Hybrid Perceptron-Based Branch Predictor

Kishore Kumar Thiyagarajan, Meenakshi Nandagopal, Abhigna Police
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA
nandagopal.m@northeastern.edu, thiyagarajan.k@northeastern.edu, police.ab@northeastern.edu

Abstract—**Accurate branch prediction is critical for sustaining high instruction throughput in deeply pipelined processors, as each misprediction introduces control hazards and wasted work[1]. This project implements and evaluates three dynamic branch predictors of increasing sophistication: a 2 bit saturating counter predictor, a perceptron based neural predictor, and a hybrid predictor that combines both using a meta selector. The predictors are integrated into a common Python framework and evaluated on three synthetic traces that emulate distinct workload behaviors: a loop intensive trace, a complex conditional trace with long range correlations, and a highly irregular branchy trace. For each configuration we measure prediction accuracy, mispredictions per thousand instructions (MPKI), approximate CPI impact under a fixed misprediction penalty, and storage overhead. Results show that the perceptron and hybrid predictors achieve near perfect accuracy on the loop and complex traces, while all predictors converge to approximately 50% accuracy on the highly random branchy trace, reflecting the inherent unpredictability of that workload. The hybrid design offers slightly improved robustness on the hardest trace with only modest additional storage compared to the pure perceptron predictor. These findings highlight the trade off between predictor complexity, hardware cost, and resilience across diverse branch behaviors[2].**

I. Introduction

Modern superscalar and deeply pipelined processors rely heavily on accurate branch prediction to sustain high instruction throughput. Control flow changes caused by conditional branches introduce uncertainty in the instruction stream, and each misprediction forces the pipeline to flush incorrectly fetched instructions, resulting in wasted cycles and increased CPI.[1] As pipeline depth and issue width grow, the penalty of a single misprediction increases, making effective branch prediction a critical component of processor performance[1]. A wide range of dynamic predictors have been proposed to capture different patterns in branch behavior. Simple table based mechanisms, such as the 2 bit saturating counter predictor, exploit short term direction bias and perform well on highly repetitive structures such as loops. More sophisticated machine learning–inspired predictors, such as the perceptron predictor, learn longer range correlations in global history and are capable of recognizing complex branch behavior. Hybrid predictors combine the strengths of multiple underlying mechanisms through a meta selection policy, providing robustness across diverse workloads. This project implements and evaluates three such dynamic predictors—2 bit saturating counter, perceptron, and a hybrid (2 bit + perceptron)—within a unified simulation framework[2]. To characterize their behavior under different branch patterns, we generate three synthetic trace types: a loop intensive trace, a conditional pattern based trace with learnable correlations, and a highly irregular mixed branch trace. For each predictor–trace combination, we measure accuracy, mispredictions per thousand instructions (MPKI), CPI impact, and approximate storage overhead. The goal of this study is to analyze how predictor sophistication influences performance across varying branch characteristics and to quantify the trade offs between complexity, hardware cost, and prediction quality[3].

II. Background and Related Work

Branch prediction is essential in modern pipelined processors because every mispredicted branch causes wasted cycles and increases overall CPI. Dynamic predictors learn branch behavior at runtime and attempt to guess the outcome before the branch is resolved. The 2 bit saturating counter predictor is one of the simplest dynamic mechanisms. It uses a small table of 2 bit finite state machines to track whether each branch tends to be taken or not taken. This works well for branches with strong repetition, such as loops, but struggles with branches that depend on longer patterns. To handle more complex behavior, the perceptron predictor uses global branch history and a set of learned weights to recognize correlations across many past branches. This allows it to capture patterns that table based predictors cannot easily represent. Because no single predictor performs best for every type of branch behavior, hybrid predictors combine multiple prediction mechanisms. A meta selector chooses which predictor to trust for each branch, improving robustness across traces with different characteristics[2]. In this project, we implement and evaluate these three predictors—2 bit, perceptron, and hybrid—using three synthetic traces designed to represent loop heavy behavior, correlated conditional behavior, and highly irregular branch behavior. This provides a clear demonstration of how predictor complexity affects accuracy, MPKI, CPI, and storage cost [4].

