk. 572.8 per day, and Tk. 569.6 per day for preventative replacement, preventive repair, and the two-level approach, respectively, in case study II. The two-level technique appears superior, demonstrating a 22.4 % reduction in costs compared to the existing corrective maintenance approach. Nevertheless, implementing preventive repair, which is less complex than two-level action, also yielded a favorable outcome with a 21.9 % cost reduction. However, compared to corrective maintenance, the preventive replacement approach demonstrates a cost savings of 16.9 %.

The reason for the better performance of Case Study II is the long lifetime of its components. The components of the leather industry experience more frequent breakdowns compared to the railway industry. Thus, opportunistic preventive maintenance reduced less costs in the leather industry than in the railway industry. In spite of that, both of the case studies showed significant cost reductions per machine per day, which will be a very large amount for the whole industry annually. Moreover, OM strategies were applied only for major components. If it is possible to implement it for all components, more cost savings will be possible.

So, the evaluation results of the model for both of the case studies show significant cost savings in compared with corrective maintenance (CM). Thus, the proposed OM strategies will be beneficial for the manufacturing industries, especially in developing countries like Bangladesh.

6.4. Sensitivity analysis considering the effect of varying C_{PV} on the average maintenance cost

This section conducts a sensitivity analysis to clearly identify the impact of the OM strategies on the average maintenance cost and to achieve more validated evaluation results for the proposed OM strategies.

- **Case Study I:**

As shown in Fig. 18, when the cost of preventive replacement (C_{PV}) is between 50 % and 75 % of C_R (cost of replacement), strategy 3, i.e., the two-level maintenance approach, is the best of all three strategies. But when the C_{PV} decreases to 25 % of C_R, strategy 1, i.e., preventive replacement, has the lowest maintenance cost. Although such a level of decreasing of C_{PV} is not possible, this is happening because preventive replacement is advantageous for low replacement costs as well as the shorter lifetime of the components, as is happening in case study I.

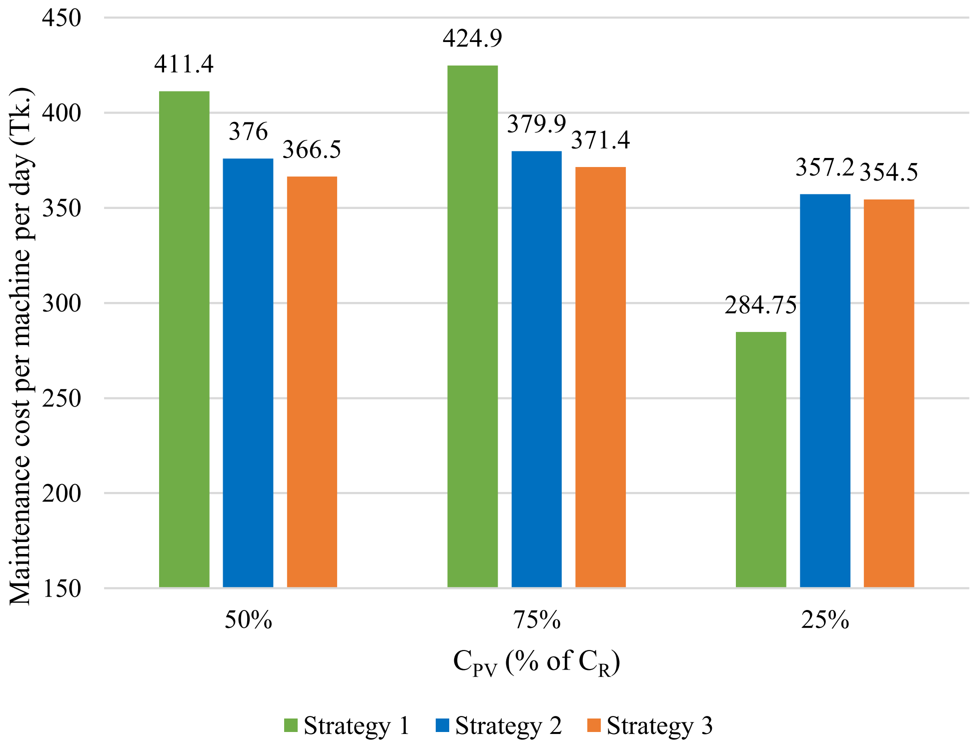


Fig. 18. Sensitivity analysis of maintenance costs for case study I with varying C_{PV}.

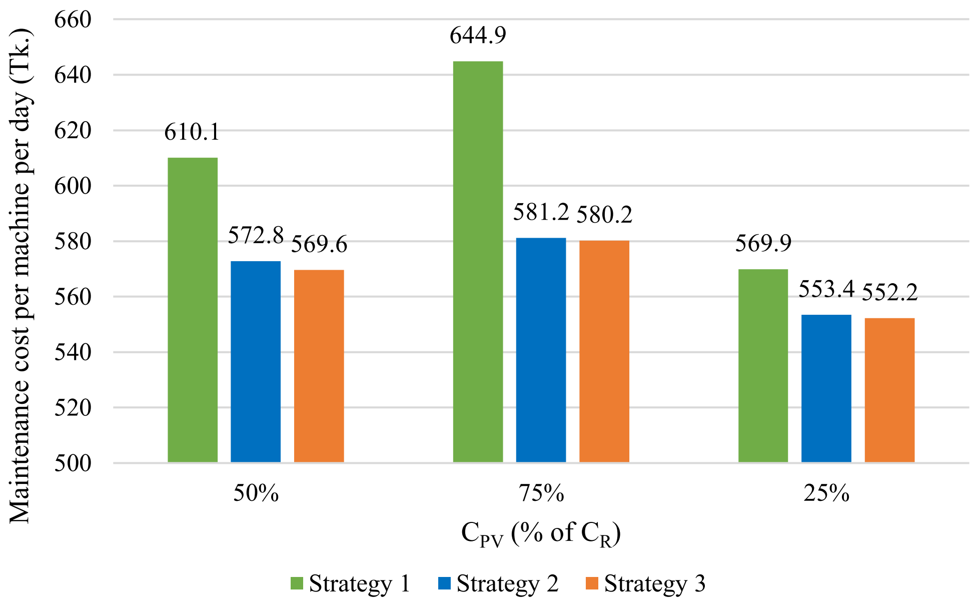


Fig. 19. Sensitivity analysis of maintenance costs for case study II with varying C_{PV}.

- **Case Stuy II:**

As shown in Fig. 19, strategy 3, the two-level maintenance approach remains the best strategy at all varying C_{PV}. Here, strategy 2, i.e., preventive repair, is very close and next better OM strategy. This is occurring due to the high replacement costs of the wheel lathes' components, which is a practical consideration.

7. Conclusions and future work

In this empirical study, the impact of opportunistic maintenance (OM) optimization has been analyzed for manufacturing-based industries in developing countries like Bangladesh. For this purpose, three OM strategies have been proposed for multi-unit manufacturing systems, where preventative maintenance is viewed as replacement, repair, and a two-level approach. The significance of the proposed OM strategies has been assessed by applying those strategies to two important industrial sectors: leather and railways in Bangladesh. The costs associated with the suggested strategies are evaluated using simulation optimization (SO) techniques. Graphical results illustrate the significance of OM strategies in lowering the average cost of maintenance. The proposed two-level maintenance approach is the best option among the three proposed OM strategies for both case studies, which show cost savings of 16.9 % and 22.4 % compared with the current CM approach for the footwear and railway industries, respectively. However, these approaches are universal and can be implemented at any opportunity due to failures or stoppages of the machines because of any other reasons. Also, the main things that matter to the industry when making maintenance decisions are the cost and availability of the system. This study presents a simulation optimization approach that will help decision makers to implement the best strategy for maintenance at the right time. It is anticipated that the leather and railway industries as well as other manufacturing industries will benefit from the proposed OM strategies.

A multi-component multi-unit manufacturing system has many important components, but to avoid complexity of the simulation process, the study only focused on the major components of each unit or machines. For future research, implementing OM strategies for almost all significant components can bring more cost savings. The research adopted a simulation optimization technique to evaluate the OM strategies. So, future research can be done via analytical techniques to assess costs and expenses more precisely. It would also be interesting to consider alternative maintenance policies with distinct reliability implications and investigate the impact of those policies on cost, availability, or any other decision parameters.

CRediT authorship contribution statement

Md. Ariful Alam: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Md. Rafiquzzaman:** Writing – review & editing, Supervision, Methodology, Investigation, Conceptualization. **Md. Hasan Ali:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis. **Gazi Faysal Jubayer:** Visualization, Validation, Software, Resources, Methodology, Formal analysis, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank the FWLF and Bangladesh Railway Authority for giving the opportunity to collect the necessary data for the research. The authors are also grateful to the Khulna University of Engineering & Technology, Bangladesh for supporting throughout the work and offering high-performance computing facilities.

