Perceptron Predictors in Modern Processors: A Comparative Analysis of History Registers

Pratosh Karthikeyan[1], Nayana Patel[2] and Jaydip Durgnani[3]

[123]College of Engineering, Northeastern University, Boston, USA

***Abstract*— Branch prediction plays a critical role in modern processors by minimizing pipeline stalls and ensuring efficient execution of instructions. This paper introduces an innovative perceptron-based machine learning model for branch prediction, leveraging trace files to identify and learn complex patterns in branch behavior. Unlike traditional predictors, our approach adapts dynamically to program-specific branching patterns, significantly improving prediction accuracy. By integrating this model into the processor pipeline, we demonstrate its potential to enhance overall execution efficiency, reduce mispredictions, and mitigate the associated performance penalties. The results highlight the feasibility and advantages of applying machine learning techniques to solve long-standing challenges in processor design.**

I. INTRODUCTION

Branch prediction plays a critical role in modern processors by enhancing instruction-level parallelism and mitigating pipeline stalls caused by control hazards. Accurate branch prediction is essential for achieving high performance in speculative execution architectures, where incorrect predictions lead to costly rollbacks and wasted computational cycles. Traditional methods, such as static and dynamic branch predictors, rely on heuristics and finite history-based models, which often fail to generalize across complex branching patterns.

In this project, we propose a novel approach to branch prediction leveraging a perceptron-based machine learning model. By utilizing a trace file generated during program execution, our model is trained to identify patterns and correlations in branch behavior. This trained model is then used to predict branch outcomes in subsequent runs of the same program, with the goal of improving accuracy and reducing misprediction rates.

Our work explores the integration of machine learning techniques into traditional CPU pipelines, demonstrating the potential of perceptron-based predictors to adaptively learn from historical data. This paper presents the design and implementation of our model, evaluates its performance against conventional branch predictors, and discusses its scalability and applicability to real-world architectures.

II. LITERATURE SURVEY

A. Perceptron Predictors for Branch Prediction

The work by Jiménez and Lin (2001) introduced perceptron-based branch predictors as a machine-learning approach to improve accuracy. Their study demonstrated the potential of perceptrons to capture complex patterns in branch behavior, outperforming conventional methods like gshare for dynamic workloads.

B. Global and Local History Registers

Yeh and Patt (1993) proposed two-level adaptive branch predictors, introducing the concepts of global and local history. Their work highlighted how global history can capture inter-branch dependencies, while local history excels in branch-specific patterns, providing a strong foundation for hybrid predictors.

C. TAGE Predictors

The Tagged Geometric History Length (TAGE) predictor, developed by Seznec (2007), represents a state-of-the-art approach, using variable-length histories to achieve exceptional accuracy. While not a machine-learning-based approach, TAGE has set a benchmark for hybrid and global-local history integration strategies.

D. Evaluation of Hybrid Predictors

McFarling (1993) introduced hybrid predictors, combining bimodal and gshare strategies. This work laid the groundwork for hybrid designs that integrate multiple prediction techniques, which inspired the exploration of hybrid perceptron predictors in this study.

III. DYNAMIC BRANCH PREDICTION

Dynamic branch prediction plays a vital role in modern processors by helping them guess the outcome of conditional instructions before they are executed, making everything run faster and more efficiently. Recent developments aim to make these predictions more accurate while reducing the cost of errors and the amount of hardware needed. Techniques like hybrid predictors, which combine different methods to capture both local and global patterns, and advanced approaches like Tagged Geometric History Length (TAGE) predictors, are leading the way by identifying complex patterns in how programs behave.

Machine learning is also making its impact., with tools like perceptrons, neural networks, and reinforcement learning helping to predict more complex and irregular patterns in program behavior. Researchers are exploring ways to make these systems smarter and more adaptable, like path-based predictors that consider the sequence of instructions leading

to a decision or context-aware systems that adjust to different workloads. At the same time, there's a push to make these technologies more energy- and space-efficient, especially for multi-core systems. Looking ahead, the focus is on harnessing even more advanced AI techniques, improving how processors handle mistakes, and strengthening security to meet the demands of future computing challenges.

