



Deliverable D2.5

Report and Artifact: Learning Processor Models

Deliverable

<i>Number and title:</i>	D2.5 – Report and Artifact: Learning Processor Models		
<i>Work package:</i>	WP2 (Reliability and Resilience)		
<i>Lead author:</i>	Lijun Zhang (IISG)		
<i>Contributors:</i>	Andrea Turrini (IISG)		
<i>Reviewers:</i>	Pablo Castro (UNRC), Thomas Noll (RWTH)		
<i>Due date:</i>	M41 (2025-02-28)	<i>Dissemination level:</i>	Public
		<i>Version:</i>	1.0 (final)
<i>Submission date:</i>	2025-02-25	<i>Pages:</i>	8

Version history

<i>Version</i>	<i>Date</i>	<i>Notes</i>
1.0	2025-02-25	First official release

Project

<i>Title:</i>	Models in Space Systems: Integration, Operation, and Networking		
<i>Acronym:</i>	MISSION	<i>Start date:</i>	01-10-2021
<i>GA no.:</i>	101008233	<i>Duration:</i>	56 months
<i>Call:</i>	H2020-MSCA-RISE-2020	<i>Website:</i>	mission-project.eu



This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233. This document reflects only the authors' view. It does not represent the opinion of the European Union, and the European Union is not responsible for any use that may be made of the information it contains.

1 Introduction

Hardware operating in space, such as the circuitry used in a satellite orbiting Earth, is subject to different failures than hardware used on the ground, mainly induced by the severely reduced protection effects of the Earth atmosphere and magnetic field. Besides being exposed to a wide range of temperatures, e.g., the satellite may be exposed to sunlight, or it could be in Earth's shadow, circuitry could also be affected by cosmic radiation and cosmic particles, which can cause temporary modifications to the data stored in memory or permanent damages to the electronic components controlling the satellite's functionalities.

Commercial off-the-shelf (COTS) components are widely used in limited-budget missions such as the ones involving small satellites like CubeSats, a main reason for that is that COTSs are cheaper and easier to obtain than components specifically designed and certified to work in space. Moreover, space-qualified components are usually larger and heavier than COTSs, hence, they consume more energy to function. Power consumption is yet another issue to be taken into account when using space-qualified components in CubeSats, since this kind of small satellites are severely constrained on their size and weight on launch, and battery-based energy storage has to be traded with the other components inserted in the satellite, affecting the functionality and the satellite's lifetime once launched and deployed in space. To increase the autonomy of the satellite when solar panels are not providing enough power (for instance, when the satellite is orbiting in the Earth's shadow, so no light from the Sun can be used to generate energy), larger batteries are needed and these can be installed if COTS are used instead of space-qualified components. However, due to being in space with reduced protection from Earth's atmosphere and magnetic field, COTS are more likely to suffer single event effects (SEEs) that are caused by a single, high-energy particle, hitting the internal electronics of the COTS and that can influence the satellite functionality in multiple forms. SEEs can be just soft, non-destructive errors, such as bits having the wrong value due to transient pulses altering the input/output value of logic or support circuitry, or as changes of the bits stored in memory cells and registers. Other SEEs can be even more severe, causing hard errors or being potentially destructive, such as an induced high current in the circuitry, exceeding the device specifications, and this can cause physical damage to the hardware gates.

It is thus important, for the success of the mission, to evaluate in advance whether COTS components, intended to be used in a satellite, provide enough guarantees about their functionality even when single effect events occur. This can be done by applying system verification, where SEEs are modelled in the system as probabilistic events affecting the state of the COTS component. In our approach the resilience of COTS with regard to single event effects that can occur in space environments (which is actually a safety and reliability problem) is modelled and assessed by means of security techniques. This is achieved by analyzing variations of the intended behavior of a component within a well-known privacy protection framework, differential privacy.

