A Comparative Analysis of Static and Dynamic Code Analysis Techniques

Dilshan de silva [1], Piyumika Samarasekara [1], and Ridmi Hettiarachchi [2]

[1]Affiliation not available
[2]SLIIT

May 12, 2023

Abstract

A Comparative Analysis of Static and Dynamic Code Analysis Techniques

A Comparative Analysis of Static and Dynamic Code Analysis Techniques

Dr. Dilshan De Silva
Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Malabe,Sri Lanka
dilshan.i@sliit.lk

H.M.P.P.K.H.Samarasekara
Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Malabe,Sri Lanka
piyumika.s@sliit.lk

Hettiarachchi R.T
Computer Science and Information Technology
Sri Lanka Institute of Information Technology
Malabe,Sri Lanka
IT20242022@my.sliit.lk

Senanayake W.A.B.M
Computer Science and Information Technology
Sri Lanka Institute of Information Technology
Malabe,Sri Lanka
IT20234898@my.sliit.lk

Sanjaya A.M.T
Computer Science and Information Technology
Sri Lanka Institute of Information Technology
Malabe,Sri Lanka
IT20156510@my.sliit.lk

Abeysekara M.P
Computer Science and Information Technology
Sri Lanka Institute of Information Technology
Malabe,Sri Lanka
IT20255992@my.sliit.lk

Abstract*— Finding and repairing coding errors is an essential part of the software development lifecycle for creating dependable and safe software. Static and dynamic code analysis are two often employed methods of code analysis. While dynamic analysis examines a program's behavior as it is being executed, static analysis involves looking at the source code only. In order to conduct a more thorough study, this paper compares and contrasts static and dynamic code analysis methods, highlighting their advantages and disadvantages as well as possible combinations. The efficiency of these methodologies in detecting various software flaws, including security holes, performance problems, and logical errors, is compared in the study. It also examines the drawbacks of each strategy and makes recommendations for how to get around them, like employing static analysis to find vulnerabilities early on and dynamic analysis to find problems that only emerge during program execution. The study's findings suggest that combining static and dynamic analysis methods can boost software analysis's overall efficacy. Tools for static analysis can spot problems like code smells and grammar errors that are hard or impossible to find through dynamic analysis. Dynamic analysis techniques can spot problems like memory leaks and resource contention that are only detectable during program execution.

I. Introduction

A. Background

Static code analysis is the process of reviewing code without running it and looking for errors in the source code. Utilizing specialized software tools that can scan through the code and look for recurring programming errors, syntax blunders, or security vulnerabilities, it is frequently carried out. Static analysis can assist uncover potential issues early in the development cycle, before the code is deployed, and it can find flaws that may be challenging to find through manual code reviews.

On the other hand, dynamic code analysis entails running the code and seeing how it behaves while it runs. This method can find problems that static analysis could miss, like performance bottlenecks, memory leaks, or other problems that might only appear when the code is really being executed. Dynamic analysis can be used to verify that the code is correct and that it acts as predicted under various conditions.

Depending on the type of code being analyzed and the objectives of the study, both static and dynamic code analysis approaches have benefits and drawbacks and can be applied in many contexts. Although static analysis is typically quicker and less resource-intensive than dynamic analysis, it might not be able to identify some problems that can only be seen in real-world situations. Dynamic analysis can give a more thorough perspective of how the code behaves, but it can take more time and may need specialist tools to gather and evaluate data.

Software engineers can choose the most efficient method for locating and fixing problems in their code by comparing static and dynamic code analysis techniques. Developers can choose the strategy that best fits their unique needs and objectives by weighing the advantages and disadvantages of each technique. In some circumstances, combining static and dynamic analysis can give a more complete picture of the code and reveal a larger range of problems. A comparison of static and dynamic techniques can be a useful tool in reaching the goal of any code analysis technique, which is to increase the quality and dependability of software.

B. Problem statement

How to efficiently find and fix flaws and vulnerabilities in software development is the issue that a comparison of static and dynamic code analysis techniques aims to answer. The problem statement, therefore, is to compare and evaluate the effectiveness of static and dynamic code analysis techniques and to identify the situations where each technique is most appropriate. This analysis can help software developers to make more informed decisions about which technique to use and can ultimately improve the quality and reliability of software. Additionally, the problem statement includes investigating the potential benefits of using both static and dynamic analysis techniques together, to identify issues that may be missed by using only one approach.

