Pipeline Conflict in Processors: An Approach for Examining and Resolving Pipeline Conflict Using Machine Learning

[1]UGWUNNA, O. Charles, [2]BELONWU, S. Tochukwu, [2]UGOH, Daniel, [3]JOSHUA, John & [4]ADEKOYA, A.Mathew.

[1]Department of Computer Science, Wigwe University, Isiokpo, Rivers State, Nigeria.
[2]Department of Computer Science, Nnamdi Azikiwe University, Awka, Anambra State, Nigeria.
[3]Department of Computer Science, Ahmadu Bello University, Zaria, Kaduna State, Nigeria.
[4]Department of Physics, Wigwe University, Isiokpo, Rivers State, Nigeria

:charles.ugwunna@wigweuniversity.edu.ng; +(234) 8037921011

Received: 22.10.2024 **Accepted: 22.11.2024** **Published: 22.11.2024**

Abstract:
This study explores the impact of pipeline conflicts on processor reliability and performance, focusing specifically on data hazards, one of three primary types of pipeline conflicts (the others being control hazards and structural conflicts). Data hazards arise from dependencies between instructions, causing stalls that reduce pipeline efficiency. The research applies machine learning to detect and mitigate these conflicts, using a dataset of artificial instruction sequences, each labeled as either conflict-free or containing one of three data hazard types: Read After Write (RAW), Write After Read (WAR), or Write After Write (WAW). Two machine learning models—logistic regression and Support Vector Machine (SVM)-were evaluated for their effectiveness in identifying pipeline conflicts. The logistic regression model achieved 96% accuracy and high precision, recall, and F1-scores across all categories, indicating its strong ability to accurately classify pipeline conflicts. In contrast, the SVM model achieved lower accuracy (83%) and performed inconsistently across classes, excelling in some but struggling with others, suggesting difficulties in recognizing certain conflict patterns.

Keywords: Pipeline conflicts, processors, Machine Learning, Logistic Regression, control hazards, data hazards, Support Vector Machine and structural conflicts

1. Introduction

SINCE the 1960s, pipelined computer architecture has garnered considerable attention in the pursuit of faster and more cost-effective systems. Pipelining involves breaking down the execution of instructions into smaller stages and processing them concurrently, allowing for improved performance without a significant increase in overall system cost. As technology has advanced, the availability of faster and more affordable Large-Scale Integration (LSI) circuits has further propelled the potential of pipelining, making it a promising approach for future computer architecture [1].

The problem of pipeline conflicts has become a significant difficulty in processor design and performance optimization. Utilizing the structural and behavioural traits that distinguish pipelined circuits from other circuits can greatly increase the productivity and scalability of pipelined circuit verification, as demonstrated by the work of MD Aagaard [2] in their paper titled A Hazard-Based Correctness Statement for Pipelined Circuits. Our analysis is based on their formal model of pipelines, which extends a state machine with particular information on state variable read/write interactions and parcel transfer across stages. Furthermore, their rigorous definition of correctness, which is based on data, structural, and control dangers, provides a thorough method for assessing pipeline reliability. This work proposes a fresh paradigm that utilizes the ideas and techniques provided by MD Aagaard [2]to effectively investigate and solve pipeline problems. With machine learning models trained on massive datasets, we can now detect, identify, and manage such conflicts in real time, ensuring increased processor efficiency as well as longevity in complex and rapidly evolving computing environments.

A wide range of fields have recently shown a great deal of interest in machine learning techniques. Machine learning offers a very potential substitute for conventional dispute resolution procedures by using data-driven methodologies to automatically find, analyze, and resolve pipeline problems. By training models on historical data and including features that can detect patterns, machine learning algorithms can predict the likelihood of disagreements in the future.This enables the processor's throughput and performance to be optimized through the use of proactive and adaptive techniques.
By training models on historical data and adding variables that may represent the features of conflicts, machine learning algorithms can recognize patterns and predict the likelihood of disagreements in the future. This allows for the utilization of preemptive and flexible approaches to optimize the processor's throughput and performance.

Furthermore, machine learning models can adjust to shifting workloads, dynamic circumstances, and system architecture. They are well adapted for making decisions in real-time in intricate and quickly changing contexts since they cancontinuously learn and update their knowledge based on fresh facts [3].