References

[1] S.H.A. Rahmati, A. Ahmadi, K. Govindan, A novel integrated condition-based maintenance and stochastic flexible job shop scheduling problem: simulation-based optimization approach, Ann. Oper. Res. 269 (2018) 583–621.
[2] B. Basker, A. Manan, T. Husband, Simulating maintenance work in an engineering firm: a case study, Microelectr. Reliab. 16 (5) (1977) 571–581.
[3] L. Wang, J. Chu, W. Mao, An optimum condition-based replacement and spare provisioning policy based on Markov chains, J. Qual. Maint. Eng. 14 (4) (2008) 387–401.
[4] I. Alsyouf, The role of maintenance in improving companies' productivity and profitability, Int. J. Prod. Econ. 105 (1) (2007) 70–78.
[5] C. Zhang, F. Qi, N. Zhang, Y. Li, H. Huang, Maintenance policy optimization for multi-component systems considering dynamic importance of components, Reliab. Eng. Syst. Saf. 226 (2022) 108705.
[6] A. Syamsundar, V.N.A. Naikan, S. Wu, Estimating maintenance effectiveness of a repairable system under time-based preventive maintenance, Comput. Ind. Eng. 156 (2021) 107278.
[7] C. Stenström, P. Norrbin, A. Parida, U. Kumar, Preventive and corrective maintenance–cost comparison and cost–benefit analysis, Struct. Infrastruct. Eng. 12 (5) (2016) 603–617.
[8] H. Dui, C. Zhang, T. Tian, S. Wu, Different costs-informed component preventive maintenance with system lifetime changes, Reliab. Eng. Syst. Saf. 228 (2022) 108755.
[9] H. Ab-Samat, S. Kamaruddin, Opportunistic maintenance (OM) as a new advancement in maintenance approaches: a review, J. Qual. Maint. Eng. 20 (2) (2014) 98–121.
[10] H. Saranga, Opportunistic maintenance using genetic algorithms, J. Qual. Maint. Eng. 10 (1) (2004) 66–74.
[11] H. Abdollahzadeh, K. Atashgar, M. Abbasi, Multi-objective opportunistic maintenance optimization of a wind farm considering limited number of maintenance groups, Renew. Energy 88 (2016) 247–261.
[12] C. Zhang, W. Gao, T. Yang, S. Guo, Opportunistic maintenance strategy for wind turbines considering weather conditions and spare parts inventory management, Renew. Energy 133 (2019) 703–711.
[13] D. Wang, et al., An opportunistic maintenance strategy for wind turbines, IET Renew. Power Gener. 15 (16) (2021) 3793–3805.
[14] J. Nilsson, A. Wojciechowski, A.-B. Strömberg, M. Patriksson, L. Bertling, An evaluation approach for opportunistic maintenance optimization models for nuclear power plants, in: Proceedings of the *IEEE PES General Meeting*, IEEE, 2010, pp. 1–7.
[15] B. Xu, S. Xu, Q. Zhang, Y. Zhang, Maintenance scheduling of transmission bays considering selective opportunistic maintenance strategy, in: Proceedings of the *IEEE 3rd International Conference on Green Energy and Applications (ICGEA)*, IEEE, 2019, pp. 40–44.
[16] A. Chateauneuf, R. Laggoune, Condition based opportunistic preventive maintenance policy for utility systems with both economic and structural dependencies– application to a gas supply network, Int. J. Pressure Vessels Piping 165 (2018) 214–223.
[17] A. Bakhtiary, J.A. Zakeri, S. Mohammadzadeh, An opportunistic preventive maintenance policy for tamping scheduling of railway tracks, Int. J. Rail Transp. 9 (1) (2021) 1–22.
[18] B. Davies, J. Andrews, Railway track availability modelling with opportunistic maintenance practice, Proc. Inst. Mech. Eng. 236 (8) (2022) 907–919. *Part F: Journal of Rail and Rapid Transit*.
[19] Q. Alam, A. Rahman, S.R.U. Islam, The Economic Development of Bangladesh in the Asian Century, Routledge, US, 2021.
[20] R.K. Mobley, Maintenance Fundamentals, Elsevier, 2011.
[21] J. Moubray, Reliability-Centered Maintenance, Industrial Press Inc., 2001.
[22] C. Franciosi, A. Voisin, S. Miranda, S. Riemma, B. Iung, Measuring maintenance impacts on sustainability of manufacturing industries: from a systematic literature review to a framework proposal, J. Clean. Prod. 260 (2020) 121065.
[23] J. McMorland, M. Collu, D. McMillan, J. Carroll, A. Coraddu, Opportunistic maintenance for offshore wind: a review and proposal of future framework, Renew. Sustain. Energy Rev. 184 (2023) 113571.
[24] M. Colledani, M.C. Magnanini, T. Tolio, Impact of opportunistic maintenance on manufacturing system performance, CIRP Ann. 67 (1) (2018) 499–502.
[25] J. Wang, V. Makis, X. Zhao, Optimal condition-based and age-based opportunistic maintenance policy for a two-unit series system, Comput. Ind. Eng. 134 (2019) 1–10.
[26] P. Zhou, P. Yin, An opportunistic condition-based maintenance strategy for offshore wind farm based on predictive analytics, Renew. Sustain. Energy Rev. 109 (2019) 1–9.
[27] J. Wang, Y. Miao, Y. Yi, D. Huang, An imperfect age-based and condition-based opportunistic maintenance model for a two-unit series system, Comput. Ind. Eng. 160 (2021) 107583.
[28] M. Li, X. Jiang, R.R. Negenborn, Opportunistic maintenance for offshore wind farms with multiple-component age-based preventive dispatch, Ocean Eng. 231 (2021) 109062.
[29] C. Su, Z.-y. Hu, Y. Liu, Multi-component opportunistic maintenance optimization for wind turbines with consideration of seasonal factor, J. Cent. South. Univ. 27 (2) (2020) 490–499.
[30] L. Li, Y. Wang, K.-Y. Lin, Preventive maintenance scheduling optimization based on opportunistic production-maintenance synchronization, J. Intell. Manuf. 32 (2) (2021) 545–558.
[31] P. Patra, U.K. Dinesh Kumar, Opportunistic and delayed maintenance as strategies for sustainable maintenance practices, Int. J. Qual. Reliab. Manag. (2024) ahead-of-print.
[32] X. Li, Y. Ran, B. Chen, F. Chen, Y. Cai, G. Zhang, Opportunistic maintenance strategy optimization considering imperfect maintenance under hybrid unit-level maintenance strategy, Comput. Ind. Eng. 185 (2023) 109624.
[33] F. De Carlo, M.A. Arleo, Imperfect maintenance models, from theory to practice, Syst. Reliab. 335 (2017) 335–354.
[34] C.E. Ebeling, An Introduction to Reliability and Maintainability Engineering, Waveland Press, 2019.

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-onbenchmarks-standards-and-evaluations/

Enhanced deep learning based decision support system for kidney tumour detection

Taha ETEM [a,*], Mustafa TEKE [b]

[a] Cankkiri Karatekin University, Faculty of Engineering, Computer Engineering, Cankiri, Turkey
[b] Cankkiri Karatekin University, Faculty of Engineering, Electrical-Electronics Engineering, Cankiri, Turkey

A R T I C L E I N F O

Keywords:
Kidney cancer
Image processing
Decision support system
Deep Learning

A B S T R A C T

This study presents a high-accuracy deep learning-based decision support system for kidney cancer detection. The research utilizes a relatively large dataset of 10,000 CT images, including both healthy and tumour-detected kidney scans. After data preprocessing and optimization, various deep learning models were evaluated, with DenseNet-201 emerging as the top performer, achieving an accuracy of 99.75 %. The study compares multiple deep learning architectures, including AlexNet, EfficientNet, Darknet-53, Xception, and DenseNet-201, across different learning rates. Performance metrics such as accuracy, precision, sensitivity, F1-score, and specificity are analysed using confusion matrices. The proposed system outperforms different deep learning networks, demonstrating superior accuracy in kidney cancer detection. The improvement is attributed to effective data engineering and hyperparameter optimization of the deep learning networks. This research contributes to the field of medical image analysis by providing a robust decision support tool for early and rapid diagnosis of kidney cancer. The high accuracy and efficiency of the proposed system make it a promising aid for healthcare professionals in clinical settings.

Introduction

Kidney cancer, also known as renal cancer, is a serious and potentially life-threatening disease that affects thousands of people worldwide each year. Exploring the nature of kidney cancer, its causes, symptoms, diagnosis, treatment options, and ongoing research efforts are one of the most important research topic in nowadays [1].

Kidney cancer primarily develops in the renal cells, which line the small tubes within the kidneys. The most common type is renal cell carcinoma (RCC), accounting for about 90 % of all kidney cancers [2]. According to global statistics, kidney cancer is among the top 10 most common cancers in both men and women, with a higher incidence in developed countries. The exact cause of kidney cancer remains unknown, but several risk factors have been identified as follows [3]. The risk increases with age, with most cases diagnosed in people over 50. Tobacco uses significantly increases the risk of developing kidney cancer. Excess body weight is associated with a higher kidney risk. High blood pressure may contribute to kidney cancer development. Certain inherited genetic conditions can increase susceptibility. Long-term dialysis patients have a higher risk [4] and certain chemicals, like trichloroethylene, may increase risk [5].

Early-stage kidney cancer often presents no symptoms. As the tumour grows, potential signs may include; blood in urine (hematuria), persistent pain in the side or lower back, unexplained weight loss, fatigue, Fever not associated with an infection and Anemia [6]. Diagnosis of kidney cancer typically involves a combination of methods; physical examination, imaging tests (CT scans, MRIs, ultrasounds), blood and urine tests and if definite results cannot be obtained with all these methods, it is necessary to apply Biopsy. Recent advancements in medical imaging and the application of artificial intelligence, particularly deep learning algorithms, have significantly improved the accuracy and speed of kidney cancer detection [7]. The prognosis for kidney cancer varies greatly depending on the stage at diagnosis. Early detection significantly improves survival rates. The five-year survival rate for localized kidney cancer (confined to the kidney) is about 93 %, but this drops to about 17 % for cases where the cancer has spread to distant parts of the body [8]. Therefore, early diagnosis of kidney cancer is very important, like all other types of cancer.

