IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Intelligent Python Code Analyzer [IPCA]

[1]Christie Thottam, [2]Nigel Fernandes, [3]Rehan Joseph, [4]Imran Mirza

[1]Student, [2]Student, [3]Student, [4]Professor

[1]Department Of Computer Engineering,

[1]Don Bosco Institute of Technology, Mumbai, India

Abstract: A revolutionary improvement in Python code development tools is the Intelligent Python Code Analyzer (IPCA). IPCA pushes the boundaries of conventional static analysis tools by utilizing state-of-the-art artificial intelligence approaches to improve efficiency and code quality. In contrast to traditional methods, IPCA offers context-aware and dynamic code analysis, giving developers smart insights. IPCA's primary goal is to optimize Python codebases by pointing out problems and making suggestions for fixes. By utilizing sophisticated algorithms, the tool offers developers practical suggestions for problem discovery and code optimization that go beyond superficial examination. IPCA can comprehend complex code links and dependencies because to the incorporation of artificial intelligence, and it can provide insightful recommendations that go beyond grammatical accuracy. Because IPCA integrates easily with well-known Python Integrated Development Environments (IDEs), it has the potential to completely transform the development workflow. The coding process is streamlined for developers by providing them with intelligent ideas and real-time feedback in their comfortable working environment. With the introduction of a revolutionary tool that not only finds inefficiencies but also makes it easier to take a proactive approach to bug avoidance, this project epitomizes a dedication to code perfection. IPCA becomes a valuable tool for Python developers, promoting a continuous development mindset and establishing a new benchmark for intelligent, context-aware code analysis.

Index Terms **- IPCA, code analysis, software development, Abstract syntax tree, command-line-interface (CLI)**

I. INTRODUCTION

Efficient code analysis tools are becoming essential in the ever changing software development world. There is an increasing demand for intelligent solutions that can understand, optimize, and improve Python code as projects get more complicated. The goal of this project is to develop an intelligent Python code analyzer by using modern approaches that were motivated by the latest IEEE research in the area. A new age of code analysis, characterized by creative methods for software understanding and optimization, has been brought about by the explosion of machine learning and artificial intelligence. Current IEEE papers—published after 2019—have been significant in influencing the state of intelligent code analyzers, especially in the context of Python programming. A notable trend in computer science education over the past few years has been the creation of more tools with the goal of improving the learning process for inexperienced programmers. IPCA is a static code analysis tool that has been carefully crafted to offer automatic feedback to students taking beginner Python classes. This study explores the IPCA module in response to the growing need for scalable and effective ways to evaluate student code, promote a better understanding of programming principles, and expedite the learning process. Static code analysis tools like as IPCA are essential for analyzing student code because they use algorithms to navigate code structures, identify syntactic and semantic problems, and produce detailed results. By closely examining the underlying algorithms for static code analysis, file validation, command-line interface processing, and server communication, this paper significantly adds to the overall functionality and efficacy of the tool. The research aims to shed light on the essential elements driving IPCA's success in the educational landscape. Static code analysis tools like as IPCA are essential for analyzing student code because they use algorithms to navigate code structures, identify syntactic and semantic problems, and

produce detailed results. By closely examining the underlying algorithms for static code analysis, file validation, command-line interface processing, and server communication, this paper significantly adds to the overall functionality and efficacy of the tool. The research aims to shed light on the essential elements driving IPCA's success in the educational landscape.

II. RESULTS

There are some existing systems closely related to the proposed idea of developing an obscene content blocker. A literature review of the proposed systems or the existing systems was performed to analyze these systems. It gives an idea of how the already existing systems were made or developed, the different implementation methods used to establish software according to the required requirements and issues faced by the developers during the period of development and the most convenient or possible outcomes of each method.

The article titled 'Software Modernization Using Machine Learning Techniques' was published by Norbert Somogyi, Gabor Kovesdan proposes a novel approach of using machine learning for software modernization, which has not been explored much before. This could open up new possibilities for improving automation and precision in legacy code transformations. Presents a concrete case study demonstrating how machine learning can be applied to a specific problem in pointer conversion between C and Java. Provides a proof of concept implementation showing feasibility. Shows through experimentation that the machine learning approach significantly outperforms traditional static analysis for the pointer conversion problem, improving accuracy from 58 to 74 percent Demonstrating clear benefits. Well structured paper that gives good background on software modernization challenges, analyzes related work, and motivates the need for new techniques like machine learning. The sample data set used for training and testing the machine learning model is relatively small at 378 examples. More data would be needed to make the results more robust. The process of manually gathering accurate training data through code transformation and observation seems time consuming and limits the overall automation level. The improvements shown are on a narrow test case of pointer conversion. Benefits for larger scale modernization tasks are still unclear. No direct comparison to other machine learning techniques for this problem. Unclear if this deep learning approach is optimal. The convergence of the training process and optimal network parameters could be analyzed more thoroughly. Overall the paper makes a nice contribution demonstrating the potential of machine learning for software modernization. But more work would be needed to expand the approach and evaluate benefits more conclusively. The pros highlight the promising possibilities, while the cons indicate limitations and areas for improvement.[1]

