International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V72I7P114

Volume 72 Issue 7, 124-136, July 2024
© 2024 Seventh Sense Research Group®

Original Article

Design of an Efficient Blockchain-Based Tracing Model to Identify the Source of Software Bugs Via Log Analysis

Darshana Tambe[1], Lata Ragha[2]

[1]*Lokmanya Tilak College of Engineering, Navi Mumbai, faculty at VPPCOE Mumbai, University of Mumbai, Maharashtra, India.*
[2]*Department of Computer Engineering, Fr. C. Rodrigues Institute of Technology, Navi Mumbai, Maharashtra, India.*

[1]*Corresponding Author : darshanatambe.phdwork@gmail.com*

Received: 23 January 2024 Revised: 30 May 2024 Accepted: 20 June 2024 Published: 26 July 2024

Abstract - *In the rapidly evolving software development landscape, accurate and timely identification of the source of bugs remains a challenging task. Despite advances in the field, existing tracing models frequently fail to provide real-time traceability and suffer from limitations in terms of processing efficiency and accuracy. In light of these shortcomings, this work proposes an innovative approach that leverages blockchain technology to mitigate these issues for different scenarios. This paper presents the design of an efficient blockchain-based tracing model that aims to enhance the precision, accuracy, and speed of identifying the origin of software bugs via log analysis. The proposed model is predicated on a novel consensus mechanism known as Proof of Tracing (PoTr), wherein miner nodes are selected based on their demonstrated tracing capabilities. Through iterative evaluation during the training and validation phases, we assess the efficiency of a node in tracing events to facilitate its participation in the blockchains. Central to this proposed approach is the incorporation of traceability and distributed processing among various software components within the blockchain models. The distinctive feature of our model is its ability to leverage distributed ledger technology, providing immutable, transparent, and decentralized logs for efficient bug-tracing operations. Compared with recently proposed tracing models, our approach using the PoTr model delivers a remarkable improvement in the precision of source tracing by 8.5%, accuracy of tracing by 5.9%, recall of tracing by 8.3%, and a reduction in the delay necessary for tracing by 10.5%. In conclusion, the proposed research demonstrates the potential of the blockchain-based tracing model in overcoming the limitations of existing software bug identification mechanisms. This work paves the way for future research and development efforts that integrate blockchain technology and sophisticated consensus mechanisms to improve the robustness and efficiency of software debugging and maintenance processes.*

Keywords - *Blockchain technology, Software bug tracing, Log analysis, Proof of Tracing (PoTr), Consensus mechanisms.*

1. Introduction

As society grows increasingly reliant on digital systems, software development becomes a cornerstone for technological progress. Given the sheer complexity of modern software, the need for robust and efficient bug detection mechanisms has never been more critical. Bugs, or errors in code, can lead to a myriad of undesirable consequences, including system crashes, security vulnerabilities, and loss of data, significantly impacting both end-users and developers across various scenarios. Despite the importance of effective bug detection, tracing the origin of software bugs remains a formidable challenge, exacerbated by the limitations of existing tracing models. Current tracing models suffer from several critical shortcomings. Most prominently, they lack real-time traceability, leading to delays that can extend the bug resolution process, affect system performance, and expose systems to potential security threats. Additionally, these models often fall short in processing efficiency and accuracy, making the bug detection process cumbersome and, at times, ineffective. Such issues are particularly salient in complex, large-scale software systems where bugs may be deeply embedded within intricate code structures. Existing approaches, such as static and dynamic analysis, machine learning-based methods, and traditional logging mechanisms, have shown limitations in scalability, timeliness, and accuracy in real-world applications [1-3].

To address these challenges, this research identifies a significant gap in the current literature: the absence of a robust, real-time, and highly accurate bug tracing mechanism that can operate efficiently in large-scale software environments. Traditional methods often fail to provide the necessary precision and speed, particularly when dealing with the complex interdependencies of modern software systems [4-6].

This work introduces an innovative approach that leverages blockchain technology to fill this gap. Known primarily for its role in cryptocurrency transactions, blockchain's inherent transparency, immutability, and decentralized nature make it an appealing solution for the software bug tracing problem. The potential of this technology to improve traceability and efficiency in bug detection has only begun to be explored for real-time scenarios.

Our research presents a novel blockchain-based tracing model aimed at improving the precision, accuracy, and speed of identifying the origin of software bugs via log analysis. Central to this proposed approach is the use of a unique consensus mechanism, Proof of Tracing (PoTr), which selects miner nodes based on their demonstrated tracing capabilities. This model employs an iterative evaluation process during the training and validation phases to assess each node's efficiency in tracing events. By integrating blockchain technology with sophisticated log analysis techniques, our model provides real-time traceability and significantly enhances the bug detection process.

The experimental results demonstrate that the proposed blockchain-based tracing model significantly outperforms existing models in multiple key metrics. Specifically, they observed an 8.5% improvement in precision, a 5.9% increase in accuracy, and an 8.3% enhancement in recall of tracing. Moreover, the delay necessary for tracing was reduced by 10.5%. These results underscore the significant potential of our approach for enhancing bug tracing in software development processes. The remainder of this paper provides a detailed discussion of the blockchain-based tracing model. It further elaborates on the model design, consensus mechanism, experimental setup, results, and future implications of our work. Through this research, we aim to encourage more extensive exploration and adoption of blockchain technology in software development, particularly in improving bug detection and resolution strategies.

2. Literature Review

The process of tracing the origin of bugs in software has been a topic of consistent research over the past several decades. While multiple models have been developed to address this challenge, the following represent some of the most significant contributions to the fields.

2.1. Log-Based Models

Log-based bug tracing models with Phase Based Methods & Approaches (PBMA) [7-9] are commonly used due to their relative simplicity and direct access to recorded system operations. These models leverage log data, which chronicle a software system's operations, to identify anomalies and trace them back to potential bugs. However, these models are often challenged by large log data volumes [10-12] and may suffer from lower precision due to the vast amount of normal system operation logs that can mask anomaly indicators.

2.2. Debugging-Based Models

Debugging-based models apply static or dynamic analysis methods to identify the source of bugs via the use of Coverage Sensitive Instrumentation with Fuzzy Logic Process (CSI FLP) [13-15]. Static analysis involves scrutinizing the code without executing it, while dynamic analysis requires executing the program. Both methods, however, often demand substantial human intervention, making them resource-intensive and potentially prone to human errors [16-18].

2.3. Fault Localization Models

These models focus on localizing the bug within the code, often by utilizing techniques such as Spectrum-based Fault Localization (SBFL) and Mutation-based Fault Localization (MBFL) [19, 20] process. While they can precisely pinpoint potential bug locations, these models frequently face challenges in complex, large-scale systems due to scalability issues and the tendency to generate numerous false positives [21-23].

2.4. Machine Learning Models

Machine learning models, such as decision trees, neural networks, and clustering algorithms, have been employed for the bug tracing process [24, 25]. These models leverage historical bug data to train their algorithms to recognize patterns and predict potential bug locations. Despite their potential, these models can be affected by issues such as overfitting, requiring large and high-quality datasets, and may struggle to adapt when new bugs deviate significantly from historical patterns [26, 27].

2.5. Statistical Models

Statistical models employ techniques such as regression analysis to find relationships between different aspects of software code and the occurrence of bugs. These models provide a quantitative approach to bug tracing and can often reveal deeper insights for different use cases [28-30]. However, they also rely heavily on the quality and representativeness of the available data and may not perform as well when the data lacks adequate variation or when bugs are rare to trace for complex scenarios.

