A Comprehensive analysis of the CBP 2025 Branch Predictors entries: LVCP, RVA-Toru, and TAGE-SC 2025

Saumya Patel

February 6, 2026

Contents

1 Introduction **1**

2 Load Value Correlator Predictor (LVCP) **2**
2.1 Details . . . 2
2.2 Load Value Correlator vs Baseline . . . 2
2.2.1 Performance Summary . . . 2
2.2.2 Analysis . . . 3
2.3 Discussion: Why LVCP Helps Some Workloads More Than Others . . . 3
2.3.1 When Load Values Actually Help . . . 3
2.3.2 Why Web Benchmarks Improve a Lot . . . 3
2.3.3 How This Differs from Register-Value-Aware Predictors . . . 3
2.3.4 Limits: Timing and Tracking Size . . . 3

3 Register-Value-Aware Branch Predictor **5**
3.1 Register Value Aware Predictor vs Baseline Predictor . . . 6
3.1.1 Summary Statistics . . . 7

4 TAGE-SC 2025 André Seznec Report **8**
4.1 Overview . . . 8
4.2 Optimization Features of TAGE 2025 . . . 8
4.3 TAGE SC 2025 vs TAGE SC-L 2016 . . . 9
4.3.1 Performance by Framework Type . . . 10
4.3.2 Analysis . . . 10
4.3.3 Top 20 Most Improved Traces . . . 10

5 TAGE-SC-L Alberto Ros Predictor Analysis **12**
5.1 Outline of the Predictor . . . 12
5.2 Predictor vs Baseline Predictor . . . 12
5.2.1 Framework Performance Comparison . . . 13
5.3 Direct Predictor Comparison . . . 13
5.3.1 Summary Statistics . . . 13
5.3.2 Trace-by-Trace Comparison (MPKI) . . . 13
5.3.3 Biggest Improvements (Seznec Better) . . . 13
5.3.4 Biggest Regressions (Seznec Worse) . . . 14

6 Delta Analyses and Conclusions **15**
6.1 LVCP vs. RVA-Toru: The Generalization of Value Prediction 15
6.1.1 Analysis of the Delta 15
6.1.2 Visual Analysis of the Performance Gap 15
6.1.3 Where LVCP Still Wins 16
6.2 TAGE-SC-L (Seznec) vs. RVA-Toru: History vs. Data 16
6.2.1 Analysis of the Delta 16
6.3 LVCP vs. TAGE-SC-L: Load Data vs. Control-Flow History 17
6.4 TAGE-SCL (Seznec) vs. TAGE-SC-L (Ros): Implementation Trade-offs 17
6.4.1 Analysis of the Delta 17

7 Overall Predictor Rankings **18**

8 Conclusion **18**

9 Scope for Future Work: Exploring the Limits of Branch Prediction **19**
9.1 Phase 1: Studying Alternative Prediction Styles 19
9.2 Phase 2: An Idealized "Infinite-Hardware" Oracle 20

10 References **21**

1 Introduction

The field of branch prediction has reached a level of sophistication where further performance gains are rarely achieved through incremental structural modifications. Instead, meaningful advancements increasingly depend on a granular understanding of why certain branches remain mispredicted and which specific information sources, whether path history, data values, or execution context, are most capable of resolving them. The 2025 Championship Branch Prediction (CBP) competition offers a unique framework to explore these questions across a diverse range of workloads and state-of-the-art architectures evaluated under a unified methodology.

This report represents a critical continuation of my multi-stage investigation into high-performance branch prediction. This work picks up directly from my second assignment, where I analyzed the 2016-era TAGE-SC-L baseline to identify the fundamental bottlenecks inherent in traditional history-based techniques. By establishing that reference point, I was able to determine which workloads are inherently difficult to predict and precisely where mispredictions tend to concentrate when the system relies solely on path history.

Building upon those initial findings, the objective of the current study is to analyze "optimization deltas." Rather than merely reproducing established results, this report examines how specific architectural innovations introduced in the CBP 2025 entries alter prediction behavior relative to established baselines. The analytical focus shifts from aggregate averages to trace-level deltas, worst-case performance, and workload-specific trends. This approach is designed to distinguish between mechanisms that address fundamental bottlenecks and those that simply reshuffle performance across different benchmarks.

To achieve this, the report investigates three distinct classes of predictors. First, the Load Value Correlator Predictor (LVCP) is evaluated as a targeted attempt to resolve data-dependent branches through explicit correlation with recent memory load values. Second, the Register-Value-Aware predictor (RVA-Toru) is examined for its ability to generalize this concept by incorporating register values directly into the prediction process. Finally, refined history-based methods, including the 2025 TAGE-SC design by André Seznec, are analyzed to determine the limits of traditional path-based correlation when compared to modern value-aware models.

Throughout this analysis, the goal is to look beyond industry standard metric of Mispredictions Per Kilo-Instruction (MPKI) reductions. By examining where regressions occur and how misprediction ceilings evolve, this study extracts broader architectural insights into the relationship between history, data, and design complexity. Ultimately, this report serves as both a performance evaluation and an exploratory study into the future of robust and effective branch prediction.

2 Load Value Correlator Predictor (LVCP)

2.1 Details

The main idea is that TAGE can struggle with hard-to-predict (h2p) branches when the decision depends on data values. TAGE mainly uses branch history (taken / not taken patterns). But for some branches, the same history can lead to different outcomes because the decision is based on a value loaded from memory, not just the path.

LVCP helps by using three key structures:

- **Load Marking Table**: Marks which loads are worth tracking.
- **Load Tracking Queue**: Keeps recent loads (their PC and value) so branches can look them up.
- **Correlation Table**: Uses (branch PC + recent load info) to predict taken/not-taken when there is a strong match.

A key question is: which load should the branch use? LVCP focuses on recent loads. If a load is too old, it becomes less useful and can be moved out of the "recent" set. **Another question is: how does it connect a load to a branch?** The branch looks back at recent loads and uses a hash of the branch PC, the load PC, and the load value. If that same combination happened before, LVCP can reuse what happened last time (taken or not taken).