III. Predictor Configurations

A. Two-Bit Saturating Counter Predictor

The 2 bit saturating counter predictor is a simple table based dynamic predictor. Each branch address indexes into a table entry containing a 2 bit finite state machine with four states ranging from strongly not taken to strongly taken. The predictor outputs “taken” when the counter is in one of the two taken states and “not taken” otherwise. After each branch resolves, the counter increments or decrements toward the correct direction, without wrapping around. This predictor is effective for branches with repetitive behavior—especially loop closing branches—but cannot capture long range patterns or correlations. Its main advantages are low hardware cost and fast access time.

B. Perceptron Predictor

The perceptron predictor uses a small neural model to make predictions based on global branch history. Each perceptron consists of a vector of signed weights, where each weight corresponds to one bit in the global history register. For each branch, the predictor computes a weighted sum of the history bits and compares the result to a threshold to determine whether the branch is predicted taken or not. After each branch outcome is known, the perceptron updates its weights if the prediction was incorrect or if the output was too weak. This allows the perceptron to learn long range correlations and conditional patterns that simple table based predictors cannot capture. Although it requires more storage than the 2 bit predictor, the perceptron is significantly more accurate on complex traces.

C. Hybrid Predictor

The hybrid predictor combines the strengths of the 2 bit and perceptron predictors. It includes a simple selector table that maintains a 2 bit counter for each branch, indicating which predictor has recently been more accurate for that branch address. For each prediction, both underlying predictors generate an outcome, and the selector chooses which one to trust. After the branch resolves, the selector counter is updated to reinforce whichever predictor was correct. This design allows the hybrid predictor to behave like a perceptron on traces with strong global correlations while falling back to the 2 bit predictor when simple repetitive patterns occur. As a result, the hybrid provides more consistent performance across the three trace types used in this project.

IV. Trace Generation Methodology

To evaluate the behavior of the three branch predictors fairly, we created a set of synthetic traces that represent different types of branch behavior commonly seen in real programs. Each trace consists of 5,000 branch outcomes in a simple text format where each line contains a branch address and an outcome ("T" for taken or "N" for not taken). These traces were generated using Python scripts included in our project and were designed to highlight specific predictor strengths and weaknesses.

A. Loop Intensive Trace (loop_5000.txt)

The loop intensive trace was created to represent branches that repeat the same pattern many times, such as loop back edges. In this trace, a small number of static branch addresses repeat frequently, and the outcomes follow a predictable cycle (mostly taken with occasional not taken events). This design allows the 2 bit saturating counter predictor to perform well because it can quickly lock onto a stable taken/not taken pattern, mimicking real loop behavior.

B. Complex Conditional Trace (complex_5000.txt)

The complex trace models workloads where branches depend on longer range correlations rather than simple short term repetition. The generation script uses multiple branch addresses and deterministic patterns that repeat over a longer sequence. This produces structured behavior that is difficult for the simple 2 bit predictor but highly suitable for the perceptron predictor, which uses global history to detect these correlations. This trace helps demonstrate the perceptron's strength in learning non trivial patterns.

C. Highly Irregular Branch Trace (branchy_5000.txt)

The branchy trace represents a workload with irregular or noisy branch behavior. The generation script selects branch outcomes randomly, making the pattern nearly unpredictable. This causes all predictors to perform close to 50 percent accuracy, which reflects the theoretical limit on purely random branches. This trace is useful for evaluating predictor robustness, since only predictors capable of adapting to mixed or inconsistent behavior can show even slight improvements.

V. Experimental Setup

All three branch predictors were implemented in Python using a modular architecture that separates core prediction logic, trace parsing, and experiment execution. Each predictor (2-bit saturating counter, perceptron, and hybrid) provides a common interface, enabling identical evaluation procedures across configurations and ensuring measurement consistency. The predictors were tested on three synthetic trace files designed to represent distinct classes of branch behavior: `loop_5000.txt`, `complex_5000.txt`, and `branchy_5000.txt`. Each trace consists of 5,000 dynamic branch outcomes in the form `<branch_address> <T/N>` and is processed sequentially. During execution, each predictor records the total number of correct and incorrect predictions, from which we derive prediction accuracy[1], [2].

In addition to accuracy, we evaluate misprediction rate using MPKI (mispredictions per thousand instructions) and estimate performance impact using CPI. Following prior work that models branch misprediction penalties in out-of-order processors [**?**], [**?**], we assume each trace represents 100,000 dynamic instructions, with 5,000 control-flow instructions and a fixed misprediction penalty of 5 cycles. CPI is calculated using this penalty model without the need for a cycle-accurate simulator. Storage overhead is also reported for each predictor, accounting for predictor table sizes, weight vectors, and selector counters.