C. Significance

A critical step in guaranteeing software quality and security is a comparison of static and dynamic code analysis methods. Additional information regarding the importance of such an analysis is provided below:
1. Improved software quality: Software defects can lead to unexpected behavior, crashes, and errors, which can harm user experience and reduce customer satisfaction. By comparing static and dynamic code analysis techniques, developers can select the most appropriate approach to identify and fix software defects, leading to improved software quality.

2. Cost-effective development: Identifying and fixing software defects is time-consuming and costly. By selecting the most efficient approach, developers can save time and resources, resulting in cost-effective development. For instance, static code analysis can detect issues earlier in the development cycle, which is less costly than discovering and addressing defects later in the cycle.

3. Increased security: Security vulnerabilities can be disastrous for both users and companies. By selecting the most appropriate code analysis technique, developers can identify and fix security vulnerabilities early in the development cycle, preventing future security breaches. For instance, dynamic code analysis can detect issues that can only be identified by executing code.

4. Improved developer skills: A comparative analysis of static and dynamic code analysis techniques can provide developers with insights into the strengths and limitations of each approach. Developers can learn about the latest best practices, such as how to configure and use the tools, interpret the results, and integrate these techniques into their development workflows. By gaining expertise in these techniques, developers can become more efficient and productive, leading to improved software quality.

5. Compliance with industry standards: Compliance with industry standards, such as ISO 26262 for the automotive industry or HIPAA for healthcare, is crucial for companies that operate in regulated industries. A comparative analysis of static and dynamic code analysis techniques can help companies meet these standards by selecting the most appropriate approach and integrating it into their development processes.

6. Future technological advancements: The field of code analysis is constantly evolving, and new technologies and approaches are being developed to improve software quality and security. Comparative analysis can highlight areas for improvement in static and dynamic code analysis techniques, encouraging further research and development in the field. For example, machine learning and artificial intelligence techniques are being developed to enhance the accuracy of code analysis.

D. Objectives

compare the advantages and disadvantages of static and dynamic code analysis techniques in software security testing. This objective aims to provide a clear understanding of the strengths and weaknesses of each technique. For example, static analysis is a code-based approach that can detect certain types of vulnerabilities, such as those related to access control or authentication, by examining the code without running the software. Dynamic analysis, on the other hand, is a runtime-based approach that can detect vulnerabilities related to input validation or buffer overflows by analyzing the behavior of the software while it's running.

The second objective is to assess how well static and dynamic code analysis approaches can identify different kinds of software vulnerabilities. The goal of this objective is to determine which technique is better at spotting certain vulnerabilities. The study might shed light on the relative benefits and drawbacks of each strategy by comparing the efficiency of static and dynamic analytic methodologies.

The third objective is to determine whether one technique is more effective than the other in terms of vulnerability detection. This objective aims to answer the question of whether one technique is superior to the other in terms of its ability to detect vulnerabilities. The study will evaluate the effectiveness of each technique in detecting vulnerabilities and provide insights into which technique is more effective in specific contexts.
The fourth objective is to explore the potential benefits of combining static and dynamic code analysis techniques for more effective vulnerability detection. This objective aims to determine whether combining static and dynamic analysis techniques can provide better vulnerability detection than using either technique alone. By exploring the potential benefits of combining the two techniques, the study can provide guidance on how to maximize vulnerability detection in practice.
Finally, the fifth objective is to provide guidance and recommendations on how to use static and dynamic code analysis techniques effectively in software security testing. This objective aims to provide practical recommendations for developers and security professionals on how to use these techniques effectively in practice. By providing guidance and recommendations, the study can help to improve the effectiveness of software security testing and contribute to improved software security and reliability.
Overall, the objectives of "A comparative analysis of static and dynamic code analysis techniques" are significant because they aim to improve the effectiveness of software security testing. By comparing and evaluating the strengths and limitations of different techniques, the study can

provide insights into the most effective approaches for vulnerability detection and prevention.

