HUAWEI

香 港 大 學

THE UNIVERSITY OF HONG KONG

ELEC4848 Senior Design Project 2021-2022

# Evaluate Prediction Accuracy of SOTA Branch Predictors on WSC traces

**Derek Jinyu Dong (3035636987)**

Supervisor: Dr Artemiy Margaritov (HUAWEI)

Second Examiner: Prof Hayden K.H. So

August 2022

## Acknowledgements

I would like to thank Dr Artemiy Margaritov and Dr George Wang from HUAWEI UK R&D for giving me this chance to learn branch prediction and apply the knowledge immediately in this interesting project; the discussions with Artemiy were very thought-provoking and fruitful. I also appreciate the effort from Prof Hayden K.H. So for his constructive advice.

## Abstract

Branch prediction is a critical part of modern CPU design and is relied upon by many advanced CPU technologies, such as OoO and prefetching. However, the prediction accuracy of branch predictors has rarely been investigated for Warehouse-house Computer – an increasingly deployed and demanded load type. Thus, this project aims to provide an starting points for study branch predictor on warehouse-scale computers. Newly released Google Workload Traces are converted into BT9 format and simulated using CBP-5 kit, followed by comparing SOTA predictors and various loads. Superiority of TAGE has once again been confirmed on WSC traces. Interesting features of WSC are concluded and suggestions for optimizing the branch predictor on WSC are given.

# Table of contents

# List of figures

# List of tables

# Nomenclature

**Terminologies**

*Set of trace / trace set* There are six sets of traces: CBP-5 *mobile*, *server*, Google WSC *delta*, *whiskey*, *charlie*, *merced*. Set is the basic unit for the simulation and calculation in the project. For WSC, set is also the maximum range in which traces are captured under same launch and share virtual address space.

*Unique Branch* Unique branch (UB) is defined inside a set for WSC traces and a trace for CBP-5 traces. Unique branch represents the branch instructions stored in same virtual address. An branch in an virtual address can only be counted as one unique branch no matter how many times it is executed.

*Branch Instruction* Branch instruction is the concept opposite to unique branch. If a unique branch is executed 10 times, the number of branch instructions is counted as 10, though the number of unique branch is always one.

**Acronyms / Abbreviations**

| | |
|---|---|
| BT9 | Branch Trace version 9, a trace format applied by CBP-5 |
| DS-TAGE | Dynamically Sized TAGE |
| ETPB | Easy To Predict Branch |
| HTPB | Hard To Predict Branch |
| MPKI | Misprediction Pre 1K Instructions |
| MPP | Multiperspective Perceptron Predictor |
| MPP-TAGE | Multiperspective Perceptron Predictor with TAGE |
| PC | Program Counter |
| PHT | Pattern History Table |
| PPM | Prediction by Partial Matching |
| TAGE | TAgged GEometric history length branch prediction |
| UB | Unique Cranch |
| UDUB | UniDirectional Unique Branch |
| WSC | Warehouse-Scale Computer |

# 1 Introduction

## 1.1 Background and related works

The demand for Warehouse-Scale Computer (WSC) is on a fast-growing trend, with a different architecture and load compared to PC, mobile devices and servers. However, to our knowledge, the prediction accuracy of existing predictors under this new type of load has not been explored. This project investigates the directional branch prediction accuracy of existing conditional branch predictors under WSC load and to compare the results with the accuracy of the predictors under other loads.

### 1.1.1 Origin of branch predictor

The importance of a high-accuracy branch predictor comes first and foremost from the pipeline penalty. Modern CPUs typically have dozens of pipeline stages, which means that once the prediction is wrong and the whole pipeline goes in the wrong direction, the cost of washing the pipeline and starting again is huge. IPC can be negatively impacted by each wrong prediction.

Besides, there are some critical technologies that rely on accurate branch prediction. For example, branch predictor predict the future instructions and feed those instructions into re-order buffer (ROB). Apple A14 (2020) and M1 has 600+ ROB entries[2], which is massively larger than typical 300-entry re-order buffer in conventional Intel/AMD/Arm processors. Similarly, re-order window of Intel's Golden Cove (2021) has a huge increase compared with Sunny Cove (2019) – 512 entries compared to 352[3]. The trend of the industry is to increase out-of-order buffer size, the idea behind is trying to increase the chance to get executable instruction from ROB. This put pressure on the CPU front-end: to not waste those extra ROB spaces, branch predictor must keep high accuracy prediction to keep the ROB occupied by correct instructions. Prefetching technology is another example: the industry is also heavily relying on high branch prediction accuracy to remove instruction cache misses[4].

There are actually two parts of branch prediction, direction prediction predicting whether the conditional branch is taken and target prediction predicts the target address of the jump. This project focus on directional prediction because it is simpler and thus a very good starting point for a time-sensitive project. Three important branch direction predictor schemes, gShare[5] and TAGE[6] (the scheme used by SOTA branch predictor) and Perceptron Predictor[1] are introduced later.

### 1.1.2 Design of branch direction predictors

#### gShare

GShare was first proposed by Scott McFarling in 1993. The innovative introduction of the XOR mechanism helps the GShare predictor globally identify different branches and reduce the probability of aliasing.

Gshare use a pattern history table (PHT) consisting of two-bit saturating counters, indexed by a global history register. The structure leads to an exponential growth of the size of PHT with the length of the global history[1].

#### Perceptron Predictor

The perceptron predictor was first introduced into branch predictor by Daniel A. JimCnez and Calvin Lin in 2001. Pereceptron predictor stores weights of each history direction into perceptron and indexes perceptrons only using hashed PC. The capacity increases when size of each perceptron increases, thus achieving linear growth with the history length.

The limitations of perceptron predictor are, firstly, that it is only good at predicting linearly separable branches because it uses only a single layer of perceptron, and secondly, that the earliest perceptron predictor only uses global histroy for prediction and ignores other branch features.

It's worth to mention that perceptron is becoming a majority in the newly proposed branch predictor schemes, including SRNN predictor[7] and BranchNet[8].