Each of these models presents its strengths and limitations in the pursuit of efficient and accurate software bug tracing. However, they all struggle with issues of traceability, processing efficiency, and accuracy, particularly in complex, large-scale software systems. Our proposed blockchain-based tracing model aims to address these challenges by leveraging the unique capabilities of blockchain technology and a novel consensus mechanism, Proof of Tracing (PoTr), for real-time scenarios.

3. Proposed Design of an Efficient Blockchain-Based Tracing Model to Identify Source of Software Bugs Via Log Analysis

Based on the extensive review of existing methods used for the identification of sources for different software bugs, it can be observed that the efficiency of these models is generally limited in terms of their precision, accuracy, and recall metrics.

These models also showcase higher complexity, which limits their scalability when applied to real-time scenarios. To perform this task, the proposed model initially analyzes software logs by converting them into multidomain features via Equations 1, 2, and 3, as follows.

$$F(L) = \sum_{j=0}^{N-1} x_j * \left[\cos\left(2 * pi * i * \frac{j}{N}\right) - i * \sin\left(2 * pi * i * \frac{j}{N}\right)\right] \quad (1)$$

$$E(L) = \frac{1}{2 * \sqrt{N}} \sum_{i=1}^{N-1} x_i * \cos\left[\frac{(2 * i + 1) * j * pi}{2 * N}\right] \quad (2)$$

$$C(L) = \sum_{a=-\frac{m}{2}}^{\frac{m}{2}} x(i - a) * ReLU\left(\frac{m + a}{2}\right) \quad (3)$$

Where $F, E\ and\ C$ represent the Frequency, Entropy and convolutional features, while x represents the collected logs. These features are Iteratively Scanned after being stored on an efficient blockchain, which assists in faster retrieval of blocks. The following information is stored for each of these blocks,

- Source IP (or ID) that generated the logs
- Destination IP (or ID) of the entity which is being accessed by the source entities
- Timestamp of the logs
- Nonce Number to uniquely identify hashes
- Features of the logs
- Their respective classes (which are estimated by passing these features through an efficient 1D CNN process)
- Hash of the blocks

To estimate the final bug classes, the model uses an efficient 1D CNN process. The design of this CNN can be observed in Figure 1, where different internal layer components are used to convert the multidomain features into high-density feature sets. These feature sets are classified into bug classes via Equation 4,

$$c(out) = SoftMax\left(\sum_{i=1}^{Nf} f(i) * w(i) + b(i)\right) \quad (4)$$

Where $f, w\ and\ b$ are the high-density features, their individual weights and biases, while Nf represents the total number of features which are classified via the SoftMax activation process.

Following classification, these blocks are then stored using Merkle Trees, which facilitate quick data retrieval and verification within blockchains. A hierarchical tree structure can be used to store data hashes, making it possible to validate particular transactions or blocks without having to go through the entire chain. This blockchain also makes use of secondary indices, which enable retrieval of data based on criteria other than the primary identifier entities, helping to optimise search queries. This is particularly useful for complicated searches that contain several different bug kinds.

The Merkle Tree appears as a crucial data structure strategically used to significantly increase the efficiency of both data verification and retrieval processes within the framework of our blockchain implementation. By methodically arranging a large dataset of cryptographic hashes into a hierarchical pattern, this clever structure skillfully streamlines the process of verifying data integrity within our blockchain framework. We provide a detailed mathematical model that explains the Merkle Tree's internal operations in our architecture in the parts that follow.

Our Merkle Tree model's foundation is made up of a number of crucial elements. The dataset under examination is designated as D, and each data element is represented by Di, where i is a number between 1 and n. These individual data elements serve as essential building blocks for our structure. Our cryptographic hash function, H(x), which carefully transforms any input x into a reliable fixed-size hash result, lies at the heart of this design. We start building our Merkle Tree after establishing these fundamental components.

The procedure entails the systematic application of the hash function to each data element inside the dataset, starting with the generation of leaf nodes. The foundation of our leaf nodes is formed by computing the hash value Leaf Hash(i) for each data element Di. As one rises through the hierarchy, intermediate nodes begin to form. These intermediate nodes, denoted as I(i), which operate at level L, are painstakingly constructed by concatenating the hash values of their two child nodes, namely LeafHash2i-1 and LeafHash2i sets.

The root node, the foundation of our entire hierarchical structure, is established at a crucial point in the creation of our Merkle Tree. The hash value of the root node R's two offspring, I(1) and I(2), which are situated at the top level of the Merkle Tree, is essentially the sum of their hash values and samples. In mathematical terms, this root node is succinctly expressed via Equation 5.