2.2 Load Value Correlator vs Baseline

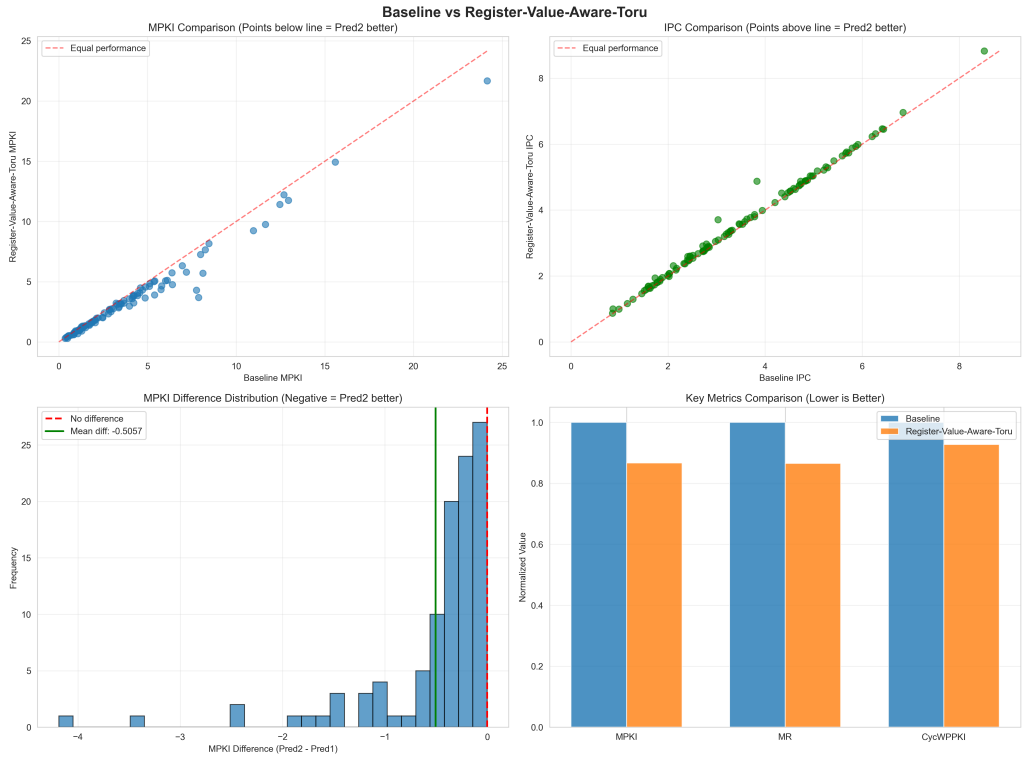

Figure 1: Load Value Correlator Predictor vs Baseline Performance

2.2.1 Performance Summary

Better performances in how many frameworks of each type:

Framework	Improve/Total	Success Ratio (%)	Total MPKI Improvement
Web	23/26	88.5	-10.4414
Int	20/37	54.1	-14.4377
Fp	5/14	35.7	-2.9306
Infra	9/16	56.3	-2.1943
Compress	1/8	12.5	-0.1299
Media	0/4	0.0	0.0000

Table 1: Performance comparison across different benchmark types

2.2.2 Analysis

As usual, it performs better than the baseline TAGE predictor. The biggest gains show up in the web benchmarks. A likely reason is that web workloads often do a lot of checks on input/state that come from memory. That creates many "load then branch" patterns. For example:

```
if (request.type == GET) { ... }
```

This would mean that usually, how these JavaScript intensive workloads are structured is that they would need memory access, and this would mean that there would be a lot of loads and stores. The load value correlator is exceptional with loads before branches. In the example above, there would be a load before the branch and this means the load value correlator would shine.

The largest overall MPKI improvement comes from the integer group. That also makes sense: many integer workloads load values from memory and then branch based on those values.

2.3 Discussion: Why LVCP Helps Some Workloads More Than Others

2.3.1 When Load Values Actually Help

LVCP works best when a branch decision depends directly on a value that was just loaded from memory. In those cases, the last load is a strong hint for what the branch will do.

If the branch depends on several values, or on values that are computed after the load (math, multiple steps, combined state), then one load value is not enough. In that situation, LVCP helps less. This is why the results are uneven. Web benchmarks improve a lot, while FP, compression, and media often show smaller gains.

2.3.2 Why Web Benchmarks Improve a Lot

Web code often follows a simple pattern: load a field, compare it, then branch. For example, reading a request type or a flag and immediately checking it. Because the load and the branch are close together, LVCP can usually find the "right" recent load and use it to predict the branch.

2.3.3 How This Differs from Register-Value-Aware Predictors

LVCP ties branch behavior to values coming from memory loads. Register-value-aware predictors look at register values, which can include both loaded values and values produced by computation. That makes them more flexible across different workloads because they can capture more kinds of data dependence than "just the last load".

2.3.4 Limits: Timing and Tracking Size

LVCP also depends on timing. If the useful load happened too long before the branch, or many other loads happen in between, the best load might not be the one LVCP picks. Also, LVCP

tracks only a limited set of recent loads. That works for "simple and recent" cases, but not for branches that depend on long-lived or complex program state.

Summary: In short, LVCP shines when control flow is tightly linked to a recent memory load (common in web-style code). It is less helpful when branches depend on longer computations or more complex state. That is why it is a strong but targeted tool for data-dependent branches.

3 Register-Value-Aware Branch Predictor

This was a pretty interesting read, and there were a lot of interesting methods that they used.

The RVA-Toru predictor uses the TAGE predictor as the foundation, and it consists of multiple tagged tables and a statistical correlator. They have also increased the size of the bimodal predictor to 128k entries.

The key innovation in this predictor is the register value awareness. Essentially this predictor uses register values to help predict branches. That is pretty interesting because initially, I thought that it would be very expensive to just compare register values every time; register reads could add to the overhead. But as I read the paper more, I was quite intrigued by the techniques that they used. I was pleasantly surprised by the quality of branch prediction as it beats André Seznec's 2025 TAGE-SC-L.

Something that I found different/interesting was that they use the term "digest" when it is just a summary. Nevertheless, a digest is a 12-bit summary of a 64-bit register value; these digests are used to track correlations between specific data patterns (like loop counters) and branch outcomes. The digests are generated based on the type of data; for example, for INT registers the digest includes the count of leading zeros, but for FP registers they use the MSBs of the exponent. Once the digests are generated they are fed into the statistical register. It then organizes the registers into banks, generates a usefulness table, and picks the most useful register. The selected digest is used to look up the prediction in the prediction table. It outputs the actual predicted branch direction.

The register component turned out to provide the largest and broadest gain out of all the other features. It was the only feature that improved accuracy across every benchmark.

RVA-Toru flags an entry as newly allocated instead of using u-bits to prevent eviction. The system tracks two outcomes for these entries: success and waste.

Apart from the register optimizations, RVA-Toru combines an arithmetic progression with a geometric one, which turned out to be near-optimal. They also improved the IMLI and added the call stack component to the statistical correlator. This can differentiate identical branches that might be called from different functions. They also increased the size of the base predictor as mentioned above.