In the early years, machine learning algorithms were used for prediction systems [9,10]. However, deep learning methods have been developed to make predictions directly from images [11]. Deep learning techniques have emerged as powerful tools for detecting kidney cancer

* Corresponding author.
E-mail address: tahaetem@karatekin.edu.tr (T. ETEM).

https://doi.org/10.1016/j.tbench.2024.100174
Received 7 August 2024; Received in revised form 5 September 2024; Accepted 5 September 2024
Available online 11 September 2024

in medical imaging [12]. These advanced artificial intelligence methods, particularly convolutional neural networks (CNNs), can analyse CT scans, MRI images, and ultrasounds to identify potential tumours with high accuracy [13]. By training on large datasets of labelled kidney images, deep learning models learn to recognize subtle patterns and features associated with cancerous growths. This approach offers advantageous of improved accuracy and reduced false positives, faster analysis of medical images, potential for earlier detection of small or atypical tumours and assistance to radiologists in their diagnostic workflow.

In this work, a deep learning-based kidney cancer detection system is designed to contribute faster diagnosis, higher accuracy and comparison of deep learning systems. First of all, enhanced deep learning system applied on a relatively large dataset that diagnoses very quickly after completing the training on the dataset. The proposed system is more successful than all studies in the literature and achieved classification success of over %99 with the help of data pre-processing and hyperparameter tunings. Finally, the superiority of the study is shown by comparing it with similar studies.

Related works

Kidney cancer, known as renal cell carcinoma (RCC), represents a prevalent type of cancer originating in the renal organs. Conventional diagnostic techniques, like imaging modalities and biopsies, often face constraints in terms of accuracy and effectiveness. The emergence of deep learning, a subset of artificial intelligence, has significantly transformed medical diagnostics by introducing innovative improvements in the recognition and categorization of renal cancer. This article investigates the application of deep learning in the detection of renal cancer, focusing on its methodologies, advantages, and challenges.

Deep learning systems, notably convolutional neural networks (CNNs), are heavily utilized in the analysis of medical images [14]. Because their ability to operate directly on raw image data is quite high [15]. Trained on vast assortments of annotated medical pictures, these structures can recognize patterns and abnormalities linked to renal cancer. Deep learning is utilized in the examination of computed tomography (CT) and magnetic resonance imaging (MRI) scans. Approaches for ameliorating pictures, like Contrast Limited Adaptive Histogram Equalization (CLAHE) and Contrast Stretching, boost the quality of these scans, thereby amplifying the accuracy of tumor recognition and categorization[16].

The incorporation of profound learning into the identification of renal carcinoma presents a multitude of benefits, such as enhanced accuracy. These structures have showcased efficacy levels similar to that of medical imaging specialists in identifying renal neoplasms, thereby diminishing the incidence of incorrect identifications [17]. Automated examination of medical images decreases the duration needed for diagnosis, enabling timely clinical judgments. Advanced learning advances non-destructive diagnostic methods, lessening the necessity for tissue samplings and their linked hazards. Furthermore, through identifying genetic indicators and tumor subgroups, advanced learning assists in formulating individualized treatment tactics for individuals [18]. The diagnosis of kidney cancer using deep learning faces several challenges, with one major hurdle being the need for carefully annotated datasets to train these models. Creating and annotating such datasets is both time-consuming and costly. The lack of transparency in deep learning models makes it difficult to understand how they make decisions, which complicates the process of validating and accepting them for clinical use. The training of deep learning architectures demands substantial computational capabilities and resources, which may not be readily accessible in all healthcare environments. Ensuring the generalizability of models across diverse patient cohorts and imaging protocols is crucial for broad clinical adoption [19,20].

Gujarathi et al. provides a comprehensive survey of the application of machine learning and deep learning algorithms in kidney cancer analysis. It highlights various deep learning models, particularly convolutional neural networks (CNNs), that have achieved radiologist-level performance in diagnosing kidney cancer [17].

Lu et al. employs a Deep Q-Network (DQN) to combine reinforcement learning with deep neural networks for identifying potential risk genes associated with clear cell renal cell carcinoma (ccRCC). The study highlights the integration of genetic factors in cancer diagnosis using deep learning and uses HPRD dataset [21].

Yanto et al. examines the impact of Contrast Limited Adaptive Histogram Equalization (CLAHE) on the classification of kidney tumours using CT scans. The enhancement technique significantly improves the accuracy of deep learning models in diagnosing kidney cancer and get % 99.12 accuracy [22].

Uhm et al. proposes a Lesion-Aware Cross-Phase Attention Network (LACPANet) for renal tumour subtype classification using multi-phase CT scans. The network focuses on lesion characteristics across different phases to enhance subtype classification accuracy. Authors use Seoul St Marry Hospital CT image dataset and get %94,26 accuracy [23].

Abdulwahhab et al. discusses various deep learning applications in medical imaging, with a focus on lung and skin cancer in the review article. It highlights how deep learning models, such as convolutional neural networks (CNNs), have improved diagnostic accuracy. They found significant advancements in the literature in the accurate detection and classification of lung and skin cancers [24].

Rossi et al. explored the benefits of risk-stratified screening for kidney cancer. Their research indicates that personalized screening protocols, based on individual risk assessments, can significantly enhance early detection rates. This approach may involve genetic, environmental, and lifestyle factors [25].

Yang et al. developed novel near-infrared fluorescent dyes for optical imaging of kidney cancer. These dyes specifically target cancer cells, allowing for more precise detection in both preclinical and clinical settings. This method could revolutionize the visualization of kidney tumours during surgery [26].

Tuncer and Alkan developed a decision support system for detecting renal cell cancer using machine learning algorithms. This system can analyse medical images and clinical data to assist radiologists in identifying kidney tumours more accurately. The accuracy of the work is % 92 with SVM classifier [27].

Although all the studies have achieved success in line with their objectives, there are some shortcomings. First of all, the low accuracy rate in some studies is a deficiency, especially considering that 50 % success is achieved even in the case where no learning is performed in classification processes consisting of 2 classes. Secondly, the low number of medical images reduces the reliability of the systems. Finally, in some studies, it has been observed that the use of a single network and the tuning of hyperparameters by default reduces the system performance. In order to overcome all these deficiencies, it is tried to show the stability of the system by comparing different networks with each other as well as comparing with the studies in the literature. The hyperparameters with the highest performance of the networks at different learning rates were optimised to obtain the highest accuracy rates in the literature.

Kidney medical scan classification dataset

Medical Scan Classification Dataset that includes kidney images with tumour and healthy kidney images, can be found available online at Kaggle [28]. Dataset contain 5000 medical images of healthy patients and 5000 medical images of kidney tumour detected patients. When the images with a healthy image in Fig. 1 and a tumour detected in Fig. 2 are examined, it is seen that it is very difficult for non-experts to detect this tumour.

As in the examples, other than horizontal section images out of a total of 10,000 images were first removed from the dataset. All of the images have 512 $\times$ 512 pixels resolution. After deleting the vertical

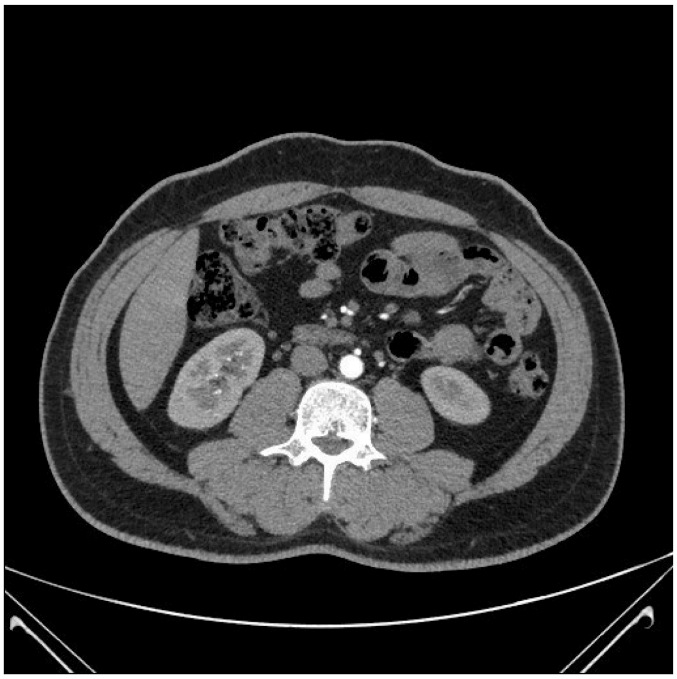


Fig. 1. Healthy Medical Image of a Patient.