2 System Verification

Traditionally, safety analysis is only through experimental methods and such a kind of analysis does not provide general theorems about the desired behaviors of the system. Interestingly, a prominent way in computer science to assure the absence of errors, or to find and report errors, is formal verification. The spectrum of key techniques in this field ranges from runtime verification, such as checking properties while executing the system, to deductive techniques such as theorem proving, to model checking [3,5]. The latter is a highly automated model-based technique assessing whether a model representing the set of all possible executions of the system satisfies a safety property describing the desirable behavior. To analyze the effect of SEEs on COTS components used in satellites, we model the latter as Markov chains and then evaluate their effect of SEEs on changing the outcomes of branch predictions, as it would be the effect of a deliberate choice of an adversary trying to extract private information from

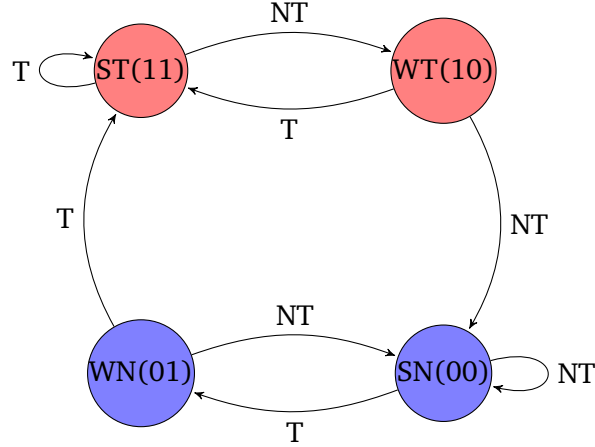


Figure 1: A typical saturating counter, for predicting the branch direction

the execution of multiple programs in a general purpose multi-threading CPU as commonly used on professional and consumer computing equipment. Branch prediction is the fundamental technique to improve the instruction-level parallelism in modern high-performance processors [8, 19, 20]. However, the security vulnerabilities such as Meltdown and Spectre exposed in recent years reveal that there are serious risks in modern branch predictor designs [1, 2, 4, 6, 7, 9, 13]. Attackers exploit these vulnerabilities to obtain the branch history (i.e., taken or not-taken branches) and detect the fine-grained execution traces of processes. For example, BranchScope [7] and Bluethunder [9] attacks steal secrets by detecting a specific branch predictor entry. Furthermore, BranchShadowing [13] also uses the shared branch prediction histories to infer fine-grained control flow in Intel SGX. Since the modern branch predictor resources are shared between different threads, in either a single-core processor or a simultaneous multi-threading processor, it leaves attackers the opportunity to maliciously perceive branch history information across different processes and privileges.

The root cause of the branch predictor vulnerabilities is that branch prediction is based on saturating counters, and the update strategy of such counters is deterministic. A saturating counter encodes the execution history information of the branch instructions, by incrementing its value when the branch is taken and decrementing it when it is not taken within the range of values given by the size of the counter. The branch prediction uses the value of the saturating counters to decide what branch of a branching instruction is more likely to be taken, so the CPU executes the initial instructions in the predicted branch to speed up the global execution under the assumption that the branch was correctly guessed. As soon as the branch choice is actually performed, either the prediction was correct and the already executed instructions are valid (so time has been saved), or the instructions in the wrong branch were executed but this does not affect the overall running time. This determinism in updating the saturating counter makes it easy for an attacker to control the saturating counter's state to construct a side-channel attack, to extract information from the data manipulated by the processor in the branch predicted wrongly. As the fundamental building block of branch predictors, the saturating counter provides an excellent cost-efficient way of reducing the penalty due to conditional branches and is widely used in various branch prediction designs from simple GShare predictor to the latest TAGE-type predictor [10–12, 15–18, 21].

3 Learning COTS Components

Branch predictor side-channel attacks require the critical capability to prime and probe saturating counters. The saturating counter encodes the execution history information of branch instructions, which

