Taming Wild Branches: Overcoming Hard-to-Predict Branches using the Bullseye Predictor

Emet Behrendt
emet@student.ubc.ca
The University of British Columbia

Shing Wai Pun
swpun@student.ubc.ca
The University of British Columbia

Prashant J. Nair
prashantnair@ece.ubc.ca
The University of British Columbia

Abstract

Accurate branch prediction is critical to the performance of modern out-of-order processors. Although the CBP-2016 champion TAGE-SC-L combines geometric-history tables, a statistical corrector, and a loop predictor, more than half of its remaining mispredictions come from a few "hard-to-predict" (H2P) branches. These branches appear under widely varying global histories, so successive allocations in TAGE thrash and are evicted before their usefulness counters mature. Furthermore, studies show that simply enlarging the tables yields only marginal benefit.

We propose augmenting a 159 kB TAGE-SC-L with a lightweight H2P subsystem, which we called the *Bullseye* predictor, that (i) identifies troublesome PCs in a set-associative H2P Identification Table (HIT) and (ii) steers them to a pair of branch-specific perceptron predictors. The first one is driven by hashed local history and the second one by a folded global history. A short trial phase records head-to-head accuracy in an H2P cache. Thereafter, a branch becomes perceptron-resident only if a perceptron's sustained accuracy and output magnitude exceeds dynamic thresholds, after which TAGE updates for that PC are suppressed to avoid pollution. The HIT, cache, and perceptron weights add 28 kB, operate fully in parallel with the TAGE-SC-L predictor, and target the H2P tail with a higher fidelity. This helps deliver, on average an MPKI of 3.4045 and a CycWpPKI of 145.09.

1 Introduction

Modern out-of-order CPUs depend heavily on aggressive speculative execution, where each fetched instruction block is predicted to either fall through or take a branch long before the branch condition is resolved. A single misprediction triggers a pipeline flush, drains the front-end, and requires re-fetching along the correct path, typically incurring a penalty of 15–30 cycles. As a result, the design and accuracy of the branch predictor have direct and significant performance implications, affecting instruction throughput, energy efficiency, and speculation-dependent microarchitectural features.

Branch prediction accuracy remains a first-order design constraint, shaping front-end bandwidth, retirement rates, and security mitigation strategies. Over the past three decades, predictors have evolved from simple bimodal schemes to sophisticated hybrid, perceptron-based, and multi-component designs. The TAGE-SC-L predictor, the CBP-2016 competition winner, combines tagged geometric history tables (TAGE) with a statistical corrector and a loop predictor to cover a broad spectrum of control-flow behaviors. However, despite this architectural complexity, TAGE-SC-L still leaves substantial accuracy on the table, especially for a small but critical subset of dynamic branches.

'6th Championship Branch Prediction (CBP)', June 21, 2025, Tokyo, Japan

Despite TAGE-SC-L's sophisticated integration of long-range geometric histories, statistical correction, and loop-specialized components, its residual mispredictions are heavily skewed. Lin et al. demonstrated that over 50% of mispredictions in a 30-million-instruction SPEC-INT 2017 trace originate from fewer than ten static "hard-to-predict" (H2P) branches [2]. These branches exhibit highly volatile control-flow behavior where consecutive dynamic instances of the same static branch often occur under vastly different global histories. This makes retrieving a consistent predictor state difficult for TAGE's tag-matching mechanism. As a result, new entries are frequently allocated with poor matching confidence. Because TAGE's usefulness is limited to increment only on correct predictions, these entries are quickly deemed ineffective and evicted. This leads to a self-reinforcing cycle of misprediction, reallocation, and eviction, depriving the predictor of the long-lived entries necessary to learn stable correlations for H2P branches.

An H2P branch may see hundreds of distinct TAGE entries within a single millisecond of execution, none surviving long enough to gather confidence. Intuitively, one might expect that adding more storage, such as enlarging each component table or appending extra history lengths, would amortize the thrashing. Yet idealized studies by Seznec [5] and the independent branch-runahead framework of Pruett et al. [3], demonstrate that even a hypothetical TAGE-SC-L with unbounded capacity converges only marginally beyond the accuracy of the published 64 kB design. Capacity alone cannot compensate for the entropy inherent in the branch's context. Thus, the predictor tends to lack a representation capable of generalizing across diverging histories that precede each H2P instance.