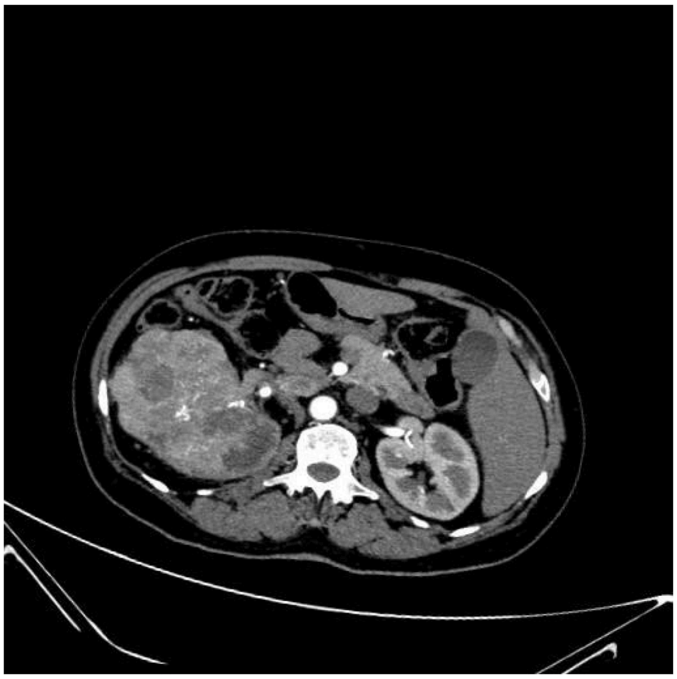


Fig. 2. Tumour Detected Medical Image of a Patient.

versions of the same images, the study was carried out with a total of 3251 healthy images and 3152 tumour images and all of the images are resized to 224 × 224 pixels for the efficiency of deep learning networks. Since vertical angle images are images of the same patients and reduce the success of the predictive system, it was deemed appropriate to perform training and testing only with horizontal section images. An example of removed vertical angle images is shown in Fig. 3.

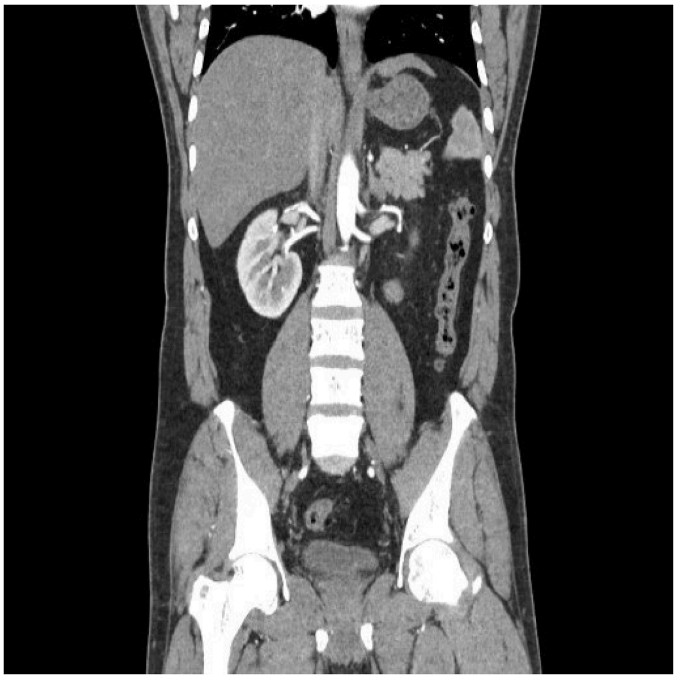


Fig. 3. Vertical Angle Image Example in the Dataset.

Materials and method

In order to establish a successful system, classification was first carried out using different deep learning structures. AlexNet, EfficientNet, Darknet-53, Xception, DenseNet-201 networks offered the best performance for the proposed system.

AlexNet is considered the model that started the deep learning revolution and holds an important place as one of the cornerstones of modern artificial intelligence research. The most important factor in this progress was the emergence of GPU designs and computational operations. AlexNet is an important model in the field of deep learning and image recognition. Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton worked to develop AlexNet in 2012, which gained significant recognition for winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition. AlexNet, with its 8 layers consisting of 5 convolutional layers and 3 fully connected layers, is widely recognized as a groundbreaking model that effectively showcased the power of deep learning. The incorporation of the ReLU activation function in AlexNet led to improved learning speed and efficiency. To prevent overfitting, AlexNet utilizes the dropout technique, which enhances the model's generalization by randomly deactivating specific neurons. Additionally, AlexNet leverages GPU parallel processing to optimize training on large datasets, resulting in enhanced speed and efficiency[29].

EfficientNet is created by Google and it presents an innovative strategy for scaling Convolutional Neural Network (CNN) architectures. The unique scaling technique of this model adjusts the width, depth, and resolution of various CNN models with great skill. EfficientNet achieves higher accuracy using fewer parameters and less computational resources. The EfficientNet series includes models ranging from B0 to B7, with B0 being the most compact and fastest, and B7 being the largest and most powerful. In our study, we used the EfficientNet-B0 model as the foundation, with the other versions being scaled adaptations of it. Compared to other popular CNN models, EfficientNet provides better accuracy and requires less computational resources, especially in evaluations using the ImageNet dataset. These advantages have made EfficientNet a preferred choice for both academic research and industrial applications. The innovative scaling methodology and remarkable performance of EfficientNet render it exceptionally effective for a diverse array of deep learning assignments [30].

The Darknet-53 is a key CNN structure used in the YOLOv3 algorithm, developed by Joseph Redmon and Ali Farhadi. Known for its outstanding performance in object recognition tasks, the Darknet-53 consists of 53 layers, enabling it to capture more complex features and achieve higher accuracy. To build a more profound network, Darknet-53 incorporates Residual Blocks inspired by the ResNet architecture. These blocks effectively address the issue of vanishing gradients in deep networks by using bypass connections, making it easier to train deeper models. In comparison to its predecessor, Darknet-19, Darknet-53 offers improved accuracy and increased Frames Per Second (FPS) efficiency.. Consequently, YOLOv3, which is constructed on Darknet-53, excels in executing prompt and effective object recognition [31].

Xception, short for "Extreme Inception," is a system designed to improve the structure of Convolutional Neural Networks (CNNs) in deep learning. Created by François Chollet and introduced in 2017, Xception builds upon Google's Inception framework and introduces significant improvements. It employs depthwise separable convolutions to enhance the efficiency of the Inception framework, reducing computational costs while improving model accuracy. Xception represents a revolutionary approach to enhancing both efficiency and accuracy in deep learning frameworks, combining the strengths of the Inception framework with the benefits of depthwise separable convolutions to create faster and lighter models [32].

DenseNet-201 stands out as a version of the Dense Convolutional Network (DenseNet) structure, specifically distinguished by its 201 layers. The team of Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger introduced DenseNet-201 in 2016, with a focus on improving information flow and enhancing gradient propagation through closely connected layers. This design ensures that each layer receives input from all previous layers, promoting weight reuse and improved information flow. The primary use of 3×3 filters in DenseNet-201 helps to reduce computational costs. Additionally, bottleneck layers in the form of 1×1 convolutional layers are integrated to further streamline computations and minimize parameter usage. Transition layers, consisting of a 1×1 convolutional layer and a 2×2 average pooling layer, are incorporated to connect dense blocks and manage dimensions. The output of each layer in DenseNet is impacted by a specific growth rate, ensuring efficient and effective network performance. This dictates how many channels each layer's output contributes. In DenseNet-201, the growth rate is typically set as 32. Dense connections among layers enable gradients to propagate more smoothly, thus mitigating the vanishing gradient issue. Due to dense connections, superior performance can be attained with fewer parameters. This enables weights to be reused and renders the model more concise. DenseNet excels in various computer vision tasks, such as image categorization and object recognition, providing high accuracy and efficiency. [33]. DenseNet-201, specifically, is preferred in scenarios that require accurate and effective deep learning architectures. It attained the highest accuracy level in the research.

A crucial element in network performance is the acquisition speed, a hyperparameter that governs the degree to which weights are modified during the instruction of a machine learning model. The speed of acquiring information specifically controls how much the weights change at each step of the instructions. Setting an appropriate acquisition speed allows the model to learn quickly and effectively, while an incorrect speed can slow down the learning process. Improvement algorithms like gradient descent adjust the model's weights based on the loss function's derivative, and the acquisition speed determines the degree of these adjustments. This highlights the acquisition speed's importance as a critical parameter in training deep learning models. A too small acquisition speed can slow down the training, while a too large speed can cause instability. Applying different optimization techniques to fine-tune the acquisition speed leads to a more efficient and effective training process [34].

Network performance metrics

A confusion matrix is a table used to assess the performance of classification algorithms by comparing the predicted class labels to the real class labels. It is particularly useful for dataset with multiple classes. The confusion matrix is comprised of four key components:

1. True Positives (TP): The number of instances correctly classified as positive.
2. True Negatives (TN): The number of instances correctly classified as negative.
3. False Positives (FP): The number of instances incorrectly classified as positive (Type I error).
4. False Negatives (FN): The number of instances incorrectly classified as negative (Type II error).

An Example of the Confusion Matrix is shown in Table 1.

To evaluate the confusion matrix, it is necessary to calculate various performance metrics. The metrics can be summarized as follows:

1. Accuracy: Proportion of samples that the model predicted correctly.
2. Precision: It shows how many of the positive predictions are actually positive.
3. Recall (Sensitivity or True Positive Rate): It shows how many of the true positives were predicted correctly.
4. F1-Score: It is the harmonic mean of Precision and Recall. It is a balanced performance measure.
5. Specificity (True Negative Rate): It shows how many of the true negatives were predicted correctly.