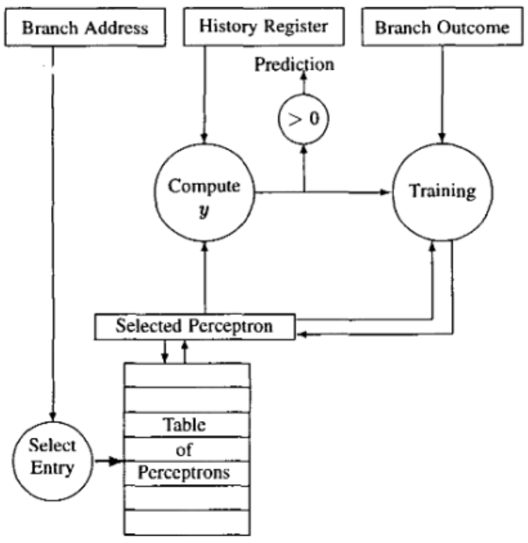


Fig. 1.1: Perceptron Predictor Block Diagram[1].

#### TAGE

The history length is a key question for branch predictor. Longer histories enable accurate predictions for some HTP branches while taking more time to warm-up, reducing accuracy for ETP branches. Geometric history length[9] helps find a balance for this question: predictor with shorter history length provides predictions early on before longer predictors finishes warm-up stage. PPM data compression algorithm can be a efficient way to manage and select from predictors with various history length.

TAGE, derived from a PPM-like predictor [10] by André Seznec, Pierre Michaud (2006), combines the advantage of geometric history length and PPM data compression algorithm, thus enables more efficient storage of detected patterns.

### 1.1.3 New trends in branch prediction field

Some new trends and concepts in branch prediction field are briefly mentioned and core reference are cited for future study.

#### Security issues

As the security and privacy of computer systems has become increasingly emphasised, research interest has attracted into the security of branching predictors under side-channel attacks. Side channel attack is based on leaked information from the physical implementation of the cryptosystem rather than theoretical weaknesses in the encryption algorithm. Under this kind of attack, shared directional branch predictor can be manipulated by the attacker and the prediction and history target can be leaked[11]. It is difficult for mitigation technique to achieve both performance and isolation, but there are some meaningful attempts[12].

#### Offline training

Offline training refers to training of the predictor before the execution of the program and its counterpart is runtime branch prediction. The limited resources on a chip and the low latency requirements limit the scale and complexity of branch prediction algorithms. However, it is possible to improve predictor accuracy by extracting the properties of branch traces in advance for offline training. Offline learning takes place in two main stages: compiling and profiling.

### 1.1.4 Warehouse-scale computer (WSC)

In the section, the general idea of WSC is briefly mentioned, followed by emphasising the key features of the WSC that may affect the work of branch predictor.

WSCs are a kind of datacenter that provide large-scale, high-performance, low-cost Internet, computing and storage services[13]. Born out of the demand for large-scale computing for a small amount of applications (e.g. webmail, web search, online translation), in recent years it has gradually taken over a large amount of computing and storage demands that would otherwise have been carried out locally as the price of computing power has fallen.

However, WSCs are fundamentally distinct from traditional benchmarks due to the need for on-demand scalability, elasticity and availability. Unlike traditional datacenter where a number of independent and different machines are physically put together, machines in WSC are more homogeneous, emphasising internal connection. Machines often have similar hardware architectures, run on the same software platform, and are connected and managed under the same system management layer.

WSCs run a small number of super-scale tasks, usually on thousands of machines simultaneously, flexibly allocating computing power according to demand. Super-scale tasks can have two significant consequences:

1. Data parallelism arises from large datasets that need to be processed. These large-scale datasets often require a large amount of computation for each parallel (sub) task, which may lead to periodic, regular instruction streams performed by a single core over a period of time.
2. Executables are also getting bigger. The increased range of instructions makes it harder for branch predictors to predict correctly. In addition, WSC has a greater instruction throughput and typically has a larger reorder window (i.e. out-of-order buffer), resulting in a greater reliance on high accuracy branch predictions. However, the branch predictor design is not specifically optimised for this critical and unique load.

## 1.2 Motivations and Objectives

WSC has been in increasing demand and deployment in recent years, and its unique load characteristics are expected to have an impact on branch predictors. Prediction accuracy under such loads has been a gap and is waiting to be investigated. May 2022, google released google datacenter's actual instruction and memory access traces, making research on WSC more feasible and convenient.

Here are the main goals of the project. Note that attention is only focused on the directional branch predictors at this stage.

1. (completed) To get familiar with the branch prediction field and compose a well-categorised review of the field, including brief development history, the latest trends and SOTA design.
2. (completed) To setup the required infrastructures on the HUAWEI laptop and server and launch the branch predictor simulator under default traces.
3. (completed) To do necessary data processing on Google traces so that it can be accepted by the CBP-5 simulator without losing required information.
4. (core objective, completed) To evaluate the accuracy of direction prediction vs capacitance of branch predictors (TAGE, perceptron) on Google Workload Traces and compare the results with the prediction accuracy under conventional load.

## 1.3 Report Organization

The report is structured as follows: the second section discusses the method used in the simulation. This includes the introduction of simulation infrastructure: CBP-5 simulator kit and Google Workload traces. The method of converting Google WSC traces into CBP-5 BT9 file is also discussed. In the next section, the results of the simulation are presented and discussed in the following order: traces statistic analysis, impact of trace sets on simulation and impact of predictors on simulation results. Some discussion about the conduct on the project is also made in this section. Finally, conclusions are drawn based on the current progress.

# 2 Methodology

The core task of the project is to investigate the prediction accuracy of directional branch predictors under WSC workload. To achieve this, WSC branch traces that fit real-world WSC load scenarios are needed, and a sufficiently efficient and well-accepted simulator is required to make the result convincing, not to mention a high-performance platform to provide the computing resources. This section demonstrates where and how the the experiment are implemented.

## 2.1 Simulation infrastructure

### 2.1.1 5th Branch Prediction Championship (CBP-5) Kit

CBP-5 [14] is a conditional branch predictor simulator provided by Branch Prediction Championship, a competition organised by *The Journal of Instruction-Level Parallelism*. Since the first competition was held in 2004, CBP has gradually become a widely accepted branch predictor simulator.

There are several reasons why CBP-5 is very well suited to this project. Firstly, it is speedy. Unlike Gem5, QEMU and other whole-chip simulation tools, CBP-5 focuses on the directional branch predictors' simulation and is implemented in C++, thus can simulates far faster than Gem5 and QEMU. Secondly, its simplicity reduces potential hurdles when being applied to other trace formats (e.g. Google Workload Traces). The smooth learning curve also makes it more suitable for this time-critical project. Finally, there are useful branch instruction traces (223 training traces) and SOTA predictors (winner submission of the competition) attached to this simulator kit and can help streamline project workload.