The article titled 'Code Vulnerability Identification and Code Improvement using Advanced Machine Learning' was published by Laneesha Ruggahakotuwa,Lakmal Rupasinghe,Pradeep Abeygunawardhana focuses on an important problem of identifying vulnerabilities in source code to prevent exploits and security breaches. This could have high practical impact. Proposes using a combination of static and dynamic analysis to detect vulnerabilities, providing complementary approaches. Aims to not just detect vulnerabilities but also automatically suggest fixes to correct the code. This could save significant developer time. Leverages machine learning and deep learning techniques which are well suited for automatically identifying patterns and issues in code. Provides concrete examples of detecting and fixing SQL injection and cross-site scripting vulnerabilities. Developing as an open source plugin makes it accessible and extensible. The machine learning approaches are discussed at a high level without details on specific models, algorithms, or training process. Limited evaluation with only two vulnerability examples. More thorough evaluation needed. Unclear how effective the automatic fix suggestions would be across a broad range of vulnerabilities. Additional comparison to existing tools would be useful to benchmark performance. Lacks analysis of false positive and false negative rates. Uncertain how the approach would handle new types of vulnerabilities not seen during training. Overall the paper explores an important direction of using ML to secure code. But more algorithmic and evaluation details would strengthen the approach. The pros highlight promising possibilities while cons indicate areas needing further development.[2]

The article titled 'Machine Learning based Static Code Analysis for Software Quality Assurance' was published by Gergana Vladova, Stefan Konopik, Andr´e Ullrich, Eldar Sultanow Capgemini addresses the important practical problem of improving code quality and reducing bugs in large mission-critical software systems. Proposes a novel approach of using machine learning for static code analysis to find hidden flaws not detectable by standard tools. Provides a concrete case study demonstrating the approach on a large 800,000+ lines of code system used by the German Federal Employment Agency. Explains the algorithms and techniques in sufficient technical detail, including association rule mining. Shows examples of actual code quality issues identified that led to bugs in production. Prototype integration with Eclipse provides accessible implementation for developers. The training data relies on existing hand-written code which may also contain flaws. Only evaluates on a single proprietary system, more case studies needed. Limited explanation for how new/changing

code influences the trained model over time. Additional comparison to traditional static analysis tools would be useful. No quantitative evaluation of issues found or improvements measured. Focused only on code quality, could expand to security, performance, etc. Overall the paper demonstrates a promising usage of machine learning for an important software engineering problem. The pros show it tackles a real-world issue with an innovative approach, while the cons suggest ways the techniques could be expanded and evaluated more thoroughly.[3]

III. EXISTING SYSTEM

3.1 Findings in the existing system

A number of systems are available in the current computer science education tool landscape that are designed to improve the learning process for inexperienced programmers. These technologies use static code analysis to create interactive learning experiences, enhance code quality, and provide automatic feedback. The main conclusions from a comparative study of several systems, each making a distinct contribution to the field of programming instruction, are examined in this part. One goal that many systems have in common is to increase the quality of the code. These programs find and fix problems with coding standards, style infractions, and possible flaws using static code analysis techniques. The focus on code quality is in line with the main objective of encouraging developers and learners to adopt best coding techniques. Interactive learning is prioritized on a number of platforms. These solutions give real time feedback while coding activities are being performed by incorporating static code analysis. By giving users instant feedback on the style and accuracy of their code, this interactive method facilitates iterative learning and improves the learning process. Some sites use a gamified strategy for their coding tasks. Users take part in coding challenges and tournaments, which promotes teamwork and competitiveness within a community. This gamification technique promotes users to hone their coding abilities through entertaining and stimulating exercises, which enhances the learning environment. A unique feature of some systems is their support for several programming languages, which gives users access to a single online development environment. Because of its inclusivity, Python is just one of the many languages that developers and students can experiment with and learn. This kind of customization helps the platform meet a wide range of learning requirements. Numerous platforms clearly state that they are meant to be instructive. To help students learn programming, these platforms include structured courses, interactive activities, and exams. These systems strive to improve the educational landscape by making coding more accessible, especially for novices, by offering guided educational content. Online coding communities encourage users to report problems and discuss them in a group setting, which promotes cooperation. This collaborative feature encourages peer learning and knowledge exchange. By participating in conversations about code quality, efficiency, and other possible solutions, users help learners feel more connected to one another. Coding challenges are an integral part of the platforms of these groups. Users take on real-world issues and provide solutions for review. In addition to strengthening coding abilities, this method offers a real-world setting in which to apply programming ideas, enhancing the overall educational process. Some programs function as integrated tools inside the workflows of developers, with an emphasis on code quality analysis. These resources support developers in upholding coding standards and seeing any problems early on in the process. The industry's emphasis on integrating code quality techniques throughout the development lifecycle is in line with the smooth integration of these technologies.