3.1 Register Value Aware Predictor vs Baseline Predictor

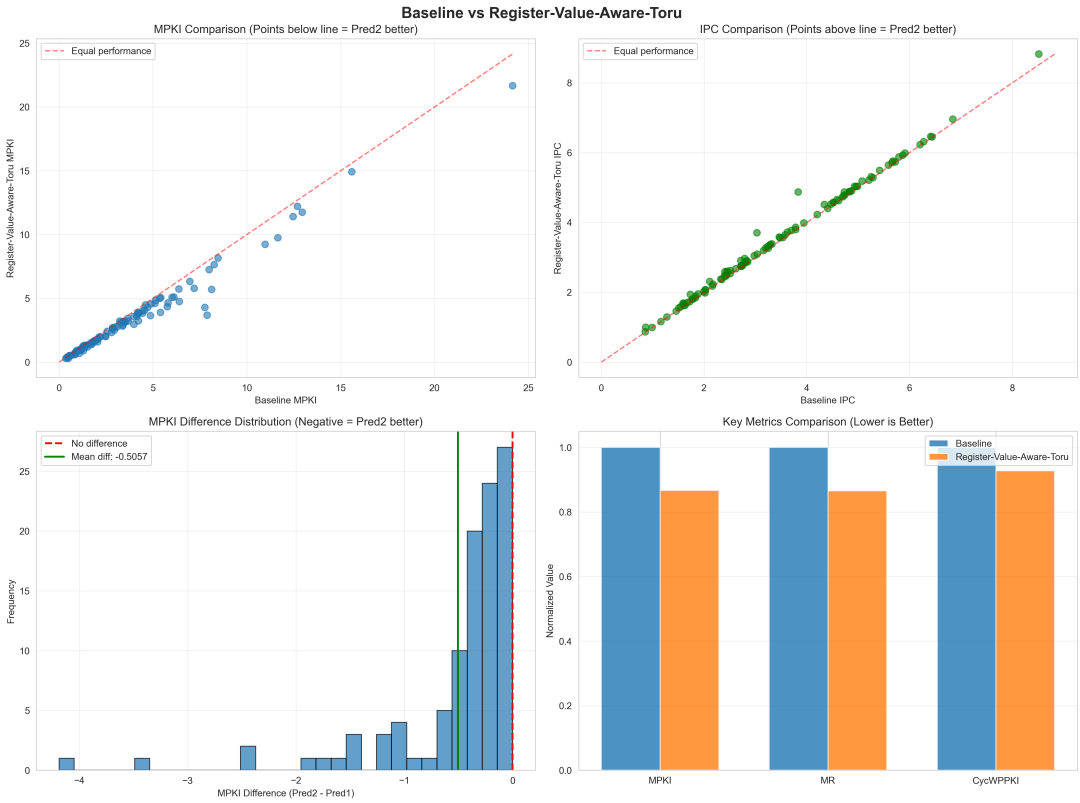

Figure 2: Register Value Aware Predictor vs Baseline Comparison

Okay first things first, in the graph (bottom right) I can see that there is a significant gain in MPKI. In terms of raw numbers, the MPKI for the register value aware predictor is 13.28% better than the baseline. There "isn't much" gain in IPC but the misprediction rate is lesser for the predictor. The IPC improvements tell me that the improvements are central to hard-to-predict branches. I'm pretty sure most of these improvements are because IRL most branches are value dependent and this predictor exactly exploits that. Since the predictor sits on top of TAGE with its smart use of registers to predict branches, it is able to maximize its gains.

From the graphs, I can see near-zero regressions. It means that the predictor only helps when it is useful and when the baseline doesn't perform the best. When the values don't correlate, the predictor falls back to baseline behavior.

A good thing is that around 15/112 benchmarks reported "similar" MPKIs, so the rest of them have significant or "pretty good" gains in MPKI.

It is not very surprising that a lot of the improvements in the MPKI (where the difference is more than 1) belong to the INT benchmark. Now that intuitively makes sense since most of the INT benchmarks are used to do integer computations, array indexing, control flow decisions, etc. But another thing to consider is that the success rate (i.e., #improved/total) is 75%, which means that INT benchmarks have a wide variety. I'm assuming it does well when it involves branches that use register values, and it turns out that the MPKI speedups are very noisy, either having significant improvements or having little to no improvement, mostly leaning towards the improvement side. RVA-Toru performs well on INT because it exploits value.

Metric	Baseline	RVA-Toru	Difference
MPKI	3.8080	3.3022	↓ 13.28% better
MR	2.9679	2.5681	↓ 13.47% better
IPC	3.4636	3.5450	↑ 2.35% better
Cycles	14824660.91	14401963.47	↓ 2.85% better
BrPerCyc	0.4203	0.4309	↑ 2.54% better
MispBrPerCyc	0.0100	0.0089	↓ 10.23% worse
CycWPPKI	161.6047	149.9141	↓ 7.23% better

Table 2: Summary statistics comparison

3.1.1 Summary Statistics

Obviously `MispBrPerCyc` would improve if MPKI improves. Moreover, all benchmarks show any sort of visible improvement (even though we might call them similar due to inherent error margins and noise).

Another thing that was quite noticeable is that MPKI reductions occur on traces with already high baseline MPKI, suggesting that the predictor sort of "clutches" on branches where history-based correlation is weak. This indicates that the register value aware predictor is able to exploit value correlation where history-based predictors fail.

The variance in INT benchmark improvements reflects the heterogeneity of integer workloads. While value-dependent control benefits significantly from register correlation, pointer-driven and indirect branches limit the effectiveness of value-aware mechanisms, leading to uneven but predominantly positive gains.

One other good thing about the predictor is that it is able to defer to the baseline TAGE predictor when register value correlation is weak or absent, preventing performance degradation. This is evident from the lack of significant regressions across benchmarks.

4 TAGE-SC 2025 André Seznec Report

4.1 Overview

By digging around, and looking at the TAGE implementation of the same TAGE-SC predictor of 2016, I found that it had some differences in the MPKI as mentioned in the abstract of the paper. Below are the differences:

- MPKI Difference is that the 2025 TAGE-SC shows a roughly 15.6% lower MPKI (3.363 vs 3.986), a drop of 0.623 MPKI, though direct comparison is approximate due to different trace sets (CBP-5 vs CBP2025).
- 2016 Optimizations: Bank-interleaving for TAGE tables, partial associativity on medium histories (2-3% MPKI gain), enhanced neural SC with IMLI counters, global backward history, and multiple local histories (total SC benefit ∼8%).
- Both use TAGE core (geometric histories, tagged tables), SC for bias correction, but the 2025 predictor focused on more SC tables and new history forms like region/target IMLI for hard-to-predict branches.