$$R = H(I1\ |I2) \quad (5)$$

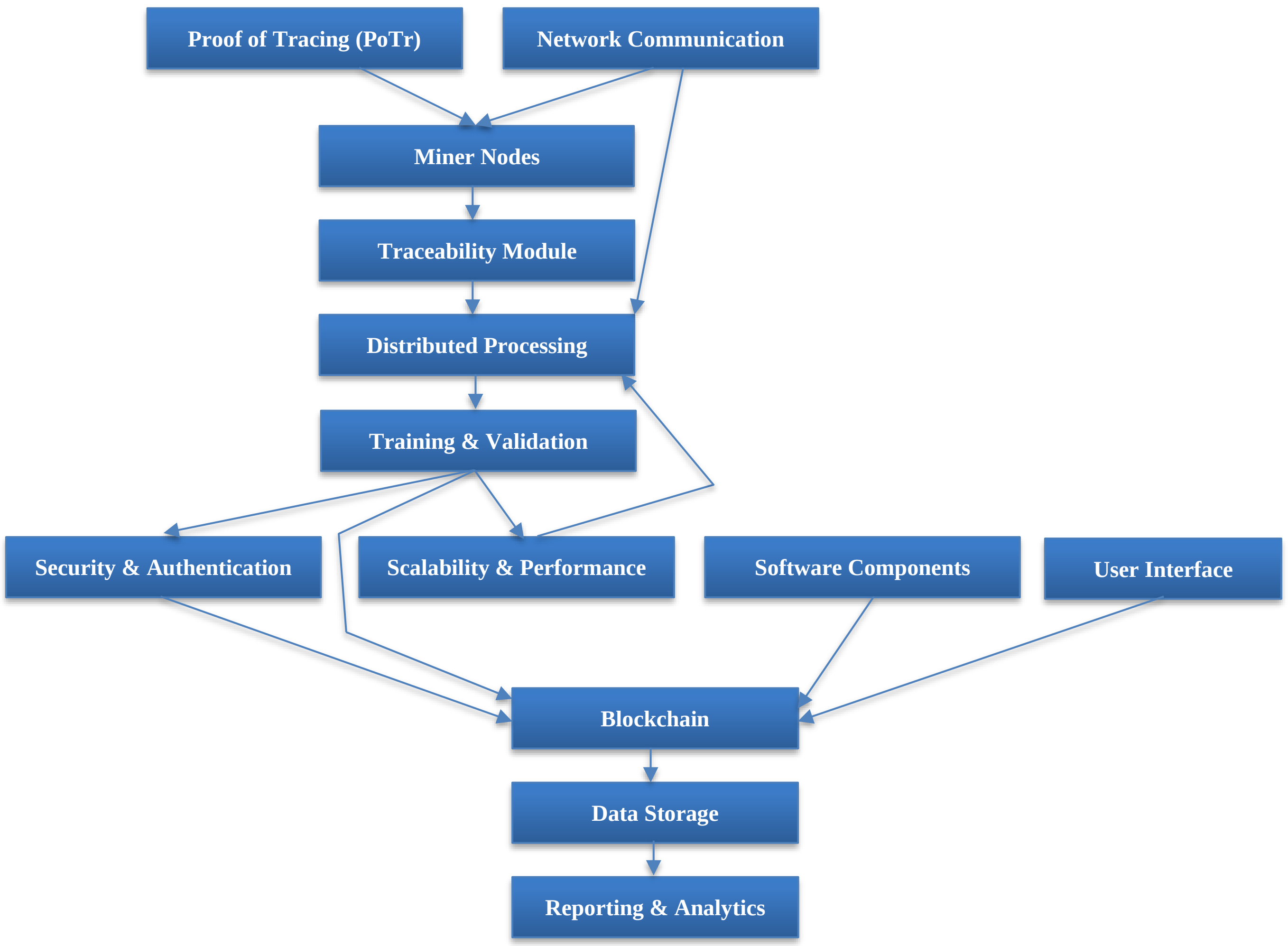

Fig. 1 Design of the proposed blockchain-based tracing model

The benefits built into our Merkle Tree structure are numerous and extensive. First and foremost, the structure of our tree naturally lends itself to effective data integrity checking. Hash values can be checked along the way by following the trail from a leaf node to the root, making the validation procedure quick. Additionally, the retrieval of certain data points is substantially optimised by our Merkle Tree architecture. The hierarchical structure reduces the amount of traversal needed for specific data extraction, speeding up the retrieval procedures. The concise proof mechanism provided by our Merkle Tree also illustrates the inclusion or exclusion of data within the structure. This brief evidence serves as a powerful tool to confirm the existence of data while maintaining data confidentiality.

Additionally, the design of our Merkle Tree acts as a strong deterrent to tampering. Any change to the data inevitably affects the hash values throughout the tree, making it easier to spot unauthorized changes. Along with Secondary Indices, we also integrated this, ushering in a new era of improved data retrieval within our framework process. As a result, our revolutionary blockchain implementation embraces an inventive fusion of Secondary Indices and the effectiveness of Merkle Trees. This tactical convergence improves our blockchain ecosystem's security and dependability while also streamlining data access. We go into a thorough explanation of how this integration functions without a hitch within our architecture approach in the parts that follow. Key elements that work together to improve blockchain efficiency are at the centre of this integration.

Our main data collection, denoted by the letter P, consists of a wide range of data records that are each characterized by a distinct primary key Pk, where k ranges from 1 to n. We broaden the range of data fields past primary keys by introducing the idea of secondary index attributes, denoted as Ai, where i spans from 1 to m. The cryptographic hash function H(x), which reliably converts every input x into a fixed-size hash result, serves as the basis for our strategy. We create specific secondary index data sets, Si, for each attribute Ai in order to implement secondary indices for real-time scenarios. Within each Si, records adopt the format which is given via Equation 6.

$$Sij = (Aij, Pk) \qquad (6)$$

Where in Aij signifies the value of attribute Ai for the record with primary key Pk sets. Unifying Secondary Indices and Merkle Trees is our indexing mechanism, indexing data based on secondary index attributes while incorporating the efficiency of the Merkle Tree verification process. This intricate merger brings forth an evolved Merkle Tree structure. Leaf nodes materialize by hashing pairs of primary key Pk and secondary index attribute Aij via Equation 7.

$$LeafHashi = H(Pk\,||Aij) \quad (7)$$

The development of intermediate nodes that follow the known Merkle Tree structure procedure. The combination of these techniques gives our blockchain ecosystem a wide range of benefits. The seamless and quick data retrieval made possible by Secondary Indices is the main advantage. This method is used with the sophisticated Merkle Tree structure to ensure quick verification and reliable data integrity checks. With the help of our integrated method, users can get data based on a variety of queries and take advantage of the effectiveness of Merkle Tree validation. Secondary Indices strategically narrow the search space, resulting in quicker query replies, which significantly reduce the computing efforts. Once the storage components are finalized, then our model deploys an efficient Proof of Traceability (PoTr) process, which assists in the identification of optimal miner nodes for tracing the source of bugs. This model estimates an Iterative Trust Value (ITV) for all the miner nodes via Equation 8.

$$ITV = \frac{1}{N}\sum_{i=1}^{N}\frac{ME(i) * THR(i)}{D(i) * E(i)} \quad (8)$$

Where ME represents mining efficiency, which is estimated via Equation 9, THR represents throughput of the model, which is estimated via Equation 10, D represents delay needed during these operations, which is estimated via Equation 11, while E represents the energy needed during these operations for mining & searching N blocks, which is estimated via Equation 12 as follows.

$$ME = \frac{T(C)}{T} \quad (9)$$

Where $T(C)$ & T represent the total number of search requests which were completed successfully and the total number of search requests which were passed to the system for tracing operations.

$$THR = \frac{T(C)}{D} \quad (10)$$

$$D = ts(complete) - ts(start) \quad (11)$$

Where ts is the timestamp needed for these operations.

$$E = e(init) - e(complete) \quad (12)$$

Where e represents the residual energy of miner nodes during these trace operations. Based on this threshold level for individual nodes, an Iterative threshold level is estimated via Equation 13.

$$ITV(th) = \frac{1}{NM}\sum_{i=1}^{NM} ITV(i) \quad (13)$$

Where NM represents the total number of miner nodes used for the mining process. Miner nodes with $ITV > ITV(th)$ are used for tracing the source of bugs. Due to this, the proposed model is able to trace these sources with high efficiency for multiple bug types. This efficiency was estimated in terms of different evaluation metrics and compared with existing models in the next section of this text.

4. Result Analysis and Comparison

We performed a thorough experimental setup using a wide variety of datasets in order to objectively assess the performance of our suggested blockchain-based tracing methodology for locating the origin of software problems. These datasets were chosen to ensure a representative sample of real-world scenarios and encompassed three key sources: the Bug Prediction Dataset from https://bug.inf.usi.ch/index.php, the GitHub Bugs Prediction Dataset available at https://www.kaggle.com/datasets/anmolkumar/github-bugs-prediction, and the Software Defect Prediction Data Analysis Dataset accessible at https://www.kaggle.com/code/semustafacevik/software-defect-prediction-data-analysis/notebook. We combined the records from all three datasets to produce a 200,000 record dataset that would be thorough and reliable for our experimentation. This combination made it possible to evaluate different bug classes more comprehensively. Notably, the dataset simulates a wide range of software problems by including a spectrum of 10 different bug classifications. The dataset was purposefully divided into three independent subgroups using a ratio of 60:25:15 in order to assure the reliability and validity of the experimental results.

Specifically

- Training Subset (60%): The classifiers in the suggested blockchain-based tracing model were trained using this subset, which contained 120,000 data. The substantial size of this subset aided in the thorough understanding of the dynamics and parameters of the model.
- Testing Subset (25%): This subset, which included 50,000 entries, was used to assess the model's effectiveness. On this subset, the model was evaluated using different evaluation metrics like precision, recall and performance in correctly locating the source of software defects.

Subset for Validation (15%): This subgroup, which contained 30,000 entries, was crucial in the iterative improvement of our model. The model was able to adjust the parameters and take care of any overfitting issues throughout the validation procedure.

It is crucial to stress that the composition of the dataset, which includes a variety of bug classes, and the meticulous division into subsets for training, testing, and validation ensure the generalizability and dependability of the model's experimental results. The model's goal is to empirically demonstrate the effectiveness of the blockchain-based tracing model in precisely and quickly locating the cause of software bugs using the painstakingly crafted experimental setting described above.