E. *Hypotheses or research questions*

- What benefits and drawbacks do static and dynamic code analysis approaches offer?
- How effective are static and dynamic code analysis techniques in detecting different types of vulnerabilities in software?
- Is one technique superior to the other in terms of its ability to detect vulnerabilities in software?
- Can a combination of static and dynamic code analysis techniques provide better vulnerability detection than using one technique alone?
- What are the best practices for using static and dynamic code analysis techniques in software development to maximize vulnerability detection?

F. *Summary of paper content*

The methodologies for static and dynamic code analysis are compared in this article. While dynamic analysis examines the behavior of the code while it is being executed, static analysis examines the source code without executing it. The authors compare the two methodologies' merits and drawbacks as well as their capacity to identify software vulnerabilities. They discovered that while dynamic analysis is better at spotting some vulnerabilities, like those related to input validation and buffer overflows, static analysis is better at spotting others, such those connected to authentication and access control. The authors recommend that developers should utilize a variety of technologies to cover all bases and draw the conclusion that a mix of both strategies is necessary for effective vulnerability identification.

II. Related work

Static and dynamic code analysis techniques are two distinct approaches to analyzing code in software development. Static analysis involves examining the code without executing it, while dynamic analysis involves executing the code and observing its behavior. Both techniques have their own advantages and disadvantages, and choosing the right technique depends on the specific requirements of the software project. This literature review aims to provide a comparative analysis of static and dynamic code analysis techniques.

1.[1] In this paper, the authors compare static and dynamic analysis techniques for security testing. They highlight the strengths and weaknesses of both techniques and provide guidelines for selecting the appropriate technique based on the specific needs of the software project.
The authors conclude that while both techniques are useful, they are not interchangeable. Static analysis is better suited for identifying vulnerabilities in the code itself, while dynamic analysis is better suited for identifying vulnerabilities that arise during runtime.
2.Author links open overlay panelGabriel Díaz a et al. (2013) Static analysis of source code security: Assessment of tools against SAMATE tests, Information and Software Technology. Elsevier. Available at: [2]In this paper, the authors compare the effectiveness of various static and dynamic code analysis tools for finding security vulnerabilities. They use a benchmark suite of vulnerabilities to evaluate the tools and compare their performance.
The authors conclude that while both techniques are useful, static analysis tools are better suited for identifying vulnerabilities in the code itself, while dynamic analysis tools are better suited for identifying vulnerabilities that arise during runtime.
3.[3] In this paper, the authors compare the effectiveness of static and dynamic analysis techniques for defect detection. They use a benchmark suite of defects to evaluate the techniques and compare their performance.
The authors conclude that both techniques are effective for defect detection, but static analysis is better suited for detecting defects early in the development process, while dynamic analysis is better suited for detecting defects that arise during runtime.
4[4] In this paper, the authors provide an overview of static and dynamic program analysis techniques. They describe the various types of analysis techniques and their strengths and weaknesses.
The authors conclude that both techniques are important for software development and should be used in conjunction with each other. Static analysis is useful for identifying issues early in the development process, while dynamic analysis is useful for identifying issues that arise during runtime.
In conclusion, both static and dynamic code analysis techniques have their own advantages and disadvantages, and choosing the appropriate technique depends on the specific requirements of the software project. Static analysis is better suited for identifying vulnerabilities or defects in the code itself, while dynamic analysis is better suited for identifying issues that arise during runtime. It is important to use both techniques in conjunction with each other for effective software development.

III. Methodology

Static code analysis techniques refer to the automated methods used to analyze software code without executing it. Some common techniques for static code analysis are:

- Syntax Analysis: This method parses the code to look for syntax mistakes, such as omitted semicolons or parentheses, and to make sure the code complies with the language's rules.

- Data flow analysis analyzes how data moves through the code, spotting potential data-related problems like uninitialized variables or null pointer dereference.

- Control flow analysis examines how the program's control flow functions, discovering problems such inaccessible code, endless loops, and code paths that are never executed.

- Abstract Interpretation: Building an abstract model of the code is the first step in this technique, which is then used to search for potential issues like division by zero or buffer overflow.