The TAGE 2016 predictor was not designed to be effective in hardware, but only to win a competition, due to the unreasonable number of SC tables in the 2016 implementation. Later a realistic TAGE predictor was presented and as cited:

> "Realistic" meaning that the author estimated that it could be implemented for an aggressive instruction front-end predicting an instruction block with up to 4 branches (at most one taken) per cycle.

```
The CBP2025 TAGE-SC is derived from CBP2016 TAGE-SC-L, and replicates
   most of the features that would prevent any reasonable direct
   hardware implementation: huge number of distinct tables, complete
   table interleaving in TAGE, use of local histories, unrealistic
   prediction latency, .. It features the new optimizations on
   allocation/replacement policy on TAGE-SC proposed in [11] as well as
    the optimizations on the IMLI components in SC.
```

4.2 Optimization Features of TAGE 2025

The optimization features of the TAGE 2025 that aren't there in the 2016 version:

- On a misprediction, a lot of the entries from different tables are allocated at the first time, and by setting the U-counter of the first entry, it is protected against replacement. Moreover, the U-counter is also set directly to 2 which supports faster eviction.
- It uses probabilistic counters that are determined by the confidence in the prediction (provided by the longest matching counter), to filter the allocation of entries.
- Uses 2-way skewed associativity.

The structural correlator has also been "improved" or so they say. Most of these optimizations are done from the article/book about TAGE in 2024 *[André Seznec. 2024. TAGE: an engineering cookbook. Technical Report 9561]*.

Something that I found pretty interesting was that they use a tagged IMLI (Inner Most Loop Iterator) to solve the issue of the fall-through misprediction in the last loop iteration. It

is tagged because considering two loops, you would notice that the counters would overwrite each other if it was just a single IMLI, so to solve that they have a tagged IMLI. **Something that was interesting to me is that they use two IMLI tables, and both of them have different purposes.** The other one is the branch context IMLI; it is useful for branches that don't have fixed loop iterations, so they use histories based on the previous anchor branch. Anchor branches do cause biases, but due to the correlator in a BrIMLI, it will make a safe biased prediction; it is still counting based rather than pattern based but it is kinda more biased now.

The branch predictor still uses 2 global history-based components from the GEHL (geo history length predictor):

- XOR PC and GHR
- XOR PC with (longest matching prediction and global history)

Funny thing, they got rid of the loop predictor; it did not seem to help a lot apparently. Marginal gains and it took space also so they nuked it lmao.

4.3 TAGE SC 2025 vs TAGE SC-L 2016

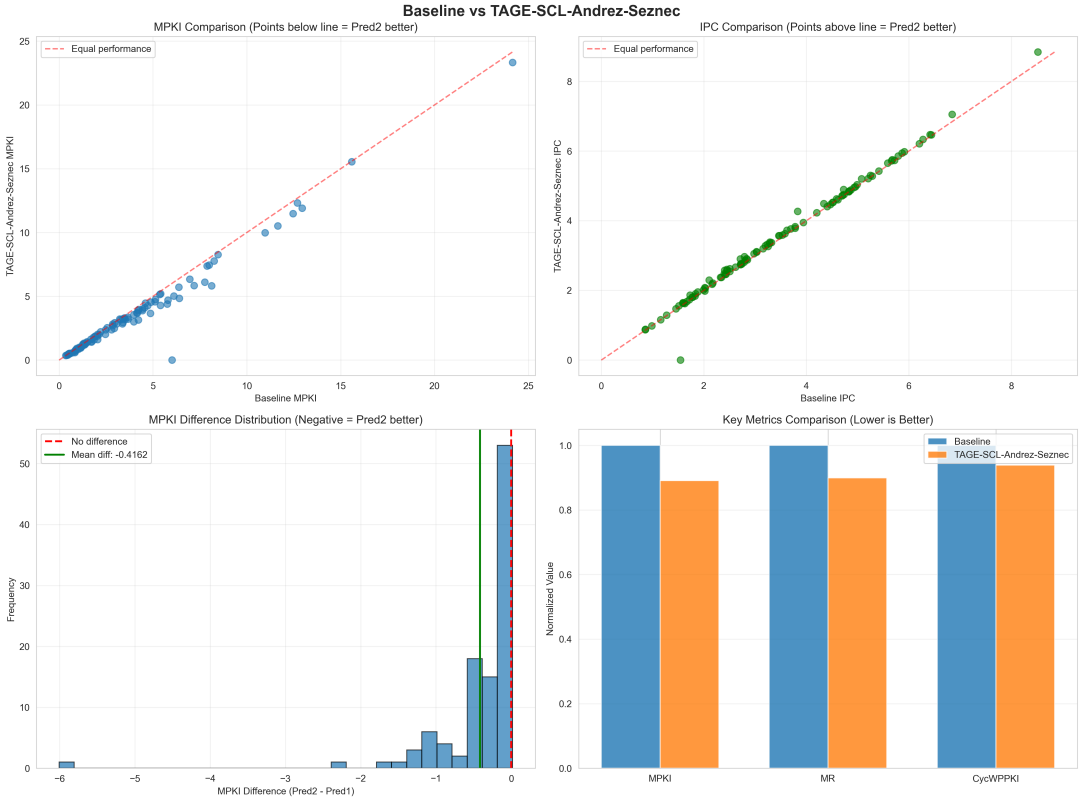

Figure 3: TAGE-SC 2025 vs Baseline Comparison

If we take a look at the plots we can see that for the top right plot, the MPKI of the TAGE-SC 2025 is slightly below the baseline predictor. There are some outliers.

My plot generating scripts and report generating scripts report similar numbers if the difference in the MPKI is more than 0.1, and I feel like I had to set that threshold to account for any noise or errors in the execution. I feel like 0.1 is a lot and I should have set the $\pm$ variation to be 0.05 but I feel like it is better to be safe than not.

But apart from that, most of the MPKIs for all the traces were above the threshold of 0.1 which is good. To be specific, what I found most interesting was that the web benchmarks were

deemed the most hard to improve in the first report that I presented initially, but here the web benchmarks show the most improvements. It is probably because of the probabilistic allocation of entries, since you now have a confidence level that is provided by the longest matching counter.