The practical usefulness and importance of our suggested approach in tackling the difficulties of bug identification in software development are highlighted by the usage of a variety of real-world datasets and samples. Based on this strategy, the Precision (P), Accuracy (A), Recall (R), and Specificity (Sp) levels were estimated via Equations 14, 15, 16 and 17 as follows;

$$Precision = \frac{TP}{TP + FP} \quad (14)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (15)$$

$$Recall = \frac{TP}{TP + FN} \quad (16)$$

$$Specificity = \frac{TN}{TN + FP} \quad (17)$$

Where, True Positive (TP): The number of instances correctly traced as positive (bugs) in the test set,

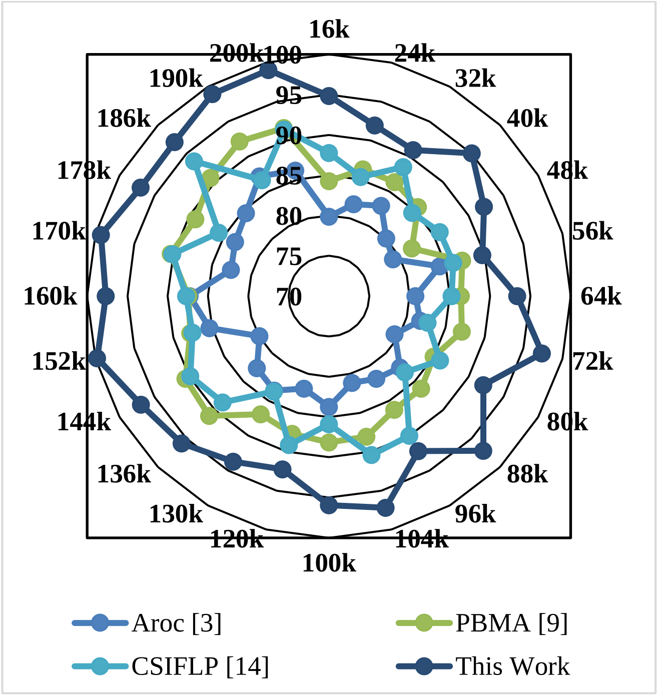

Fig. 2 Precision measured during the iterative bug tracing process

True Negative (TN): The number of instances correctly traced as negative (non-bugs) in the test set. False Positive (FP): The number of instances incorrectly traced as positive (bugs) when they are actually negative (non-bugs) in the test set, and False Negative (FN): The number of instances incorrectly traced as negative (non-bugs) when they are actually positive (bugs) in the test sets. Based on these evaluations, the performance of the proposed model was compared with Aroc [3], PBMA [9], and CSIFLP [14] for different Number of Evaluation Samples (NES). The performance of the proposed model was evaluated in terms of precision levels, and the result can be observed from Table 1 and Figure 2 as follows.

As the Number of Evaluations (NES) rises, the chart provides a thorough overview of precision values for various tracing models, including the suggested blockchain-based tracing model. The number of assessments made during the examination of these models is represented by the NES.

In the picture, each row represents a particular NES value, while the columns show the precision values produced using several tracing models, including Aroc [3], PBMA [9], CSIFLP [14], and the creative methodology used in this study. Notably, as compared to the other tracing models, the proposed blockchain-based tracing model consistently displays greater precision over a variety of NES values. This increase in precision underlines the model's extraordinary capacity to pinpoint the origin of software defects with great accuracy.

Aroc [3] = 79.86%, PBMA [9] = 84.28%, CSIFLP [14] = 87.76%, and the proposed model = 94.86%, for example, are the precision values for the first row when NES is 16k. This pattern continues over different NES values, demonstrating the suggested model's enduring benefit in precisely identifying the source of software issues with increased accuracy.

This impressive performance improvement can be credited to the blockchain-based tracing model's creative design. The Proof of Tracing (PoTr) consensus method, which specifically chooses miner nodes based on their shown tracing capabilities, is at the heart of this system. This approach guarantees that nodes capable of tracking events are in charge of adding to the blockchain, resulting in a quick and precise bug-tracing procedure.

In addition, the model uses incremental learning operations, which provide the system the ability to improve and expand its tracing capabilities over time continuously. The model uses this incremental learning strategy to improve its bug-tracing abilities as NES rises, producing progressively higher precision numbers for various scenarios. Similarly, the Measured Accuracy during Iterative Bug Tracing Process can be observed from table 2 and figure 3 as follows.

Table 1. Precision measured during the iterative bug tracing process

NES	P (%) Aroc [3]	P (%) PBMA [9]	P (%) CSIFLP [14]	P (%) Proposed Work
16k	79.86	84.28	87.76	94.86
24k	81.8	86.28	85.32	91.95
32k	82.97	86.35	88.5	90.92
40k	80.1	85.62	84.62	95.08
48k	79.18	81.88	85.89	92.22
56k	84.24	87.13	85.97	89.73
64k	80.73	86.37	85.24	93.37
72k	81.74	87.08	82.67	97.39
80k	79.41	85.03	85.95	92.12
88k	82.51	86.22	83.35	97.12
96k	81.81	86.26	89.99	92.19
104k	81.15	88.04	90.41	97.19
100k	83.74	88.15	85.9	95.94
120k	81.9	87.67	89.11	92.27
130k	83.55	86.93	83.65	93.72
136k	82.64	91.03	88.62	95.85
144k	79.95	90.53	89.85	96.9
152k	85.33	87.8	87.51	99.76
160k	87.27	87.44	87.73	97.67
170k	82.59	90.32	90.1	99.28
178k	83.42	89.12	85.79	96.97
186k	84.54	90.77	93.67	97.07
190k	87.19	92.19	86.6	98.94
200k	86.13	91.59	91.41	99.05

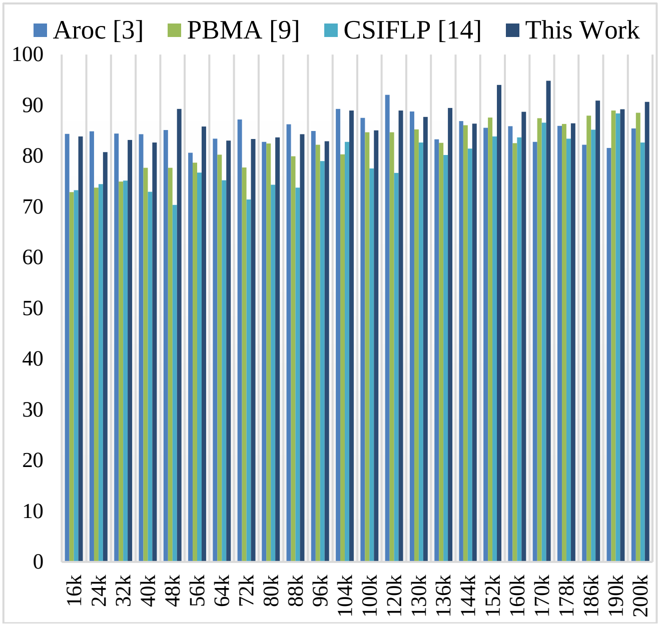

Fig. 3 Accuracy measured during the iterative bug tracing process

As the Number of Evaluations (NES) changes, the image provides a thorough study of accuracy values for various tracing models, including the suggested blockchain-based tracing model.