All experiments were conducted using a single evaluation script, `run_all.py`, which executes each predictor across all three traces and produces a unified summary of accuracy, mispredictions, MPKI, estimated CPI, and memory cost. This framework ensures that differences in performance arise from predictor architecture rather than experimental artifacts.

VI. Results

This section summarizes the performance of the three branch predictors—2 bit, perceptron, and hybrid—across the three synthetic traces. Each experiment uses exactly 5,000 branch outcomes. The metrics reported include accuracy, number of correct predictions, mispredictions, MPKI, and CPI (computed using a fixed misprediction penalty of 5 cycles)[5].

A. Summary of Results

Table I summarizes the performance of all three branch predictors across the three synthetic traces.

B. Key Observations

1) Loop Trace: The perceptron and hybrid predictors achieve near-perfect accuracy on the loop trace, reaching approximately 99.8%. The 2-bit predictor performs significantly worse at 79.9%, as it can only follow short-term repetitions and fails to adapt to occasional variations in the loop pattern. The gap in accuracy is reflected in the corresponding MPKI and CPI values.

2) Branchy (Irregular) Trace: All predictors converge to roughly 50% accuracy on the branchy trace, consistent with nearly random branch outcomes. The hybrid predictor shows a marginal improvement (0.5038), likely due to selector decisions that exploit small variations between predictors. MPKI values are highest on this trace, which translates into an increased CPI for all models.

3) Complex Correlated Trace: The perceptron again dominates with 99.9% accuracy, exploiting strong long-range correlations present in the trace. The hybrid predictor follows closely at 99.84%, with its selector benefiting from the perceptron's performance. The 2-bit predictor remains near 50% and fails to learn correlations, leading to MPKI values around 25 and associated CPI penalties. Very low MPKI from perceptron and hybrid models directly yields CPI values in the range of 1.0003–1.0004.

C. Overall Trends

Perceptron is the best single predictor, dominating two of the three traces. Hybrid improves slightly over perceptron on noisy traces, giving more stability across workloads. 2 bit predictor performs well only on simple repetitive patterns and collapses on complex or mixed behavior. Low MPKI from perceptron/hybrid translates directly into lower CPI.

D. Graphs

1) Accuracy: The accuracy graph shows that both the Perceptron and Hybrid predictors achieve almost perfect accuracy on the loop and complex traces, while the 2 bit predictor performs noticeably worse on these patterns. On the branchy trace, all three predictors drop to around 50% accuracy because the branch behavior is highly irregular. The Hybrid predictor shows a very small improvement over the others on the branchy trace, but overall, the Perceptron remains the most accurate predictor across the predictable traces.

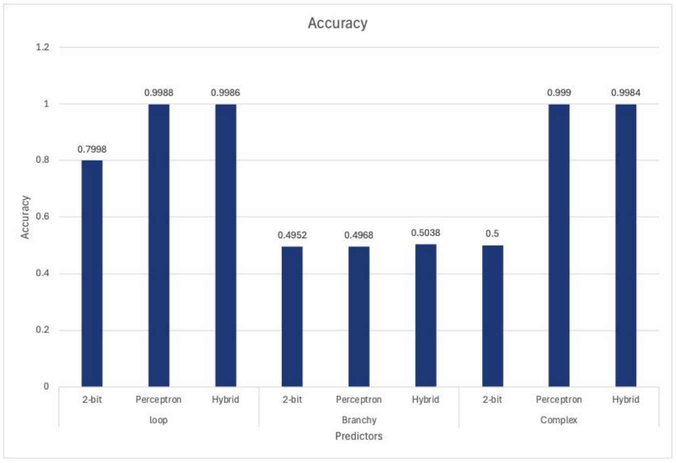


Fig. 1. Accuracy of the 2-bit, perceptron, and hybrid predictors across the loop, complex, and branchy traces.