4.3.1 Performance by Framework Type

Better performances in how many frameworks of each type:

Framework	Improve/Total	Success Ratio (%)
Web	25/26	96.2
Fp	9/14	64.3
Int	25/37	67.6
Infra	7/16	43.8
Compress	1/8	12.5

Table 3: Performance improvement by framework type

4.3.2 Analysis

The results show that the TAGE-SC 2025 predictor achieves a consistently lower MPKI than the baseline implementation, with an overall improvement of approximately 11

A notable exception appears in a small subset of the infrastructure benchmarks. Of the few cases where the baseline predictor performs better, three out of four occur within the *infra* category. While these differences are minor in magnitude, they suggest a systematic pattern rather than random noise.

One plausible explanation relates to the learning behavior of the two predictors. The TAGE-SC 2025 design introduces several mechanisms, such as confidence-filtered allocation, aggressive entry protection, and multi-allocation policies, that intentionally slow down or suppress learning for branches deemed noisy. These features are highly effective for modern, irregular workloads but can be less suitable for small, regular loops commonly found in infrastructure benchmarks. In contrast, the baseline design updates its prediction structures more directly and deterministically, allowing it to converge faster on simple, repetitive patterns.

In this context, *deterministic convergence* refers to the straightforward saturation of prediction counters based solely on observed branch outcomes, without delayed updates or confidence gating. For stable loops with highly predictable behavior, this simpler learning process can occasionally provide a slight advantage. However, it is important to note that the observed differences are extremely small (on the order of ± 0.01 MPKI), suggesting that these regressions are not practically significant and may be partially attributable to measurement noise.

Finally, the largest performance gains are observed in the integer (INT) benchmarks. This behavior aligns well with expectations: INT workloads tend to be branch-heavy and exhibit irregular control flow due to complex program logic and compile-time decision points. The TAGE-SC 2025 predictor excels in this regime, benefiting from its additional IMLI tables and enhanced allocation strategies, which are specifically designed to handle noisy and alias-prone branch behavior.

Overall, these results reinforce the conclusion that TAGE-SC 2025 represents a clear improvement over the baseline, particularly for modern, complex workloads, while only marginally underperforming in a small number of simple, highly regular cases.

4.3.3 Top 20 Most Improved Traces

Table 4: Top 20 Most Improved Traces

#	Trace	Baseline MPKI	TAGE-SCL MPKI	Improvement	Status
1	web_19_trace	6.0075	0.0000	-6.0075	✓ Better
2	int_16_trace	8.1253	5.8279	-2.2974	✓ Better
3	web_7_trace	7.7526	6.1046	-1.6480	✓ Better
4	int_17_trace	6.3949	4.8436	-1.5513	✓ Better
5	int_7_trace	5.7589	4.3978	-1.3611	✓ Better
6	int_8_trace	7.1931	5.8380	-1.3551	✓ Better
7	int_9_trace	4.8555	3.6628	-1.1927	✓ Better
8	int_1_trace	11.6458	10.5130	-1.1328	✓ Better
9	int_22_trace	5.3965	4.2827	-1.1138	✓ Better
10	web_11_trace	5.7895	4.7055	-1.0840	✓ Better
11	web_15_trace	6.1054	5.0241	-1.0813	✓ Better
12	web_20_trace	4.2180	3.1562	-1.0618	✓ Better
13	int_30_trace	12.9427	11.9056	-1.0371	✓ Better
14	int_29_trace	12.4572	11.4747	-0.9825	✓ Better
15	int_36_trace	3.9821	3.0001	-0.9820	✓ Better
16	int_2_trace	10.9652	9.9876	-0.9776	✓ Better
17	int_21_trace	24.1502	23.3263	-0.8239	✓ Better
18	web_25_trace	6.3793	5.7130	-0.6663	✓ Better
19	int_32_trace	6.9551	6.3487	-0.6064	✓ Better
20	web_12_trace	4.4363	3.8768	-0.5595	✓ Better

5 TAGE-SC-L Alberto Ros Predictor Analysis

5.1 Outline of the Predictor

The TAGE-SC-L predictor by Alberto Ros, while similar to André Seznec's implementation, introduces distinct optimizations on top of the base TAGE architecture.

Key differences include:

- **Hybrid Sequence Design**: The predictor employs a quadratic sequence initially, then transitions to a generalized geometric sequence with increasing multipliers. This design leverages the rapid growth of quadratic sequences early on, followed by the more moderate progression of geometric sequences. This approach simplifies hardware implementation by allowing direct-mapped tables instead of set-associative ones, reducing MPKI.
- **Enhanced Confidence Mechanism**: The confidence rates have been improved to better balance decisions between the statistical correlator and TAGE components. The loop predictor is treated as the most confident component with final, non-overridable decisions, a departure from the 2016 predictor which allowed conflicts between the SC and loop predictor. The loop predictor is only selected when confidence level is ≥ 2.

5.2 Predictor vs Baseline Predictor

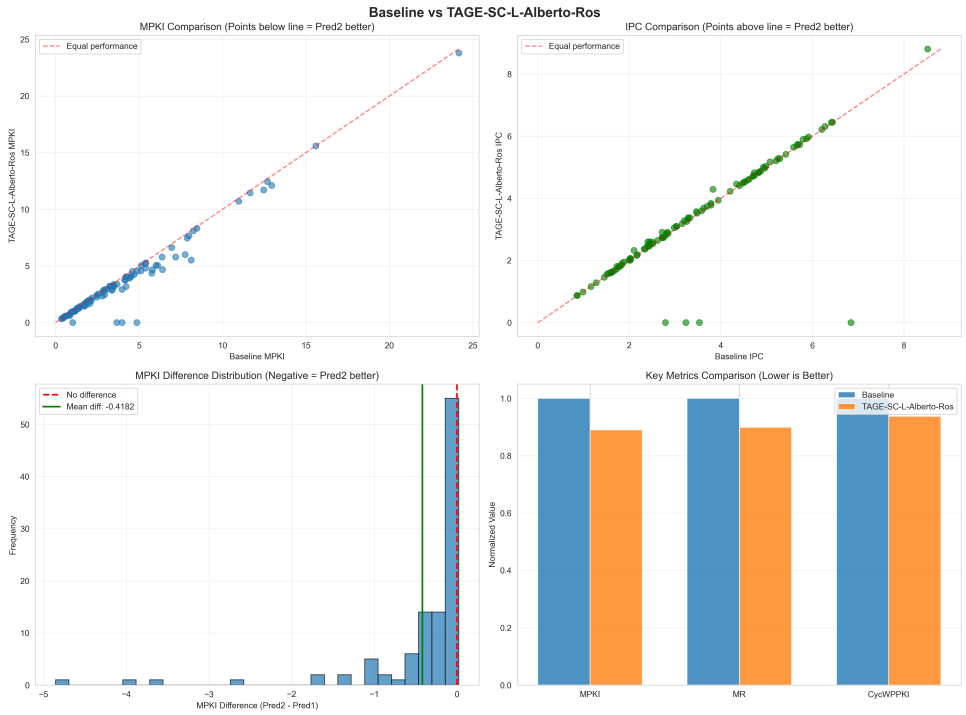

Figure 4: Performance comparison plots (Ros vs Baseline)

Upon analyzing the performance graphs, minimal differences were observed between this predictor and André Seznec's implementation. The improvements are nearly identical, if not slightly worse in some cases.