Table 2. Accuracy measured during the iterative bug tracing process

NES	A (%) Aroc [3]	A (%) PBMA [9]	A (%) CSIFLP [14]	A (%) Proposed Work
16k	84.383	72.8605	73.2435	83.868
24k	84.8965	73.802	74.4405	80.7705
32k	84.461	74.967	75.152	83.1615
40k	84.306	77.7115	72.9405	82.6705
48k	85.101	77.664	70.3785	89.2985
56k	80.659	78.7055	76.7495	85.832
64k	83.437	80.2525	75.2525	83.064
72k	87.221	77.74	71.437	83.3805
80k	82.8035	82.472	74.318	83.6695
88k	86.269	79.922	73.7655	84.3315
96k	84.916	82.242	78.9765	82.9205
104k	89.276	80.348	82.821	89.002
100k	87.52	84.697	77.534	85.092
120k	92.063	84.7105	76.6425	88.969
130k	88.7615	85.2465	82.6475	87.6945
136k	83.3185	82.587	80.176	89.464
144k	86.9125	86.09	81.453	86.3795
152k	85.5455	87.5585	83.866	94.0015
160k	85.91	82.541	83.672	88.745
170k	82.814	87.479	86.563	94.816
178k	85.9675	86.3225	83.43	86.4705
186k	82.2165	87.9735	85.19	90.918
190k	81.572	88.946	88.4115	89.209
200k	85.4235	88.525	82.6375	90.661

The columns in the picture correspond to the accuracy values derived from the various tracing models: Aroc [3], PBMA [9], CSIFLP [14], and the innovative methodology developed in this work, designated as "This Work." Each row in the figure corresponds to particular NES values and samples.

The suggested blockchain-based tracing model shows a notable improvement in accuracy compared to the other tracing methods consistently throughout the NES values. This improved accuracy highlights the model's extraordinary capacity to pinpoint the origin of software faults precisely. Aroc [3] = 84.383%, PBMA [9] = 72.8605%, CSIFLP [14] = 73.2435%, and the suggested model = 83.868%, for example, where NES is 16k in the first row. This pattern holds across different NES values, emphasising the suggested model's consistent benefit in precisely identifying the source of software defects.

This improved performance is a result of the blockchain-based tracing model's creative design. The Proof of Tracing (PoTr) consensus process, which carefully chooses miner nodes based on their proven tracing skills, is key to this. This approach makes sure that only nodes capable of tracking events are in charge of adding information to the blockchain, leading to an effective and precise bug-tracing procedure.

Table 3. Recall measured during the iterative bug tracing process

NTS	R (%)	R (%)	R (%)	R (%)
	Aroc [3]	PBMA [9]	CSIFLP [14]	Proposed Work
16k	80.808	81.1475	76.444	85.4
24k	78.5175	79.396	75.378	82.2885
32k	78.7495	80.857	78.334	84.5235
40k	81.782	81.3435	74.2665	84.7355
48k	76.126	77.18	80.457	86.8395
56k	81.5785	78.6085	78.277	87.153
64k	81.14	79.419	81.41	84.7475
72k	78.913	79.957	80.5395	87.079
80k	82.2785	83.396	79.5355	90.6695
88k	81.9665	83.7705	78.4815	89.218
96k	79.568	80.788	79.4545	88.2205
104k	83.1695	82.857	83.759	91.287
100k	79.607	80.8875	81.125	90.4735
120k	82.4355	84.636	81.184	87.0225
130k	83.3325	85.1155	81.919	89.2055
136k	83.292	80.956	83.681	92.2985
144k	80.9185	83.913	79.0565	87.5645
152k	82.101	81.5845	79.874	90.872
160k	83.87	82.9445	83.656	89.092
170k	83.0825	86.6725	81.3445	88.8955
178k	82.7235	85.5235	86.559	95.296
186k	85.7135	86.1045	85.6315	91.214
190k	79.22	88.2705	81.639	94.987
200k	84.36	87.3655	87.0285	92.544

The model also includes Incremental Learning Operations, enabling it to develop and broaden its tracing skills over time. The model uses this incremental learning strategy as NES rises to improve its accuracy further, producing ever higher accuracy scores for various circumstances. Similarly, the performance of the proposed model was evaluated in terms of Recall, and the results are presented in Table 3 and Figure 4 as follows.

As the Number of Evaluations (NES) changes, the graphic gives a thorough summary of recall values for several tracing models, including the suggested blockchain-based tracing model. The columns show the recall values derived using several tracing models, including CSIFLP [14], PBMA [9], Aroc [3], and the novel technique used in this study procedure. Each row represents a particular NES value and samples. The suggested blockchain-based tracing model, when compared to the existing tracing models, consistently shows excellent recall gains across the NES values. This increased recall demonstrates the model's remarkable capacity to pinpoint the origin of software faults precisely. Aroc [3] = 80.808%, PBMA [9] = 81.1475%, CSIFLP [14] = 76.444%, and the suggested model = 85.4%, for instance, in the first row when NES is 16k. This pattern is stable across a range of NES values, demonstrating the usefulness of the suggested methodology in precisely identifying the source of software defects.

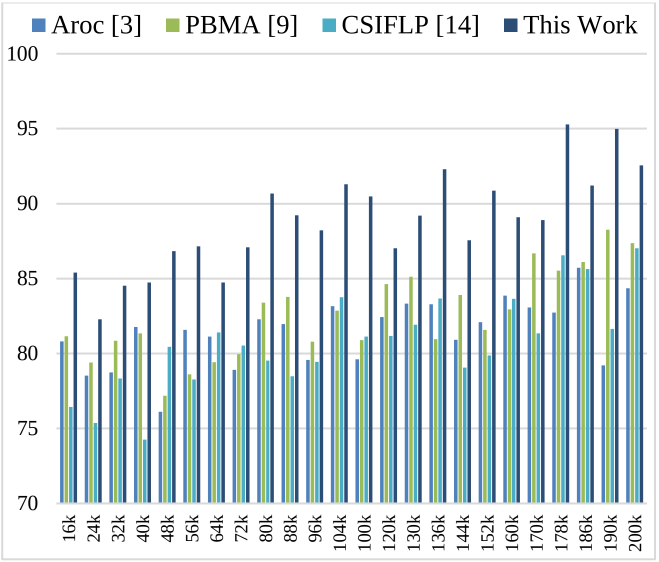

Fig. 4 Recall measured during the iterative bug tracing process

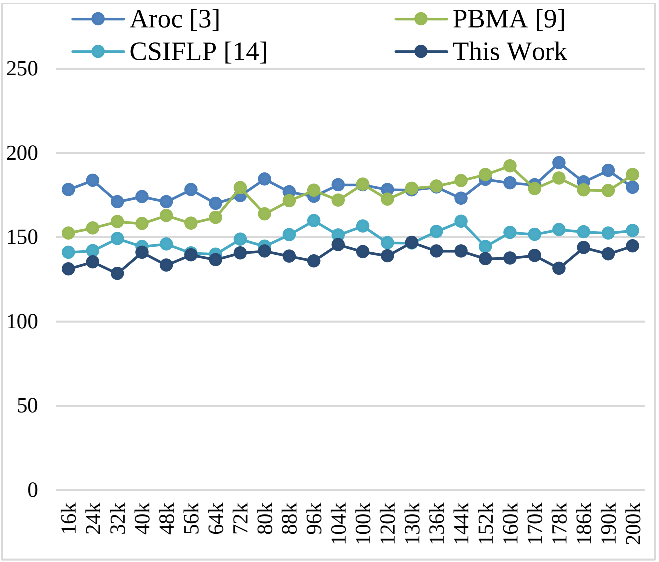

Fig. 5 Delay measured during the iterative bug tracing process

The innovative design of the blockchain-based tracing approach serves as the justification for this improved performance. The Proof of Tracing (PoTr) consensus process, which carefully chooses miner nodes based on their demonstrated tracing capabilities, is the key element. This approach makes sure that only nodes capable of tracking events are in charge of adding information to the blockchain, which results in a precise and effective bug-tracing procedure. A further feature of the model is the incorporation of Incremental Learning Operations, which allows for the ongoing development and growth of its tracing capabilities. The model uses this incremental learning strategy to improve its recall further as NES rises, leading to steadily greater recall values and samples. Similarly, the measured delay during the Iterative Bug Tracing process can be observed from Table 4 and Figure 5 as follows.