Remember that precise reflection and adjustment are necessary for methods based on machine learning for pipeline conflict resolution to be implemented successfully. Important elements that can affect the model's efficacy include the selection of relevant and high-quality training data, the development of reliable machine-learning algorithms, and the choice of suitable features. When using machine learning models in real-world situations, potential biases and interpretability issues with data privacy models should also be taken into account.

During the study process, we have chosen to concentrate on applying machine learning to anticipate, address, and lessen data dangers. Allowing several instructions to run concurrently runs the risk of them interfering with one another and creating conflicts. When instructions in a pipeline are executed concurrently, several different kinds of conflicts may occur. One of the main challenges of pipelined systems' high performance is data risks [4].

By enabling the use of complex scheduling and forwarding algorithms, pipelined designs can be greatly enhanced by machine learning's predictive power. These models highlight important paths and potential data dependencies within the pipeline, allowing for the intelligent sequencing of instructions or the inclusion of appropriate routing methods to mitigate dangers. The primary goal of this research project is to determine how machine learning may assist in developing more efficient pipeline designs in addition to predicting possible hazards.

Technological developments present a significant opportunity to transform computer system design and optimization through the incorporation of machine learning into pipelined systems. To identify and address pipeline conflicts, particularly data hazards, in pipelined computer architecture, this research suggests applying machine learning models.

2. REVIEW OF RELATED WORK

Pipeline architecture came into existence because the need for faster and more cost-effective systems became critical. It was created because it can help match the speeds of various subsystems without duplicating the cost of the entire system involved. Pipelining was developed as a way to improve processors by allowing multiple instructions to be executed simultaneously, but of course, for every advantage, there will also be a disadvantage, which brought about pipeline conflicts in the architecture [5].

To improve the RISC cores' resilience and dependability, Tahar and Kumar [6] introduced a methodical technique for explicitly validating pipeline conflicts in these processor architectures. Through the process of formalizing and classifying the various kinds of conflicts that may occur when instructions are executed simultaneously in the pipeline, the researchers prepared the way for a thorough examination. This methodology is significant because it streamlines the verification process by introducing automated proof techniques that are customized for each unique type of conflict. Interestingly, when conflicts are found, the research produces the associated circumstances that lead to their occurrence, offering insightful information to designers on how to resolve and eradicate them. The authors demonstrated the applicability and effectiveness of their proposed approach through illustrative examples of the DLX RISC processor. Through this significant contribution, the research community gained an invaluable resource to systematically investigate, detect, and ultimately mitigate pipeline conflicts in RISC cores, fostering the development of more dependable and efficient processor designs.

Many strategies have been put out over time to resolve the pipeline conflicts in computer architecture, some of which are:

- InstructionScheduling: As part of an effort to construct an optimizing compiler for a pipelined architecture, a code reorganization approach has been developed to minimize the amount of runtime pipeline interlocks. ILP processors present a higher challenge to compilers, and most of these processors fall short of performance targets in the absence of sophisticated compiling techniques. These methods are mostly used during the code generation and high-level implementation stages of instruction scheduling [7].

- Register renaming: This technique removes erroneous data dependencies between register operands of succeeding instructions in sequential code, such as write after read (WAR) and writes after write (WAW). By removing associated precedence restrictions from the instructions' execution order, this technique raises the IPC (number of instructions performed per cycle) by enabling more instructions to be processed in parallel each cycle. A thorough grasp of this intricate process can be attained by investigating the register renaming design space[8].

Tongsima et al. (1998) suggested a new loop scheduling algorithm intended to lower the risk of data in applications involving digital signal processing. The SHARP tool, which schedules a pipelined data flow graph to multiple pipelined units, included this algorithm. The tool's architecture minimises execution hazards and hides data threats.

Hwang and Hwang [9] present the idea of using code reorganization in pipelined architectures as an alternative for pipeline interlocks. To avoid the need for intricate hardware solutions, code reorganization involves rearranging machine-level instructions during compilation. The study demonstrates the NP-completeness of the problem and suggests a heuristic algorithm for its solution. Verifying the approach, empirical data from the MIPS processor design are presented. Additionally, the effect of code reorganization on the compiler system was discussed, providing some useful context. In summary, the research delves into the possibilities of rearranging code to improve pipelined architectures and optimize processor designs.

The work by Charvát et al. [10] shows an automated way to find flaws in single-pipelined microprocessors, such as data and control hazards. This is meant to make the verification process better. This method is implemented in the tool Hades.