3.2 Limitations

Although current static code analysis tools offer useful information about the style, quality, and possible problems in the code, they frequently have drawbacks in terms of user interest and educational focus. A lot of technologies don't specifically focus on helping inexperienced programmers in an educational environment. Furthermore, the learning process might not be interactive or customized to meet the needs of each particular student, and the feedback that is given could be general.

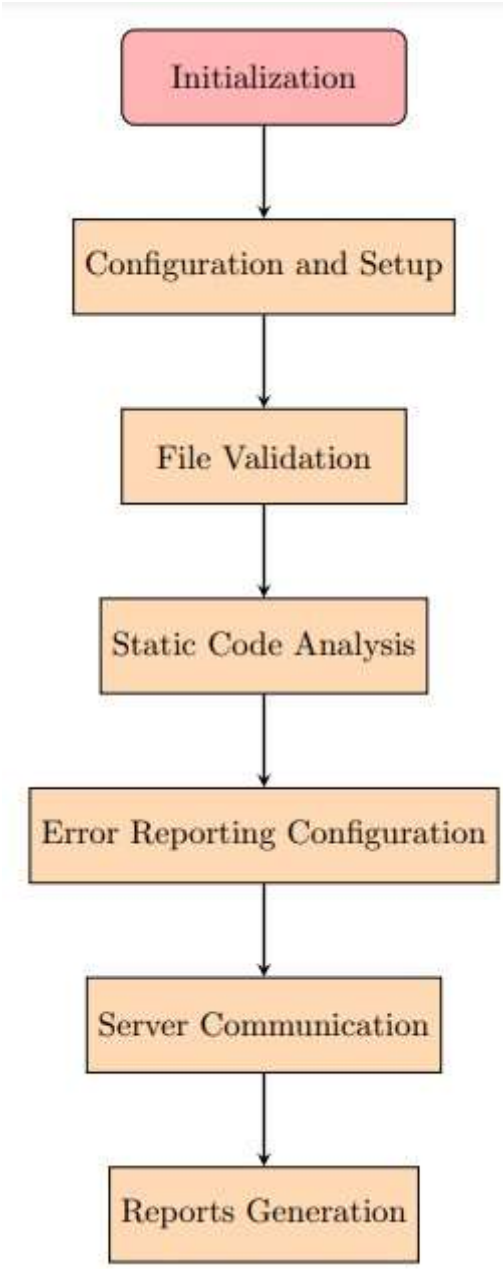
3.3 Proposed Work

In order to overcome these constraints, IPCA focuses on meeting the requirements of inexperienced programmers taking beginning Python classes. Its architecture makes use of instructional concepts to provide focused and helpful feedback, which improves the learning process. By providing users with in-depth analysis of their code, IPCA's interactive feedback system guarantees that users gain a deeper comprehension of programming principles. The tool stands out for its emphasis on customization and user-friendly interactions, which makes it an invaluable resource in the field of education. Using this framework, you can emphasize IPCA's benefits in resolving the constraints that have been found, as well as its function in facilitating efficient learning and offering customized feedback to inexperienced programmers.