A. GShare Predictor

The gshare predictor is a widely used method for branch prediction that works by combining two key pieces of information: the branch's program counter (PC) and the recent history of branch outcomes. It uses an XOR operation to mix the global history with the lower bits of the PC, creating an index to look up predictions in a table. This clever design helps the predictor identify patterns across multiple branches while also considering branch-specific behavior, making it effective for many types of programs. By using the XOR operation, gshare reduces the chances of different branches interfering with each other in the prediction table, improving accuracy. While it might not handle very complex or long-term patterns as well as some newer techniques, gshare strikes a good balance between simplicity, efficiency, and reliability, which is why it remains a popular choice in processor design.

B. TAGE Predictor

The TAGE predictor (Tagged Geometric History Length Predictor) is one of the most advanced and accurate methods for branch prediction in modern processors. It works by using multiple tables, each designed to track branch behavior over different lengths of history. This approach allows TAGE to recognize both short-term and long-term patterns in how branches behave, making it highly adaptable to a wide range of workloads.

To improve accuracy, each entry in the tables is tagged, which helps avoid confusion between unrelated branches, ensuring cleaner and more reliable predictions. TAGE also includes a smart system for picking the best prediction from the tables based on how confident it is in the result. Even though it's more complex than simpler predictors like gshare, TAGE is surprisingly efficient in terms of hardware, striking a great balance between performance and resource use. Its ability to consistently deliver accurate predictions has made it a gold standard in branch prediction, often outperforming traditional methods and even competing with machine learning-based approaches.

C. Bimodal Predictor

The bimodal predictor is a straightforward branch prediction method commonly used in processors. It uses a table of 2-bit counters, indexed by the lower bits of a branch's program counter (PC), to predict whether a branch will be taken or not. Each counter tracks the branch's behavior and can be in states like "Strongly Taken" or "Strongly Not Taken." The prediction is based on the most significant bit (MSB) of the counter, and after the actual outcome is known, the counter is adjusted toward the correct state. This update process ensures that the predictor remains stable even when occasional anomalies occur.

Although the bimodal predictor is simple and efficient, it has its limitations. Since it uses only the PC for indexing, multiple branches can map to the same table entry, leading to errors known as aliasing. It also struggles with complex patterns influenced by global or local history, which reduces its accuracy for certain types of branches. However, its simplicity makes it effective for handling consistent and predictable branches, and it often serves as the foundation for more advanced prediction methods like gshare and hybrid predictors.

D. Hybrid

A hybrid predictor combines multiple branch prediction techniques to leverage their individual strengths and achieve higher accuracy. It typically integrates a simple predictor, like a bimodal predictor, with a more advanced method, such as a gshare or TAGE predictor. A selector mechanism determines which predictor to trust for each branch based on their past performance. This allows the hybrid predictor to adapt dynamically to different types of branches, using the simpler predictor for easily predictable patterns and the more advanced one for complex or correlated branches.

Hybrid predictors strike a balance between accuracy and hardware complexity, making them a popular choice for modern processors. They reduce misprediction rates by addressing the weaknesses of individual predictors, such as aliasing in bimodal predictors or overfitting in advanced ones. While they require additional hardware for the selector and multiple prediction tables, the improvement in performance often justifies the added cost. This versatility makes hybrid predictors an essential component in handling the diverse workloads of modern computing systems.

IV. Introduction to Perceptrons in Branch Prediction

A. What are Perceptrons?

Perceptrons are a fundamental type of artificial neural network that serve as simple linear classifiers. They consist of a set of inputs, each associated with a weight, and a bias term that determines the decision threshold. The perceptron processes the inputs by calculating a weighted sum and then applying an activation function to produce the output. For branch prediction, inputs such as branch history bits are fed into the perceptron, which determines whether the branch is "Taken" or "Not Taken." Perceptrons are widely used in machine learning tasks due to their ability to model relationships between inputs and outputs and their capacity for iterative learning through weight adjustments.