In contrast, the study by Parrot et al. [11] extends the model to investigate the design space of pipelined systems and

validate temporal features, with an emphasis on pipeline design optimization using Timed Petri Nets. Furthermore, Najjar et al. [12] study explores the use of pipeline techniques for hazard identification and mitigation in a hardwired programmable processor.

By demonstrating several approaches and techniques to improve the dependability and effectiveness of pipelined circuit verification, these works jointly highlight the significance of addressing risks in pipelined circuits to guarantee accurate execution and performance optimisation.

3. METHODOLOGY

The procedure commences with data gathering when relevant information regarding processor instructions is obtained. This could contain historical information, system logs, or performance indicators that highlight the traits and chance of developing conflict.

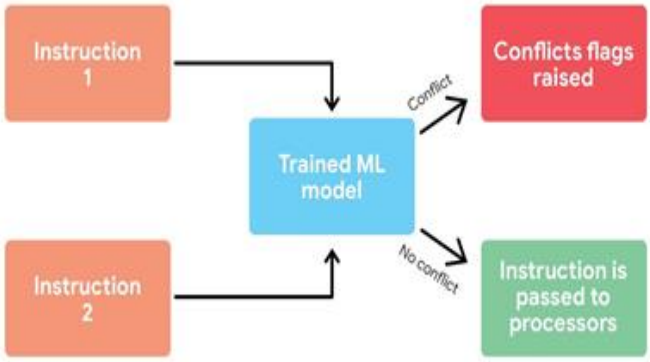

Figure 1: System architecture

After data is gathered, the preparation stage makes sure it is consistent and suitable for the model. Cleaning up the data, dealing with missing values, and normalizing the input features are all part of preprocessing. Figure 1 shows how to do this. As soon as the data is ready in a standard format, it may be added to the machine learning model.

After training, the machine learning model is integrated into the processing system. This integration could entail embedding the model into the existing pipeline or integrating it into the processor architecture. When the model is smoothly integrated, it becomes a crucial part of the decision-making process for evaluating instructions.

Individual instructions or batches of instructions are run through the model for analysis during the evaluation phase. The model looks at the instructions' properties and makes a prediction about whether or not conflicts will arise from them. An indication of the possible risks connected to each instruction is given by this prediction.

One component of the methodology is the flag-raising system, which is activated as soon as conflicts are detected. The model alerts the relevant systems or stakeholders when it finds a conflict by raising a flag or sending out a notification. Real-time conflict detection and alerting enable proactive measures and timely response. Following conflict identification, the method incorporates conflict resolution strategies. These strategies which are implemented in reaction to the conflicts discovered might involve rearranging instructions, making use of pipelining techniques, or changing the processor architecture. By lowering the number of conflicts, the system's overall performance can be improved.

Throughout the process, the model's performance is regularly tracked and evaluated. The model is gradually enhanced and modified based on feedback regarding its efficacy and accuracy. Iteratively evaluating the model ensures that its ability to accurately assess pipeline conflicts will always be dependable and effective.

Using this method makes the machine learning model that the processing system has developed an essential component for analyzing instructions and anticipating conflicts. Proactive decision-making and efficient dispute resolution are made possible by its integration, which enhances processor performance.

3.1 Data Generation and Analysis

The purpose of this dataset is to provide synthetic data for pipeline identification of conflict studies. By focusing on detecting pipeline conflicts in instruction sequences, it aims to facilitate the development and evaluation of machine learning models. Using this dataset, scholars and practitioners can analyze and explore various approaches to detecting and resolving pipeline disputes.

With a specific focus on data risks, the dataset is made to represent several information operations that are implemented in assembly code. These instructions mimic real-world instruction sequences and real-world data hazard conditions. The labels in this dataset correspond to the following danger scenarios:

(i) Read-After-Write (RAW) dependency occurs when a subsequent instruction reads a register or memory location that was previously written to by another instruction.

(ii) Write-After-Read (WAR) dependency occurs when a subsequent instruction writes to a register or memory location that was previously read from another instruction.

(iii) Write-After-Write (WAW) dependency: Two subsequent instructions write to the same register or memory location.