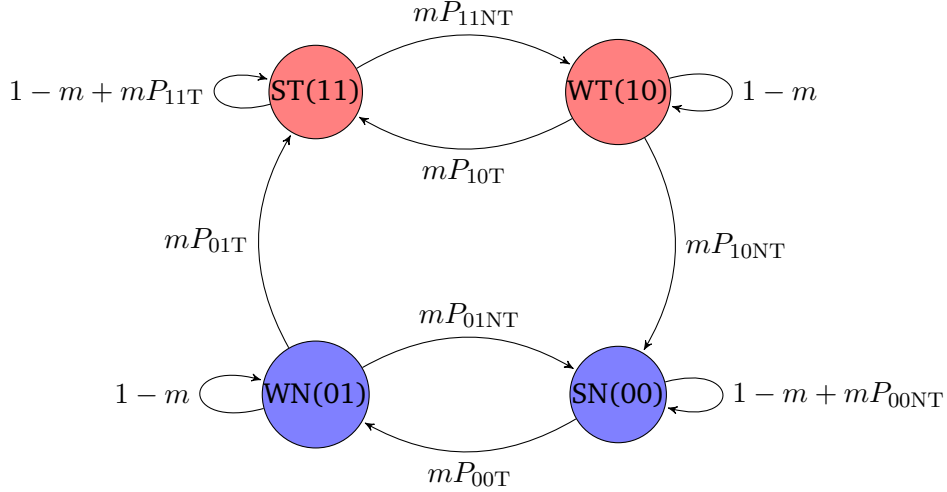


Figure 2: A 2-bit probabilistic saturating counter

may also contain sensitive branch instructions. The attack is the process of decoding the information in the saturating counter. Since the state transition of a conventional saturating counter is deterministic, it is easy for an attacker to infer the direction of the victim's branch by counting the branch prediction results (correct prediction or misprediction). Figure 1 shows a 2-bit prediction counter as a finite state machine, having four states: SN (strongly not taken), WN (weakly not taken), WT (weakly taken) and, ST (strongly taken). There are two input symbols: T (taken) and NT (not taken), indicating the execution direction of a thread on the saturating counter. The most significant bit of the saturating counter predicts the direction of the branch, and the other bits provides hysteresis, thereby the branch predictor requires two successive mispredictions to change the prediction of direction, which is T (taken) or NT (not taken) of the input alphabet and each is indicated by red states or blue states. In short, a prediction counter can be used to generate taken/not- taken predictions and change its own states by reading the actual execution direction.

The prediction is correct if the saturating counter is in state ST or WT and the next input is T, or in state SN or WN and the next input is NT. Otherwise a misprediction occurs. The attacker judges the prediction result of the branch predictor by measuring the execution time of probe process (e.g., reading rdtsc/rdtscp register or related hardware performance counter in x86 ISA). In [7, 14], the attacker thread tries to find the cut-off point of the saturating counter, which corresponds to the first time that a correct prediction shows after a bunch of mispredictions. On finding the cut-off point, the attacker can infer that the victim thread's execution direction is T or NT, by setting up a side-channel attack, and with this, gain information about the possible values that are involved in the branch selection, for instance the secret values like passwords and private keys involved in an authentication process.

When evaluating the robustness of a saturating counter against adversarial attacks tailored to exploit branch predictor vulnerabilities, a promising solution to mitigate branch predictor vulnerabilities is to introduce probabilistic updates into saturating counter design [14]. It changes the conventional deterministic state transition mode to a probabilistic state transition mode. The probabilistic saturating counter greatly reduces the ability of the attacker to perceive the saturating counter's state, as it adds noise to the actual state and decreases the usefulness of the information extracted through the side-channel attack.

Figure 2 shows the probabilistic counterpart of the 2-bit saturating counter depicted in Figure 1. Here, deterministic transitions have been replaced by probabilistic transitions, resulting in a Markov chain, by weighting the outgoing transitions with a parameter $m \in [0, 1]$ that works as a probabilistic threshold in the counter. When a transition is made, a random number is generated and compared with the

