# TAGE-SC 2025 Andrez Seznec Report

## 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 affective in hardware, but only to win a completition, 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
```

### 1.1 Optimization Features of TAGE 2025

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

- 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 ucounter is also set directly to 2 which supports faster eviction.
- It uses probablistic 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 optimization are done from the article/book or whatever about tage in 2024 [André Seznec. 2024. TAGE: an engineering cookbook. Technical Report 9561. Inria. 1–73 pages. https://hal.science/hal-04804900]. something that i found pretty interesting was that they use a tagged IMLI, ie a tagged inner most loop iterator to solve the issue of the fall through mis prediction in the last loop iteration of a loop. its tagged because considering two loops, you would notice that they counters would overwrite each other if it was just a single IMLI, so to solve that the have a tagged IMLI. **something that was interesting to me that they use two IMLI tables, and both of them have differnt purposes** the other that was is the branch context IMLI, it is useful for branches that dont have fixed loop iterations, so they use histories based on the previous anchor branch. anchor branches do cause biases, but due to the corelator in an BrIMLI, it was make a safe biased prediction, its still counting based ratherr than pattern based but its 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 (longestmathcprediction and globla history)

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

# 2 TAGE SC 2025 vs TAGE SC-L 2016

If we take a look at the plots we can see that for the top right plot we can see that, 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 similiar 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 +- variation to be 0.05 but i feel like its 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 intially but here the web bencmarks show te most improvements. its probably because of the probabilisitic allocation of entries, since you now have a confidence level that is provided by the longest matching counter.

## 2.1 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 1: Performance improvement by framework type

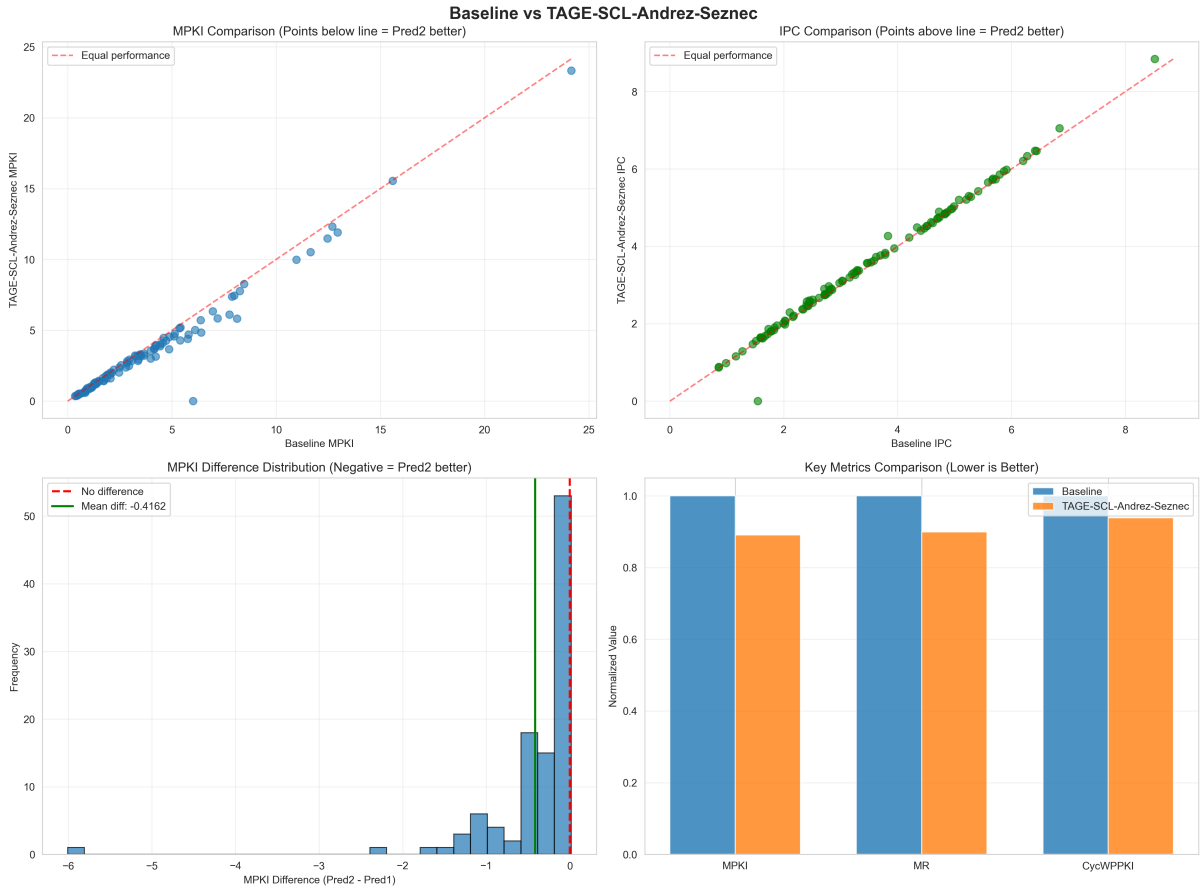


Figure 1: TAGE-SC 2025 vs Baseline Comparison

## 2.2 Analysis

More over when i look at the MPKI i found that the mpki for the 2025 predictor is better. there is an 11% performance gain which is pretty good considering the relative efficiency of the predictors. the tage sc 2025 is better in almost all scenarios in this case.

One interesting observation that i made was the in the cases that the baseline impplementation was better was all in the case of the infra frameworks mostly. 3 out of the 4 cases where the baseline predictor was better was in the case of the infra benchmarks. i dont have much synthesis ability on branch prediction so i did some asking around with perplexity and this is what it said:

```
TAGE-SC 2025's optimizations suppress and delay learning (confidence-
    filtered allocation, aggressive protection, multi-allocation), which
     helps noisy modern workloads but hurts small, regular infra loops
    that need fast, deterministic convergence.
The 2016 design learns these stable patterns more directly, so it wins
    on the few infra benchmarks despite being worse overall.
```

What i understand from this is that the optimizations support more nosiy workloads, ie loops that are very irregular, and kinda messes up on workloads that are more stable needing more deterministic convergence and what i mean by deterministic convergence is that everytime a branch happens a counter saturates based on what the result of the branch is. there is no sense of delayed learninng or randomness. the data structures saturate based on the outcome of the branches. again this probably doesnt make a difference since the 2025 version is much smarter than 2016 and the outcomes only differ by +- 0.01 or less so its probably the noise.

Another thing i noticed was that the biggest improvements i noticed were for the int benchmarks, and it makes sense for the int benchmarks since the int benchmarks are quite noisy due to them compile time branhces that are very control flow and and branch heavy. the tage 2025 shines due to its 2 imli tables and optimizations, and it being more efficient for noisy workloads.

### 2.3 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 |