IV. METHODOLOGY

The study project's technique entails a methodical analysis of the IPCA codebase. The first step in the investigation is a thorough codebase inspection, where each component is examined in detail to determine its function and contribution. The goal of this phase is to provide a fundamental understanding of IPCA's architecture by emphasizing important elements and how they work together. After the preliminary analysis, the paper concentrates on the algorithmic underpinnings of IPCA, with specific attention to tree architectures, visitor patterns, and code duplication verification. An essential component of IPCA's capabilities, static code analysis, is examined in detail using these algorithmic foundations. The study then moves on to the IPCA feedback creation processes, including the approaches for producing feedback messages and grouping them logically. This stage seeks to understand how IPCA gives users constructive feedback, which is in line with the overarching objective of improving the learning process for inexperienced programmers. A thorough investigation of server connection methods is also included in the process, with a focus on safe data transfer and the possible input of anonymized data. This feature demonstrates IPCA's dedication to privacy concerns and data integrity in educational settings. The study also examines the setup and configuration procedures, including loading IPCA setups. Comprehending IPCA's customization and configuration capabilities is essential to appreciating its versatility in accommodating a range of user preferences and educational environments. We look closely at the validation techniques used in IPCA, including decorator functions, special exceptions, and set ends transformations. The purpose of this phase is to clarify how IPCA maintains the robustness and dependability of the validation of its teaching tools. Drawing on findings from prior studies, the investigation also addresses cross-platform compatibility problems within IPCA. Evaluating IPCA's practical utility requires an understanding of how it handles issues and maintains consistency across various operating systems. Lastly, an examination of the HTML and Markdown reporting formats used by IPCA is part of the methodology. This stage assesses how well IPCA's reporting systems enable user interpretation and interaction with the findings of analysis. By using this methodology, the research hopes to provide a deeper understanding of IPCA's architecture and functionality, which will be of great value to the field of computer science teaching tools. The methodical inspection of every component guarantees a thorough and in-depth analysis, setting the stage for well-informed conversations and possible advancements in the field of static code analysis for educational objectives.

V. FLOWCHART



The initialization of the IPCA module paves the way for the thorough examination of Python code in the first stage. This stage creates the fundamental framework required for further assessments. IPCA comes with a plethora of configuration choices that cover a wide range of parameters. By taking this step, users can customize the tool to meet their own needs and preferences while also ensuring flexibility and adaptability. A thorough file validation procedure is carried out before analysis begins. This ensures that the input files are legitimate Python files, averting possible problems and guaranteeing a seamless analysis procedure. This step, which is the core of IPCA, entails running sophisticated algorithms for static code analysis. The identification of syntactic and semantic problems in the code helps to provide a comprehensive picture of its structure and quality. Carefully designed, the output reporter improves user experience. This stage makes sure that the code analysis results are presented in a way that makes it easy to understand the issues that have been found. Strong logic for server communication is incorporated into IPCA, potentially enabling the submission of anonymized data. This feature supports data-driven and cooperative changes in line with modern methods. The creation of thorough analysis reports marks the culmination of the IPCA procedure. These reports help users make well-informed decisions by giving them insightful information about possible areas for improvement and the quality of the code.

VI. ALGORITHMS USED

Tree Traversal Algorithm

Through its navigation of the Abstract Syntax Tree (AST), the technique makes hierarchical exploration of Python code easier. Using pre-order, in-order, and post-order tree traversal techniques, IPCA analyses code constructions in a methodical manner. This algorithm makes sure that the AST is thoroughly examined, which lays the groundwork for further analysis.

Syntactic Analysis Algorithm

Python code is parsed to determine syntactic components and grammatical structure. Using the ast module that comes with Python, the algorithm analyses and parses code in a methodical manner to identify syntactic patterns. Comprehending the syntactic structure facilitates further analyses and improves IPCA's comprehension of code composition.

Semantic Analysis Algorithm

Putting more emphasis on the correctness and meaning of the code than just its syntax. By analyzing variables, functions, and their relationships, IPCA uses semantic analysis techniques to understand the code's intent. Semantic analysis improves IPCA's ability to recognize logical mistakes and offer thorough feedback.

Error Checking Algorithm

Recognizing and disclosing frequent style infractions and programming faults. The code is regularly checked by the algorithm, which ensures adherence to coding standards by using established rules and patterns. By identifying problems with variable naming standards, indentation, and other style-related issues, IPCA ensures code quality.

Feedback Generation Algorithm

Producing insightful feedback messages based on findings from static code analysis. The algorithm improves user interpretation by grouping feedback into logical sets like the MessageSet structure. The goal of IPCA is to provide newbie programmers with feedback that is both actionable and understandable, hence improving their learning process.

Server Communication Algorithm

Establishing safe channels of communication between a centralized server and IPCA. Secure data transfer can be facilitated by implementing protocols, possibly with the use of HTTPS or other secure techniques. permits IPCA to send anonymized data to a central server, which aids in the development of tools and instructional methods.

File Validation Algorithm

Before analysis, make sure the input files are legitimate Python files. Verifies file extensions, formats, and basic syntax to avoid problems with further analysis. IPCA relies heavily on file validation to prevent needless errors and guarantee correct code analysis.

Configurability and User Interaction Algorithm

Putting into practice algorithms that manage user interactions and setups, improving user experience. involves loading configuration files, handling user preferences for an intuitive interface, and processing command-line arguments. Because of this algorithmic feature, IPCA is more user-friendly overall and can adjust to different preferences.