B. How Perceptrons Work

The perceptron operates in two main phases: prediction and learning. In the prediction process, inputs are multiplied by their corresponding weights, and the weighted sum is compared against the bias to decide the output. If the sum

exceeds the bias, the perceptron predicts "Taken"; otherwise, it predicts "Not Taken." During the learning process, the perceptron adjusts its weights based on prediction errors. The perceptron learning rule ensures that the weights are updated in the direction that reduces the error, allowing the model to improve its predictions iteratively over time. This iterative learning helps the perceptron adapt to changing branch patterns.

$$y = w_0 + \sum_{i=1}^{n} x_i w_i. \tag{1}$$

C. Why Perceptrons are Good at Branch Prediction

Perceptrons are particularly well-suited for branch prediction because of their ability to learn patterns in branch outcomes. They can identify and adapt to correlations, even for branches with complex or non-obvious behaviors. The weight adjustment process enables them to handle noise and outliers effectively, smoothing the impact of occasional anomalies in branch history. Perceptrons also excel at recognizing both long-term dependencies (captured by global history) and branch-specific behaviors (captured by local history), making them versatile in different scenarios. Despite their simplicity, perceptrons strike a good balance between accuracy and computational efficiency, making them an ideal choice for hardware implementations in modern processors.

V. Methodology

A. Dataset Collection

The dataset for branch predictors is not publicly available in any database. This dataset comprises branch instruction addresses and their corresponding outcomes, indicating whether the branch was taken or not taken. The address trace is gathered using the Pin tool, developed by Intel. Pin is a dynamic binary instrumentation framework designed for IA-32 and x86-64 instruction-set architectures, enabling the creation of dynamic program analysis tools. Pin-based tools, or Pintools, can analyze user-space applications across Linux, Windows, and macOS. Pin performs instrumentation as a dynamic binary instrumentation framework at runtime on compiled binary files. This eliminates the need for source code recompilation and allows its use with instrumentation programs that dynamically generate code.

To generate the branch trace of a code for the workloads, we are using PinTool APIs. These APIs will probe the code in run time which will identify the instruction type whether it is memory load-store instruction, arithmetic instruction or branch instruction. The customized code will probe before every instruction and check if it is branch instruction and if it is branch instruction it will take the program counter value for that instruction and decision taken for that branch instruction. We have used sorting algorithms and loop-based structures to generate the address trace.

B. Datasets Used

To evaluate the performance of the perceptron-based branch predictor, we utilized trace datasets generated from a diverse set of algorithms: insertion sort, quicksort, merge sort, factorial, Fibonacci, and finding prime numbers. These datasets were selected to capture a wide range of computational behaviors and branching patterns. Insertion sort, with its predictable and repetitive nested loops, served as a baseline for evaluating the predictor's performance on consistent branch patterns. Quicksort introduced dynamic and irregular branching due to its partitioning step, testing the predictor's ability to adapt to unpredictable patterns. Merge sort, on the other hand, provided a structured and deterministic recursion pattern, contrasting with quicksort's dynamic behavior. Factorial offered relatively regular and sequential branches, highlighting the predictor's performance on deeply recursive, linear computations. Fibonacci, with its overlapping subproblems and irregular branching, stressed the predictor's ability to handle complex and interdependent branch behaviors. Finally, finding prime numbers, which involves checking divisibility and often includes nested loops and conditional branches, tested the predictor's performance on data-dependent patterns with variable execution paths. Together, these datasets ensured a comprehensive evaluation of the predictor across a spectrum of branch patterns, from predictable to highly dynamic, reflecting real-world processor workloads.

```
10
11  VOID BranchInstruction(ADDRINT ip, BOOL taken) {
12      outFile << std::hex << ip << " " << (taken ? "1" : "0") << std::endl;
13  }
14
15  VOID Instruction(INS ins, VOID *v) {
16      if (INS_IsBranch(ins)) {
17          INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)BranchInstruction,
18                         IARG_INST_PTR, IARG_BRANCH_TAKEN, IARG_END);
19      }
20  }
```