The use of different metrics is important in decision support systems such as cancer prediction in the study. For example, if healthcare professionals are required to examine medical images more carefully, especially those containing tumours, the Recall parameter used here will show how high the tumour detection rate is.

Proposed network designs

Flow chart of the proposed prediction system is shown in Fig. 4. Firstly, Kidney images were extracted from the dataset. Then, all medical images were resized to the same size and images with errors and low resolutions were deleted. Since there are both horizontal and vertical angle images of the same patient in the dataset and since it is difficult to detect most kidney tumours in vertical angle images, these images were excluded. It was also tested with deep learning systems that it is difficult to detect tumours from vertical angled images, and it was determined that vertical angled images decreased the success in the training of all networks.

After completing all the operations on the dataset, a balanced dataset containing 6404 images with approximately equal number of tumour and healthy images was obtained. The obtained dataset was randomly divided as 75 % training and 25 % test. Then, the most successful 5 different networks are shown in the study results according to the validation results by training on many deep learning networks. Also, hyperparameter tuning plays a crucial role in optimizing deep learning networks. The choice of optimizer and loss function significantly impacts model convergence and performance. Learning rate, a key hyperparameter, influences the speed and stability of training; too high a rate may cause overshooting, while too low a rate can lead to slow convergence. Batch size affects both training speed and generalization, with larger batches potentially offering more stable gradients but at the cost of memory. The number of epochs determines how many times the

Table 1
An example of confusion matrix.

	Real Positives	Real Negatives
Predicted Positives	TP	FP
Predicted Negatives	FN	TN

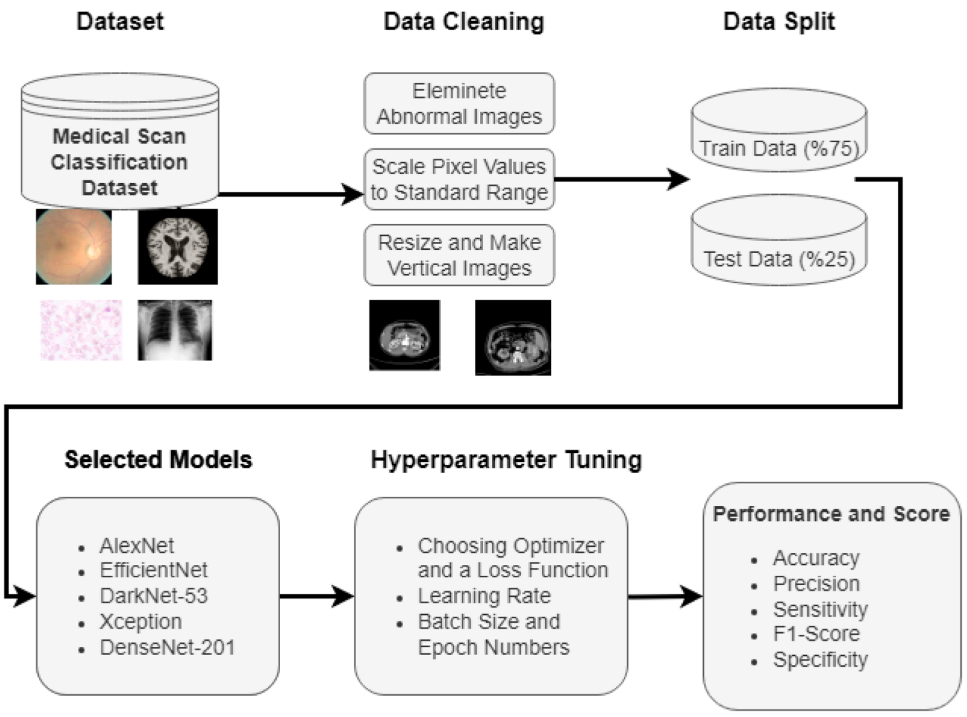


Fig. 4. Proposed System Flow Chart.

model sees the entire dataset, balancing between underfitting and overfitting. Careful tuning of these hyperparameters can dramatically improve a model's accuracy, efficiency, and generalization capabilities.

Results

The performance metrics of the network models used in the study against the learning rate are given in the Table 2.

As seen in the Table 2 above, the DenseNet-201 network gave the highest performance. When the learning rate, one of the most important parameters affecting the network performance, was changed, different model Xception network gave higher results. While the effect of the learning rate on Alexnet is 0.001, it shows 50 % performance; When training is performed with a rate of 0.0001, the performance has increased up to 98 %. Confusion matrices were used to obtain metrics. The confusion matrices of the selected deep learning networks are given below in Fig. 5, 6, 7, 8, 9, 10 and 11.

According to all these analyses, it was determined that the best performance was obtained when the DenseNet-201 learning rate was shown as 0.0001. Apart from DenseNet-201 network, Xception LR= 0.001 and EfficientNet LR= 0.0001 networks can also be used in decision support mechanisms by making improvements. When the confusion matrices of these networks are examined, it is very important for decision support systems that the margin of error usually occurs in healthy individuals while detecting tumour images very accurately.

Benchmarking

The comparison table of the proposed method and other kidney cancer detection systems are shown in Table 3.

In the context of kidney cancer detection, deep learning techniques have shown varying degrees of success across different studies [41]. On public datasets, researchers like Türk et al. [40] employed a hybrid V-Net-based model, yielding an accuracy of 97.7 % with 210 CT images, while Ma et al. [39] proposed a Heterogeneous Modified Artificial Neural Network (HMANN) and achieved an accuracy of 97.5 % with 400 CT scans.

In comparison, our proposed method utilizing the DenseNet-201 architecture significantly outperformed these models, achieving an accuracy of 99.75 % on a much larger public dataset consisting of 10,000 CT images. This improvement is attributed to the extensive dataset size, effective preprocessing steps, and hyperparameter optimization, making it one of the most reliable systems for kidney tumor detection to date. The results suggest that our model not only generalizes well to unseen data but also sets a new benchmark in terms of accuracy, making it a strong candidate for integration into clinical decision support systems.

Table 2
Score of the deployed models.

Deep Learning Model	Learning Rate	Accuracy	Precision	Sensitivity	F1-Score	Specificity
AlexNet	0.001	0.5078	0.25	0.50	0.50	.050
EfficientNet-b0	0.001	0.9550	0.9581	0.9557	0.9550	0.9557
DarkNet-53	0.001	0.7820	0.8465	0.7854	0.7727	0.7854
DenseNet-201	0.001	0.9750	0.9758	0.9754	0.9750	0.9754
Xception	**0.001**	**0.9950**	**0.9950**	**0.9951**	**0.9950**	**0.9951**
AlexNet	0.0001	0.9813	0.9813	0.9814	0.9813	0.9814
EfficientNet-b0	0.0001	0.9582	0.9608	0.9588	0.9581	0.9608
DarkNet-53	0.0001	0.9663	0.9675	0.9667	0.9663	0.9667
DenseNet-201	**0.0001**	**0.9975**	**0.9975**	**0.9975**	**0.9975**	**0.9975**
Xception	0.0001	0.9794	0.9799	0.9797	0.9794	0.9797

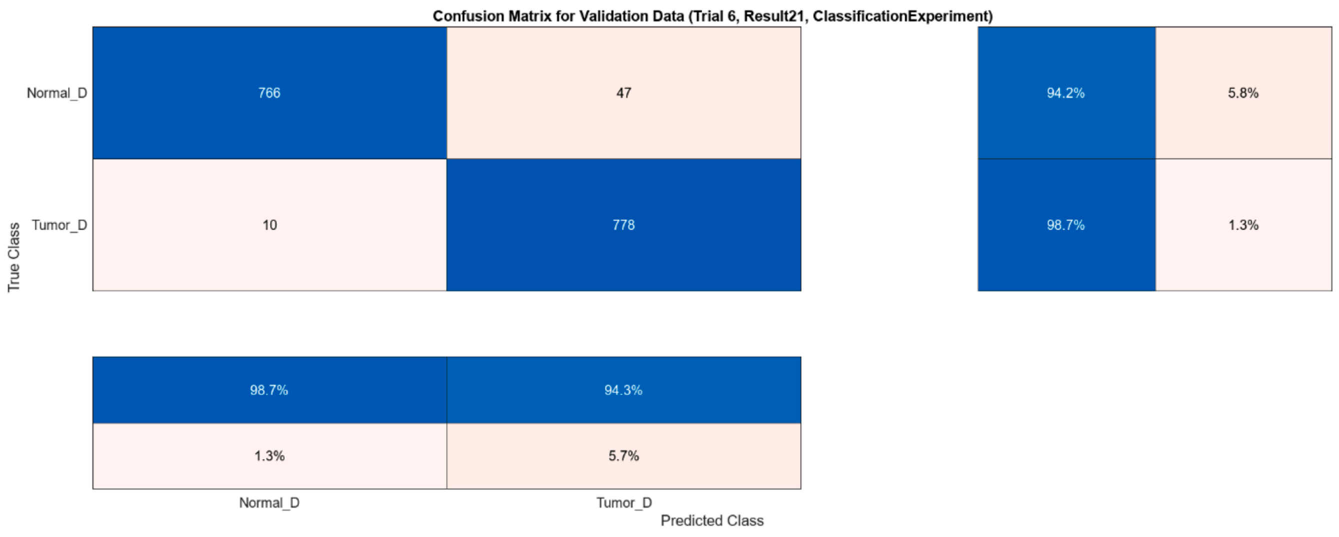


Fig. 5. Confusion matrice of AlexNet LR: 0,0001.