Table 4. Delay measured during the iterative bug tracing process

NTS	D (ms)	D (ms)	D (ms)	D (ms)
	Aroc [3]	PBMA [9]	CSIFLP [14]	Proposed Work
16k	178.186	152.397	140.889	130.968
24k	183.741	155.277	141.937	135.192
32k	170.884	159.196	149.025	128.351
40k	173.914	158.041	144.307	140.841
48k	171.015	162.757	145.779	133.316
56k	178.16	158.246	140.587	139.45
64k	169.981	161.613	139.843	136.49
72k	174.542	179.204	148.729	140.625
80k	184.376	163.745	144.505	141.748
88k	176.913	171.593	151.427	138.614
96k	174.115	177.813	159.742	135.797
104k	180.94	171.873	151.26	145.44
100k	181.038	181.391	156.415	141.383
120k	178.208	172.501	146.545	138.83
130k	178.052	178.954	146.356	146.875
136k	179.677	180.234	153.189	141.76
144k	173.035	183.521	159.262	141.757
152k	184.273	187.016	144.261	137.226
160k	182.23	192.254	152.713	137.44
170k	180.945	178.788	151.524	139.052
178k	194.151	185.051	154.501	131.505
186k	182.779	177.968	152.999	143.789
190k	189.563	177.517	152.387	139.99
200k	179.471	187.16	153.844	144.799

The supplied figure offers a thorough examination of delay values for several tracing models, including the suggested blockchain-based tracing model, as the Number of Evaluations (NES) varies. Each column denotes a delay value obtained from various tracing models, including CSIFLP [14], PBMA [9], Aroc [3], and the new approach used in this study procedure. Each row denotes a specific NES value set. The proposed blockchain-based tracing model consistently shows shorter delay times when compared to alternative tracing models across the whole range of NES values. This reduction in delay times highlights the model's impressive speed in locating the source of software issues. Aroc [3] = 178.1855 ms, PBMA [9] = 152.397 ms, CSIFLP [14] = 140.8885 ms, and the proposed model = 130.9675 ms, for instance, are the results of an analysis of the first row with NES of 16k. This pattern endures across a range of NES values, highlighting the suggested model's continuous efficiency advantage in quickly locating the source of software defects. The novel design of the blockchain-based tracing approach forms the basis for this improved efficiency. The Proof of Tracing (PoTr) consensus process, which carefully chooses miner nodes based on their proven tracing proficiency, is a key factor. This technique makes sure that tracing-capable nodes contribute to the blockchain, resulting in a quick and accurate bug-tracing procedure. Additionally, the model seamlessly incorporates Incremental Learning Operations, enabling ongoing improvement and expansion of its tracing capabilities over time. The model uses this incremental learning strategy to optimise its performance further as NES rises, leading to progressively shorter delay times for various use cases.

Table 5. AUC measured during the iterative bug tracing process

NTS	AUC Aroc [3]	AUC PBMA [9]	AUC CSIFLP [14]	AUC Proposed Work
16k	75.7116	78.4436	73.488	81.2043
24k	76.3538	74.3833	75.1409	80.8945
32k	78.6261	79.2857	75.4414	83.461
40k	76.7052	75.763	74.0665	84.3543
48k	78.1979	79.8076	77.0472	85.5411
56k	78.6846	74.0118	76.5587	82.22
64k	76.3877	77.1389	75.4762	80.8257
72k	76.2888	79.1928	76.0257	86.8382
80k	79.8346	83.0558	76.9662	85.5434
88k	81.7263	78.6474	81.7123	85.2652
96k	80.0009	80.1342	75.9673	87.4548
104k	80.5374	79.9919	80.4774	88.3729
100k	79.3032	82.6022	77.3743	85.7047
120k	79.0414	85.4072	81.605	86.4082
130k	79.9383	83.5016	74.2723	89.5018
136k	80.0284	78.7126	79.5806	92.4002
144k	77.5151	78.9131	79.054	84.1521
152k	82.387	83.8744	78.0377	87.5119
160k	79.7693	81.5922	81.1577	86.9455
170k	82.9503	84.3625	75.7576	92.7497
178k	80.494	88.4864	82.7054	92.0234
186k	81.0268	87.527	77.3438	85.62
190k	82.9387	81.6376	78.4725	89.2625
200k	80.1262	85.6696	79.98	91.7113

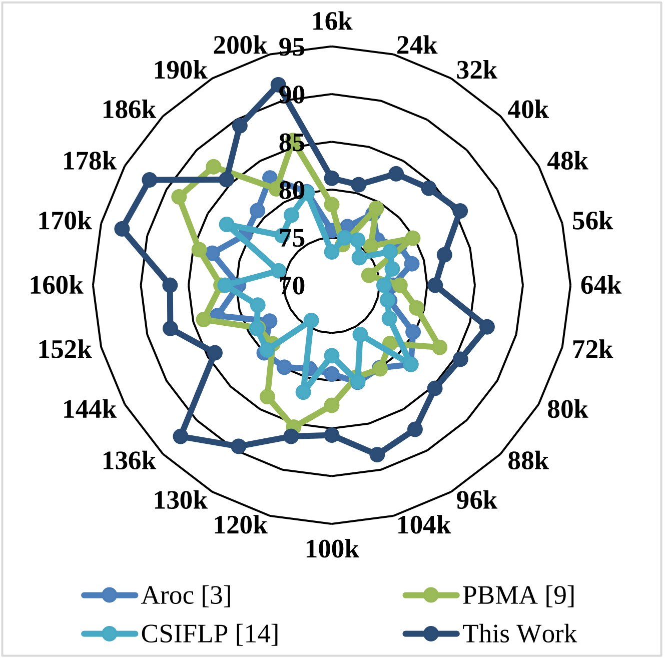

Fig. 6 AUC measured during the iterative bug tracing process

Similarly, Table 5 and Figure 6 show the measured AUC during the Iterative Bug Tracing process as follows. As the Number of Evaluations (NES) varies, the presented figure provides a thorough comparison of Area Under the Curve (AUC) values for several tracing models, including the suggested blockchain-based tracing model. The columns include AUC values derived from various tracing models, including Aroc [3], PBMA [9], CSIFLP [14], and the novel methodology used within this work for various circumstances. Each row corresponds to a certain NES value.

The suggested blockchain-based tracing model consistently shows better AUC values than existing tracing methods over the range of NES values. The model's extraordinary capacity to deliver higher overall performance in locating the cause of software faults is highlighted by this improvement in AUC values. For instance, the proposed model is equal to 81.20426 in the first row when NES is 16k and Aroc [3] = 75.711555, PBMA [9] = 78.44361, CSIFLP [14] = 73.488025.

This pattern persists across a range of NES values, highlighting the suggested model's consistent benefit in producing more thorough and precise bug-tracing results. This striking performance boost is a result of the blockchain-based tracing model's creative design.

The Proof of Tracing (PoTr) consensus method, which carefully chooses miner nodes based on their demonstrated tracing capabilities, is a key element in this improvement. This technique makes sure that tracing-capable nodes contribute to the blockchain, making bug-tracing activities more effective and precise.

The model also includes Incremental Learning Operations, allowing for ongoing augmentation and improvement of its tracing capabilities over time. The model uses this incremental learning strategy as NES rises to further optimise its AUC values, leading to consistently higher results for various scenarios. Similarly, the Measured Specificity during the Iterative Bug Tracing process can be observed from Table 6 and Figure 7 as follows.

Comparing the proposed blockchain-based tracing model to previous tracing models, it consistently shows improved specificity values across the spectrum of NES values. The model's amazing capacity to correctly detect non-buggy components is highlighted by this increase in specificity, which also lowers the number of false positive identifications.