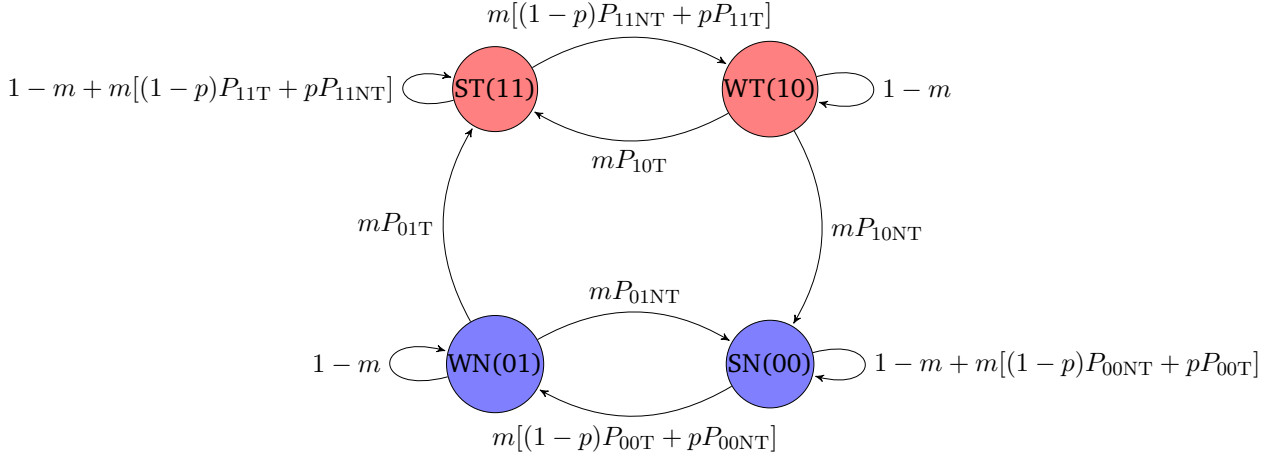


Figure 3: The Markov chain for analyzing probabilistic saturating counters

threshold m . If the generated number is smaller than m , then the counter remains in its current state; otherwise, the transition is carried on, with relative probabilities P_{iT} and P_{iNT} , where i is the state identifier, P_{iT} is the statistical probability of executing the target branch with direction “taken” and P_{iNT} is the statistical probability with direction “non-taken”; as usual, we have $P_{iT} + P_{iNT} = 1$. The values of P_{iT} and P_{iNT} can be learned from simulations on the COTS components: by repeatedly executing the programs controlling the different parts and functionalities of the satellite on synthetic inputs, standing for the values that would be provided by the satellite sensors (such as battery charge level, solar panel energy production, satellite gyroscopic orientation, and so on), we can collect data about how frequently directions “taken” and “non-taken” occur in the saturating counter and use these data as statistical estimation for P_{iT} and P_{iNT} in the different states i .

The value of m can be used to represent the effect of SEEs on the value of the saturating counter, since it can be used to model the fact that an SEE can prevent the counter to be updated correctly when a branch is chosen. However, the fact that the SEE can alter the way branches are accounted is not modelled in the Markov chain shown in Figure 2. To account for even more changes induced by SEEs in the values stored in the saturating counter, we add another parameter $p \in [0, 1]$, which governs how directions “taken” and “not-taken” are actually evaluated; the corresponding Markov chain is shown in Figure 3. We retain the original probability update mechanism of the Markov chain depicted in Figure 2, that is, the original transition will be updated with a probability m . At the same time, the new probability parameter $p \in [0, 1]$ is introduced in states ST and SN: when a T is read in state ST, there is still a probability p of going to state WT, and a probability $1 - p$ of remaining in state ST. When NT is read in state ST, with probability p we stay in state ST, and with the remaining probability $1 - p$ we transition to state WT. A similar modification is applied to state SN. In practice, the parameter p represents the probability that the direction that has been actually chosen has been accounted wrongly, that is, as it would have been the opposite direction.

As a result, the probabilistic saturating counter still uses the probabilistic update mechanism, but only with different update probabilities for state ST and SN. For state ST, when T is read, there is a probability of $1 - mp$ of staying in ST and a probability of mp of going to state WT; when N is read, there is a probability of $1 - m + mp$ staying in ST and of $m - mp$ to state WT. Therefore, whether NT or T is read, as long as $p \neq 0$, there is a positive probability to stay in state ST or to go to state WT. A similar analysis holds for state SN. It can be found that our newly designed probabilistic saturating counter generalizes the previous saturating counters: when using $m = 1$ and $p = 0$, it becomes the conventional saturating counter shown in Figure 1; when $m \neq 1$ and $p = 0$, it corresponds to the probabilistic saturating counter depicted in Figure 2.