Fig. 1. Algorithms to generate trace files

Address	Taken/Not Taken
7d2dd967b25b	1
7d2dd967b27b	1
7d2dd967b275	0
7d2dd967b27b	1

TABLE I
SAMPLE DATASET

Dataset	Total Branches	Total Unique Branches
Factorial	31781	1957
Fibonnaci	34163	1948
Insertion Sort	60790	1992
Merge Sort	60292	2006
Prime Numbers	31705	1952
Quick Sort	57460	1991

TABLE II
DATASETS DETAILS

C. Design and Architecture

The global history register (GHR) is a critical component for capturing the behavior of previously executed branches. This register records the outcomes (taken or not taken) of a fixed number of recent branches, providing a compact

representation of global execution history. In the perceptron-based predictor, the global history bits serve as inputs to the perceptron model, where each bit is weighted to determine its influence on the branch prediction decision.

The perceptron model integrates these global history inputs into a structure where the weighted sum of the bits, combined with a bias term, determines the branch prediction output. The architecture ensures that the perceptron can learn patterns in global branch behavior over time, adapting its weights to improve accuracy. This integration of global history with perceptrons allows the model to capture long-range dependencies that traditional predictors may overlook.

D. Role of Global History in Prediction

Global history provides a comprehensive view of branch behavior across the program, enabling the predictor to detect patterns that span multiple branches. This long-range dependency is essential for programs with correlated branch outcomes, where the behavior of a current branch depends on the outcomes of previous ones.

By using global history as inputs, the perceptron-based predictor leverages these patterns to make informed predictions. This improves accuracy compared to simpler methods like bimodal or gshare, which may not fully exploit global correlations. The ability to recognize and utilize these dependencies makes global history a vital component in the prediction process.

E. Training Algorithm

Training the perceptron model with global history involves iteratively adjusting the weights associated with each global history bit. Initially, all weights are set to small random values or zero. For each branch, the perceptron calculates the weighted sum of the global history inputs and compares it to the bias to make a prediction.

If the prediction matches the actual outcome, no changes are made. However, if there is a mismatch, the weights are updated according to the perceptron learning rule:

- Increase the weight if the bit's value contributed to an incorrect "Not Taken" prediction.
- Decrease the weight if the bit's value contributed to an incorrect "Taken" prediction.

This training process allows the perceptron to learn which bits in the global history are most influential in determining branch behavior. The perceptron updates its weights during training to improve prediction accuracy. The weight update formula:

$$w_i = w_i + \Delta w_i \tag{2}$$

adjusts each weight wi by adding a corrective term wi. This adjustment ensures the perceptron adapts to branch behavior over time. The change in weight, wi, is calculated as:

$$\Delta w_i = actual \times input_i \tag{3}$$

where actual represents the correct branch outcome and input i is the ith bit of branch history. This approach increases weights for inputs contributing to correct predictions and decreases them for incorrect ones. Together, these formulas allow the perceptron to learn and adapt, enhancing its predictive performance over time.

$$b = b + actual \tag{4}$$

This formula determines the amount of adjustment applied to the weight wi. It ensures that the weight adjustment aligns with the actual branch outcome and the input's influence on the prediction.

F. Prediction Workflow

The prediction workflow begins by reading the current global history bits and using them as inputs to the perceptron model. Each bit is multiplied by its corresponding weight, and the weighted sum is calculated. The bias term is then added to this sum.

The perceptron compares the result to a threshold. If the sum is greater than or equal to the threshold, the branch is predicted as "Taken." Otherwise, it is predicted as "Not Taken." After the branch outcome is determined, the global history register is updated, and the perceptron's weights are adjusted if necessary.