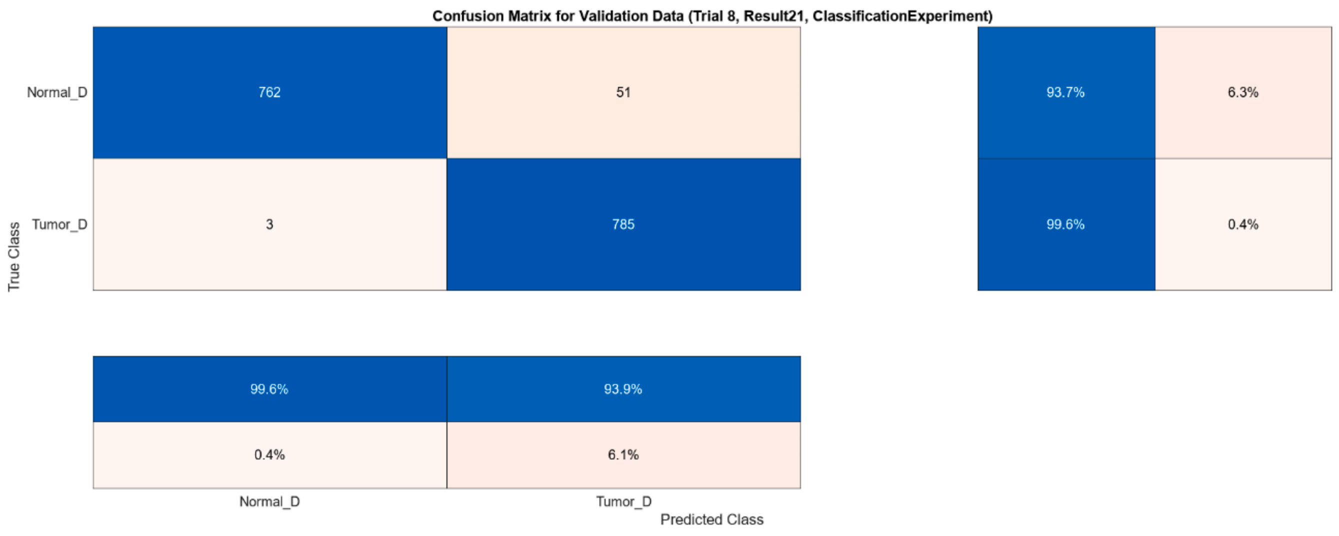


Fig. 6. Confusion matrice of DarkNet-53 LR: 0,0001.

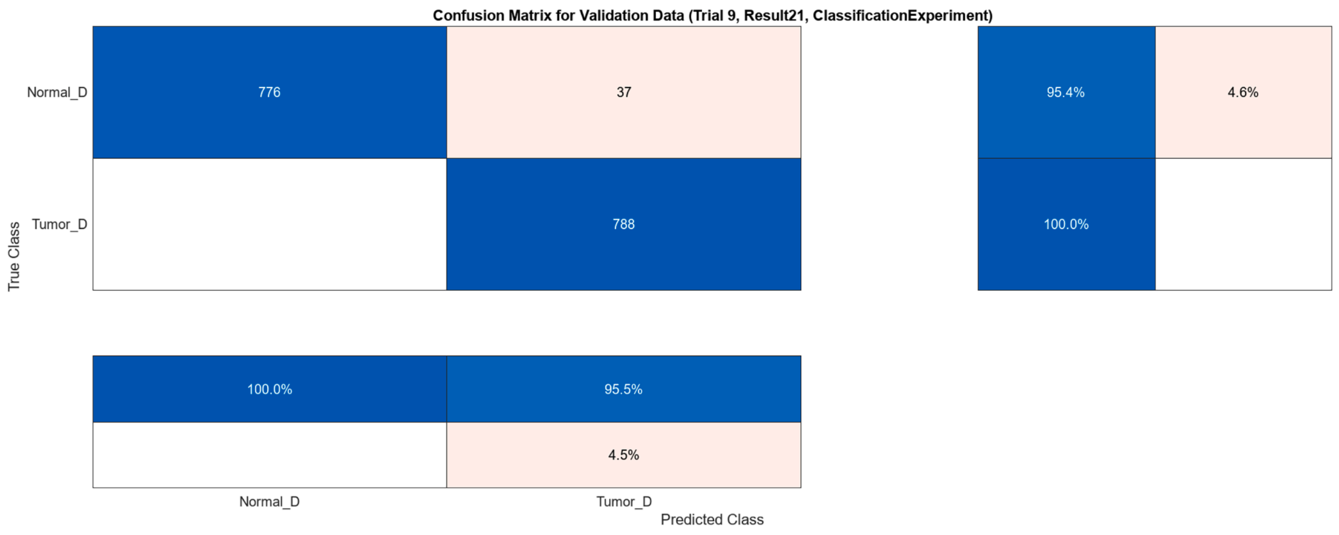


Fig. 7. Confusion matrice of DenseNet-201 LR: 0,0001.

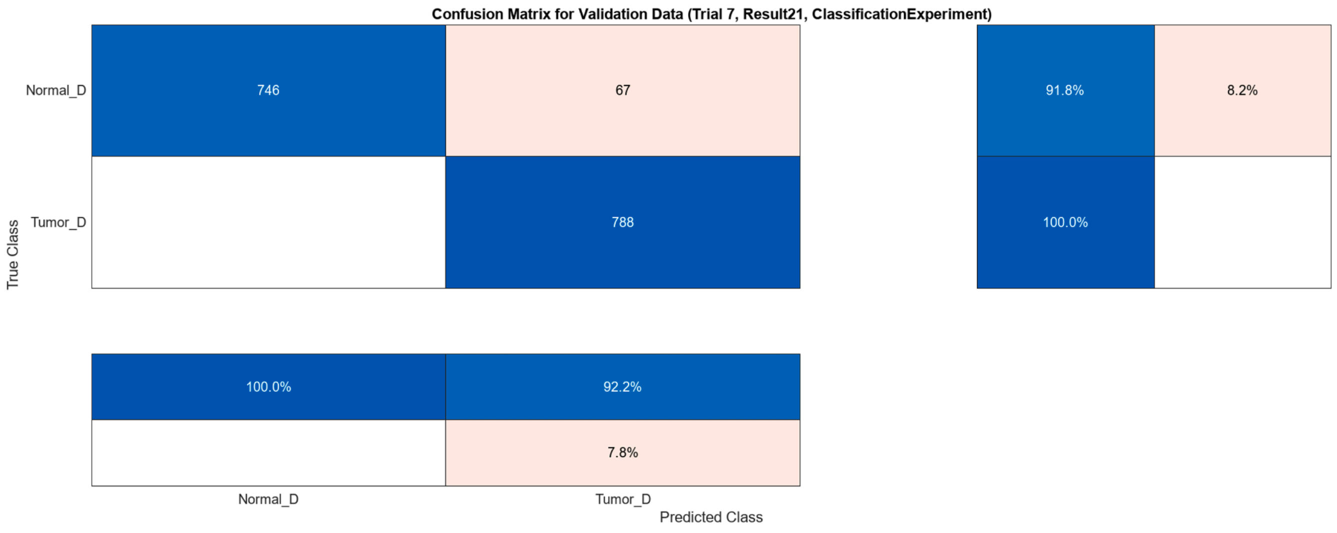


Fig. 8. Confusion matrice of EfficientNet LR: 0,0001.

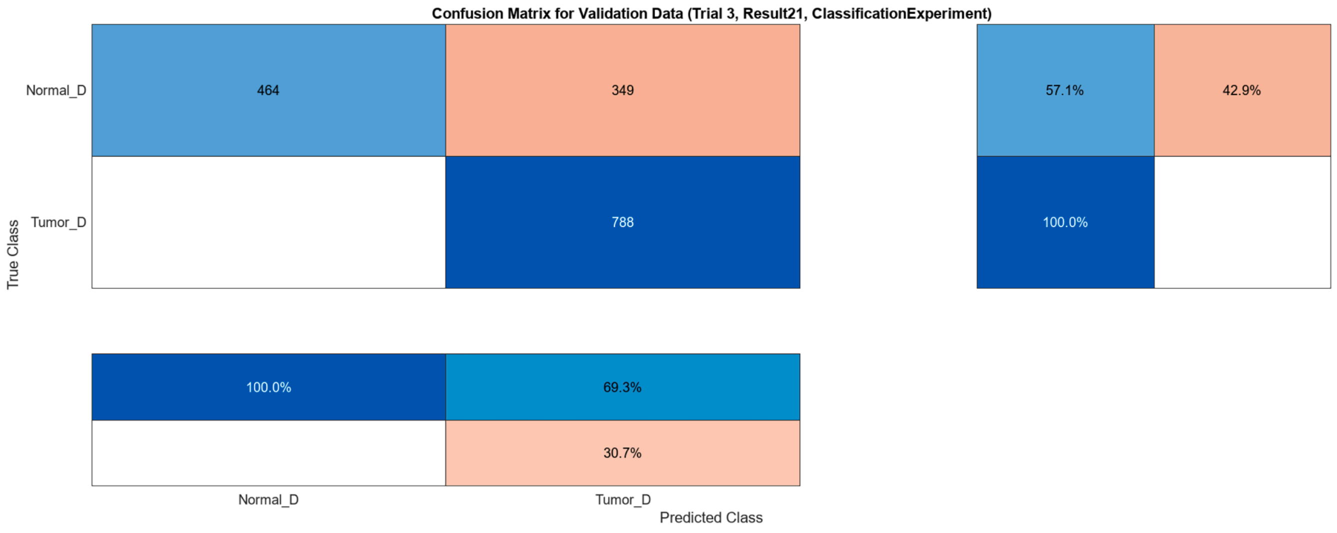


Fig. 9. Confusion matrice of DarkNet-53 LR: 0,001.