#### CBP-5 traces

223 traces are given in the CBP-5 kit and separated into four catagories: 19 long mobile traces, 61 short mobile traces, 4 long server traces and 139 short server traces. Long traces have 1 billion instructions and shot traces have 100 million instructions. In this project we combine all 80 mobile traces as CBP-5 *mobile* trace set, and all 143 server traces as CBP-5 *server* trace set.

#### CBP-5 predictors

11 prize-winning predictors are given in the CBP-5 kit, including SOTA **TAGE-SC-L**[15] (8KB & 64KB), **MTAGE+SC**[16] (Unlimited size), **Multiperspective Perceptron Predictor**[17] (8KB, 64KB & Unlimited), **Multiperspective Perceptron Predictor with TAGE**[18] (8KB, 64KB & Unlimited) and **Dynamically Sized TAGE**[19] (8KB & 64KB). All predictors are simulated but the discussion on predictors is focused on TAGE-SC-L, MTAGE-SC and Multiperspective Perceptron with TAGE.

### CBP-5 trace format: BT9

CBP-5 uses *Branch Trace version 9* (BT9) as its input file format. A complete branch instruction trace is translated into the a flow control graph (e.g. Figure 2.1) and its node table, edge table and edge sequence are stored in a BT9 file, therefore being a space-saving format.

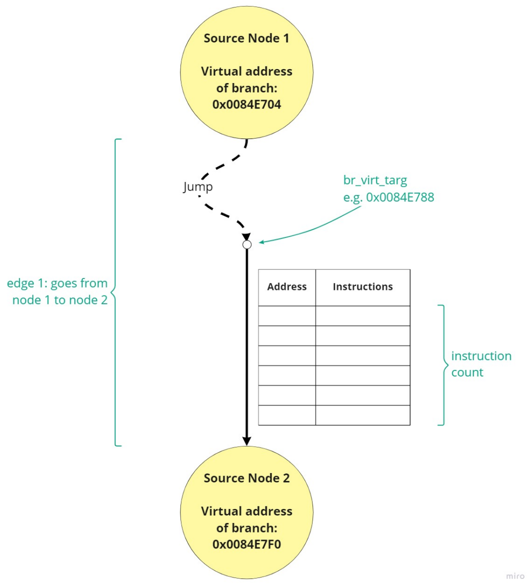

Source Node 1
Virtual address of branch: 0x0084E704
Jump
br_virt_targ e.g. 0x0084E788
edge 1: goes from node 1 to node 2
Address
Instructions
instruction count
Source Node 2
Virtual address of branch: 0x0084E7F0
miro

**Fig. 2.1:** BT9 format example: control flow graph

## 2.1.2 Google workload traces

Released May 2022, the Google Workload Traces[20] project is driven by the rapid growth of warehouse-scale computers (WSCs) in today's computing market and hopes to improve research in the WSC architecture by sharing real WSC workload data. The traces are captured from Google datacentre using *DynamoRIO*'s *drmemtrace*, including instruction and memory address traces (.memtrace.gz) as well as branch traces (branch_trace.<tid>.csv.gz). Each representing a unique WSC application, there four trace folders named *delta*, *whiskey*, *charlie* and *merced*. All traces inside a folder are captured in a single launch thus sharing virtual address space.

As the first published WSC traces, Google workload traces fit well with this project by offering a domain different from all traditional benchmarks. At the stage the focus is on the branch traces.

## 2.1.3 Computing platform

The simulation is power by a HUAWEI server equipping two Intel(R) Xeon(R) Gold 6161 CPUs and supporting a maximum of 88 threads. Computing resource is accessed by SSH, and a Docker container is created on the server to provide a safe working environment. All simulation data is analysed by *numpy* and plotted by *matplotlib*.

## 2.2 BT9 Format conversion

This section stems from the mismatch between the CBP-5 and WSC branch trace formats; CBP-5 uses the BT9 format, while WSC traces are captured by DynamoRIO and therefore use the DynamoRIO format. There are two potential solutions to the mismatch, either designing a program that converts Google WSC traces into BT9 format or modifying the frontend part of CBP-5. The former option is taken in this project, beacuse designing a program that only formats the WSC trace would make the risk more manageable than modifying the simulator with thousands of lines of code. Besides, modified CBP-5 may reduce the acceptance of the results.

This is one of the core objectives of the project and takes the major project period. Since a WSC trace is structured as a sequential branch instruction list while a BT9 file is stored as node table (unique branches), edge table and edge sequence (abbreviated branch instructions), the foundamental idea of conversion is simple: read branch instruction one by one trace file, update the node table and edge table, and then append the interpreted edge into edge sequence.

However, because of the mismatched information, some unexpected problems were encountered during the conversion leading to a one-week delay.

### Behaviour, size and instruction count

Though the behaviour information is expected in BT9 format, it is not used in the predictor. Thus we block the behaviour entry in BT9.

Instruction count refers to the number of non-branch instructions executed between two branch instructions. Whatever instruction size is used in the calculation, some instruction count are fractions or negative numbers. The invalid instruction count implies that in contrast to the fixed-size instructions in the CBP-5 traces, instructions in Google WSC traces have various sizes (and is presumed to be x86_64 instruction set)

### Branch type conversion

The classification of branch types in two formats are different. In BT9, each branch must take a label from each of the following classes:

| | |
|---|---|
| **Type** | Return, Jump, Call |
| **Directness** | DIRECT, INDIRECT |
| **Conditionality** | CONDITIONAL, UNCONDITIONAL |

However in Google WSC traces, branches are categorised into eight types, as shown in the first column of Table 2.1. Obviously, the conversion cannot be done directly, and three problems are listed below.

1. No Google branch types are specified the Conditionality except *conditional jump*

   Based on an observation that all branches in Google WSC traces are 100% taken except *conditional jump*, it is assumed that all other types are unconditional.