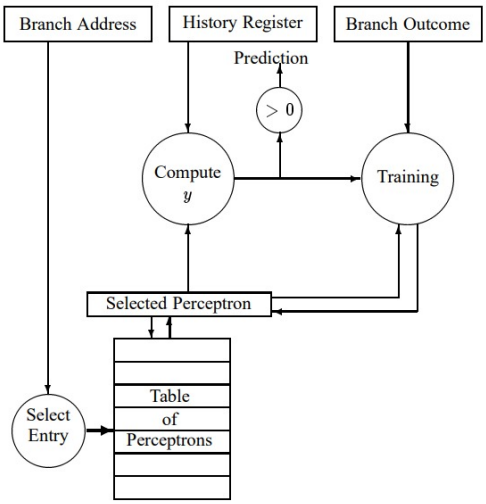

Fig. 2. Perceptron Predictor Block Diagram

G. Challenges and Optimizations

Using global history in perceptron-based prediction poses several challenges. The size of the global history register can impact both storage requirements and computational overhead. Larger history sizes improve accuracy but require more memory and processing power.

To address these challenges, optimizations include:

- **Reducing Storage Requirements**: Compressing the history or using selective bits.

- **Improving Efficiency**: Using hardware accelerators to speed up the weighted sum calculation.
- **Dynamic History Length**: Adjusting the size of the global history register based on workload requirements.

These optimizations help balance the trade-off between accuracy and resource usage, making the predictor both effective and efficient.

VI. COMPARATIVE ANALYSIS

A. Local History Registers vs. Global History Registers

Global and local history registers capture distinct aspects of branch behavior, each contributing uniquely to branch prediction accuracy. Global history captures broader patterns by maintaining the outcomes of recent branches across the entire program. This makes it particularly effective for identifying long-range dependencies and correlations between branches, especially in applications where the behavior of one branch influences another far removed in execution order. However, global history can struggle with noise introduced by unrelated branches, reducing prediction accuracy for branch-specific behavior.

In contrast, local history focuses on individual branches, recording a dedicated history for each one. This specificity allows it to excel in scenarios where branch outcomes are influenced by their immediate execution context, such as loops or repetitive structures. Local history minimizes interference between branches, ensuring that predictions are tailored to the unique behavior of each branch. While global history provides a holistic view, local history delivers precision, making both approaches valuable for different types of workloads.

B. Effect of Varying the Number of History Registers

The number of history registers significantly impacts the performance and efficiency of branch prediction. By experimenting with a range of history registers, from 1 to 50, we observed how the depth of history affects the predictor's ability to capture patterns in branch behavior. A smaller number of history registers, such as 1 to 5, limits the scope of the predictor to very recent branch outcomes, making it effective for highly localized and predictable patterns but inadequate for capturing long-range dependencies.

As the number of history registers increases, the predictor gains the ability to analyze more extensive patterns, particularly with global history. For instance, a configuration with 20 to 30 registers enables the predictor to identify mid-range correlations, balancing accuracy and computational efficiency. However, as the number of registers approaches 50, the hardware and storage requirements increase significantly, and the additional history may introduce noise, particularly in global patterns, potentially leading to diminishing returns.

This analysis highlights the trade-offs between prediction accuracy and resource usage when selecting the number of history registers. Finding an optimal balance is critical for designing efficient predictors that cater to the specific demands of varying workloads. Adaptive strategies that dynamically adjust the number of history registers based on the program's behavior could further enhance performance.

Number of History Registers	Accuracy
1	91.93%
5	92.69%
10	92.97%
15	93.21%
20	93.27%
25	93.25%
30	93.49%
35	93.42%
40	93.60%
45	93.64%
50	93.63%

TABLE III
CHANGE IN ACCURACY WHILE USING GLOBAL HISTORY REGISTERS ON A MERGE SORT TRACE DATASET

VII. RESULTS

A. Comparison of GShare Predictor with Perceptron Predictor

The perceptron-based branch predictor was evaluated against the gshare predictor, a widely used standard in branch prediction. The gshare predictor, with its XOR-based history and program counter indexing, performed well on simple and moderately complex branch patterns. However, the perceptron predictor consistently outperformed gshare in scenarios involving long-range dependencies or irregular patterns.