VII. TOOLS AND TECHNOLOGIES USED

To guarantee resilience, effectiveness, and maintainability, a variety of techniques and technologies are heavily employed in the creation of the Intelligent Python Code Analyzer (IPCA). An outline of the essential elements that make up the IPCA codebase is given below: Python, a popular programming language renowned for its clarity and conciseness, is mostly used in the development of IPCA. Python is a good choice for teaching tools since it supports the objective of giving inexperienced programmers a user-friendly environment. IPCA's fundamental functionality, which uses well-established methods and approaches, is mostly dependent on code analysis libraries. Notable libraries are Tokenize, which is used to tokenize Python source code—a critical step in static code analysis—and Astroid, which is used for manipulating and analyzing abstract syntax trees (ASTs). To improve its functionality, IPCA incorporates external libraries. Distinguished instances comprise Click, which is employed to generate an intuitive command-line interface (CLI) for IPCA, and Requests, which is crucial for managing HTTP requests and enabling server interaction. ConfigParser is used to handle configuration settings, allowing configuration files to be loaded and parsed. This gives configuration flexibility for IPCA parameters based on particular needs. IPCA uses Sphinx, a program that makes it easier to create excellent documentation from Python code, for documentation needs. This makes the codebase more

understandable. Git is used by IPCA to administer version control, which guarantees effective code tracking, versioning management, and collaboration. This makes it possible for coordinated development, teamwork, and efficient code management. Incorporating this extensive array of instruments and technologies into the account of IPCA's evolution affords openness regarding the technological basis. Readers, researchers, and developers can better grasp the various interconnected tools that make IPCA function by using this information. Changes can be made to ensure clarity and relevancy based on the particular technologies utilized in the codebase.

VIII. Code Quality Metrics

For any educational tool to be effective and maintainable, the code must be of a high quality. Multiple code quality criteria are used to conduct a thorough evaluation of the Intelligent Python Code Analyzer (IPCA). The main measures used to evaluate the stability and effectiveness of IPCA's codebase are highlighted in this section. One basic metric used to measure code complexity is cyclomatic complexity. In order to maintain low cyclomatic complexity numbers, IPCA designs sophisticated algorithms with simplicity in mind. These algorithms are used for server communication, file validation, and static code analysis, which results in reduced complexity scores. Duplication of code can cause problems and make it harder to maintain. By utilizing algorithms to detect and reduce code duplication, IPCA tackles this issue. Regular audits are performed on the codebase to reduce redundancy and improve readability. A composite metric called the Maintainability Index takes into account a number of variables that affect the maintainability of code. High maintainability index values are emphasized by IPCA, which encourages readability, clarity, and adherence to best practices in code design. This is especially important for a learning tool meant for inexperienced programmers. In-depth testing is essential to the advancement of IPCA. An large suite of unit tests is included with the codebase to ensure thorough coverage in a variety of scenarios. High test coverage is the goal of IPCA, which is in line with best practices in software development and improves robustness and dependability. One of the main components of IPCA's code quality policy is adherence to PEP 8, the official Python style standard. The naming standards, code formatting, and general stylistic coherence receive special attention. PEP 8 compliance is enforced by automated tools, which helps maintain a standardized and clean codebase. Finding chances for refactoring is one way to approach continuous improvement. Periodic code reviews and component analysis, particularly for static code analysis and feedback creation, aid in identifying areas that need improvement. The main goals of refactoring are to increase code clarity and optimize algorithms. For documentation to be comprehensible, it must be of high quality. IPCA creates well-organized, succinct, and lucid documentation using Sphinx. Regular assessments of the documentation guarantee its relevance and accuracy, which helps to maintain the high caliber of the documentation overall. The assessment of these code quality criteria demonstrates IPCA's dedication to providing a reliable, efficient, and maintainable learning resource. By upholding strict guidelines and regularly evaluating the codebase, IPCA seeks to offer a dependable environment for improving the education of inexperienced programmers. Based on changing best practices and the demands of the educational technology sector, these criteria may need to be adjusted.