3.1.1 **Description of the Dataset**

The three columns that make up the dataset hold the essential information required for detecting data hazards.

a) Instruction 1: The first instruction in a sequence of instructions is shown in this column. It includes the instruction's textual representation in assembly language format. Operations like arithmetic (ADD, SUB), memory (LOAD, STORE), logical (AND, OR), and shifts (SHIFT) are examples of operations that are represented as strings in instructions. The registers (R0, R1, etc.) and immediate values are the operands of the instructions.

b) Instruction 2: This column represents the subsequent instruction in the instruction sequence. It follows the same format as Instruction 1 and represents the instruction that has a potential data hazard with Instruction 1. The data hazard can be of three types: Read-after-Write (RAW), Write-after-Read (WAR), or Write-after-Write (WAW).

c) Conflict Type: This column represents the type of data hazard present between Instruction 1 and Instruction 2. It specifies whether the pair of instructions exhibits a RAW, WAR, WAW hazard, or no conflict.

The dataset includes synthetic samples for each type of data hazard as well as instances with no conflicts. Random values are generated for registers and immediate values to introduce variations and randomness in the data.

3.1.2 **Dataset Generation Process**

This dataset was synthetically generated using predefined rules to simulate different types of pipeline conflicts (data hazards) [13, 14]. The generation process adheres to the following rules:

a) The first instruction, designated as instruction 1, is chosen and written to a register or memory address. This is known as the read-after-write dependency (RAW) mechanism. Next, a different instruction is chosen, indicated by instruction 2, which reads from the register or memory address that instruction 1 had written to. After that, the instructions are given back and marked "RAW." Instruction 2 must depend on Instruction 1 in this connection.

b) Write-After-Read Dependency (WAR): Instruction 2, which reads from a register or memory address, is the first instruction chosen. Next, a different instruction is chosen, indicated by instruction 2, which writes to the same register or memory address that instruction 1 was reading. After that, the instructions are given back and marked "WAR."

c) Writing to a register or memory address is the first instruction chosen in a write-after- write (WAW) system. This is indicated by instruction 1. Next, a different instruction is chosen, indicated by instruction 2, which writes to the same register or memory address as instruction 1. After that, the instructions are given back and marked "WAW."

Table 1: Sample of the Dataset

S/N	Instruction 1	Instruction 2	Conflict Type
0	SUB R10, R0, 34	OR R5, R10, R2	RAW
1	XOR R10, R0, 19	LOAD R10, R0, 36	WAW
2	AND R4, R1, R2	SHIFT R2, R4, R3	WAR
3	OR R10, R1, R2	LOAD R10, R10, R3	WAR
4	SHIFT R5, R1, R2	AND R9, R5, R3	WAR
5	AND R1, R0, 43	ADD R1, R1, R2	RAW
6	MUL R2, R1, R2	MUL R8, R3, 10	No_CONFLICT
7	MUL R13, R0, 31	OR R10, R13, R2	RAW
8	OR R15, R0, 20	MUL R15, R0, 0	WAW
9	OR R0, R0, 16	ADD R0, R0, 8	WAW

Table 1 above displays a sample of the data set created with pre-established rules to model various pipeline conflict scenarios. The sample dataset can be gotten from the link. https://github.com/Dev-180Memes/pipeline-conflict-detection. For a machine learning model to effectively learn the differences between instructions that would generate data hazards and instructions that wouldn't, the need to generate data on instructions that wouldn't generate any hazards arose. The data on non-conflict-generating data was simulated using the following rules:

(i) An instruction 1 which writes to a register Rx and reads from registers, R1 and R2 is first selected.

(ii) An instruction 2 which writes to a register Ry and reads from registers, R3 and performs operation on the immediate value of 10 is also selected

(iii) The instructions along with the label "No CONFLICT" are then returned.

The dataset described in Table 1 is synthetic and based on predefined rules. It does not in any way represent real-world data and should be used only for educational or research purposes.

3.2 Machine Learning Model Development

From the data generated in Table 1 above we have a resulting dataset with three columns named:

(i) Instruction 1

(ii) Instruction 2

(iii) Conflict type

Each column contains data that would be used in training a machine learning model that can adequately predict data hazards in instruction sequences. Each of these instructions and the conflict type they present are all represented as strings. Machine learning models are only capable of training numbers (integers or floats) which means we would need to represent each of these strings as numbers.