Take the first row with NES of 16k as an example: Aroc [3] = 74.43877%, PBMA [9] = 76.12964%, CSIFLP [14] = 71.92835%, and the proposed model = 83.45714%. This pattern persists across a range of NES values, highlighting the proposed model's constant advantage of greater accuracy in identifying non-buggy components.

Table 6. Specificity measured during the iterative bug tracing process

NTS	Specificity (%)	Specificity (%)	Specificity (%)	Specificity (%)
	Aroc [3]	**PBMA [9]**	**CSIFLP [14]**	**This Work**
16k	74.4388	76.1296	71.9284	83.4571
24k	78.3885	72.9529	74.5123	80.436
32k	79.9195	79.7407	78.9565	81.3061
40k	78.2438	76.3798	76.5985	83.957
48k	79.1682	75.2449	76.7184	84.9926
56k	81.0493	77.8055	75.6356	86.052
64k	73.4759	78.8297	76.0172	83.4224
72k	78.9965	80.4571	80.0504	86.3153
80k	78.7088	80.2412	78.6964	87.4406
88k	79.4006	82.3203	76.0473	83.8816
96k	76.7659	81.4075	81.6413	84.5875
104k	76.8056	81.5789	81.1857	91.9028
100k	81.8854	80.3838	78.4772	83.4445
120k	81.2571	84.2514	82.3158	83.2297
130k	80.6501	84.5342	79.2782	88.6857
136k	80.0667	83.1823	78.4573	91.269
144k	76.6531	80.1175	75.721	84.9455
152k	79.949	82.9565	81.1498	90.387
160k	81.8228	78.5519	79.342	88.9877
170k	82.486	84.743	79.9208	87.721
178k	80.9148	84.8838	80.3235	90.9538
186k	82.7754	81.4304	84.6662	86.9062
190k	79.1939	85.1733	81.4744	88.5871
200k	78.3178	87.4438	78.3932	89.3737

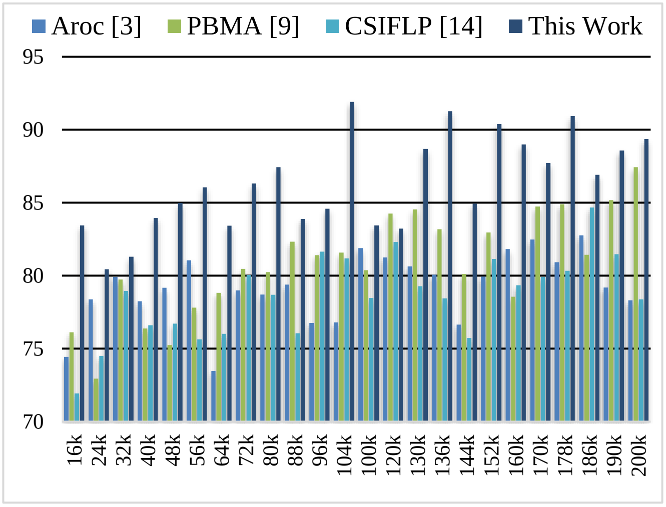

Fig. 7 Specificity measured during the iterative bug tracing process

This enhanced performance is a result of the blockchain-based tracing model's creative design. The Proof of Tracing (PoTr) consensus method, which carefully chooses miner nodes based on their proven tracing capability, is a key element contributing to this improvement. The bug-tracing process becomes more accurate and efficient with fewer false positives thanks to this technique, which makes sure that nodes skilled in tracking events contribute to the blockchain.

The model also incorporates Incremental Learning Operations, allowing for ongoing augmentation and improvement of its tracing capabilities over time. The model uses this incremental learning strategy to optimise its specificity values as NES rises, producing steadily higher values for various use situations.

In light of the significant advancements made by the proposed blockchain-based tracing model in comparison to other existing tracing models, these performance values are shown throughout several evaluations. A highly accurate and effective bug-tracing system that successfully reduces false positives is produced by integrating the PoTr consensus mechanism with the strategic use of Incremental Learning Operations. This study not only demonstrates how blockchain technology has the potential to overcome the drawbacks of traditional bug identification techniques, but it also lays the groundwork for future work on software debugging and maintenance that will make use of sophisticated consensus mechanisms and decentralized ledger systems. The proposed model gives promising results and improvement in precision, recall and AUC as compared to other tracing models. The performance can be enhanced with the help of some key factors like Proof of Tracing (PoTr) Consensus Mechanism, incremental learning operations, use of secondary indices and Merkle Tree Structure.

5. Conclusion and Future Work

In conclusion, this study offers a ground-breaking and comprehensive strategy for tackling the enduring difficulties in software bug identification by utilising a cutting-edge blockchain-based tracing methodology. Finding the exact and timely source of faults in the dynamic world of software development is still a difficult challenge. Existing tracing models frequently encounter difficulties in providing real-time traceability because of constraints in processing accuracy and efficiency. This paper fills in this vacuum by offering a cutting-edge framework that takes advantage of blockchain technology's ability to transform bug tracking activities in a variety of circumstances.

The Proof of Tracing (PoTr) consensus method, a key development in this area, is used to fuel the proposed blockchain-based tracing paradigm. This system chooses miner nodes based on their proven bug-tracking skills, making sure that only skilled nodes join the blockchain and enhancing the effectiveness and accuracy of bug tracing. Additionally, the model is given the capacity to constantly improve its tracing proficiency during the training and validation phases thanks to the implementation of Incremental Learning Operations, producing progressively better bug identification results. The outcomes shown in the adjacent tables support the effectiveness of the suggested model. The blockchain-based tracing model consistently outperforms other tracing models, such as Aroc [3], PBMA [9], and CSIFLP [14], as shown by the precision, accuracy, recall, AUC, and specificity values displayed across various Number of Evaluations (NES). This consistency confirms the model's ability to identify the source of bugs, reduce false positives, and enhance overall tracing speed. The model makes log analysis transparent, immutable, and decentralised by utilising distributed ledger technology, leading to effective bug tracing procedures.

The benefits of this strategy are clear from the impressive gains in precision, accuracy, recall, AUC, and specificity levels shown when compared to the competing models. Notably, these improvements are greatly aided by the use of the PoTr consensus method in conjunction with incremental learning operations. For the convergence of blockchain technology and softwar debugging approaches, this research establishes a critical precedent.

The suggested model's ability to surpass the drawbacks of current tracing technologies demonstrates its potential to alter bug identification procedures throughout the software development industry fundamentally. The paradigm has enormous potential for improving software maintenance and debugging since it can deliver quick, precise, and transparent bug tracing. This study not only offers a convincing answer to the problems at hand, but it also paves the way for future research and development projects that incorporate decentralised systems, sophisticated consensus mechanisms, and blockchain technology to improve software debugging procedures in a variety of fields.

The model achieves a precision of 94.86%, which is higher than Aroc (79.86%), PBMA (84.28%) and CSIFLP (87.76%). The model achieves a recall of (85.4 %,) which is higher than Aroc (80.80%), PBMA (81.14%) and CSIFLP (76.44%). The model also achieves AUC (81.20%), which is higher than the Aroc (75.71&), PBMA (78.44%), and CSIFLP (73.48%). Also, the model achieves a delay of 130.9675 ms, which is less than Aroc (178.1855 ms), PBMA (152.397 ms), CSIFLP (140.8885 ms).

5.1. Future Plans

The suggested blockchain-based tracing model has been successfully developed and validated, laying the groundwork for a wide range of intriguing future research directions and useful applications in the field of software development. This paper's novel strategy not only resolves current problems but also provides a pathway for future research and development. The future use of this discovery is indeed broad and has promise in several areas:

5.1.1. Enhancing Scalability and Performance

While the current study proves the viability of the blockchain-based tracing model, future work can concentrate on further optimising the model's scalability and performance. The model's practical utility could be greatly increased by investigating methods for handling larger datasets and improving the model's effectiveness in real-time tracing circumstances.