We introduce Bullseye, a compact H2P subsystem that augments a 159.3 kB TAGE-SC-L to address these challenges. Bullseye first pinpoints the few branches that dominate the residual error and then hands them off to branch-specific perceptrons. Detection is handled by the H2P Identification Table (HIT), a set-associative array that records each static branch's execution and misprediction counts. A branch becomes H2P-active only after it surpasses adaptive thresholds on executions, mispredictions, and accuracy limits that tighten as the number of active H2P branches grows. Real workloads seldom activate more than eight to ten PCs simultaneously, so the HIT remains small and latency-neutral.

Once Bullseye flags a program counter, the branch enters a brief trial phase: TAGE-SC-L and two perceptron engines predict in parallel while an H2P cache tracks their head-to-head accuracy. Suppose either perceptron maintains higher accuracy than TAGE-SC-L and produces outputs above a dynamic magnitude threshold. In that case, Bullseye promotes the branch to perceptron-resident status and suppresses further TAGE updates for that PC, eliminating table pollution. Branches that fail the trial fall back to the HIT with negligible overhead.

Bullseye's prediction engines comprise two lightweight perceptrons that capture patterns that TAGE's geometric indexing misses. One leverages hashed-window local history for fine-grained, per-branch behavior; the other uses a folded global history vector to learn long-range correlations. During each fetch, the HIT probe, both perceptron evaluations, and the baseline TAGE-SC-L lookup proceed in parallel. A single-cycle arbiter selects a perceptron's output when its confidence dominates; otherwise, TAGE-SC-L supplies the prediction. By confining extra complexity to the stubborn H2P tail, Bullseye sharpens overall accuracy while preserving the critical-path speed of the underlying predictor.

2 Background and Related Work

Modern branch predictors can be broadly classified into TAGE-based and perceptron-based categories.

2.1 The TAGE-based Predictor

The TAGE predictor architecture combines a tagless bimodal table of PC-indexed 2-bit counters with multiple tagged components indexed using geometrically increasing history lengths. Each tagged entry includes a prediction counter, a partial tag, and a usefulness counter. The final prediction is selected from the tagged component with the longest matching history. TAGE-SC-L enhances this core design with two additional components: (i) a statistical corrector (SC) that aggregates predictions from biased, global, and local history tables to override low-confidence TAGE outcomes, and (ii) a loop predictor (L) optimized for detecting constant-iteration loops [6]. TAGE-SC-L was the winning submission in CBP-2016 and serves as the baseline for the predictor proposed in this work.

2.2 The Perceptron-Based Predictor

Perceptron-based predictors take a fundamentally different approach by framing branch prediction as a form of single-layer neural inference. Each prediction is computed as a weighted sum of recent branch outcomes combined with a bias term; the sign of the resulting sum determines whether the branch is predicted as taken or not taken [1]. Weights are updated at retirement based on the correctness of the prediction, allowing the perceptron to learn long-term correlations that traditional counter-based predictors typically miss. While perceptrons are well-suited for linearly separable patterns, they introduce higher latency and storage costs, especially as history lengths grow and weight vectors expand.

2.3 Why use H2P-Tailored Predictors?

While continued process scaling has enabled significantly larger branch predictor budgets, idealized studies reveal diminishing returns in accuracy per kilobyte as predictor size increases [5]. Even with sophisticated designs like TAGE-SC-L, detailed execution traces show that a small number of hard-to-predict (H2P) branches account for a disproportionate share of mispredictions, limiting overall IPC gains [2]. These insights motivate hybrid predictor architectures, such as the one proposed in this work, that retain TAGE-SC-L's low-latency, high-throughput backbone while integrating lightweight perceptron models explicitly targeted at these high-impact, difficult-to-model branches.