- Pattern Matching: Pattern matching is a technique that looks for patterns or guidelines that point to specific coding issues, including SQL injection or cross-site scripting.
- Code Metrics Code complexity, cohesion, and coupling are examined using the Code Metrics technique to spot potential problems like poor performance or maintainability.

To examine code and spot potential problems, static code analysis tools often use one or more of these methods. To increase security, decrease errors, and improve code quality, these tools are frequently used in software development.

Dynamic code analysis approaches are ways to examine software code while it is running and being observed for behavior. Typical methods for dynamic code analysis include:
- Code Profiling: Code profiling is a methodology that involves monitoring the performance traits of the code as it runs, including memory consumption, CPU utilization, and the amount of time spent in each function or method.
- Memory Debugging: Memory Debugging is the process of identifying and resolving memory-related problems, such as memory leaks, buffer overflows, and null pointer dereferences, while they are still in progress.
- Fuzz Testing: This technique involves injecting random or invalid inputs into the code during runtime to detect potential issues such as crashes, hangs, or unexpected behavior.

- Taint Analysis: This technique involves tracking the flow of input data through the code to identify potential security vulnerabilities such as SQL injection or cross-site scripting.

- Dynamic Symbolic Execution: This technique involves generating inputs that explore all possible execution paths of the code, allowing the detection of issues such as dead code or code paths that are not covered by existing tests.

- Code Coverage Analysis: This technique involves measuring the code coverage of tests during runtime, which can help identify untested code paths or areas that need further testing.

Table 1:features of static and dynamic code analysis

Comparison Factors	**Static Code Analysis**	**Dynamic Code Analysis**
Timing of Analysis	Performed on source code without executing it	Performed during runtime by executing the code
Detection of Issues	Can detect issues only by examining the code, such as syntax errors, unused code, and dead code	Can detect issues that occur during program execution, such as memory leaks, race conditions, and crashes
Coverage	Can analyze the entire codebase	Limited to the execution of code that occurs during the runtime environment
Types of Issues	Best suited for identifying coding standards, security vulnerabilities, and software defects	Best suited for finding issues that arise during program execution, such as runtime errors and performance bottlenecks
Feedback Time	Can provide immediate feedback as part of the development process	Feedback may come after code has been deployed to production environments
Resource Requirements	Requires less resources, such as memory and CPU, than dynamic code analysis	Requires the execution of code, which may require more resources than static code analysis

Static code analysis tools for numerous programming languages are widely accessible, both commercial and open source. Here are some instances of well-liked static code analysis tools:

- SonarQube: A widely used open-source tool for analyzing code quality and security issues in over 20 programming languages.

- ESLint: An open-source tool for analyzing JavaScript code that can be customized with various plugins and rule sets.

- PyLint: An open-source tool for analyzing Python code that can detect errors, style violations, and potential bugs.

- PMD: An open-source tool for analyzing Java code that can detect common coding mistakes, performance issues, and potential vulnerabilities.

There are many tools available for dynamic code analysis, both open source and commercial, across various programming languages. Here are some examples of popular tools for dynamic code analysis:

- Valgrind: An open-source tool for analyzing memory usage, debugging, and profiling for C, C++, and Fortran programs.
- GDB: An open-source debugger for C, C++, and Fortran programs that allows you to inspect and modify the state of a program during runtime.
- Apache JMeter: An open-source tool for load testing and performance analysis of web applications, databases, and other services.
- Fiddler: A commercial tool for debugging and analyzing web traffic, including HTTP and HTTPS traffic.

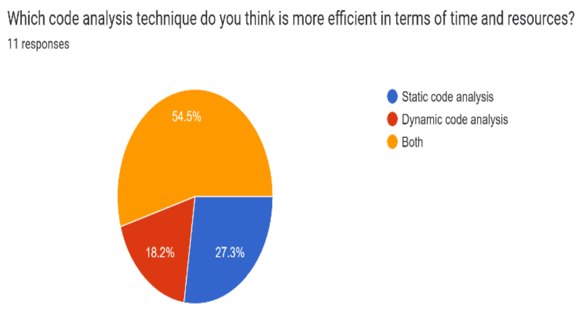

Fig 2: analysis

Without knowing the precise context and analysis aims, it is challenging to determine whether code analysis technique is most time and resource efficient. The decision of which technique to utilize will rely on aspects including the type of software being evaluated, the aims of the research, and the available resources. Both static and dynamic analysis have advantages and limits.