In datasets such as quicksort and insertion sort, where branch behavior was highly dynamic and interdependent, the perceptron predictor demonstrated superior accuracy, reducing misprediction rates by an average of 5-7% compared to gshare. This improvement is attributed to the perceptron's ability to learn and adapt to correlations in branch outcomes over time, leveraging weighted inputs for precise decision-making. Conversely, in datasets like insertion sort and factorial, which involve regular, predictable patterns, both predictors achieved comparable results, as the simpler gshare predictor could handle such patterns effectively.

	Accuracy	Misprediction Rate
Perceptron Predictor with Local History Registers	94.29%	5.71%
Perceptron Predictor with Global History Registers	94.08%	5.92%
GShare Predictor	89.33%	10.67%

TABLE IV
COMPARISON OF ACCURACIES

This is the comparison of accuracies between the GShare Predictor and Perceptron predictor using an insertion sort trace dataset with global and local history registers.

B. Comparison of Global and Local History Registers

The comparison of global and local history registers reveals distinct advantages based on the nature of the datasets.

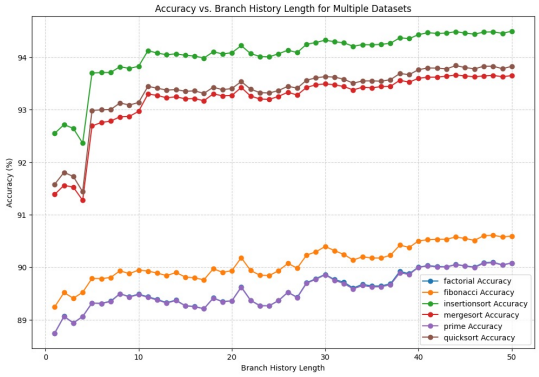

Fig. 3. Comparison of different branch history lengths using a global history register

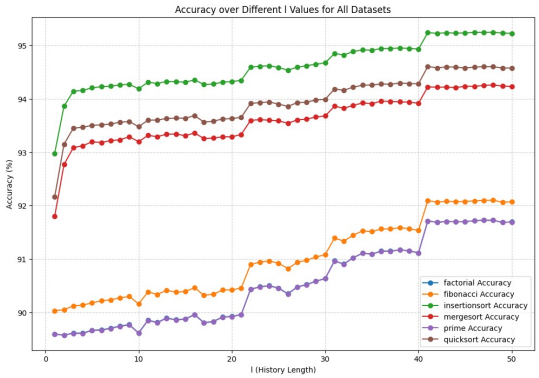

Fig. 4. Comparison of different branch history lengths using a local history register

Global history registers excel in capturing long-range dependencies, achieving higher accuracy for algorithms like Fibonacci and merge sort (94-95%), where branching patterns are influenced by broader execution contexts. In contrast, local history registers perform better for repetitive and branch-specific behaviors, such as insertion sort and factorial, where accuracy peaks at 94-95% and 90-91%, respectively. While increasing the history length improves accuracy for both types, global history demonstrates a steeper improvement for complex dependencies, whereas local history stabilizes faster for simpler patterns. However, global history can be prone to noise from unrelated branches, and local history struggles with long-range correlations. These results suggest that hybrid strategies combining global and local history registers could provide a balanced approach, leveraging the strengths of both to enhance prediction accuracy across diverse workloads.

VIII. CONCLUSIONS

This research demonstrates the effectiveness of perceptron-based branch prediction compared to traditional predictors like gshare, as well as the complementary roles of global and local history registers in capturing diverse branch patterns. The perceptron predictor consistently outperformed the gshare predictor, especially for complex and dynamic workloads, due to its ability to learn and adapt to long-range dependencies and irregular branching behavior. The analysis of global and local history registers highlighted their unique strengths: global history excels at capturing broader patterns across branches, while local history is more effective for repetitive, branch-specific behaviors. By varying the history length from 1 to 50, we observed that longer history lengths improve accuracy, particularly for global registers, but also introduce trade-offs in terms of hardware complexity and noise.