3 Bullseye: High-Level Design Overview

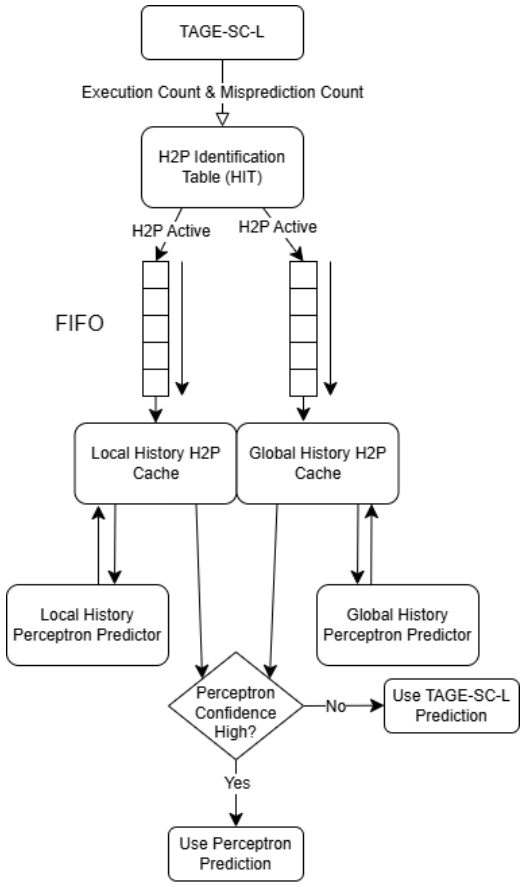

Figure 1: A high-level overview of the architecture of the Bullseye prediction system.

Figure 1 shows the control flow through Bullseye's H2P pipeline. Execution begins on the baseline TAGE-SC-L, which supplies both predictions and two running statistics to the H2P Identification Table (HIT). Namely, the per-PC execution count and misprediction count. When a branch's counters exceed Bullseye's adaptive thresholds, the HIT flags it H2P-active and enqueues the PC, in FIFO order, into two small tag-RAMs: the local-history H2P cache and the global-history H2P cache. These caches hold only a handful of H2P PCs observed in practice ($\leq$10), ensuring constant-time look-ups without inflating front-end latency.

On every fetch, both caches probe in parallel. A hit launches the corresponding local-history or global-history perceptron predictor, which reads its weights, produces an output, and forwards the result to a confidence arbiter. If either perceptron's magnitude and running win rate exceed Bullseye's dynamic threshold, the arbiter overrides TAGE-SC-L with the perceptron prediction; otherwise, the system returns to the baseline forecast. This selective substitution confines the extra latency of neural evaluation to the rare H2P branches while preserving the single-cycle critical path for the typical case, thereby sharpening overall accuracy without compromising pipeline depth.

4 Bullseye Predictor Operation

Our branch predictor uses a naively scaled-up 159.3 kB version of the provided TAGE-SC-L with additional logic to identify and predict H2P branches. This section details the functionality of each key component in the branch predictor.

4.1 Hard-to-Predict Identification Table (HIT)

Bullseye locates key H2P branches with a small, set-associative *Hard-to-Predict Identification Table (HIT)*. Each HIT entry maintains three running statistics for a static branch b:

- $\mathrm{Exec}(b)$: cumulative dynamic executions,
- $\mathrm{Mispred}(b)$: cumulative TAGE-SC-L mispredictions,
- $\mathrm{Acc}(b) = 1 - \mathrm{Mispred}(b)/\mathrm{Exec}(b)$: running accuracy.

4.1.1 Running Statistics. Let N_{H2P} denote the *current* number of branches already classified as hard-to-predict (H2P-active) and therefore resident in the perceptron layer. A new branch becomes H2P-active, and is queued into both the local- and global-history H2P caches, when it first satisfies the adaptive rule set in Equations (1) (a–d):

$$\mathrm{Exec}(b) \;\geq\; 2048 + 16\,N_{\mathrm{H2P}}, \tag{1a}$$

$$\mathrm{Mispred}(b) \;\geq\; 256, \tag{1b}$$

$$\mathrm{Acc}(b) \;<\; f\big(N_{\mathrm{H2P}}\big). \tag{1c}$$

$$f(N) = \begin{cases} 1 - 0.01\,N/32, & N < 32, \\ 0.95 - 0.01(N - 32), & 32 \leq N \leq 71, \\ 0.60, & N > 71. \end{cases} \tag{1d}$$

4.1.2 Leveraging the Statistics. Equation (1a) raises the execution threshold by 16 for every additional H2P-active branch, ensuring that transient bursts cannot flood the perceptron layer. Equation (1b) enforces a fixed lower bound of 256 mispredictions, filtering out seldom-executed yet highly biased branches. The piece-wise function $f(N)$ in Equation (1d) tightens the accuracy ceiling as N_{H2P} grows, dropping from $> 95\%$ when the perceptron layer is empty to 60% at saturation ($N_{\mathrm{H2P}} > 71$). Because empirical workloads rarely exceed $N_{\mathrm{H2P}} \leq 10$, the HIT remains compact.