IX. Educational Platform Integration

The incorporation of IPCA (Intelligent Python Code Analyzer) into educational platforms signifies a revolutionary step in augmenting students' entire learning experience, specifically in classes focused on coding. IPCA creates a responsive and dynamic learning environment by giving students immediate feedback on their programming assignments through real-time code analysis. Teachers can use IPCA's automated code checking, style enforcement, and fault discovery features to easily integrate it into their curricula. This integration makes it easier for teachers to evaluate student work, making it possible for them to evaluate and comment on coding projects more quickly. The capacity of IPCA to automate the code verification process is one of its primary features. In addition to saving teachers a great deal of time, this feature guarantees that students receive assessments quickly, allowing them to revise their work and quickly learn from their mistakes. IPCA enforces coding style, which goes beyond basic code inspection. Students who are taught good programming methods from the start have a solid basis for developing clear, maintainable code. In addition to helping students with their coursework, this proactive approach to coding standards also gets them ready for real-world coding challenges. Additionally, IPCA is a useful tool for error detection. Students can improve their coding skills and learn from typical blunders by identifying faults in the code. Programmers are becoming more skilled and self-aware as a result of this iterative learning process, which IPCA facilitates. In summary, students taking coding classes benefit greatly from the incorporation of IPCA into educational systems. With features like style

enforcement, automated checking, real-time code analysis, and error detection, IPCA enables teachers and students to promote a culture of excellence in programming abilities and continual progress.

X. SECURITY CONSIDERATIONS

The IPCA server ought to use secure coding techniques since it is in charge of processing and examining student code. It is essential to have secure data storage methods, output encoding, and input validation in place to stop common web-based vulnerabilities like SQL injection and cross-site scripting (XSS) attacks. Encryption techniques should be used since IPCA processes and stores user data, protecting sensitive data both during transmission and storage. You can use Secure Sockets Layer (SSL) or its descendant Transport Layer Security (TLS) to create secure channels of communication. Code execution vulnerabilities may arise because IPCA assesses code created by students. In order to isolate and restrict student code execution and stop unwanted access to system resources, strong sandboxing techniques are used. There is a chance of code execution vulnerabilities because IPCA assesses student generated code. In order to avoid unauthorized access to system resources, it is important to implement strong sandboxing mechanisms that isolate and contain student code execution. To find and fix such vulnerabilities, the IPCA codebase and server architecture undergo routine security audits. Both manual code reviews and automated methods can improve IPCA's security posture. It is imperative that the IPCA server and related components adopt secure configuration procedures. To reduce the attack surface, this entails setting up firewalls, access controls, and server hardening techniques.

XI. SCALABILITY AND PERFORMANCE

The architecture of IPCA is made to grow smoothly with the volume and complexity of codebases seen in learning environments. By utilizing the Astroid library to manipulate abstract syntax trees (ASTs), IPCA can now analyze code hierarchies more effectively and manage multi-file projects without sacrificing performance. The IPCA's modular design makes it simple to incorporate new checks and transforms and guarantees that it may be adjusted to meet changing educational needs. Several important criteria, such as analysis time and resource consumption, are taken into account in order to quantify IPCA's performance. The application is designed to give quick feedback, so teachers and students can get the results of their analyses right away. Performance benchmarks show how effectively IPCA processes code, which makes it a useful tool in instructional settings where time is of the essence. When tokenization techniques are applied during static code analysis, IPCA can process code more quickly, which improves overall performance. During code analysis, IPCA makes a number of optimizations to increase its effectiveness. To minimize redundant computations and speed up subsequent analyses, caching mechanisms are utilized to store code that has already been analyzed. This optimization is especially helpful when handling assignments that need iterations or frequent changes to the code. Furthermore, IPCA distributes analytic jobs using parallelization techniques, which maximizes resource utilization and reduces analysis time, particularly in situations involving large-scale codebases. IPCA's performance and scalability match the various needs of both individual students and educational institutions. IPCA's architecture assures consistent performance, making it ideal for usage in both advanced software engineering and basic programming courses. This facilitates learning. Because of its versatility, the tool can be used by instructors to handle a wide range of student assignments with varying degrees of difficulty.

XII. CONCLUSION

In conclusion, the Intelligent Python Code Analyzer (IPCA) proves to be a crucial instrument for teaching programming and demonstrates a dedication to effectiveness and flexibility. By utilizing the Astroid and Tokenize libraries for code analysis, IPCA guarantees a stable educational setting. User interaction and server communication are improved by the incorporation of the Click and Requests libraries. Effective version control and documentation are facilitated by Sphinx and Git. IPCA empowers teachers by supporting several codebases and providing quick, multifile project analysis. Its incorporation into courses is expected to improve students' coding skills and provide a deeper understanding of programming. Transparency in technology is a hallmark that provides developers and researchers with information. The flexibility of IPCA to adjust to new pedagogies and technological developments is essential. The evolution of IPCA may be accelerated by the incorporation of machine learning, interoperability with learning management systems, and customization opportunities. IPCA welcomes open-source contributions and is based on community

engagement. Its success comes from motivating skilled programmers as much as from technological prowess. The article outlines how IPCA will develop further and influence programming education around the world. Receipts convey appreciation to those who have contributed. The scholarly foundation is established by references, and author biographies demonstrate knowledge. Contact details encourage cooperation and guarantee PyTA's continuous development. This final section summarizes IPCA's journey and looks forward to a dynamic programming education landscape. The story of the tool is created through group contributions, encouraging international cooperation for continued progress.