2. The *interrupt* and *context switch* do not correspond to any of the BT9 label.

   This problem indeed exists and cannot be address directly. However, noting that only a tiny fraction of branch instructions are *interrupt* or *context switch* (verified in Section 3.1),

| Google WSC type | BT9 type | *problem encountered* |
|---|---|---|
| *direct call* | CALL+DIR+UCD | 1 |
| *indirect call* | CALL+IND+UCD | 1 |
| *direct jump* | JMP+DIR+UCD | 1 |
| *indirect jump* | JMP+IND+UCD | 1 |
| *conditional jump* | JMP+DIR+CND | 3 |
| *return* | RET+IND+UCD | 1 |
| *interrupt* | JMP+DIR+UCD | 2 |
| *context switch* | JMP+DIR+UCD | 2 |

**Table 2.1:** Branch type conversion result

they are ignored in this project by being converted into JMP+DIR+UND, a common and undirectional type that won't affect conditional branch predictors.

3. The *conditional jump* has no clear Directness

   Because the project studies the condition branch predictor only (i.e. predict taken or not-taken), the directness does not affect the simulation results. *conditional jump* interpreted as DIRECT branch because direct branches are the majority of all branches.

# 3 Results

This section presents the results of the simulation and gives some analysis and discussion about the results. An analysis of the trace statistics is given first, including trace lengths and trace composition (conditional branches and single-direction branches). Immediately afterwards the simulation results are compared between the trace sets and some useful WSC trace features are obtained, inspiring some thoughts on the causes of the prediction accuracy under the WSC workload. In the third section TAGE[15, 16] (SOTA) and some other predictors [19, 18, 17] are compared. Finally, a review of the completion of the project is given and limits are discussed in detail.

## 3.1 Traces statistics

When studying new trace sets, it is natural and necessary to first study the statistical properties of the sets themselves. The four WSC sets (*delta, whiskey, charlie* and *merced*) show significant differences from each other, indicating that WSC workload branch traces strongly depend the application run on WSC.

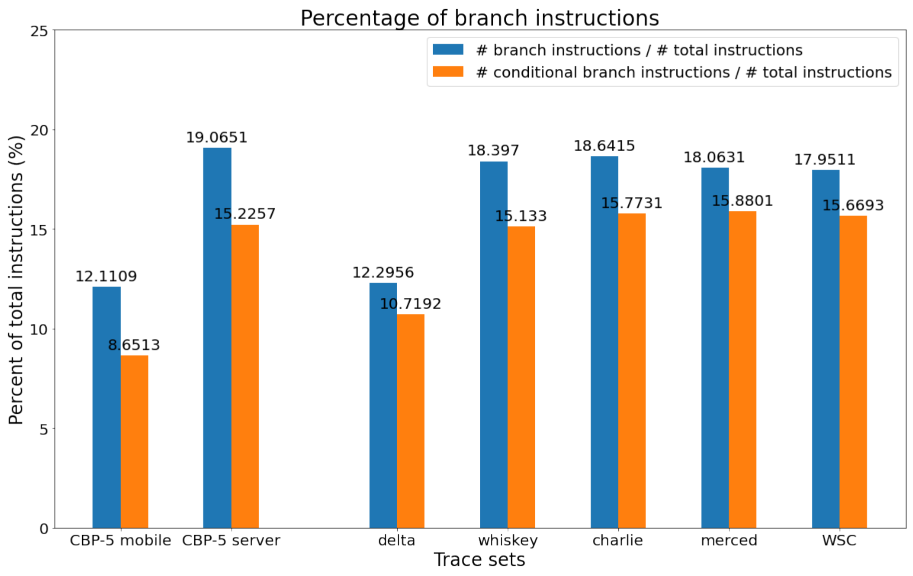


**Fig. 3.1:** Branch instruction and conditional branch instruction percent

### Fraction of branch instructions

As this project focused on conditional branch only, the proportions of branch instruction and conditional branch instructions are studied first Figure 3.1. The proportions of both branch instructions and conditional branch instructions in WSC sets are almost equal to those in CBP-5 *server* and higher than in CBP-5 *mobile*, suggesting a higher similarity between WSC and CBP-5 *server* than CBP-5 *mobile*. Also note that *delta* have fewer branch instructions and conditional branch instructions than other WSC sets, which is one of the major causes why *delta* has the smallest MPKI among the WSC sets.

Besides, 0.00127% instructions are *interrupt* and 0.00142% instructions are *context switch* in WSC sets. This verifies the previous assumption in Section 2 that *interrupt* and *context switch* take small proportion of the sets and that ignoring them has a negligible effect on the simulation results.

### Trace lengths

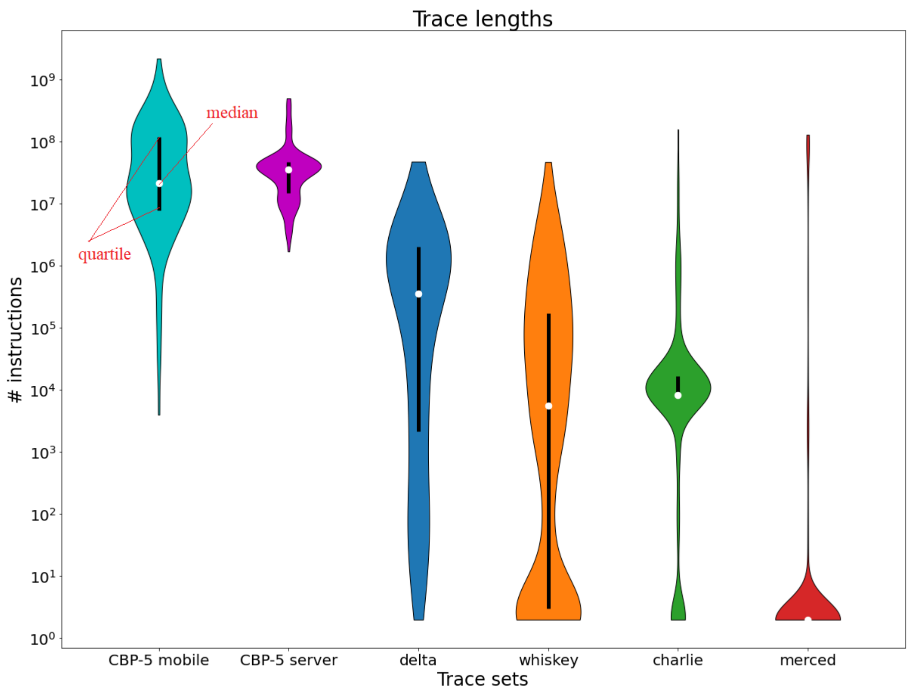


**Fig. 3.2:** Trace lengths of six trace sets

In this report, since both CBP-5 and WSC traces show similar proportions of branch instruction, trace length is defined as the number of all types of instructions in a trace. Figure 3.2 shows the trace length distribution of each trace set, where the vertical coordinates indicate the trace length and each violin plot represents a set, with wider indicating more traces locate at the corresponding length.