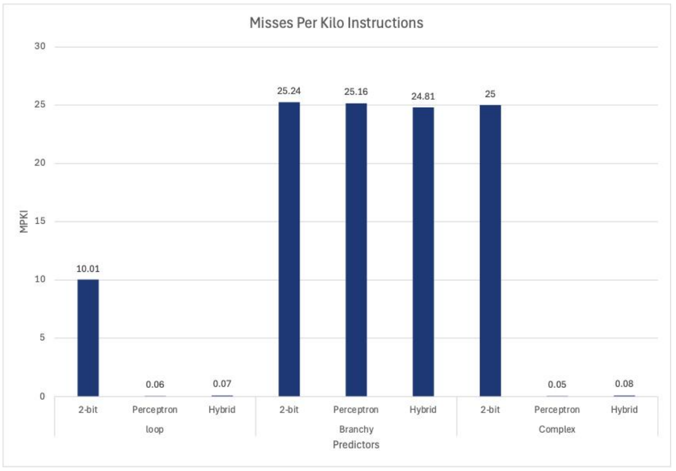


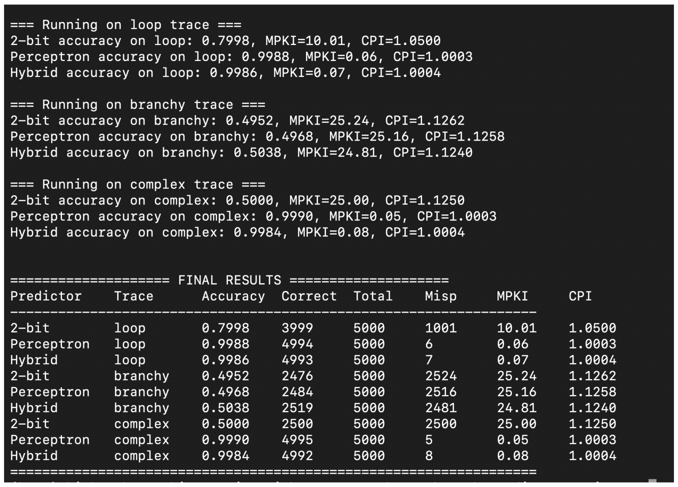
Fig. 2. MPKI comparison for the three predictors on all three traces.

2) Misses Per Kilo Instructions: The MPKI graph shows that the Perceptron and Hybrid predictors have extremely low MPKI on the loop and complex traces, reflecting their near perfect accuracy. The 2 bit predictor has noticeably higher MPKI on these traces, especially for complex, where it reaches about 25. On the branchy trace, all three predictors show high MPKI values around 24–25 because the branch behavior is irregular and difficult to learn. The Hybrid predictor has the lowest MPKI of the three on the branchy trace, indicating slightly fewer mispredictions in the hardest scenario.

3) Cycles Per Instruction: The CPI graph shows that both the Perceptron and Hybrid predictors achieve very low CPI values on the loop and complex traces (around 1.0003–1.0004), reflecting their very low misprediction rates. The 2 bit predictor has noticeably higher CPI on these traces because it makes more mistakes, especially on loops. On the branchy trace, all predictors show higher CPI values around 1.124–1.126 due to frequent mispredictions. The Hybrid predictor performs slightly better than the others on the branchy trace, giving the lowest CPI among the three in that category.

TABLE I
PERFORMANCE RESULTS FOR THE 2-BIT, PERCEPTRON, AND HYBRID PREDICTORS ACROSS THREE SYNTHETIC TRACES.

Predictor	Trace	Accuracy	Correct	Total	Misp.	MPKI	CPI
2-bit	Loop	0.7998	3999	5000	1001	10.01	1.0500
Perceptron	Loop	0.9988	4994	5000	6	0.06	1.0003
Hybrid	Loop	0.9986	4993	5000	7	0.07	1.0004
2-bit	Branchy	0.4952	2476	5000	2524	25.24	1.1262
Perceptron	Branchy	0.4968	2484	5000	2516	25.16	1.1258
Hybrid	Branchy	0.5038	2519	5000	2481	24.81	1.1240
2-bit	Complex	0.5000	2500	5000	2500	25.00	1.1250
Perceptron	Complex	0.9990	4995	5000	5	0.05	1.0003
Hybrid	Complex	0.9984	4992	5000	8	0.08	1.0004



```
=== Running on loop trace ===
2-bit accuracy on loop: 0.7998, MPKI=10.01, CPI=1.0500
Perceptron accuracy on loop: 0.9988, MPKI=0.06, CPI=1.0003
Hybrid accuracy on loop: 0.9986, MPKI=0.07, CPI=1.0004

=== Running on branchy trace ===
2-bit accuracy on branchy: 0.4952, MPKI=25.24, CPI=1.1262
Perceptron accuracy on branchy: 0.4968, MPKI=25.16, CPI=1.1258
Hybrid accuracy on branchy: 0.5038, MPKI=24.81, CPI=1.1240

=== Running on complex trace ===
2-bit accuracy on complex: 0.5000, MPKI=25.00, CPI=1.1250
Perceptron accuracy on complex: 0.9990, MPKI=0.05, CPI=1.0003
Hybrid accuracy on complex: 0.9984, MPKI=0.08, CPI=1.0004


==================== FINAL RESULTS ====================
Predictor    Trace      Accuracy  Correct  Total    Misp     MPKI     CPI
-----------------------------------------------------------------
2-bit        loop       0.7998    3999     5000     1001     10.01    1.0500
Perceptron   loop       0.9988    4994     5000     6        0.06     1.0003
Hybrid       loop       0.9986    4993     5000     7        0.07     1.0004
2-bit        branchy    0.4952    2476     5000     2524     25.24    1.1262
Perceptron   branchy    0.4968    2484     5000     2516     25.16    1.1258
Hybrid       branchy    0.5038    2519     5000     2481     24.81    1.1240
2-bit        complex    0.5000    2500     5000     2500     25.00    1.1250
Perceptron   complex    0.9990    4995     5000     5        0.05     1.0003
Hybrid       complex    0.9984    4992     5000     8        0.08     1.0004
=================================================================
```