Table 5: Side-by-Side Framework Performance Comparison

Framework	TAGE Alberto Ros		TAGE Seznec	
	Improve/Total	Success %	Improve/Total	Success %
Web	25/26	96.2%	25/26	96.2%
Fp	8/14	57.1%	9/14	64.3%
Int	20/37	54.1%	25/37	67.6%
Infra	4/16	25.0%	7/16	43.8%
Compress	1/8	12.5%	1/8	12.5%
Media	3/4	75.0%	3/4	75.0%
Average	61/109	55.96%	70/105	66.7%

5.2.1 Framework Performance Comparison

5.3 Direct Predictor Comparison

Even when comparing overall MPKI differences, there is minimal variance between the two implementations.

5.3.1 Summary Statistics

Table 6: TAGE-SC-L Alberto Ros vs TAGE-SCL André Seznec

Metric	Alberto Ros	Seznec	Difference
MPKI	3.3898	3.3918	↑ 0.06% worse
MR	2.6697	2.6687	↓ 0.04% better
IPC	3.3537	3.5081	↑ 4.60% better
Cycles	14106643.7	14447201.9	↑ 2.41% worse
BrPerCyc	0.4012	0.4257	↑ 6.10% better
MispBrPerCyc	0.0089	0.0092	↑ 2.42% better
CycWPPKI	151.4364	151.7579	↑ 0.21% worse

5.3.2 Trace-by-Trace Comparison (MPKI)

- TAGE-SC-L Alberto Ros wins: 27 traces
- TAGE-SCL André Seznec wins: 78 traces
- Ties: 0 traces

5.3.3 Biggest Improvements (Seznec Better)

Trace	Alberto Ros	Seznec	Improvement
web_19_trace	5.0555	0.0000	↓ 5.0555
int_1_trace	11.4406	10.5130	↓ 0.9276
int_2_trace	10.7227	9.9876	↓ 0.7351
int_22_trace	4.8095	4.2827	↓ 0.5268
int_21_trace	23.7948	23.3263	↓ 0.4685

5.3.4 Biggest Regressions (Seznec Worse)

Trace	Alberto Ros	Seznec	Regression
int_9_trace	0.0000	3.6628	↑ 3.6628
web_3_trace	0.0000	3.5523	↑ 3.5523
web_14_trace	0.0000	3.3625	↑ 3.3625
fp_6_trace	0.0000	0.8550	↑ 0.8550
int_16_trace	5.5268	5.8279	↑ 0.3011

6 Delta Analyses and Conclusions

This section analyzes the trace-level performance differentials ("deltas") between the evaluated predictors. By isolating where specific mechanisms succeed or fail, we distinguish between fundamental architectural advantages and stochastic variance. While individual traces exhibit high variance, the consistency of directional improvements across a majority of workloads indicates that the observed deltas reflect systematic architectural effects rather than stochastic noise.

6.1 LVCP vs. RVA-Toru: The Generalization of Value Prediction

Statistic	Value (MPKI)
Average Δ ($MPKI_{LVCP} - MPKI_{RVA}$)	+0.1664
Std Dev	0.4111
RVA Win Rate	87.6% (92/105 traces)

Table 7: Delta Statistics: LVCP vs. RVA-Toru (Positive Δ indicates RVA is better)

6.1.1 Analysis of the Delta

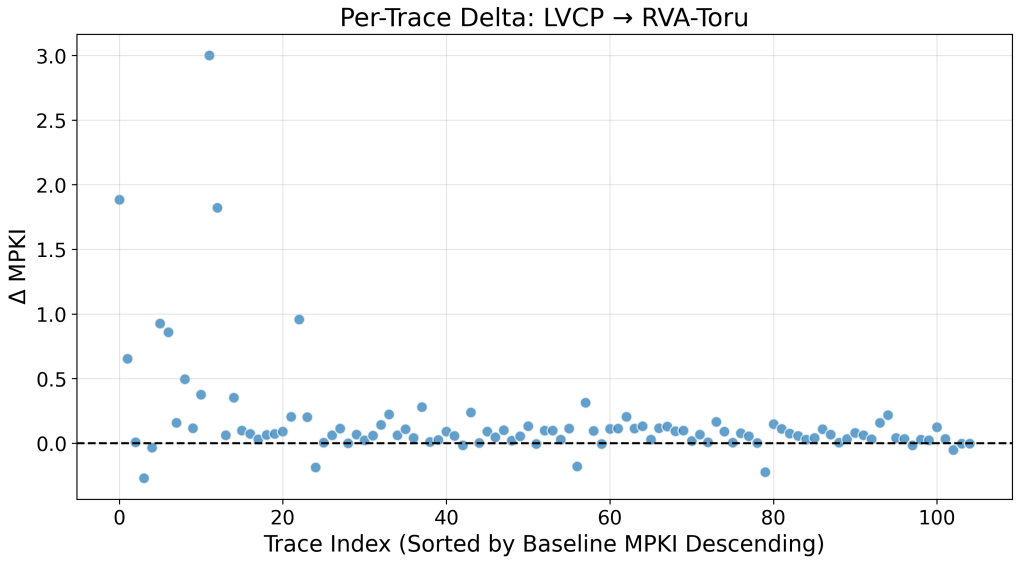

Figure 5: Per-Trace MPKI Delta: LVCP vs. RVA-Toru

6.1.2 Visual Analysis of the Performance Gap

Figure 5 illustrates where the performance improvements originate. The traces are ordered by difficulty, with the hardest workloads (highest baseline MPKI) shown on the left.

Two clear patterns stand out.

- **Large gains on the hardest traces:** The largest improvements, where $\Delta MPKI$ exceeds 1.0, are concentrated on the left side of the plot (roughly indices 0–15). These traces represent cases where the baseline and LVCP perform particularly poorly. RVA-Toru's strong improvements here suggest that it is capturing dependencies that LVCP cannot observe, especially in workloads with complex data-driven behavior.
- **Stable behavior on easy traces:** Moving toward the right side of the figure (indices 40–100), which corresponds to easier traces, the deltas cluster tightly around zero. This

indicates that RVA-Toru does not degrade performance when value correlation is weak or unnecessary. Instead, it behaves similarly to LVCP on these workloads.