While 90% CBP-5 traces longer than $10^6$ instructions, WSC traces are significantly shorter and less concentrated with many traces even shorter than 100 instructions (particularly *whiskey* and *merced*). Since Google Workload Traces[20] are raw data directly captured from the machine and each trace represents a software thread, we presume very short traces represent idle threads while the program is running. As the predictors require a learning process, too short traces do not reflect the prediction accuracy of the predictor, and an example is thousands of traces containing only 16 instructions reporting an MPKI of 62.5, well above the MPKI reported by other long traces.

### Uniting traces

Since short traces contribute unreasonably high MPKIs but have very limited impact on actual system performance, the default methods of CBP-5 that counts the MPKI of all traces and calculate the arithmetic mean for each set is no longer reasonable for Google Workload Traces. Three new solutions were proposed to overcome this problem:

1. Filter traces shorter than a certain length
2. **Unite all the traces in a set after the simulation.**
3. Do not reset the predictor between traces in the same set.

Confirmed with Google, this solution 2 is supported by the fact that all traces within the same trace set are for the same run of the application, therefore share the address space. In a set if two branches occurring in different traces share same virtual address they are recognised as same unique branch, thus the total executed number and misprediction number within a set can be obtained for each unique branch, and this is called "uniting traces" in this report. In solution 2, each trace is still simulated independently and the MPKI is calculated after uniting the simulation results of all the traces in a set. Solution 1 requires a reasonable criteria to filter traces which we do not currently have and solution 3 is time consuming, requiring additional code and another simulation. Given the limited project duration, solution 2 is applied and WSC traces are united within each sets.

One of the idea raised from uniting WSC traces is can CBP-5 traces also be united? This idea is objected by Table 3.1, whose first row represents the number of UBs in each set and second row is obtained directly adding up the number of UBs in each individual trace). The third row shows the ratio of the first row to the second row, it is called commonality ratio and it reflects the proportion of branches that are shared by different traces within a set; the lower the commonality ratio, the more likely it is that the set of traces was captured under a single launch. Both CBP-5 sets have a much higher commonality ratio than WSC traces, implying that they are less likely captured under same software launch.

Comparing the united mean of WSC sets with the arithmetic mean of CBP-5 may be questioned to be unscientific. However, the CBP-5 traces are parallel and independent with each other so it is logically to calculate their arithmetic mean; whereas the traces in a WSC set are captured under a single launch, contain complex interconnections and are not equivalent (some threads carry the main workload) so it is reasonable to introduce weight by uniting traces.

| | CBP-5 | | Google Workload Traces | | | |
|---|---|---|---|---|---|---|
| | *mobile* | *server* | *delta* | *whiskey* | *charlie* | *merced* |
| # unique branches (UBs) | ~~152K~~ | ~~456K~~ | 63K | 904K | 499K | 290K |
| # adding up UBs | 444K | 2,234K | 1,088K | 10,481K | 20,035K | 56,328K |
| commonality ratio | 34.3% | 20.4% | 5.81% | 8.63% | 2.49% | 0.515% |
| # average UBs (top 6) | 35.5K | 82.7K | 23.0K | 166.5K | 282.1K | 133.1K |

**Table 3.1:** Commonality reatio & Average UBs (top 6). The commonality ratio reflects the proportion of branches that are shared by different traces within a set; the lower the ratio, the more likely it is that the set of traces was captured under a single launch.

### Number of unique branches (UBs)

The reason for the interest in the number of UBs is because of Aliasing. Aliasing is a well-known phenomenon that affects predictor accuracy by limiting the predictor's ability to accurately identify UBs. Due to the limited capacity of the predictor, a great amount of UBs in a trace can put high pressure on the predictor capacity, thus causing aliasing and reducing predictor accuracy.

Considering the large number of short traces in WSC sets, the average of the six traces with the most UBs are calculated and appended to Table 3.1. According to the averaged UBs, *server* has two times more than *mobile*, in line with the simulation results where the MPKI of *server* is more than twice that of the *mobile*. Most WSC traces (i.e. except *delta*) have 1.6-3.4 times more UBs than *server*, implying a much higher MPKI than *server* can be expected (though this is not the case, as discussed in Section 3.4).

As to *delta*, not like other WSC sets, it has less UBs than *mobile*, implying that its MKPI can be even lower than mobile (verified by the simulation result). Besides, the great variation between WSC applications is again confirmed: delta looks more like *mobile* than a WSC set.

### Fraction of unidirectional unique (UU) branches

Unidirectional unique branches (UDUBs) are of particular interest because they have only one choice (T or N) and are easy to predict while occupying the same amount of resources as other more complex UBs under the present predictor schemes. Research on them may help optimize predictor designs for UU branches to saving valuable capacity. Since uniting is not valid for CBP-5 sets, six traces with the largest number of UBs are selected and averaged (Table 3.2) for both CBP-5 *mobile* and CBP-5 *server*. The fraction of taken UU branches, not-taken UU branches, their summation and the ratio of taken UU branches to not-taken UU branches are obtained for selected CBP-5 traces and united WSC sets.

| *mobile* | | *server* | |
|---|---|---|---|
| trace name | # UBs | trace name | # UBs |
| SHORT_MOBILE-33 | 41984 | SHORT_SERVER-5 | 89864 |
| LONG_MOBILE-14 | 38832 | LONG_SERVER-2.res | 89864 |
| SHORT_MOBILE-32 | 36949 | SHORT_SERVER-6.res | 88689 |
| SHORT_MOBILE-35 | 33592 | LONG_SERVER-3.res | 88689 |
| SHORT_MOBILE-36 | 30839 | SHORT_SERVER-8 | 69612 |
| LONG_MOBILE-15 | 30839 | LONG_SERVER-4.res | 69612 |

**Table 3.2:** Selected CBP-5 traces with the largest number of unique branches (UBs)

It's clearly shown in Table 3.3 that the fraction of UU branches are similar and about 80% for both CBP-5 and WSC, hinting at the potential for optimisation for UU branches. Whoever the ratio of taken UU branches to not-taken UU branches are significantly different in CBP-5 and WSC. Depending on different WSC sets not-taken UU branches can be 5 12 times more than taken UU branches, compared with 1.5 times for *mobile* and 2.1 times for *server*. More not-taken UU branches indicate that WSC is a stronger biased workload, which reduces the direction prediction difficulty. Another point worth mentioning is that the WSC still show significant differences about

UU branches between sets: *merced* has the smallest proportion of UU branches while *whiskey* has the largest.