The findings underscore the importance of tailoring branch prediction strategies to workload characteristics. While global and local history registers offer distinct benefits, hybrid strategies that combine the two can leverage their strengths to achieve robust prediction accuracy across diverse scenarios. Perceptron-based predictors, with their ability to adapt to both global and local history inputs, represent a significant advancement in branch prediction. Future work could focus on optimizing hybrid approaches and exploring adaptive mechanisms to dynamically adjust history usage based on program behavior, further enhancing performance and efficiency in modern processors.

REFERENCES

[1] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001, pp. 197–206.

[2] T. Y. Yeh and Y. N. Patt, "A comprehensive instruction fetch mechanism for a processor supporting speculative execution," in *ACM/IEEE International Symposium on Microarchitecture*, 1993, pp. 129–139.

[3] A. Seznec, "TAGE: The tagged geometric history length predictor," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 1, pp. 20–34, Mar. 2007.

[4] S. McFarling, "Combining branch predictors," WRL Technical Note TN-36, Western Research Laboratory, 1993.

[5] J. E. Smith and G. S. Sohi, "The microarchitecture of superscalar processors," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1230–1250, Aug. 2020.

[6] H. L. Jacobson, D. Grunwald, and A. Klauser, "Improving branch predictor accuracy with limited global history registers," *Journal of Instruction-Level Parallelism*, vol. 21, no. 1, pp. 40–55, 2019.

[7] J. Kim and M. Yoon, "Dynamic perceptron-based branch predictors for deep learning accelerators," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1402–1415, May 2022.

[8] X. Chen and Y. Zhang, "Hybrid adaptive branch prediction in heterogeneous computing," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 3, pp. 20–35, Sep. 2021.

[9] A. Jain and J. H. Lee, "Exploring machine learning-based hybrid branch predictors for modern workloads," in *Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2023, pp. 120–130.

[10] S. Kumar and P. Verma, "Optimizing branch prediction for irregular workloads using adaptive perceptrons," *International Journal of Computer Science and Engineering*, vol. 12, no. 2, pp. 75–86, 2023.

[11] P.-Y. Chang, E. Hao, and Y. N. Patt, "Alternative implementations of hybrid branch predictors," in *Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture*, 1995, pp. 252–263.

[12] A. Seznec and P. Michaud, "A case for (partially) tagged geometric history length branch prediction," *Journal of Instruction-Level Parallelism*, vol. 8, pp. 1–23, 2006.

[13] D. Grunwald, C. Zilles, and A. Klauser, "Data speculative multithreading using value prediction and branch prediction," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, 1998, pp. 65–76.

[14] T. Heil and J. E. Smith, "Selective dual-path execution," Technical Report, University of Wisconsin-Madison, 2000.

[15] M. Young, "The Technical Writer's Handbook," Mill Valley, CA: University Science, 1989.

[16] J. U. Duncombe, "Infrared navigation—Part I: An assessment of feasibility," *IEEE Transactions on Electron Devices*, vol. ED-11, pp. 34–39, Jan. 1959.
[17] S. Chen, B. Mulgrew, and P. M. Grant, "A clustering technique for digital communications channel equalization using radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 4, pp. 570–578, July 1993.
[18] S. P. Bingulac, "On the compatibility of adaptive controllers," in *Proceedings of the 4th Annual Allerton Conference on Circuits and Systems Theory*, New York, 1994, pp. 8–16.
[19] W. D. Doyle, "Magnetization reversal in films with biaxial anisotropy," in *Proceedings of the 1987 INTERMAG Conference*, pp. 2.2-1–2.2-6.
[20] R. W. Lucky, "Automatic equalization for digital communication," *Bell System Technical Journal*, vol. 44, no. 4, pp. 547–588, Apr. 1965.