Overall, the figure shows that RVA-Toru delivers its benefits where they matter most, on difficult traces, while remaining stable on simpler ones. This supports the claim that RVA-Toru is a robust generalization of LVCP rather than a risky specialization.

The empirical data demonstrates that RVA-Toru effectively subsumes the benefits of the Load Value Correlator Predictor (LVCP). RVA-Toru achieves a lower MPKI on 87.6% of traces. This dominance is structurally inherent:

- **Architectural Superset:** LVCP correlates branch outcomes strictly with *memory load values*. RVA-Toru correlates with *register values*. Since all loaded data must traverse a register to influence a branch (in Load-Store architectures), RVA captures the same information entropy as LVCP.
- **Computational Scope:** RVA-Toru captures information LVCP misses: values derived from arithmetic operations (e.g., loop induction variables, bitwise shifts) rather than raw memory loads.
- **Digest Efficiency:** LVCP suffers from timing windows, if a load is too old, it is evicted. RVA-Toru's use of 12-bit "digests" allows it to track dependencies over longer instruction windows with less storage overhead than storing full 64-bit load values.

Conclusion: The RVA predictor effectively renders the standalone LVCP largely redundant for general-purpose prediction by generalizing the concept of value correlation from the memory domain to the register domain.

6.1.3 Where LVCP Still Wins

Although RVA-Toru dominates overall, LVCP outperforms RVA on 12.4% of traces. These cases likely correspond to tightly coupled load–branch idioms where the branch outcome depends on a recently loaded value with minimal intervening computation. In such scenarios, LVCP's direct memory-value correlation can be marginally more precise, while RVA's value digest abstraction may dilute short-lived correlations. These cases are structurally narrow, reinforcing the conclusion that LVCP provides limited incremental coverage beyond RVA-Toru.

6.2 TAGE-SC-L (Seznec) vs. RVA-Toru: History vs. Data

Statistic	Value (MPKI)
Average Δ ($MPKI_{RVA} - MPKI_{TAGE}$)	-0.1393
Std Dev	0.4437
RVA Win Rate	71.2% (74/104 traces)

Table 8: Delta Statistics: RVA-Toru vs. TAGE-SCL (Seznec)

6.2.1 Analysis of the Delta

This comparison represents the critical inflection point in CBP 2025: the limit of Control-Flow History vs. Data-Flow Correlation.

- **The History Wall:** TAGE-SC-L relies on path history. When different data values flow through the exact same control path (e.g., a data-dependent sort or hash check), TAGE suffers from aliasing. No amount of history length can resolve a branch that is purely a function of a register value.

- **Orthogonal Information:** RVA-Toru injects orthogonal information (register contents) into the prediction hash. The significant improvement ($\Delta \approx -0.14$ MPKI) confirms that modern misprediction bottlenecks are largely data-dependent, not history-dependent.
- **Hybrid Robustness:** RVA-Toru sits atop a TAGE-like base. It defaults to TAGE when register correlation is weak. The lack of significant regressions (points above the line in Figure 2) confirms that RVA is a safe, additive optimization.

6.3 LVCP vs. TAGE-SC-L: Load Data vs. Control-Flow History

This comparison highlights the fundamental distinction between data-local correlation and long-range control-flow history.

LVCP excels on branches whose outcomes depend on recently loaded memory values, such as bounds checks or flag-based conditionals. In contrast, TAGE-SC-L captures long-term path correlations that arise from repetitive control-flow patterns, independent of data values. Neither predictor strictly subsumes the other: LVCP struggles when relevant loads fall outside its temporal window, while TAGE-SC-L fails on branches that are purely data-dependent.

This complementarity explains why RVA-Toru, which unifies history-based prediction with register-value correlation, outperforms both approaches individually.

6.4 TAGE-SCL (Seznec) vs. TAGE-SC-L (Ros): Implementation Trade-offs

Statistic	**Value (MPKI)**
Average Δ ($MPKI_{Ros} - MPKI_{Seznec}$)	+0.0617
Std Dev	0.1539
Seznec Win Rate	77.0%

Table 9: Delta Statistics: Ros vs. Seznec (Positive Δ indicates Seznec is better)

6.4.1 Analysis of the Delta

While the MPKI delta (+0.0617) appears marginal, the comparison highlights a trade-off between algorithmic complexity and hardware realism.

- **Algorithmic Purity:** Seznec's implementation (TAGE-SCL 2025) utilizes complex features like skewed associativity and aggressive update policies, yielding a statistically significant win rate (77%). It represents the theoretical ceiling of history-based prediction.
- **Hardware Feasibility:** Ros's implementation utilizes direct-mapped tables and simplified confidence mechanisms. The fact that Ros's predictor performs within ≈ 0.06 MPKI of Seznec's complex design suggests that the marginal utility of aggressive associativity is diminishing.

Conclusion: The 2025 Championship demonstrates that purely history-based improvements (Seznec vs. Ros) have reached a point of diminishing returns. The significant leap in performance comes from breaking the abstraction layer and utilizing data values directly (RVA-Toru). Future high-performance predictors must likely incorporate register-data awareness as a standard component.

7 Overall Predictor Rankings

This section provides a consolidated, high-level comparison of all evaluated branch predictors across the full set of performance metrics. While earlier sections focused on trace-level deltas and architectural behavior, the goal here is to summarize *overall effectiveness* by looking at absolute averages and relative rankings side by side.

Table 10 reports the average values for each predictor across five key metrics: mispredictions per kilo-instructions (MPKI), branch miss rate, instructions per cycle (IPC), average cycle count, and cycles wasted per kilo-instructions (CycWPPKI). For each metric, predictors are implicitly ranked according to whether lower or higher values are better. A composite score is then computed by summing each predictor's rank across all metrics, where a lower score indicates stronger overall performance.

Predictor	MPKI	Miss Rate	IPC	Cycles	CycWPPKI	Composite
Register-Value-Aware-Toru	3.3022	2.5681	3.5450	14,401,963	149.9141	**6**
TAGE-SC-L (Alberto Ros)	3.3898	2.6697	3.3537	14,106,644	151.4364	**13**
TAGE-SCL (Seznec)	3.3918	2.6687	3.5081	14,447,202	151.7579	**14**
Load Value Correlator	3.4687	2.7183	3.5159	14,561,855	153.8107	**18**
Baseline	3.8080	2.9679	3.4636	14,824,661	161.6047	**24**

Table 10: Overall predictor performance and composite ranking across all metrics

Several trends are immediately apparent. Register-Value-Aware-Toru ranks first in nearly every category, achieving the lowest MPKI, lowest miss rate, highest IPC, and lowest wasted cycles per kilo-instruction. This dominance results in a composite score that is substantially better than all other predictors.