In contrast to UU branches, hart-to-predict (HTP) branches are also of interest and are discussed this in later sections.

| | CBP-5 | | Google Workload Traces | | | |
|---|---|---|---|---|---|---|
| | *mobile* | *server* | *delta* | *whiskey* | *charlie* | *merced* |
| Taken UU branches | 30.42% | 25.90% | 6.25% | 12.31% | 9.38% | 6.43% |
| Not-taken UU branches | 46.40% | 54.39% | 73.51% | 70.16% | 68.44% | 62.10% |
| All UU branches | 76.82% | 82.83% | 79.75% | 82.46% | 77.82% | 68.53% |
| Not-taken / Taken | 1.536 | 2.101 | 11.77 | 5.700 | 7.298 | 9.653 |

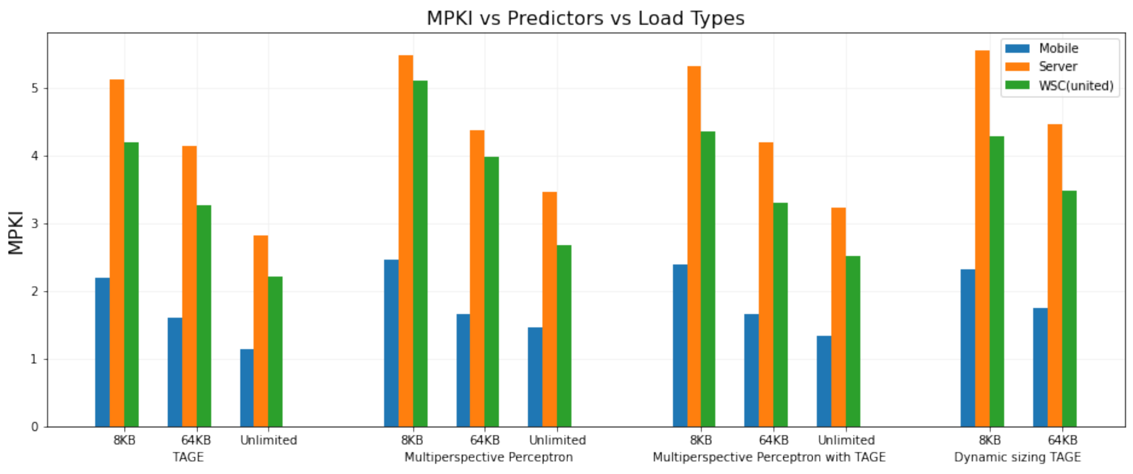
**Table 3.3:** Fraction of unidirectional unique branches (UDUBs). CBP-5 data is obtained from six selected traces, while WSC data is obtained from united WSC sets.

## 3.2 Impact of trace sets on MPKI

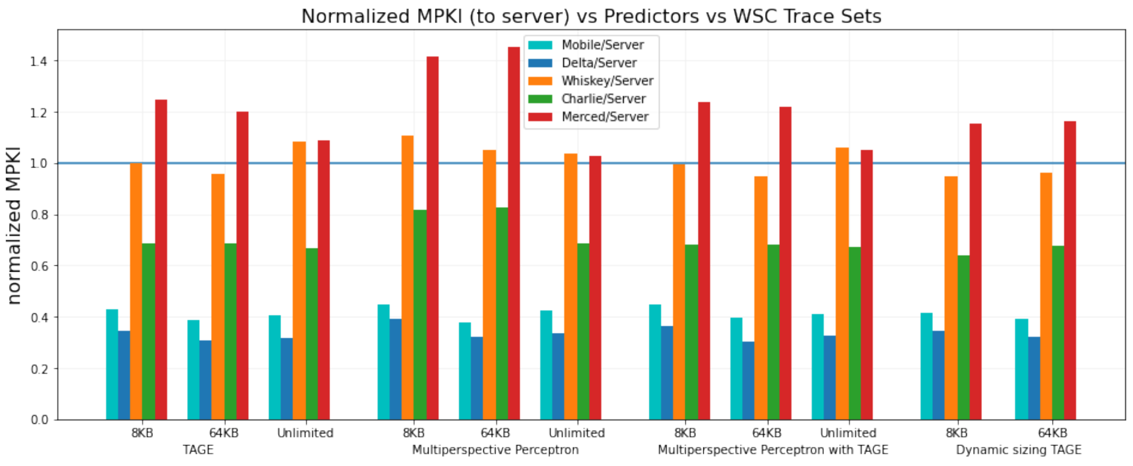
The simulation results are discussed in this section, mainly around the differences between trace sets. Despite the high pressure put on the predictors' capabilities by the large number of UBs, the WSC sets reports MPKIs that are similar to or even slightly lower than the MPKI of *server*. This result means that due to specific tasks and similar context[13] WSC is a more predictable workload. Note that WSC sets continue to show a huge variation in simulation results, emphasising the importance of application-specific analysis.

The MPKI are shown in Figure 3.3a. Contrary to previous expectation, MPKI of united WSC is 15.7% lower than server, suggesting that there are other factors increase WSC prediction accuracy (e.g. very few applications or large fraction of not-taken UDUBs). Since WSC sets are highly differentiated, all sets are shown in Figure 3.3b. Normalizing sets to server is to compare the adaptability of different predictors to WSC in later section. MPKI proves that *whiskey, charlie* and *merced* put higher pressure on predictor capacity than *delta*, due to much more UBs.

The paradox of WSC sets (except *delta*) having more UBs but lower MPKI than *server* suggests that WSC has some features makes branches highly predictable, which is in line with previous summaries of WSC features from a higher level. Despite WSC sets have more UBs, they also have more not-taken UUBs (thus strongly biased), more specific application and larger-scale parallel instruction context.