XIII. ACKNOWLEDGEMENTS

We would like to convey our deep thanks to the organization of Don Bosco Institute of Technology who assisted in the creation of this project for the insightful recommendations and inspiration they provided as they led us through the process.

We also want to express our gratitude to Don Bosco Institute of Technology for providing all the facilities needed to complete this project. Finally, as one of the team members, let me express my gratitude to every member of my group for their cooperation and support.

REFERENCES

[1] N. Somogyi and G. K¨ovesd´an, "Software Modernization Using Machine Learning Techniques," 2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI), Herl'any, Slovakia, 2021, pp. 000361- 000365, doi: 10.1109/SAMI50585.2021.9378659. keywords: Automation;Machine learning;Static analysis;Tools;Maintenance engineering;Aging;Software;code modernization;static analysis;machine learning,

[2] L. Ruggahakotuwa, L. Rupasinghe and P. Abeygunawardhana, "Code Vulnerability Identification and Code Improvement using Advanced Machine Learning," 2019 International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 2019, pp. 186-191, doi: 10.1109/ICAC49085.2019.9103400. keywords: Machine learning;Software;Tools;Feature extraction;Security;Static analysis;Neural networks;Vulnerability;Machine learning;Deep learning;CVE,

[3] E. Sultanow, A. Ullrich, S. Konopik and G. Vladova, "Machine Learning based Static Code Analysis for Software Quality Assurance," 2018 Thirteenth International Conference on Digital Information Management (ICDIM), Berlin, Germany, 2018, pp. 156-161, doi: 10.1109/ICDIM.2018.8847079. keywords: Machine learning;Data mining;Software;Tools;Prototypes;Employment;association rule mining;Machine Learning;Static Code Analysis;German Federal Employment Agency,

[4] M. Feathers, Working Effectively with Legacy Code. USA: Prentice Hall PTR, 2004.

[5] M. S. Harrison and G. H. Walton, "Identifying high maintenance legacy software," Journal of Software Maintenance, vol. 14, no. 6, p. 429–446, Nov. 2002.

[6] H. Huijgens, A. van Deursen, and R. van Solingen, "Success factors in managing legacy system evolution: A case study," in 2016 IEEE/ACM International Conference on Software and System Processes (ICSSP), 2016, pp. 96–105.

[7] A. Cockburn, Agile Software Development: The Cooperative Game (2nd Edition) (Agile Software Development Series). Addison-Wesley Professional, 2006.

[8] H. M. Sneed and T. Dombovari, "Comprehending a complex, distributed, objectoriented software system: a report from the field," in Proceedings Seventh International Workshop on Program Comprehension, 1999, pp. 218–225.

[9] Xiaomin Wu, A. Murray, M. . Storey, and R. Lintern, "A reverse engineering approach to support software maintenance: version control knowledge extraction," in 11th Working Conference on Reverse Engineering, 2004, pp. 90–99.

[10] Jelber Sayyad Shirabad, T. C. Lethbridge, and S. Matwin, “Mining the maintenance history of a legacy software system,” in International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., 2003, pp. 95–104.

[11] A. Terekhov and C. Verhoef, “The realities of language conversions,” IEEE Software, vol. 17, no. 6, pp. 111–124, 2000.

[12] mtSystems, “mtSystems documentation,” https://www.mtsystems.com/, 2020, accessed: 2020-10-05.

[13] H. M. Sneed, “Migrating pl/i code to java,” in 2011 15th European Conference on Software Maintenance and Reengineering, 2011, pp. 287–296.

[14] P. Newcomb and G. Kotik, “Reengineering procedural into objectoriented systems,” in Proceedings of 2nd Working Conference on Reverse Engineering, 1995, pp. 237–249.

[15] H. Gall and R. Klosch, “Program transformation to enhance the reuse potential of procedural software,” in Proceedings of the 1994 ACM Symposium on Applied Computing, ser. SAC ’94. New York, NY, USA: Association for Computing Machinery, 1994, p. 99–104.

[16] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” Journal of Machine Learning Research - Proceedings Track, vol. 9, pp. 249–256, 01 2010.

[17] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, “Malware Detection using Machine Learning and Deep Learning.,” Cryptography and Security (cs.CR); Machine Learning (cs.LG), vol. 11297, pp. 402-411, 2018.