Among the history-based designs, TAGE-SC-L (Ros) and TAGE-SCL (Seznec) perform very similarly. Ros's simpler design achieves the lowest average cycle count, while Seznec's predictor slightly outperforms it in IPC and miss rate. Their close composite scores reflect the diminishing returns of increasingly complex history-based refinements.

The Load Value Correlator improves meaningfully over the baseline across all metrics, particularly IPC and MPKI, but consistently trails predictors that incorporate register-level value information. The baseline predictor ranks last across every category, reinforcing the necessity of advanced correlation mechanisms for modern workloads.

Overall, this ranking table reinforces the central conclusion of this study: while history-based predictors have largely converged in performance, incorporating data-flow information, especially register values, provides the most consistent and system-wide gains.

8 Conclusion

This study of the CBP 2025 predictors highlights a clear shift in how modern branch prediction is improving. By comparing the Load Value Correlator (LVCP), RVA-Toru, and several TAGE-based predictors, a consistent pattern emerges: further gains from control-flow history alone are becoming harder to achieve, while data-driven techniques are providing the most meaningful improvements.

The comparison between Seznec's TAGE-SCL and Ros's TAGE-SC-L illustrates this well. Although Seznec's design uses more complex mechanisms such as skewed associativity and aggressive updates, its advantage over Ros's simpler implementation is small (about 0.06 MPKI). This suggests that increasing complexity within history-based predictors yields limited returns, especially given the added hardware cost.

The more impactful improvements in this championship come from incorporating value information. The Load Value Correlator shows that, for certain workloads, particularly web-style

programs, branch outcomes are often determined by recently loaded data rather than long control-flow histories. However, LVCP is limited in scope. It struggles when branches depend on values produced by computation rather than direct memory loads, which explains its weaker performance on many integer benchmarks.

RVA-Toru addresses this limitation by extending value correlation to register contents. This allows it to capture both loaded values and values produced by arithmetic operations. The per-trace delta analysis shows that RVA-Toru consistently improves the most difficult workloads while remaining stable on easier ones. In other words, it provides large benefits where history-based predictors fail, without introducing widespread regressions.

Final Verdict: The CBP 2025 results indicate that future gains in branch prediction will not come from larger or more complex history tables alone. Instead, meaningful progress requires incorporating information about the data being processed by the program. RVA-Toru demonstrates that register-value awareness can complement traditional history-based prediction and address cases where history alone is insufficient. As workloads continue to become more data-dependent, this hybrid approach is likely to become a standard component of high-performance branch predictors.

9 Scope for Future Work: Exploring the Limits of Branch Prediction

This work shows that RVA-Toru currently represents the strongest predictor in CBP 2025 by making effective use of register values. **The natural next question is: *how much further can branch prediction be pushed?*** My future work focuses on answering this by studying alternative design ideas and by estimating the theoretical upper bound of prediction accuracy.

I plan to approach this in two phases.

9.1 Phase 1: Studying Alternative Prediction Styles

Although TAGE-based predictors dominate the leaderboard, CBP 2025 also includes predictors that take very different approaches. These predictors may capture information that traditional history-based designs miss. I plan to perform trace-level delta analysis on several of these "outlier" designs to understand what, if anything, they contribute beyond TAGE and RVA-style predictors.

- **Multiperspective Perceptron (Jiménez):** This predictor replaces tables with a neural-style learning mechanism. I want to test whether this approach adapts faster during warm-up periods, especially in *infra* and *web* workloads where TAGE predictors often struggle early in execution.
- **Programming-Idiom Predictor (PIP) and Code Structure Correlator (CSC):** These predictors attempt to link branches to higher-level code patterns rather than raw history or values. Comparing them against RVA-Toru will help answer whether "code structure" provides new information, or whether it is simply another way of indirectly capturing data-flow behavior.
- **BALL and Bullseye Predictors:** Since this study already examined LVCP, the BALL predictor offers a natural follow-up by tracking longer dependency chains involving ALU operations and loads. This allows a direct comparison: does following a full computation chain outperform tracking only load values? I also plan to revisit Bullseye to see whether its filtering techniques could be reused to reduce predictor overhead in other designs.

9.2 Phase 2: An Idealized "Infinite-Hardware" Oracle

Even with advanced predictors like RVA-Toru, a small fraction of traces remain difficult to predict. This raises an important question: are these failures caused by limited hardware resources, or are they fundamentally unpredictable?

To explore this, I propose building an idealized "Oracle" predictor that ignores all practical hardware constraints. This predictor would not be bound by the CBP 64KB size limit and would combine the strongest ideas from all predictors into a single design.

The Oracle would include:

- **Very Long Histories and Large Tables:** History lengths extended far beyond practical limits, with fully associative structures to eliminate aliasing.
- **Comprehensive Feature Tracking:** Simultaneous use of global history, register values, memory values, and neural-style learning signals.
- **Perfect Updates:** Immediate and ideal updates to remove pipeline timing effects from the analysis.

Goal: By running this Oracle on the most stubborn traces, those that defeat all existing predictors, I aim to separate remaining mispredictions into two groups: those caused by hardware limits (and therefore potentially fixable), and those caused by true randomness in program behavior. This distinction would help define the true ceiling of branch prediction accuracy and guide future architectural research.

10 References

References

[1] A. Seznec, *TAGE-SC for CBP 2025*, Championship Branch Prediction Workshop, 2025. SiFive.

[2] A. Ros, *A Deep Dive into TAGE-SC-L*, Championship Branch Prediction Workshop, 2025. University of Murcia.

[3] T. Koizumi, T. Maekawa, M. Mizuno, M. Kuroki, T. Tsumura, and R. Shioya, *RUNLTS: Register-Value-Aware Predictor Utilizing Nested Large Tables*, Championship Branch Prediction Workshop, 2025. Nagoya Institute of Technology and The University of Tokyo.

[4] Y. Man, L. Gou, Y. Liu, M. Chen, and Y. Bao, *LVCP: A Load Value Correlated Predictor for TAGE-SC-L*, Championship Branch Prediction Workshop, 2025. Institute of Computing Technology, Chinese Academy of Sciences.

[1]

[2]

[1] ChatGPT (OpenAI) was used for conceptual clarification. No AI was used for explanation of architectural ideas, and editorial refinement of text. No experimental results or analysis were generated by the tool.

[2] Perplexity AI was used for exploratory questioning and cross-checking background information. All formulas, performance metrics, and graph generation code were written and validated by me.