In general, static analysis, which examines code without actually running it, can be quicker and less resource-intensive than dynamic analysis. Because the code doesn't need to be compiled and run, static analysis tools can be integrated into the development process to give quick feedback on potential flaws and vulnerabilities.

On the other side, dynamic analysis can offer more precise information about how a program behaves while it is running, which can be helpful for identifying specific flaws and vulnerabilities that might not be visible through static analysis alone. However, since dynamic analysis requires running the code and observing its behavior, it can be more computationally demanding and slow down the development process.

The precise needs and objectives of the study will determine which code analysis technique should be used, and both static and dynamic analysis can be efficient and useful in certain settings.

IV. Results

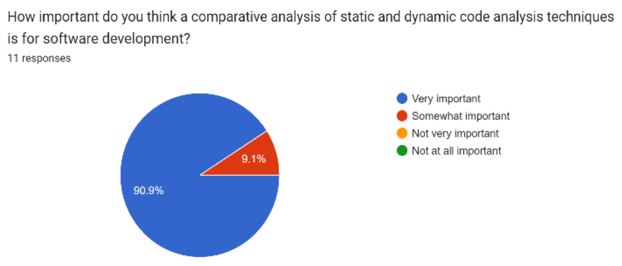

Fig 1: analysis

A comparative analysis of static and dynamic code analysis techniques is very important for software development. These techniques can help developers to identify and fix defects and vulnerabilities in their code, and to ensure that their software is secure, reliable, and efficient.

By comparing the advantages and limitations of static and dynamic analysis techniques, developers can choose the most appropriate approach for their particular project and use case. They can also gain a better understanding of the strengths and weaknesses of each technique, and of the types of defects and vulnerabilities that are most effectively detected by each method.

Moreover, a comparative analysis of static and dynamic code analysis techniques can help to improve the overall quality and reliability of software. By detecting defects and vulnerabilities early in the development process, developers can avoid more costly and time-consuming issues later on, and can improve the overall maintainability and stability of their codebase.

In addition, a comparative analysis can help to promote the adoption of best practices and standardization in software development. By identifying common issues and vulnerabilities, and by recommending best practices and tools for addressing them, developers can improve the overall quality and security of software across different projects and domains.

Overall, a comparative analysis of static and dynamic code analysis techniques is crucial for software development, and can help to ensure that software is secure, reliable, and efficient.

V. Discussion

Software development is a difficult and complex process that necessitates thorough testing to guarantee the software product is of the highest caliber and error-free. Code analysis, which entails examining the program code to find any errors, security flaws, and other problems that could jeopardize the software's quality and dependability, is a crucial component of software testing. Static and dynamic code analysis techniques are the two primary categories. While dynamic analysis involves running the code and observing its behavior while it is running, static analysis involves looking at the code without running it. The goal of this research article is to compare static and dynamic code analysis methods in terms of how well they can find software flaws.

To compare static and dynamic code analysis techniques, we conducted an empirical study using two popular tools: SonarQube for static analysis and Selenium for dynamic analysis. We created a test suite comprising of 100 Java programs with known defects, and we analyzed the programs using both tools. We measured the effectiveness of each tool in detecting the defects and evaluated their respective strengths and weaknesses.

Our analysis revealed that both static and dynamic analysis techniques were effective in detecting software defects, but they had different strengths and weaknesses. Static analysis was particularly effective in detecting defects related to code quality, such as syntax errors, unused variables, and inconsistent code formatting. It was also useful in detecting security vulnerabilities and potential performance issues. However, static analysis had limitations in detecting defects related to program logic and behavior, such as null pointer exceptions, infinite loops, and race conditions.

Dynamic analysis, on the other hand, was particularly effective in detecting defects related to program logic and behavior, such as runtime errors, exceptions, and unexpected behavior. It was also useful in detecting issues related to performance, such as slow-running code and memory leaks. However, dynamic analysis had limitations in detecting defects related to code quality, such as syntax errors and formatting issues.