**(a)** MPKI vs Predictors vs Load Types



**(b)** Normalized MPKI (to server) vs Predictors vs WSC Trace Sets

**Fig. 3.3:** MPKI histograms & Normalized MPKI of 6 trace sets and 11 predictors

## 3.3 Impact of predictors on MPKI

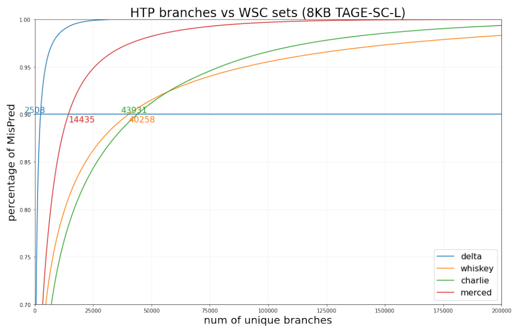
Figure 3.3 shows MPKI of all predictor. On WSC sets, TAGE scheme (TAGE-SC-L & MTAGE-SC) is still SOTA, followed by the Multiperspective Perceptron Predictor with TAGE (MPP-TAGE) and Dynamically sized TAGE (DS-TAGE). The multiperspective perceptron predictor (MPP) reports highest MPKI on all capacities. While DS-TAGE reports higher MPKI than MPP on *server*, DS-TAGE reversed the results on WSC. This draws attention to the variation of each predictor in server and WSC, thus that the normalized MPKI is plotted. 8KB & 64KB MPP has the largest normalised MPKI and worst adaptation on WSC, resulting in the highest MPKI. However, Unlimited MPP has the smallest normalised MPKI, suggesting that large MMP has the potential to do well on WSC traces. Besides, For all predictors schemes and most WSC sets, unlimited versions report smaller normalised MPKI, because the impact of UBs increasing is less significant for unlimited capacity.

## 3.4 Hard-to-predict branches (HTPB)

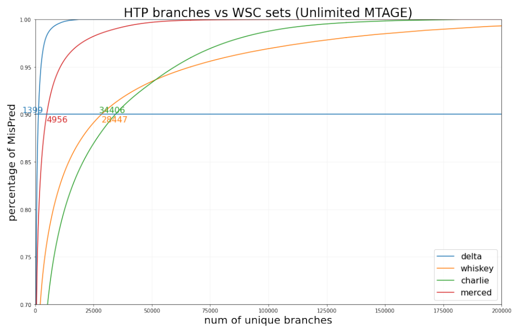
Hard-to-predict branches are those unique branches that hard to predict and contribute to the most of misprediction. By sorting unique branches using the number of mispredictions and ploting the accumulated misprediction percent, Figure 3.4 are plotted. A horizontal line is placed at 90% mispredictions to help count how many unique branches contribute 90% mispredictions to the whole set. This number is defined as the number of HTPB in this project. Four predictors are selected such that three of them are TAGE with different size, and one of them is 64KB MPP-TAGE to campare with 64KB TAGE-SC-L.

Firstly, significant variation is still observed for four WSC sets. The number of HTPB generally increase with number of UUBs, but still affected by specific feature of individual set. The more HTPB there are in a set, the larger capacity is required to address those HTPB.

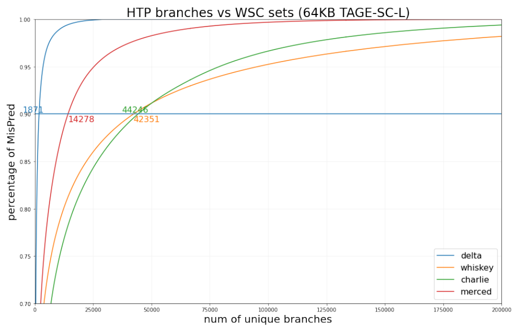
Secondly, the numbers of HTPB do not change significantly with predictors, but are suddenly reduced when unlimited capacity is applied. This indicates how many unique branches are hard to be predicted by the current scheme, ignoring the limitation from the predictor size.



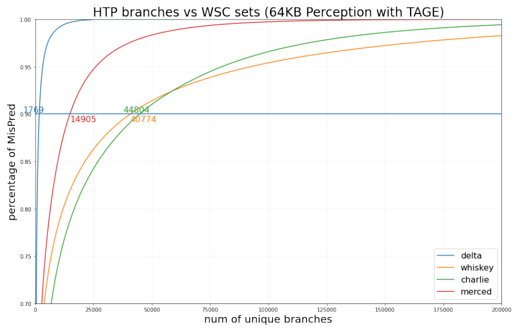
(a) 8KB TAGE-SC-L



(b) Unlimited MTAGE-SC



(c) 64KB TAGE-SC-L



(d) 64KB Perception with TAGE (MPP-TAGE)

**Fig. 3.4:** HTP branches vs WSC sets

## 3.5 Discussion of the results

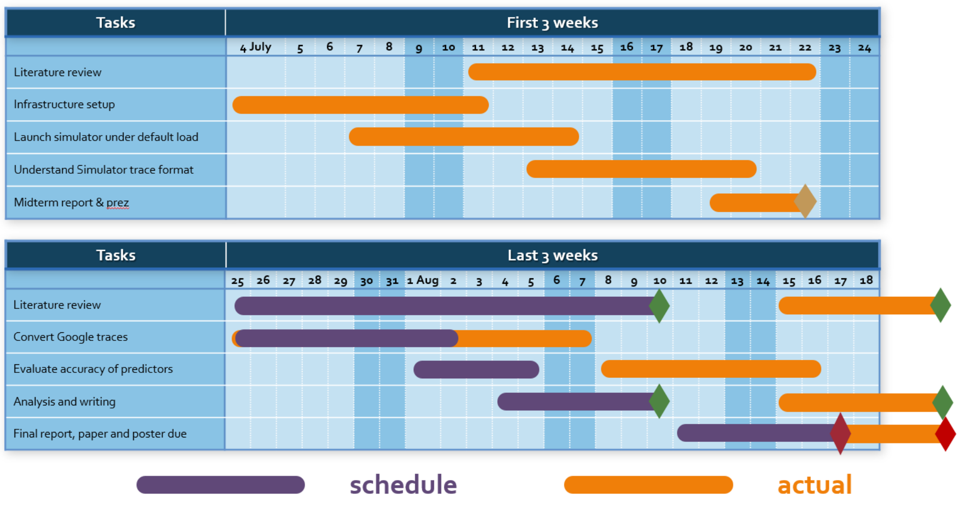


Fig. 3.5: Project progress Gantt Chart

Shown in the final gantt chart (Figure 3.5), the results of the project are generally acceptable and all planned objectives have been completed. Unexpected problems were encountered in two main areas, resulting in delays of the project:

1. Mismatch between the information provided by Google Workload Traces and expected by CBP-5 simulator.
2. Raw, unprocessed trace sets from Google Workload Traces.