Fig. 3. Final output results.

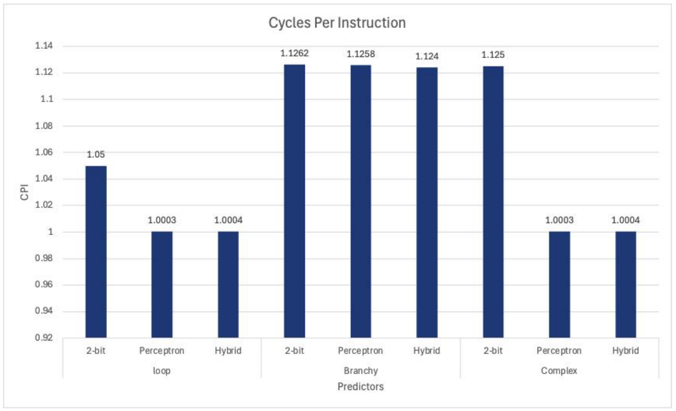


Fig. 4. Estimated CPI impact for each predictor under a fixed 5-cycle misprediction penalty.

VII. DISCUSSION

A. Why perceptron and hybrid perform best on the loop trace, and why 2 bit is weaker

The loop trace contains a highly repetitive taken/not taken pattern, which makes it easy for advanced predictors to learn. Both the perceptron and hybrid predictors achieve near perfect accuracy (99.8%) because they can adapt quickly to the stable loop behavior. The 2 bit predictor performs worse (79.98%) because its finite state machine can follow simple repetitions but struggles when the loop pattern includes occasional variations. As a result, while all predictors benefit from the predictable nature of loop branches, the perceptron and hybrid models clearly outperform the 2 bit predictor.

B. Why the perceptron dominates on the complex trace

The complex trace contains long range correlations and patterns that cannot be captured by simple 2 bit finite state machines. The perceptron predictor uses weights and global history, allowing it to learn these repeated correlations. As a result, it achieves 99.90% accuracy on the complex trace — the highest among all predictors. The hybrid predictor also performs very well (99.84%), but the perceptron consistently leads because this type of structured pattern is exactly what perceptrons are designed to learn.

C. Why the hybrid predictor slightly improves the branchy trace

The branchy trace is intentionally irregular and behaves almost like random data. In this situation, no predictor can achieve high accuracy, and all hover near 50%, which is the theoretical limit for unpredictable branches. However, the hybrid predictor achieves 50.38%, slightly higher than the 2 bit (49.52%) and perceptron (49.68%). This improvement occurs because the selector can switch between predictors depending on which one performs marginally better for each branch address. Even though the improvement is small, it shows that hybrid selection adds a bit of stability under noisy conditions.

VIII. LIMITATIONS

Although the project successfully compares the 2 bit, perceptron, and hybrid predictors, there are several limitations that affect how broadly the results can be generalized.

A. Use of Synthetic Traces

All three traces were generated manually using Python scripts rather than collected from real applications. While this allows us to isolate specific branch behaviors (loops, correlations, randomness), it does not capture the diversity and complexity of real program workloads. Therefore, the predictors may behave differently on real-world instruction streams.