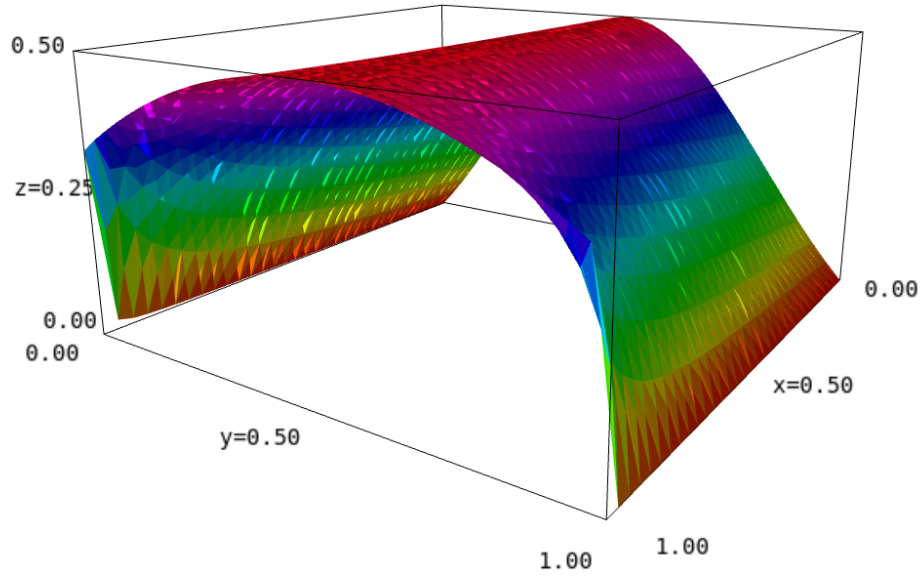


Figure 4: The misprediction rate in the steady states of the Markov chain from Figure 3

4 Analyzing COTS Components

The Markov chains depicted in Figures 2 and 3, together with the original deterministic saturating counter shown in Figure 1, allow us to compare the different success rates of an attacker to extract information via a side-channel from the saturating counter, by computing the probability of a successful attack and comparing them in the context of differential privacy. This is a well-known way to evaluate the amount of information the attacker is able to gain by observing small variations in the behavior of the system. This information then can be used to derive guarantees about the functionality of the COTS component and its resilience against single effect events that can occur in space and that can alter the intended updates of the saturating counter. In particular, small values in the differential privacy results are mapped back to higher resilience of the COTS component: since the attacker can only extract a small amount of extra information from the altered counter when compared with the unaltered counter, this means that the counter’s behavior is very similar in the altered and unaltered counters. This implies that the single effect event has had a rather limited effect on the counter inside the COTS component, so the component can be considered to be reliable in its functionality and resilient to SEEs.

In a similar manner, we can compute the misprediction probability rates for varying values of the parameters governing the transition probabilities in the probabilistic saturating counter shown as the Markov chain in Figure 3. The misprediction rate represents the fact that a prediction has been done wrongly, namely, NT was predicted in states ST and WT, or T was predicted in states SN and WN. The computed misprediction rate is shown in Figure 4, where z is the misprediction rate; x represents the probability value p governing how directions “taken” and “not-taken” are actually evaluated; and y stands for the probability P_T of taking the branch, that is, the P_{iT} value, where we assume that P_{iT} is the same for all states i to simplify the analysis. The success probability rate is then just the opposite value, i.e., $1 - z$.

By inspecting the computation of the misprediction rates on the steady states, it can be observed that it is independent of the parameter m as long as $m \neq 0$, which otherwise would make all states of the Markov chain in Figure 3 absorbing, thus making the counter useless. When $p = 0$, the misprediction rate for the saturating counter in Figure 1 is equal to that of the counter in Figure 2, which is only related to the probability P_T of taking the branch, while the misprediction rate of the probabilistic

saturating counter in Figure 3 is related to both parameters P_T and p . Moreover, when $p = 1$, P_T cannot be 0 or 1, otherwise the steady state is uncertain.

As we can see from the plot, it is heavily dependent on the value of P_T , that is, on how diverse frequently the branch is taken: if the branch is almost always ($P_T \approx 1$) or almost never ($P_T \approx 0$) taken, then misprediction is small, since we can just predict the direction that occurs in the overwhelming majority of the cases; if the branch is taken near half of the times ($P_T \approx 0.5$), then we also fail the prediction by a similar rate. However, the value of p (depicted on the x -axis of the plot) has limited effect on the misprediction rate shown in the plot, except when p is close to 1. This means that the modifications caused by SEEs on how branches are accounted for have little influence on the behavior of the COTS component, unless they interfere with almost all branch selections; this happens only in COTS components that are very vulnerable to SEEs and thus should not be used as control circuitry in a satellite as they would completely disrupt the satellite's functionality.