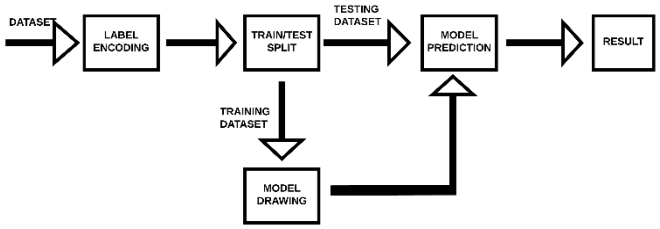

Figure 2: Machine Learning Process

Figure 2 shows the machine learning processes involved enabling the instructions to be used in the models .The following preprocessing steps were taken on each of the columns as follows:

Conflict Type: The conflict type column consists of data on the type data hazard that would be produced as a result of these instructions. In the dataset there are four (4) different types of hazards which are Read-After-Write (RAW), Write-After-Read (WAR), Write-After-Write (WAW) and No Conflict. To use this data in our ML model we need to convert it into a numerical representation, which we accomplish by label encoding[15].

After label encoding is done, the target values are now represented in numeric form. This numeric form would act as the target values for our trained ML model.

Instruction 1 and Instruction 2: Since instruction 1 and instruction 2 are both similar, the same kind of preprocessing is done to both columns. To change values of the instruction strings to a numeric representation a rule is then provided.

The instructions in the dataset are examples of instructions in a three (3) operand machine (Patterson, 1985). In a three-operand machine, each instruction consists of four (4) parts, which are the operator and three operands. The dataset consists of 10 unique operators. These operators are each changed into a numerical representation (0 - 9) and stored in an array.

After the operator has been changed, we are now movingon to the operands. The operands are each represented in two distinct ways, it's either the operand is a register or an immediate value. If the operand is a register, we remove the R in front and add the leftover number (the register number) into the array. Finally, if the operand is an immediate value, the binary representation of the number is used. These rules are implemented and the resulting data set are displayed in the Table 2 below which are accomplished by data set encoding.

Table 2: Encoded Dataset

Instruction 1	Instruction 2	Conflict Type
[1, 10, 0, 100010]	[7, 5, 10, 2]	1
[8, 10, 0, 100011]	[4, 10, 0, 100100]	3
[6, 4, 1, 2]	[9, 2, 4, 3]	2
[7, 10, 1, 2]	[4, 10, 10, 3]	2
[9, 5, 1, 2]	[6, 9, 5, 3]	2

3.3 Feature Engineering

The instruction was successfully changed into an array of numbers; the problem of having the instructions as arrays and our target (Conflict Type) is an integer still exists. To properly train our model we need to extract both our operators and operands from the array, this is done using pandas series object as displayed in Table 3 below where the operator and operands are taken out of the array

Table 3: Operator and operands

Conflict Type	Operator 1	Operand 1.1	Operand 1.2	Operand 1.3	Operator 2	Operator 2.1	Operator 2.2	Operator 2.3
1	1	10	0	100010	7	5	10	2
3	8	10	0	10011	4	10	0	100100
2	6	4	1	2	9	2	4	3
2	7	10	1	2	4	10	10	3
2	9	5	1	2	6	9	5	3

The instruction 1 has an operand and three operators that can be represented as operand 1, operator 1.1, operator 1.2 and operator 1.3. The same applies for operator 2. This logic is then applied to instruction 2 also, so that from the instructions using pandas series object we can extract and properly represent each item in the array.

4. RESULTS AND DISCUSSIONS

The results discussed in this paper were generated using each of this ML algorithms as depicted in table 4.

Table 4: Machine earning Algorithm results

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	96%	0.967	0.963	0.965
Support Vector Machine	83%	0.909	0.884	0.895

Logistic Regression: This is a popular machine learning algorithm commonly used for binary classification tasks. It is particularly useful when the outcome variable is binary or categorical. Despite its name, logistic regression is a classification algorithm rather than a regression algorithm. Among its many benefits are its efficiency, interpretability, and simplicity with other regression models. This algorithm is a robust and trendy tool for solving classification problems; because of its interpretability, it is especially helpful in situations where it is important to comprehend the factors that go into the classification decisions [16, 17].

Support Vector Machine: For tasks including regression and classification, Support Vector Machines (SVM) constitutes an effective machine learning algorithm. By maximizing the margin, it determines the most suitable hyperplane to divide data points into different classes. By using kernel functions to translate the data into a higher-dimensional space, SVM can handle both linear and nonlinear relationships between features. Because of this, SVM is useful for capturing intricate patterns. Because of the margin concept, SVM is less liable to overfitting.