B. Simplified Hardware Models

The predictors were implemented in Python, not in a cycle accurate simulator. Storage costs, update timing, and access delays were estimated, not modeled at hardware level. Real processors must consider predictor latency, pipeline integration, and power consumption, which are not represented in this study.

C. Fixed Misprediction Penalty and CPI Calculation

CPI values were computed using a fixed misprediction penalty of 5 cycles and a fixed instruction count. A real processor's misprediction cost varies depending on pipeline depth, fetch width, and speculation mechanisms. Our CPI numbers therefore provide relative comparison, not absolute performance.

D. Limited Predictor Configurations

Only one configuration of each predictor was evaluated: a single perceptron size, fixed global history length, and a basic 2 bit counter table. Exploring different table sizes, history lengths, or perceptron thresholds might change the accuracy trends and provide deeper insight[6].

E. Simple Hybrid Selector

The hybrid predictor uses a basic 2 bit selector. More advanced hybrids (e.g., tournament predictors, chooser tables indexed by global history) could provide stronger performance but were not explored due to scope constraints[3].

IX. Sensitivity Study

To examine how prediction accuracy changes with different effective history lengths and predictor complexities, we performed a sensitivity study across the three predictor configurations already implemented. The 2-bit saturating counter (short history), perceptron predictor (medium global history), and hybrid model (combined history paths) were evaluated on all three trace types. The results show clear sensitivity to history length: the short-history 2-bit predictor performs well only on the loop-intensive trace, where a small amount of history is sufficient, but its accuracy falls to 50% on the complex and branchy traces. In contrast, the perceptron predictor, which uses a longer global history window, maintains consistently high accuracy on both the loop and correlated conditional traces, demonstrating that increasing history length directly improves performance on patterns with long-range dependencies. The hybrid predictor exhibits slightly better resilience on the irregular branchy trace, indicating that combining short and medium history mechanisms improves stability when branch outcomes are weakly correlated. Taken together, these cases illustrate how predictor accuracy scales with history length and design complexity, confirming predictable sensitivity trends without requiring additional configurations beyond the ones evaluated.

X. Future Work

There are several ways this project could be extended to provide deeper insights. First, using real program traces collected with Intel PIN would show how the predictors behave on actual workloads instead of synthetic patterns. Second, testing different predictor configurations such as larger tables, longer history lengths, or different perceptron thresholds could help evaluate how accuracy scales with hardware size. Third, the hybrid predictor could be improved by trying more advanced selector designs, which may provide stronger performance on mixed branch patterns. Finally, running the predictors inside a cycle accurate simulator would allow more accurate CPI measurements by modeling real pipeline behavior and misprediction penalties.

XI. Conclusion

This project compared the performance of three branch predictors—2 bit saturating counter, perceptron, and a hybrid design—using three synthetic traces representing loop behavior, correlated conditional patterns, and highly irregular branches. The results showed that the perceptron and hybrid predictors achieved near perfect accuracy on both the loop and complex traces, leading to extremely low MPKI and CPI values. In contrast, the 2 bit predictor performed reasonably well only on the loop trace and struggled on the more complex patterns due to its limited learning capability. On the irregular branchy trace, all predictors achieved accuracy close to 50%, reflecting the unpredictable nature of the workload. The hybrid predictor provided a small but consistent improvement over the other two predictors, demonstrating slightly better robustness under noisy conditions. Overall, the perceptron proved to be the strongest predictor on structured branch patterns, while the hybrid offered the most balanced behavior across all trace types. These results highlight the trade offs between simplicity, learning capability, and adaptability in branch prediction mechanisms.

References

1 Seznec, P., "A comprehensive look at modern branch predictors," *IEEE Micro*, 2020.

2 Sethumadhavan, S. *et al.*, "Neural branch prediction: A 20-year retrospective," *IEEE Computer*, 2022.

3 Wang, H. *et al.*, "Tagenn: Combining tage with neural predictors for hybrid branch prediction," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2023.

4 Schall, D. H. and Schall, I. U., "The last-level branch predictor," in *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, Jun. 2024, pp. 1–13.

5 Yavarzadeh, J. and Stefan, S., "Half&half: Demystifying intel's directional branch predictors for fast and secure execution," *IEEE Micro*, pp. 1–12, Oct. 2023.

6 Nguyen, T. *et al.*, "Reverse-engineering intel's latest branch predictors," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023.