4.2 H2P Cache: Trial, Admission, and Eviction

After the HIT flags a branch as *H2P-active*, its PC is inserted into both the local- and global-history perceptron engines and into a small, fully- associative **H2P cache**. The cache serves two purposes: (i) it records which PCs are currently predicted by Bullseye's perceptrons and (ii) it mediates a head-to-head "trial" between each perceptron and the baseline TAGE-SC-L.

4.2.1 Trial phase. A newly admitted entry is granted a *warm-up window* of **512** dynamic occurrences to train the perceptron weights. During this window, the branch is never considered for eviction. During and after the trial phase, two counters are used to determine H2P prediction confidence. Every prediction updates a saturating *relative performance* counter based on the winning predictor. As the performance metric saturates, the stability of the *relative performance* counter is then measured with a *confidence counter* with *linear growth and exponential decay*. Confidence is incremented by $C \leftarrow \min(C{+}1,\ 255)$ if the current *relative performance* trend continues and $C \leftarrow C/2$ if the update goes against the running trend. This creates a policy that slowly rewards sustained predictor superiority yet quickly penalizes failing branches.

4.2.2 Eviction policy. A branch is evicted when either (i) the *confidence counter* has saturated in favor of TAGE-SC-L or (ii) the entry is not referenced for 2^{16} dynamic branches ("stale" timeout). Eviction occurs only when at least one HIT-qualifying branch is waiting to enter, thus guaranteeing high utility for every occupied slot. Branches evicted from the cache revert to standard TAGE-SC-L prediction but may re-enter if they later satisfy the H2P criteria in Equations (1) (a–d).

This gating mechanism ensures that Bullseye deploys its perceptron resources *only* where they continue to beat the geometric core while bounding both storage and latency. Empirically, no more than 10 PCs reside in the cache simultaneously, keeping look-ups single-cycle and energy-efficient.

4.3 Hashed-Window Local-History Perceptron

Bullseye's first neural component targets correlations in branch's *local history*. For each H2P-active PC, the predictor constructs a feature vector whose i^{th} element is the parity of a fixed-width *window* W_i of the branch's outcome history, where window sizes grow with age (e.g., W_0=4, W_1=8, W_2=16, …). Successive windows start at offsets separated by a constant *stride* S, ensuring non-overlapping coverage of up to a few hundred past outcomes.

Each feature is mapped to two independent weight words by the hash $h_i(\mathrm{PC}, W_i)$, implemented as a 32-bit XOR-shift scrambler. Dual hashing mitigates aliasing: a collision in one weight location is usually resolved by the second. The perceptron output is the integer sum of the selected weights plus a bias term. Finally, the sign of the output determines the prediction.

Weights are updated using the *dynamic-threshold* rule of Seznec and Vintan's O-GEHL predictor [4]. Specifically, a global threshold θ tracks the absolute output magnitude at misprediction time and adapts toward the smallest value that keeps training activity near 50%. A weight w is incremented (decremented) when the actual outcome is taken (not-taken) *and* $|\mathrm{output}| \leq \theta$, enabling fast convergence without saturation. This hashed-window design yields high resolution on recent local patterns while maintaining low storage per perceptron.

4.4 Folded Global-History Perceptron

Bullseye adds a global-history perceptron that captures long-range, cross-branch correlations. The feature vector consists of the most recent H_g outcomes from the *global* branch history register, folded by XOR into a fixed W_g-bit index; each bit selects a single signed weight word. With a minimal number of weights, the size of the global history perceptron is an order magnitude smaller than the local model.