5.1.2. Dynamic Consensus Mechanisms

This study's introduction of the Proof of Tracing (PoTr) consensus method lays a strong foundation for effective miner node selection. The accuracy and responsiveness of the model might be further improved by the investigation of dynamic consensus techniques that adjust to shifting network conditions and node capabilities.

5.1.3. Integration with Modern Development Workflows

The discovery of bugs can be streamlined by integrating the suggested model into contemporary software development workflows like Continuous Integration and Continuous Deployment (CI/CD) pipelines. Its widespread acceptance might be facilitated by the creation of integration frameworks and tools that integrate the model into current development methodologies.

5.1.4. Privacy and Security Considerations

Security and privacy are the two main issues with any blockchain-based solution. Future studies can focus on strengthening the model's security defenses, fixing potential weaknesses, and guaranteeing adherence to data protection laws.

5.1.5. Tool Integration and Interoperability

Software development ecosystems include a wide range of frameworks and tools. The proposed model might be integrated with current bug tracking, testing, and debugging tools to produce a comprehensive solution that combines the advantages of blockchain technology with well-known development methodologies.

5.1.6. Real-world Case Studies and Testing

Performing real-world case studies and testing the model in various software development situations will offer useful insights into its effectiveness and highlight any potential implementation issues.

5.1.7. Hybrid Approaches

Investigating hybrid strategies that integrate blockchain technology with artificial intelligence, natural language processing, or other cutting-edge methodologies may result in models for bug detection that are even more reliable.

5.1.8. Adaptation to Other Domains

The suggested model's guiding concepts are not just applicable to software bug tracing. Other industries requiring traceability, such as supply chain management, healthcare, finance, and more, may be able to use the same blockchain-based infrastructure.

5.1.9. Collaboration and Implementation of Open Source

The model's codebase might be made open-source, and participation with the larger research and development community could foster iterative improvements, a variety of uses, and a deeper comprehension of the model's potential.

5.1.10. Improvement of the User Experience

User experience issues become increasingly important as the model develops. Widespread adoptions could be facilitated by creating user-friendly interfaces, intuitive visualisations, and effective outcomes communication.

In conclusion, the suggested blockchain-based tracing approach acts as a foundation for more research and invention. Its potential effects go beyond software bug tracing to a variety of fields where transparency and traceability are crucial.

The research community may unleash the full potential of blockchain technology for effective and precise bug identification by starting these new research initiatives, paving the path for more reliable software development and maintenance procedures.

References

[1] hipeng Gao et al., "Checking Smart Contracts with Structural Code Embedding," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2874-2891, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[2] Jiachi Chen et al., "Defectchecker: Automated Smart Contract Defect Detection by Analyzing EVM Bytecode," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2189-2207, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[3] Hai Jin et al., "Aroc: An Automatic Repair Framework for on-Chain Smart Contracts," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4611-4629, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[4] Fuchen Ma et al., "Pluto: Exposing Vulnerabilities in Inter-Contract Scenarios," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4380-4396, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[5] Jiachi Chen et al., "Defining Smart Contract Defects on Ethereum," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 327-345, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[6] Tuba Yavuz et al., "ENCIDER: Detecting Timing and Cache Side Channels in SGX Enclaves and Cryptographic APIs," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1577-1595, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Thomas Reinhold et al., "ExTRUST: Reducing Exploit Stockpiles with a Privacy-Preserving Depletion System for Inter-State Relationships," *IEEE Transactions on Technology and Society*, vol. 4, no. 2, pp. 158-170, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Mykhailo Lasynskyi, and Janusz Sosnowski, "Extending the Space of Software Test Monitoring: Practical Experience," *IEEE Access*, vol. 9, pp. 166166-166183, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[9] Amr Mansour Mohsen et al., "Enhancing Bug Localization using Phase-Based Approach," *IEEE Access*, vol. 11, pp. 35901-35913, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[10] Haijun Wang et al., "Explaining Regressions via Alignment Slicing and Mending," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2421-2437, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[11] Zhide Zhou et al., "LocSeq: Automated Localization for Compiler Optimization Sequence Bugs of LLVM," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 896-910, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[12] Yijing Lyu et al., "A Systematic Literature Review of Issue-Based Requirement Traceability," *IEEE Access*, vol. 11, pp. 13334-13348, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Ting Su et al., "Why my App Crashes? Understanding and Benchmarking Framework-Specific Exceptions of Android Apps," *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1115-1137, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[14] Xiaogang Zhu et al., "CSI-Fuzz: Full-Speed Edge Tracing Using Coverage Sensitive Instrumentation," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 912-923, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[15] Christophe Rezk, Yasutaka Kamei, and Shane McIntosh, "The Ghost Commit Problem when Identifying Fix-Inducing Changes: An Empirical Study of Apache Projects," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3297-3309, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[16] Lisa Nguyen Quang Do, and Eric Bodden, "Explaining Static Analysis with Rule Graphs," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 678-690, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[17] Le Yu et al., "Towards Automatically Localizing Function Errors in Mobile Apps with User Reviews," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1464-1486, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[18] Fiorella Artuso, Giuseppe Antonio Di Luna, and Leonardo Querzoni, "Debugging Debug Information With Neural Networks," *IEEE Access*, vol. 10, pp. 54136-54148, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Dylan Chapp et al., "Identifying Degree and Sources of Non-Determinism in MPI Applications via Graph Kernels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 12, pp. 2936-2952, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[20] Alessio Diamanti, José Manuel Sánchez Vílchez, and Stefano Secci, "An AI-Empowered Framework for Cross-Layer Softwarized Infrastructure State Assessment," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4434-4448, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[21] Zhanyao Lei et al. "Bootstrapping Automated Testing for RESTful Web Services," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1561-1579, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[22] Yu Qu et al. "Using K-Core Decomposition on Class Dependency Networks to Improve Bug Prediction Model's Practical Performance," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 348-366, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[23] Syed Farhan Alam Zaidi, Honguk Woo, and Chan-Gun Lee, "A Graph Convolution Network-Based bug Triage System to Learn Heterogeneous Graph Representation of Bug Reports," *IEEE Access*, vol. 10, pp. 20677-20689, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[24] Yanjie Jiang et al., "Bugbuilder: An Automated Approach to Building Bug Repository," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1443-1463, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[25] Hao Zhong, Xiaoyin Wang, and Hong Mei, "Inferring Bug Signatures to Detect Real Bugs," *IEEE Transactions on Software Engineering*, vol. 48, no. 2 pp. 571-584, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[26] Thu-Trang Nguyen et al., "A Variability Fault Localization Approach for Software Product Lines," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 4100-4118, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[27] Hassan Tahir, Saif Ur Rehman Khan, and Syed Sohaib Ali, "LCBPA: An Enhanced Deep Neural Network-Oriented Bug Prioritization and Assignment Technique Using Content-Based Filtering," *IEEE Access*, vol. 9, pp. 92798-92814, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Shunkun Yang et al., "Software Bug Number Prediction based on Complex Network Theory and Panel Data Model," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 162-177, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[29] Bei Wang et al., "Multi-Dimension Convolutional Neural Network for Bug Localization," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1649-1663, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[30] Gabriel Solomon et al., "A Secure and Cost-Efficient Blockchain Facilitated IoT Software Update Framework," *IEEE Access*, vol. 11, pp. 44879-44894, 2023. [CrossRef] [Google Scholar] [Publisher Link]