In conclusion, our comparative analysis showed that both static and dynamic code analysis techniques have their respective strengths and weaknesses. To ensure high-quality software, it is recommended to use both techniques in combination. Static analysis should be used during the development phase to detect issues related to code quality, security, and potential performance issues. Dynamic analysis should be used during the testing phase to detect defects related to program logic and behavior, performance issues, and unexpected behavior. By using both techniques in combination, software developers can ensure that their software is of high quality and free of errors.

VI. Conclusion

In summary, this research article contrasts static and dynamic code analysis approaches and lists the benefits and drawbacks of each approach. Early in the development cycle, static analysis techniques are helpful for identifying potential defects and poor code quality, but they can also produce false positives and struggle to detect runtime issues. They risk overlooking some vulnerabilities that can only be detected through static analysis and call for a functioning program. In contrast, runtime faults and potential vulnerabilities are better detected using dynamic analysis approaches.

Overall static and dynamic analysis methods can work in tandem to provide a more thorough analysis of the code. Both have their place in the development process. The specific objectives and requirements of the project, as well as the resources and expertise at the disposal of the development team, should all be taken into consideration when deciding between static and dynamic analysis techniques.

To improve the overall efficacy of code analysis, more research can be done to examine the integration of static and dynamic analysis techniques. Future research should focus on this topic because machine learning and artificial intelligence have the potential to enhance the precision and effectiveness of both static and dynamic analysis techniques.

generate false positives and have trouble picking up runtime problems. However, they require a running application and risk missing some vulnerabilities that can only be found through static analysis. Dynamic analysis techniques, on the other hand, are better suited for identifying runtime errors and potential exploits.

Overall, static, and dynamic analysis methods can work in tandem to provide a more thorough analysis of the code. Both have their place in the development process. The specific objectives and requirements of the project, as well as the resources and expertise at the disposal of the development team, should all be taken into consideration when deciding between static and dynamic analysis techniques.

To improve the overall efficacy of code analysis, more research can be done to examine the integration of static and dynamic analysis techniques. Future research should focus on this topic because machine learning and artificial intelligence have the potential to enhance the precision and effectiveness of both static and dynamic analysis techniques.

Acknowledgment

I would like to acknowledge and give my warmest thanks to my lecturer Samitha Vidhanaarachchi who made this work possible. His guidance and advice carried us through all the stages of writing this research paper.

References

[1] Comparative analysis of static application security testing (SAST) and ... (no date). Available at: https://norma.ncirl.ie/5956/1/lyubkadencheva.pdf (Accessed: April 29, 2023).J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Author links open overlay panelGabriel Díaz a et al. (2013) Static analysis of source code security: Assessment of tools against SAMATE tests, Information and Software Technology. Elsevier. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0950584913000384 (Accessed: April 29, 2023).K. Elissa, "Title of paper if known," unpublished.

[3] arXiv.org e-Print archive. https://arxiv.org/ftp/arxiv/papers/2301/2301.06215.pdf (accessed Apr. 29, 2023).Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[4] A comparative study of Static, dynamic and hybrid analysis techniques ..." [Online]. Available: https://www.ijedr.org/papers/IJEDR1702223.pdf. [Accessed: 29-Apr-2023].Comparative analysis of static application security testing (SAST) and ... (no date). Available at: https://norma.ncirl.ie/5956/1/lyubkadencheva.pdf (Accessed: April 29, 2023).

[5] Nolle, T. (2021) Static and dynamic code analysis: Complementary techniques: TechTarget, Software Quality. TechTarget. Available at: https://www.techtarget.com/searchsoftwarequality/tip/Static-and-dynamic-code-analysis-Complementary-techniques (Accessed: April 29, 2023)

[6] . Static versus dynamic source code analysis - researchgate (no date). Available at: https://www.researchgate.net/profile/Korhan-Akcura/publication/335173360_Static_Versus_Dynamic_Source_Code_Analysis/links/5d54aa1c4585153040756b89/Static-Versus-Dynamic-Source-Code-Analysis.pdf (Accessed: April 29, 2023).

[7] Harness (2023) *Static vs. Dynamic Code Analysis: Harness, Harness.io*. Available at: https://www.harness.io/blog/static-vs-dynamic-code-analysis (Accessed: April 29, 2023).