[18] H. K. Dam, T. Tran, T. T. M. Pham, S. W. Ng, J. Grundy, and A. Ghose, “Automatic feature learning for predicting vulnerable software components,” IEEE Transactions on Software Engineering, Institute of Electrical and Electronics Engineers Inc., 2018.

[19] J. Jurn, T. Kim, and H. Kim, “An automated vulnerability detection and remediation method for software security,” Sustain., vol. 10, no.5, May 2018.

[20] N. Shakhovska, “Advances in Intelligent Systems and Computing,” Ukraine: Springer, Cham, 2016.
[21] I. P. L. Png, C.-Y. Wang, and Q.-H. Wang, “The Deterrent and Displacement Effects of Information Security Enforcement: International Evidence”, Journal of Management Information Systems , vol. 25, Taylor Francis, Ltd., pp. 125144.

[22] H. Nguyen, H. Dang, T. Le, and S. Le, “Innovative Mobile and Internet Services in Ubiquitous Computing - Proceedings of the 11th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2017)”, Torino, Italy, 10-12 July 2017, vol. 612, 2018.

[23] C. M. Chase and S. A. Jacob Abraham, Ernest A Emerson II Vijay K Garg Aleta M Ricciardi “Testing Concurrent Software Systems Committee.”

[24] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, “Fuzzing: State of the Art,” IEEE Trans. Reliab., vol. 67, no. 3, pp. 11991218, Sep. 2018.

[25] HT. B. Lee, Which Languages Are Bug Prone tim@vox.com, 2014.[Online]. Available: https://www.i-programmer.info/news/98-languages/11184-whichlanguages-are-bug-prone.html

[26] T. Abraham and O. De Vel, “A Review of Machine Learning in Software Vulnerability Research,”, Defence Science and Technology Group, Australia, 2017.

[27] R. Amankwah, P. K. Kudjo, and S. Y. Antwi, “Evaluation of Software Vulnerability Detection Methods and Tools: A Review,” Int. J. Comput. Appl., vol. 169, no. 8, pp. 2227, 2017.

[28] M. Dowd, J. McDonald, and J. Schuh, “An automated vulnerability detection and remediation method for software security”, Pearson Education, 2006.

[29] F. Yamaguchi, “Pattern-Based Vulnerability Discovery”, University of G¨ottingen, 2015.

[30] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, “Automatic feature learning for vulnerability prediction”,arXiv Prepr.arXiv1708.02368, 2017.

[31] T. B. Lee, “The Heartbleed Bug”, explained - Vox, tim@vox.com,2014.[Online].Available:https://www.vox.com/2014/4/8/5593654/ heartbleedexplainer-big-new-web-security-flaw-compromiseprivacy

[32] C. Fenton, “How to Check Open Source Code for Vulnerabilities DZone Security” Security Zone Analysis, 2017. [Online]. Available:https://dzone.com/articles/how-tocheck-open-sourcecodefor-vulnerabilities.

[33] R. Russell, “Automated vulnerability detection in source code using deep representation learning,” in 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018, pp.757762.

[34] K. Nishizono, S. Morisakl, R. Vivanco, and K. Matsumoto, “Source code comprehension strategies and metrics to predict comprehension effort in software maintenance and evolution tasks,” an empirical study with industry practitioners, in 2011 27th IEEE International Conference on Software Maintenance (ICSM), 2011, pp.473481.

[35] Q. Zoubi, I. Alsmadi, and B. Abul-Huda, “Study the impact of improving source code on software metrics”, in 2012 International Conference on Computer, Information and Telecommunication Systems (CITS), 2012,

[36] M. Nadeem, ”Why SonarQube: An introduction to Static Code Analysis” 2015. Online available from: https://dzone.com/articles/ why-sonarqube-1 (accessed 24 February 2018)

[37] N. DuPaul, ”Static Testing vs. Dynamic Testing” 2017. Online available from: https://www.veracode.com/blog/2013/12/statictesting-vs-dynamic-testing (accessed 24 February 2018)

[38] B. McCorkendale, T. Xue Feng, G. Sheng, Z. Xiaole, M. Jun, M. Qingchun, H. Ge Hua, and E.H.Wei Guo, “Systems and methods for combining static and dynamic code analysis.” U.S. Patent 8,726,392, issued May 13, 2014.

[39] A. Marchenko, and P. Abrahamsson, ”Predicting Software Defect Density: A case Study on Automated Static Code Analysis.” in Proceedings of 8th International Conference, XP 2007. Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Computer Science, 4536, DOI: 10.1007/978-3-540-73101-6

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, “Scikit-learn: Machine learning in Python”. Journal of machine learning research, 12(Oct) 2011, pp. 2825-2830.