The logistic regression model's 96% accuracy rate in classifying pipeline conflicts in the processor is demonstrated in Figure 3. All classes have consistently high precision, recall, and F1 scores, ranging from 0.94 to 0.99. This implies that the logistic regression model is highly capable of accurately detecting pipeline conflict occurrences in various categories.

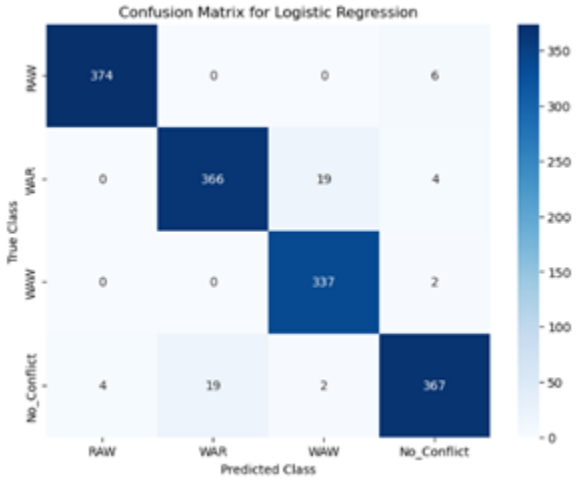

Figure 3: Confusion Matrix for Logistic Regression

On the other hand, Figure 4 shows that the SVM model performs unevenly across classes and has a lower overall accuracy of 83% s shown in table 4. It can correctly classify instances of class 0 and class 3, as evidenced by its high recall, precision, and F1-scores; however, it has difficulty classifying instances of class 1 and class 2. The inferior performance in these classes can be a sign that it's difficult to identify the precise traits and patterns associated with pipeline conflicts within those classifications.

Moreover, the logistic regression model's interpretability holds significance within the research context. It enables researchers to comprehend the fundamental elements and characteristics causing pipeline conflicts, offering insights into

the underlying causes and possible remedies.

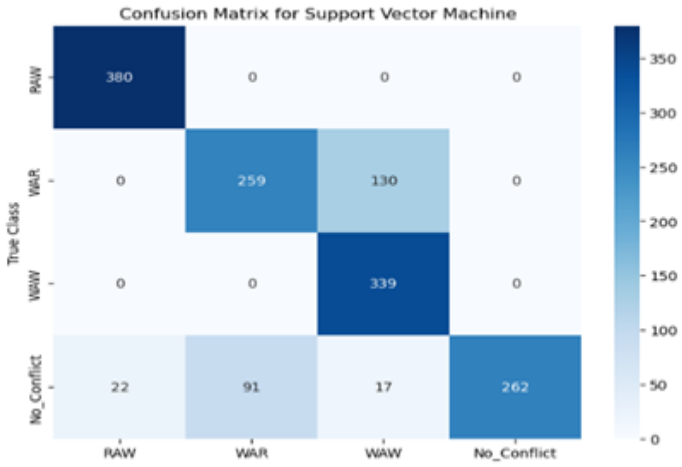

Figure 4: Confusion Matrix for Support Vector Machine

This interpretability helps identify important factors influencing conflicts and supports the development of resolution strategies, which is in line with the research goal of creating a framework for examining and resolving pipeline conflicts.

It is essential to take into account the implications of the classification results in the context of pipeline conflict investigation in addition to the performance metrics that have been discussed. The logistic regression model appears to be highly accurate at detecting and classifying pipeline conflicts based on its high precision, recall, and F1 scores across all classes. This suggests that the model can offer insightful information about the types and frequency of pipeline conflicts in the processor.

The goal of the project is to create a framework for investigating and resolving pipeline conflicts with machine learning. The robust performance of the logistic regression model in this context is in line with the goals of the study. The model's ability to precisely identify and classify pipeline conflicts can aid in the development of successful resolution strategies as well as a thorough understanding of the factors that contribute to these conflicts.

The logistic regression model's interpretability makes it even more appropriate for pipeline conflict research. The model's coefficients can be used by researchers to determine specific features and variables that have the biggest influence on pipeline conflict prediction. The interpretability of the data enables a more profound comprehension of the fundamental reasons and changes linked to pipeline conflicts, thereby enabling focused interventions and enhancements in the architecture and functionality of the processor.