Prediction and learning follow the same *dynamic-threshold* rule outlined in Section 4.3. Specifically, the perceptron sums its W_g signed weights and compares the magnitude to a shared, runtime-tuned

threshold θ:

$$\text{out}_g = \sum_{i=0}^{W_g-1} w_i \cdot x_i, \quad \text{prediction} = \text{sign}(\text{out}_g).$$

Weights are updated only when $|\text{out}_g| \leq \theta$ or the prediction is incorrect, ensuring rapid adaptation and bounded training costs.

Although its standalone accuracy gain is modest (< 1% BPC), the global perceptron provides a *backup view* that often corrects rare, history-spanning patterns missed by both the local model and TAGE-SC-L, while adding negligible footprint.

4.5 Prediction Arbitration

Whenever an H2P-resident branch is fetched, *all* engines fire concurrently: the baseline TAGE tables, the statistical corrector (SC), and Bullseye's two perceptrons. The final outcome is chosen by a lightweight, confidence-based arbiter:

i **Perceptron gate**: Each perceptron supplies a two-bit *conf* field that encodes (a) its running win-rate over TAGE-SC-L (*high* if win-rate ≥ 55%), and (b) whether |output| > θ for the current instance. A perceptron asserts *strong* confidence only when both tests pass.

ii **TAGE gate**: TAGE-SC-L asserts *strong* confidence when (a) the selected TAGE component's usefulness= 3 **or** (b) the SC overrides with a magnitude > 0.

iii **Decision rule**: If at least one perceptron has *strong* confidence and TAGE-SC-L does *not*, Bullseye chooses the perceptron outcome. Otherwise, the arbiter defaults to TAGE-SC-L. This rule preserves accuracy on easy branches while allowing neural takeover only during sustained benefits.

iv **Decision rule**: If at least one perceptron has *strong* confidence, Bullseye chooses the perceptron outcome. Otherwise, the arbiter defaults to TAGE-SC-L. This rule preserves accuracy on easy branches while allowing neural takeover only during sustained benefits.

4.6 Selective TAGE Filtering

Once a branch has delivered 128 consecutive correct perceptron predictions *without* a single TAGE-SC-L win, its updates are **filtered** – i.e., subsequent outcomes bypass all TAGE and SC tables. Filtering prevents low-utility data from evicting well-trained entries and cuts energy by avoiding unneeded SRAM writes. If the perceptron later falters (confidence drops below the *strong* threshold), filtering is automatically revoked, ensuring that valuable geometric history is never lost permanently. Empirically, this mechanism yields a modest accuracy gain (< 0.3% BPC) for negligible extra storage.

5 Experimental Results and Analysis

We present results for CycWpPKI and BrMisPKI metrics. Overall, Bullseye achieves an average CycWpPKI of 145.09 and an average BrMisPKI of 3.405.

6 Discussion

Bullseye's strength lies in its trace-invariant targeting of pathological control flow: because the HIT thresholds scale with the instantaneous H2P population rather than absolute miss rates, the predictor provides a noticeable misprediction reduction. Its mechanisms are workload-agnostic, the underlying TAGE-SC-L handles the common case, while the perceptron tier activates only for branches that statistically qualify, so porting to server, mobile, or mixed integer–floating-point suites requires no retuning. Furthermore, the lightweight H2P identification mechanism of Bullseye can be augmented with any other predictor. The main limitation is that Bullseye ignores data-dependent correlations, an avenue we explored but could not surpass marginal accuracy gains within our area budget, leaving room for future H2P engines that fuse branch history with lightweight data value predictors.

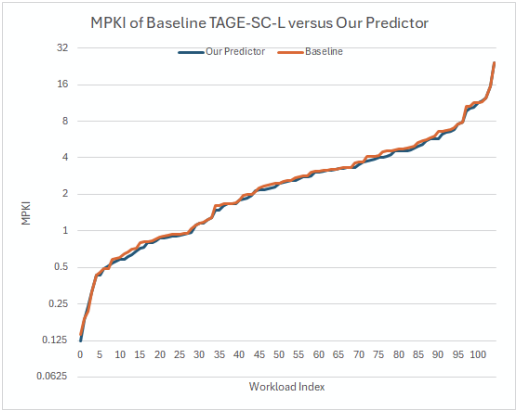

Figure 2: The BrMisPKI across all workloads for Bullseye.