Problems caused by the former include type, behaviour and size problem as discussed in Section 2.2. Though having been mostly addressed but the start of the simulation was postponed by a week from schedule because of them. The second aspect was only realised towards the end of the project, when the limited time available meant that we could only process the simulation data as much as possible (uniting traces) and could not re-simulate it after processing the raw trace.

Overall, the project has some points for improvement in the future. Firstly, pre-processing of the WSC raw data is desired. Uniting the WSC sets into one trace before simulation or modifying the simulator to avoid resetting of predictors between traces within the same set are both acceptable. Secondly, the calculation of the commonality ratio is not rigorous because it has high dependency on the number of traces in the analysed set. A better way to check the validity of uniting CBP-5 traces is to calculate pair-based commonality ratio and then average on the whole set. Or even simpler - contact the CBP-5 organizer to get confirmation. Finally, limited to the understanding on branching prediction, this project does not adequately compare and analyse the prediction accuracy of the various predictors.

# 4 Conclusion

WSC traces are proved to be significantly varied from application to application. Considering usually only a small amount of applications running on WSC and WSC has specific tasks and targetted functionality, It can be a emerging to specialize a branch predictor for common application. Besides, the superiority of TAGE has once again been confirmed on WSC traces. Despite much larger amount of unique branches giving pressure to the capacity, lower MPKI reported by WSC sets is a suggestion to some feature that makes predictino easier. Finally, according to the MPKI from predictors with unlimited capacity, the improvement of increasing capacity can be expected to be larger on WSC than server.

This project aims to provide an starting points for study branch predictor on warehouse-scale computers. Trace format conversion and simulation has been done. Interesting features of WSC are concluded and suggestions for optimizing the branch predictor on WSC are given. Further study can first try to address the limits of this project mentioned in Section 3.5, then explore ways to improve branch predictor on WSC.

# References

[1] D. Jimenez and C. Lin, “Dynamic branch prediction with perceptrons,” in Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Jan. 2001, pp. 197–206, iSSN: 1530-0897.

[2] A. Frumusanu, “Apple Announces The Apple Silicon M1: Ditching x86 - What to Expect, Based on A14,” Nov. 2022. [Online]. Available: https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive

[3] D. I. C. Frumusanu, Andrei, “Intel Architecture Day 2021: Alder Lake, Golden Cove, and Gracemont Detailed,” Aug. 2021. [Online]. Available: https://www.anandtech.com/show/16881/a-deep-dive-into-intels-alder-lake-microarchitectures

[4] Y. Ishii, J. Lee, K. Nathella, and D. Sunwoo, “Rebasing Instruction Prefetching: An Industry Perspective,” IEEE Computer Architecture Letters, vol. 19, no. 2, pp. 147–150, Jul. 2020, conference Name: IEEE Computer Architecture Letters.

[5] S. McFarling, “Combining Branch Predictors,” p. 22, 1993.

[6] A. Seznec and P. Michaud, “A Case for (partially) Tagged Geometric History Length Branch Prediction,” The Journal of Instruction-Level Parallelism, vol. 8, p. 23, Feb. 2006.

[7] L. Zhang, N. Wu, F. Ge, F. Zhou, and M. R. Yahya, “A Dynamic Branch Predictor Based on Parallel Structure of SRNN,” IEEE Access, vol. 8, pp. 86 230–86 237, 2020.

[8] S. Zangeneh, S. Pruett, S. Lym, and Y. N. Patt, “BranchNet: A Convolutional Neural Network to Predict Hard-To-Predict Branches,” in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Oct. 2020, pp. 118–130.

[9] A. Seznec, “Analysis of the O-GEometric history length branch predictor,” in 32nd International Symposium on Computer Architecture (ISCA’05), Jun. 2005, pp. 394–405, iSSN: 1063-6897.

[10] P. Michaud, “A PPM-like, tag-based branch predictor,” Journal of Instruction-Level Parallelism, vol. 7, pp. 1–10, Apr. 2005.

[11] D. Evtyushkin, R. Riley, N. C. a. E. Abu-Ghazaleh, and D. Ponomarev, “BranchScope: A New Side-Channel Attack on Directional Branch Predictor,” ACM SIGPLAN Notices, vol. 53, no. 2, pp. 693–707, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3296957.3173204

[12] I. Vougioukas, N. Nikoleris, A. Sandberg, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett, “BRB: Mitigating Branch Predictor Side-Channels,” in 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb. 2019, pp. 466–477, iSSN: 2378-203X.

[13] L. A. Barroso, J. Clidaras, and U. Hölzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition. Morgan & Claypool Publishers, San Rafael, California, Jul. 2013. [Online]. Available: https://www.morganclaypool.com/doi/abs/10.2200/S00516ED2V01Y201306CAC024

[14] "Championship Branch Prediction." [Online]. Available: https://jilp.org/cbp2016/

[15] A. Seznec, "TAGE-SC-L Branch Predictors Again," Proceedings of the 5th Championship on Branch Prediction, 2016. [Online]. Available: https://jilp.org/cbp2016/

[16] ——, "Exploring branch predictability limits with the MTAGE+SC predictor," Proceedings of the 5th Championship on Branch Prediction, p. 4, 2016. [Online]. Available: https://jilp.org/cbp2016/

[17] D. A. Jimenez, "Multiperspective Perceptron Predictor," Proceedings of the 5th Championship on Branch Prediction, p. 4, 2016. [Online]. Available: https://jilp.org/cbp2016/

[18] ——, "Multiperspective Perceptron Predictor with TAGE," Proceedings of the 5th Championship on Branch Prediction, p. 4, 2016. [Online]. Available: https://jilp.org/cbp2016/

[19] S. Pruett, S. Zangeneh, A. Fakhrzadehgan, B. Lin, and Y. N. Patt, "Dynamically Sizing the TAGE Branch Predictor," Proceedings of the 5th Championship on Branch Prediction, p. 5, 2016. [Online]. Available: https://jilp.org/cbp2016/

[20] "Google Workload Traces." [Online]. Available: https://dynamorio.org/google_workload_traces.html