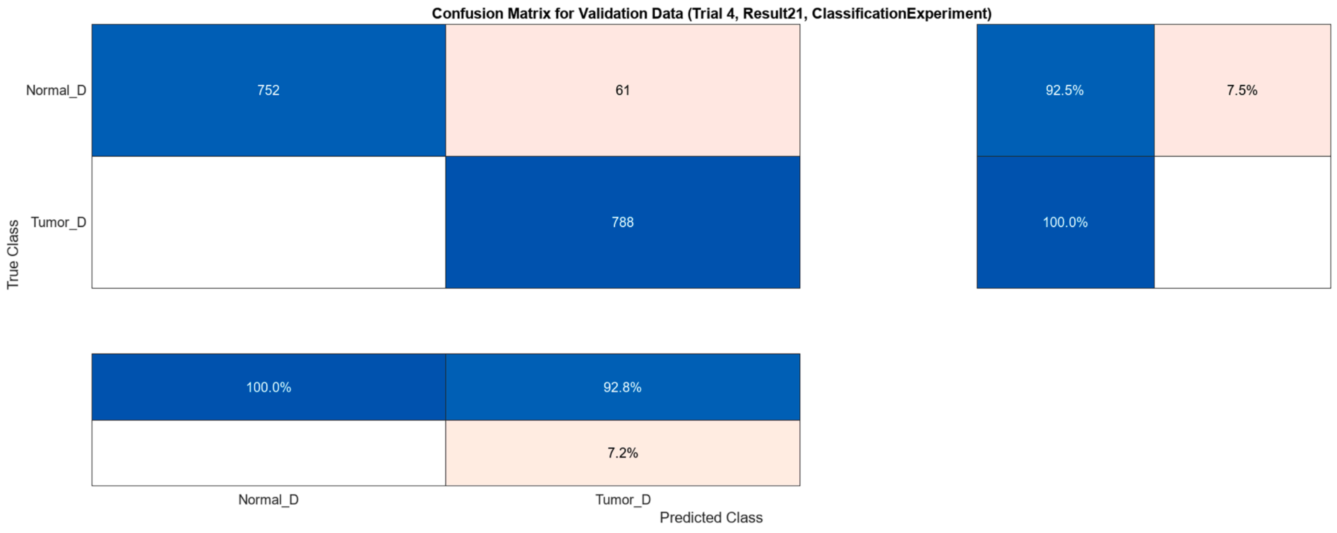


Fig. 10. Confusion matrice of DenseNet-201 LR: 0,001.

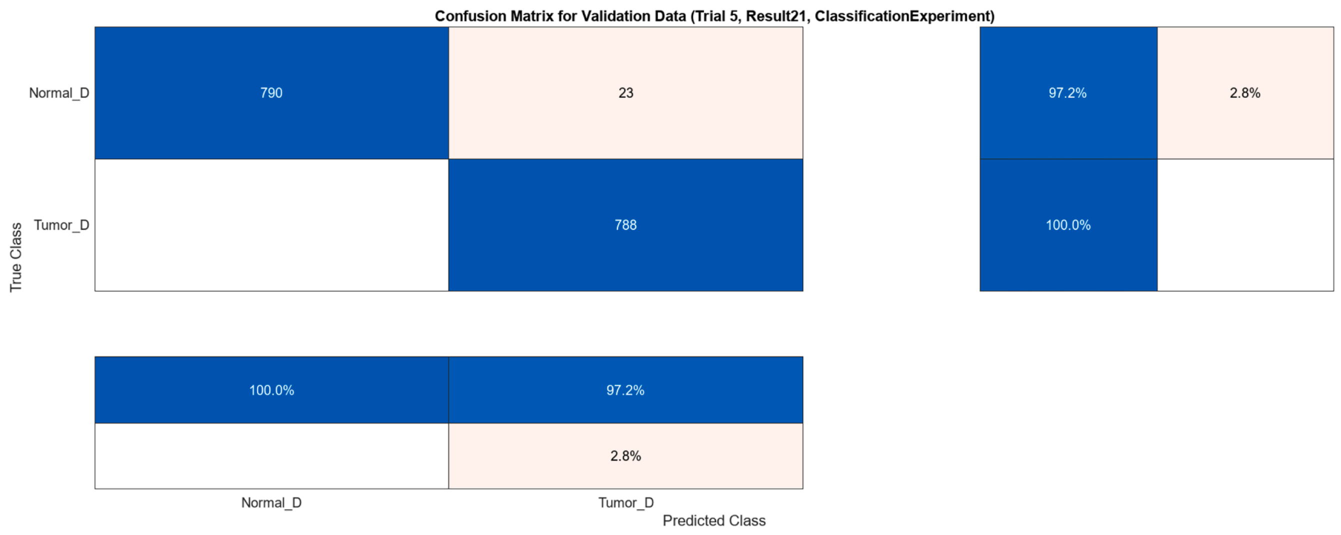


Fig. 11. Confusion matrice of Xception LR: 0,001.

Table 3
Scores of the deployed models.

Ref.	Model	Dataset	Data Size	Accuracy
[23]	LACPANet	Private	183 CT	0.9426
[27]	SVM	Private	100 CT	0.9200
[35]	YOLO	Private	300 CT	0.9320
[36]	2D CNN	Public	8400 CT	0.9200
[37]	AdaBoost, RF	Public	735 CT	0.7500
[38]	XGBoost	Public	177 CT	0.7700
[39]	HMANN	Public	400 CT	0.9750
[40]	V-Net	Public	210 CT	0.9770
This Work	DenseNet-201	Public	10,000 CT	0.9975

Conclusions

The importance of early and rapid diagnosis of serious diseases such as cancer is increasing day by day. At this point, although artificial intelligence systems are not yet in a position to take over completely, they offer great support to users even as a decision support system. In this study, a deep learning-based prediction system has been developed for kidney cancer. As can be seen in Table 3, the highest success was obtained in the comparison of the studies. The improvement of the accuracy was achieved as a result of the data engineering done on the dataset and by optimising the hyper-parameters of the deep learning networks.

For future studies, it is planned to realise a system in which deep learning networks supported by larger datasets make fully automatic decisions with high success. Also, feature extraction methods can be applied to achieve the best results for the predictive system.

Consent for publication

Not Applicable.

CRediT authorship contribution statement

Taha ETEM: Writing – review & editing, Writing – original draft, Visualization, Methodology, Formal analysis. **Mustafa TEKE:** Validation, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethics approval and consent to participate

Not Applicable.

Acknowledgements

Not Applicable.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability

The data analyzed in this study can be downloaded from the following sites: https://www.kaggle.com/datasets/arjunbasandrai/medical-scan-classification-dataset.

References

[1] H.D. Patel, et al., Survival after diagnosis of localized t1a kidney cancer: current population-based practice of surgery and nonsurgical management, Urology. 83 (1) (Jan. 2014) 126–133, https://doi.org/10.1016/J.UROLOGY.2013.08.088.
[2] A.A.W. Goller, W. Djatisoesanto, Pediatric with localized renal cell carcinoma (RCC), Radiol. Case Rep. 19 (7) (Jul. 2024) 2886–2890, https://doi.org/10.1016/J.RADCR.2024.03.022.
[3] R. De Angelis, et al., Complete cancer prevalence in Europe in 2020 by disease duration and country (EUROCARE-6): a population-based study, Lancet Oncol. 25 (3) (Mar. 2024) 293–307, https://doi.org/10.1016/S1470-2045(23)00646-0.
[4] M.Y. Lin, et al., Association of dialysis with the risks of cancers, PLoS. One 10 (4) (Apr. 2015), https://doi.org/10.1371/JOURNAL.PONE.0122856.
[5] A.S. Andrew, M. Li, X. Shi, J.R. Rees, K.M. Craver, J.M. Petali, Kidney cancer risk associated with historic groundwater trichloroethylene contamination, Int. J. Environ. Res. Public Health 19 (2) (Jan. 2022), https://doi.org/10.3390/IJERPH19020618.
[6] B. Mukherjee, et al., Recent developments in cancer vaccines: where are we? Nanotherapeut. Cancer Vaccinat. Chall. (Jan. 2022) 29–75, https://doi.org/10.1016/B978-0-12-823686-4.00018-5.
[7] H. Zhang, M. Botler, J.P. Kooman, Deep learning for image analysis in kidney care, Adv. Kidney Dis. Health 30 (1) (Jan. 2023) 25–32, https://doi.org/10.1053/J.AKDH.2022.11.003.
[8] "Survival Rates for Kidney Cancer | American Cancer Society." Accessed: Jul. 12, 2024. [Online]. Available: https://www.cancer.org/cancer/types/kidney-cancer/detection-diagnosis-staging/survival-rates.html.
[9] S. Buyrukoğlu, New hybrid data mining model for prediction of Salmonella presence in agricultural waters based on ensemble feature selection and machine learning algorithms, J. Food Saf. 41 (4) (Aug. 2021) e12903, https://doi.org/10.1111/JFS.12903.
[10] F. Türk, Investigation of machine learning algorithms on heart disease through dominant feature detection and feature selection, Signal. Image Video Process. 18

(4) (Jun. 2024) 3943–3955, https://doi.org/10.1007/S11760-024-03060-0/TABLES/6.