5 Conclusion

By analyzing the security of the probabilistic update mechanisms by combining performance evaluation with formal verification, it is possible to quantify the information an attacker can extract from the running system. By linking the parameters governing the probabilistic update mechanism of the saturating counter with the probability of having SEEs when the COTS component is operating in the orbiting satellite, we can derive guarantees about the functionality of the satellite. The update mechanism is modeled and analyzed by means of a Markov chain and the security property is specified based on a well-known privacy protection framework, differential privacy. The probability of success of the side channel attack is analyzed and studied, and the computed success rate of the attack is used to derive properties on the functionality of the COTS component. Furthermore, analysis shows that under certain circumstances the attacker is still able to accurately guess the branch execution of the victim's process, which indicates that SEEs can still disrupt the correct execution of the COTS component used to control the functionality of the satellite. Thus, a proper differential privacy defense guarantee can deduce useful saturating counter model parameters, that can be linked back to the amount of transient disruption SEEs can cause to the internal electronic gates of the COTS component.

References

- [1] O. Aciğmez, Ç. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 312–320. ACM, 2007.
- [2] O. Aciğmez, Ç. K. Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In *Cryptographers' Track at the RSA Conference*, pages 225–242. Springer, 2007.
- [3] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.
- [4] S. Bhattacharya and D. Mukhopadhyay. Fault attack revealing secret keys of exponentiation algorithms from branch prediction misses. *Cryptology ePrint Archive*, Report 2014/790, 2014. <https://ia.cr/2014/790>.
- [5] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.
- [6] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh. Understanding and mitigating covert channels through branch predictors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13, 2016.

- [7] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev. BranchScope: A new side-channel attack on directional branch predictor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*, pages 693–707. ACM, 2018.
- [8] B. Grayson, J. Rupley, G. Z. Zuraski, E. Quinnell, D. A. Jiménez, T. Nakra, P. Kitchin, R. Hensley, E. Brekelbaum, V. Sinha, and A. Ghiya. Evolution of the Samsung Exynos CPU microarchitecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 40–51, 2020.
- [9] T. Huo, X. Meng, W. Wang, C. Hao, P. Zhao, J. Zhai, and M. Li. Bluethunder: A 2-level directional predictor based side-channel attack against SGX. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020:321–347, 2020.
- [10] D. Jimenez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, 2001.
- [11] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.
- [12] C.-C. Lee, I.-C. Chen, and T. Mudge. The bi-mode branch predictor. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 4–13, 1997.
- [13] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 557–574, 2017.
- [14] K. W. Lu-Tan Zhao, Rui Hou, Y.-L. Su, P.-N. Li, and D. Meng. A novel probabilistic saturating counter design. *Journal of computer science and technology*, 36(5):1022, 2021.
- [15] S. McFarling. Combining branch predictors. Technical report, Technical Report TN-36, Digital Western Research Laboratory, 1993.
- [16] A. Seznec. The L-TAGE branch predictor. *J. Instr. Level Parallelism*, 9:1–13, 2007.
- [17] A. Seznec. A new case for the TAGE branch predictor. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 117–127, 2011.
- [18] A. Seznec. TAGE-SC-L branch predictors again. In *5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5)*, Seoul, South Korea, June 2016.
- [19] B. Sinharoy, J. A. Van Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. D. Brown, J. E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J. W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbach, T. Karkhanis, and K. M. Fernsler. IBM POWER8 processor core microarchitecture. *IBM Journal of Research and Development*, 59(1):2:1–2:21, 2015.
- [20] D. Suggs, M. Subramony, and D. Bouvier. The AMD “Zen 2” processor. *IEEE Micro*, 40(2):45–52, 2020.
- [21] D. Tarjan and K. Skadron. Merging path and gshare indexing in perceptron branch prediction. *ACM Trans. Archit. Code Optim.*, 2(3):280–300, 2005.