Branch Predictor	BrMisPKI
159 kB TAGE-SC-L + Bullseye (Total 187 kB)	3.4045
192 kB TAGE-SC-L	3.4277
159 kB TAGE-SC-L	3.4513

Table 1: Comparison of BrMisPKI Across Predictors

7 Conclusions

Bullseye demonstrates that selectively augmenting a compact 159 kB TAGE-SC-L with a lightweight, H2P-aware neural tier yields a disproportionate return on accuracy. The Hard-to-Predict Identification Table (HIT) dynamically isolates the few static branches that dominate the misprediction tail. It then admits them to local- and global-history perceptrons for a gated trial. Thereafter, it maintains thresholds that prevent resource thrashing. A confidence-based arbiter ensures that neural predictions override the TAGE-SC-L predictor only when they deliver sustained benefit, while update filtering shields TAGE-SC-L from low-value training noise. Overall, our experiments show that Bullseye achieves an MPKI of 3.405.

References

[1] D.A. Jimenez and C. Lin. 2001. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. 197–206. https://doi.org/10.1109/HPCA.2001.903263

[2] Chit-Kwan Lin and Stephen J. Tarsa. 2019. Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 228–238. https://doi.org/10.1109/iiswc47752.2019.9042108

[3] Stephen Pruett and Yale Patt. 2021. Branch Runahead: An Alternative to Branch Prediction for Impossible to Predict Branches. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-54)*. ACM, 804–815. https://doi.org/10.1145/3466752.3480053

[4] A. Seznec. 2005. Analysis of the O-GEometric history length branch predictor. In *32nd International Symposium on Computer Architecture (ISCA'05)*. 394–405. https://doi.org/10.1109/ISCA.2005.13

[5] André Seznec. 2007. The Idealistic GTL Predictor. *J. Instr. Level Parallelism* 9 (2007). http://www.jilp.org/vol9/v9paper7.pdf

[6] André Seznec and Pierre Michaud. 2006. A case for (partially) tagged geometric history length branch prediction. https://inria.hal.science/hal-03408381/document

A Cost Analysis

Table 2 provides a breakdown of the memory usage by each component.

Between prediction time and update time the following must be stored: for the local history perceptron, the local history; for the global history perceptron, the global history; and for TAGE-SC-L the requirements are unchanged from the base version with the same local and global history required to update.

Component	Details of each field of each entry and memory breakdown	Cost
TAGE-SC-L	Calculated using the built in TAGE-SC-L memory usage calculator	159.34 kB
H2P Identification Table (HIT Cache)	PC Tag: 10 bits (16 bit tag, but 10 bits stored with set-association) * 2^6 sets * 8 ways = 5120 bits Correct Prediction Counters: 16 bits * 2^6 sets * 8 ways = 8192 bits Incorrect Prediction Counters: 12 bits * 2^6 sets * 8 ways = 6144 bits	2.375 kB
Global History Perceptron and FIFO	H2P PC Tag: 62 bit PC * 16 H2P entries = 992 bits PC Queue: 62 PC bits * 64 queue entries = 3968 bits Global History: 128 bits Weights: 12 bit precision * 128 tables * 16 H2P entries = 24576 bits Perceptron Bias: 10 bit precision * 2^4 table entries * 16 H2P entries = 2560 Update Thresh. Counters: (14+7) * 16 H2P entries = 336 bits Branch Management Counters: (6+8+9+16) * 16 H2P entries = 624 bits	4.05 kB
Local History Perceptron and FIFO	H2P PC Tag: 62 bit PC * 32 H2P entries = 1984 bits PC Queue: 62 PC bits * 64 queue entries = 3968 bits Weights: 10 bit precision * 64 tables * table size 2^8 = 163840 bits Local History: 124 bit history * 32 H2P entries = 3968 bits Perceptron Bias: 12 bit precision * 2^1 table entries * 32 H2P entries = 768 bits Update Thresh. Counters: (10+7) * 32 H2P entries = 544 bits Branch Management Counters: (6+8+9+16) * 32 H2P entries = 1248 bits	21.52 kB
TOTAL		187.28 kB

Table 2: Memory Usage Breakdown