[11] G. Buyrukoğlu, S. Buyrukoğlu, Z. Topalcengiz, Comparing regression models with count data to artificial neural network and ensemble models for prediction of generic escherichia coli population in agricultural ponds based on weather station measurements, Microb. Risk. Anal. 19 (Dec. 2021), https://doi.org/10.1016/J.MRAN.2021.100171.

[12] J. Nyman, et al., Spatially aware deep learning reveals tumor heterogeneity patterns that encode distinct kidney cancer states, Cell Rep. Med. 4 (9) (Sep. 2023) 101189, https://doi.org/10.1016/J.XCRM.2023.101189.

[13] M. Mousavi, S. Hosseini, A deep convolutional neural network approach using medical image classification, BMC. Med. Inform. Decis. Mak. 24 (1) (Aug. 2024) 239, https://doi.org/10.1186/s12911-024-02646-5.

[14] S. Savaş, Enhancing disease classification with deep learning: a two-stage optimization approach for monkeypox and similar skin lesion diseases, J. Imag. Informat. Med. 37 (2) (Jan. 2024) 778–800, https://doi.org/10.1007/S10278-023-00941-7. *2024 37:2*.

[15] A. Alhudhaif, B. Almaslukh, A.O. Aseeri, O. Guler, K. Polat, A novel nonlinear automated multi-class skin lesion detection system using soft-attention based convolutional neural networks, Chaos. Solitons. Fractals. 170 (May 2023) 113409, https://doi.org/10.1016/J.CHAOS.2023.113409.

[16] F. Yanto, M.I. Hatta, I. Afrianty, L. Afriyanti, Pengaruh image enhancement contrast stretching dalam klasifikasi ct-scan tumor ginjal menggunakan deep learning, INOVTEK Polbeng - Seri Informatika 9 (1) (Jun. 2024), https://doi.org/10.35314/ISI.V9I1.4233.

[17] S.M. Gujarathi, R. Shrivastava, S. Kumar Angadi, A survey of kidney cancer analysis using machine learning and deep learning algorithms, J. Electric. Syst. 20 (6) (2024) 2491–2501. Accessed: Jul. 12, 2024. [Online]. Available: https://www.researchgate.net/publication/381887478.

[18] K. Yan, S. Fong, T. Li, Q. Song, Multimodal machine learning for prognosis and survival prediction in renal cell carcinoma patients: a two-stage framework with model fusion and interpretability analysis, Appl. Sci. 14 (2024) 5686, https://doi.org/10.3390/APP14135686. Pagevol. 14, no. 13, p. 5686, Jun. 2024.

[19] N.Goel Deepali, Padmavati Khandnor, Advances in AI-based genomic data analysis for cancer survival prediction, Multimed. Tools. Appl. (Jun. 2024) 1–28, https://doi.org/10.1007/S11042-024-19684-W/FIGURES/1.

[20] Z. Zheng, Q.H. Soomro, D.M. Charytan, Deep learning using electrocardiograms in patients on maintenance dialysis, Adv. Kidney Dis. Health 30 (1) (Jan. 2023) 61–68, https://doi.org/10.1053/J.AKDH.2022.11.009.

[21] D. Lu, et al., Identifying potential risk genes for clear cell renal cell carcinoma with deep reinforcement learning, bioRxiv. (Jun. 2024), https://doi.org/10.1101/2024.06.19.599667, p. 2024.06.19.599667.

[22] N. Jannata, F. Yanto, L. Handayani, E. Pandu, C. Kurnia, Pengaruh contrast limited adaptive histogram Equlization dalam Klasifikasi CT-scan tumor ginjal menggunakan deep learning, INOVTEK Polbeng - Seri Informatika 9 (1) (Jun. 2024) 2024, https://doi.org/10.35314/ISI.V9I1.4235.

[23] K.H. Uhm, S.W. Jung, S.H. Hong, S.J. Ko, Lesion-aware cross-phase attention network for renal tumor subtype classification on multi-phase CT scans, Comput. Biol. Med. 178 (Aug. 2024) 108746, https://doi.org/10.1016/J.COMPBIOMED.2024.108746.

[24] A.H. Abdulwahhab, et al., A review on medical image applications based on deep learning techniques, Article J. Image Graphics (2024), https://doi.org/10.18178/joig.12.3.215-227.

[25] S.H. Rossi, H. Harrison, J.A. Usher-Smith, G.D. Stewart, Risk-stratified screening for the early detection of kidney cancer, The Surgeon 22 (1) (Feb. 2024) e69–e78, https://doi.org/10.1016/J.SURGE.2023.10.010.

[26] X. Yang, et al., Optical imaging of kidney cancer with novel near infrared heptamethine carbocyanine fluorescent dyes, J. Urol. 189 (2) (Feb. 2013) 702–710, https://doi.org/10.1016/J.JURO.2012.09.056.

[27] S.A. Tuncer, A. Alkan, A decision support system for detection of the renal cell cancer in the kidney, Measurement 123 (Jul. 2018) 298–303, https://doi.org/10.1016/J.MEASUREMENT.2018.04.002.

[28] "Medical scan classification dataset." Accessed: Jul. 12, 2024. [Online]. Available: https://www.kaggle.com/datasets/arjunbasandrai/medical-scan-classification-dataset.

[29] A. Priya, V. Vasudevan, Brain tumor classification and detection via hybrid alexnet-gru based on deep learning, Biomed. Signal. Process. Control 89 (Mar. 2024) 105716, https://doi.org/10.1016/J.BSPC.2023.105716.

[30] S. Nigam, R. Jain, V.K. Singh, S. Marwaha, A. Arora, S. Jain, EfficientNet architecture and attention mechanism-based wheat disease identification model, Procedia Comput. Sci. 235 (Jan. 2024) 383–393, https://doi.org/10.1016/J.PROCS.2024.04.038.

[31] D. Pathak, U.S.N. Raju, Content-based image retrieval using feature-fusion of groupnormalized-inception-darknet-53 features and handcraft features, Optik. (Stuttg) 246 (Nov. 2021) 167754, https://doi.org/10.1016/J.IJLEO.2021.167754.

[32] C. Upasana, A.S. Tewari, J.P. Singh, An attention-based pneumothorax classification using modified Xception model, Procedia Comput. Sci. 218 (Jan. 2023) 74–82, https://doi.org/10.1016/J.PROCS.2022.12.403.

[33] G. Mohandass, G.Hari Krishnan, D. Selvaraj, C. Sridhathan, Lung cancer classification using optimized attention-based convolutional neural network with DenseNet-201 transfer learning model on CT image, Biomed. Signal. Process. Control 95 (Sep. 2024) 106330, https://doi.org/10.1016/J.BSPC.2024.106330.

[34] B. Chandra, R.K. Sharma, Deep learning with adaptive learning rate using laplacian score, Expert. Syst. Appl. 63 (Nov. 2016) 1–7, https://doi.org/10.1016/J.ESWA.2016.05.022.

[35] Y. Nyirandikumana, M. Wilson, Efficacy of YOLO deep learning algorithm kidney tumor cancer detection, Int. J. Innov. Sci. Res. Technol. 8 (2) (2023). Accessed: Jul. 17, 2024. [Online]. Available: www.ijisrt.com.

[36] "GitHub - DaliaAlzubi/Kidneytumor: a novel deep learning approach for kidney tumors multi-diagnostic models based on CT scans." Accessed: Jul. 17, 2024. [Online]. Available: https://github.com/DaliaAlzubi/KidneyTumor.

[37] F.Y. Yap, et al., Shape and texture-based radiomics signature on CT effectively discriminates benign from malignant renal masses, Eur. Radiol. 31 (2) (Feb. 2021) 1011–1021, https://doi.org/10.1007/S00330-020-07158-0.

[38] N. Schieda, K. Nguyen, R.E. Thornhill, M.D.F. McInnes, M. Wu, N. James, Importance of phase enhancement for machine learning classification of solid renal masses using texture analysis features at multi-phasic CT, Abdom. Radiol. (NY) 45 (9) (Sep. 2020) 2786–2796, https://doi.org/10.1007/S00261-020-02632-1.

[39] F. Ma, T. Sun, L. Liu, H. Jing, Detection and diagnosis of chronic kidney disease using deep learning-based heterogeneous modified artificial neural network, Fut. Generati. Comput. Syst. 111 (Oct. 2020) 17–26, https://doi.org/10.1016/J.FUTURE.2020.04.036.

[40] F. Türk, M. Lüy, N. Barışçı, Kidney and renal tumor segmentation using a hybrid V-Net-based model, Mathematics 8 (10) (Oct. 2020) 1772, https://doi.org/10.3390/MATH8101772. *2020, Vol. 8, Page 1772*.

[41] F. Turk, RNGU-NET: a novel efficient approach in Segmenting Tuberculosis using chest X-Ray images, PeerJ. Comput. Sci. 10 (Feb. 2024) e1780, https://doi.org/10.7717/PEERJ-CS.1780/SUPP-1.