5. CONCLUSION

In conclusion, with superior recall, accuracy, and precision scores than the other models in every class, the logistic regression model performs exceptionally well. Because of its accuracy in pipeline conflict classification, it is a reliable instrument for recognizing and categorizing different types of conflicts. Researchers may also find out more about the factors causing pipeline disputes thanks to the interpretability of the logistic regression model, which aids in the development of targeted remedies. In contrast, the SVM model's lower overall accuracy and uneven performance across classes demonstrate its limitations in accurately categorizing some types of pipeline conflicts. However, the benefits of the SVM model, such as its ability to handle nonlinear relationships through kernel functions, may be useful in recognizing intricate patterns linked to pipeline disputes. Because of its great efficiency and interpretability, the logistic regression model is the most suitable choice for investigating pipeline conflicts in the processor, given the goals of the research. The reliability and trustworthiness of the outcomes are increased by accuracy and uniform performance across classes. Additionally, by simplifying the model's interpretation, pipeline conflicts can be better understood, leading to more focused interventions and improved processor architecture and operation.

DECLARATIONS

Ethics approval and consent to participate: Not Applicable

Consent for publication: Not Applicable

Conflict of interests

The author declare no conflict of interests.

Authors' contributions: Authors wrote and edited the manuscript.

References

[1] D. A. Patterson, "Reduced instruction set computers," *Communications of the ACM*, vol. 28, no. 1, pp. 8-21, 1985.

[2] M. D. Aagaard, "A hazards-based correctness statement for pipelined circuits," in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, 2003: Springer, pp. 66-80.

[3] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions," *IEEE access*, vol. 7, pp. 137184-137206, 2019.

[4] S. Tongsima, C. Chantrapornchai, E. H.-M. Sha, and N. L. Passos, "Reducing Data Hazards on Multi-pipelined DSP Architecture with Loop Scheduling," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 18, pp. 111-123, 1998.

[5] C. V. Ramamoorthy and H. F. Li, "Pipeline architecture," *ACM Computing Surveys (CSUR)*, vol. 9, no. 1, pp. 61-102, 1977.

[6] S. Tahar and R. Kumar, "Formal verification of pipeline conflicts in RISC processors," in *Proc. European Design Automation Conference (EURO-DAC94), Grenoble, France*, 1994: Citeseer, pp. 285-289.

[7] P. B. Gibbons and S. S. Muchnick, "Efficient instruction scheduling for a pipelined architecture," in *Proceedings of the 1986 SIGPLAN symposium on Compiler construction*, 1986, pp. 11-16.

[8] D. Sima, "The design space of register renaming techniques," *IEEE micro*, vol. 20, no. 5, pp. 70-83, 2000.

[9] Y.-T. Hwang and J.-S. Hwang, "Efficient code generation for digital signal processors with parallel and pipelined instructions," in *1997 IEEE Workshop on Signal Processing Systems. SiPS 97 Design and Implementation formerly VLSI Signal Processing*, 1997: IEEE, pp. 243-252.

[10] L. Charvát, A. Smrčka, and T. Vojnar, "Utilizing parametric systems for detection of pipeline hazards," *International Journal on Software Tools for Technology Transfer*, pp. 1-28, 2022.

[11] R. Parrot, M. Briday, and O. H. Roux, "Timed Petri nets with reset for pipelined synchronous circuit design," in *International Conference on Applications and Theory of Petri Nets and Concurrency*, 2021: Springer, pp. 55-75.

[12] H. Najjar, R. Bourguiba, and J. Mouine, "Pipeline Hazards Resolution for a New Programmable Instruction Set RISC

Processor," *International Journal of Advanced Computer Science and Applications,* vol. 9, no. 10, 2018.

[13] D. R. Jeske, P. J. Lin, C. Rendon, R. Xiao, and B. Samadi, "Synthetic data generation capabilties for testing data mining tools," in *MILCOM 2006-2006 IEEE Military Communications conference*, 2006: IEEE, pp. 1-6.

[14] H. Murtaza, M. Ahmed, N. F. Khan, G. Murtaza, S. Zafar, and A. Bano, "Synthetic data generation: State of the art in health care domain," *Computer Science Review*, vol. 48, p. 100546, 2023.

[15] B.-B. Jia and M.-L. Zhang, "Multi-dimensional classification via sparse label encoding," in *International Conference on Machine Learning*, 2021: PMLR, pp. 4917-4926.

[16] M. P. LaValley, "Logistic regression," *Circulation,* vol. 117, no. 18, pp. 2395-2399, 2008.

[17] T. G. Nick and K. M. Campbell, "Logistic regression," *Topics in biostatistics,* pp. 273-301, 2007.