sBerti: Enhancing Berti with a Smart Stride Prefetcher for Better Coverage

Jiapeng Zhou
jzhou418@connect.hkust-gz.edu.cn
HKUST(GZ)
Guangzhou, China

Ben Chen
chanben04gz@gmail.com
HKUST(GZ)
Guangzhou, China

Kunlin Li
kli082@connect.hkust-gz.edu.cn
HKUST(GZ)
Guangzhou, China

Yun Chen
yunchen@hkust-gz.edu.cn
HKUST(GZ)
Guangzhou, China

Abstract

Hardware data prefetching remains a critical technique for mitigating the widening gap between processor speed and memory latency. Among state-of-the-art solutions, the Berti prefetcher has demonstrated significant efficacy by leveraging local delta history to manage prefetch timeliness. However, our detailed profiling of the DPC4 benchmark suite reveals that the default Berti prefetcher struggles to capture the streaming memory access patterns prevalent in AI-ML workloads. To address this limitation, we propose sBerti, a hybrid framework that augments Berti with a stride prefetcher component. This design facilitates aggressive stride prefetching with high degrees and ensures seamless cross-page boundary traversal. Evaluation demonstrates that sBerti outperforms the standalone Berti, achieving geometric mean IPC improvements of 3.9% and 1.4% under full-bandwidth and limit-bandwidth single-core configurations, respectively. Furthermore, our solution (L1D sBerti + L2 Pythia) surpasses the baseline by 1.8% in the full-bandwidth configuration, while maintaining comparable performance under limit-bandwidth configuration.

1 Introduction

Data prefetching remains a cornerstone of high-performance CPU design. Various algorithms have been proposed to predict future memory access addresses, broadly categorized into temporal, stream, stride, and spatial prefetchers. Berti [3], first introduced at the 3rd Data Prefetching Championship (DPC3), exploits page-level memory access patterns [1]. By tracking the timing of recent prefetches and memory accesses, Berti facilitates timely prefetching that dynamically adapts to runtime congestion.

However, our evaluation reveals significant limitations when applying Berti to DPC4 workloads. We observed particularly high L1D miss rates in newly introduced AI-ML and Google Trace V2 benchmarks. Detailed trace analysis indicates that AI-ML workloads (specifically Llama2 and BioGPT) exhibit stable and dense stride access patterns. Profiling suggests that Berti fails to capture these patterns effectively due to its conservative prefetch depth and the frequent interruption of prefetching streams at page boundaries, resulting in a high L1D cache miss rate.

[1]Note that an extended version of Berti was published in MICRO [2]. Unless otherwise specified, references to Berti in this paper denote the DPC3 implementation.

To address these shortcomings, we propose **sBerti**, a hybrid prefetching paradigm that augments Berti with a Smart Stride component. Inspired by the robust stride detection mechanisms in the Xiangshan processor [4, 5], our Smart Stride component employs heuristics to dynamically adjust confidence and depth for aggressive, cross-page prefetching. In the sBerti framework, the two engines operate on the same input stream but utilize distinct metadata tables to prevent history pollution. While Berti focuses on complex, page-localized patterns, the decoupled Smart Stride engine efficiently handles dense linear streams with greater continuity across page boundaries. Additionally, a deduplication buffer is implemented to filter redundant requests from the two concurrent engines.

Our evaluation demonstrates the efficacy of this hybrid approach. sBerti outperforms the standalone Berti, achieving geometric mean IPC improvements of 3.9% and 1.4% under full-bandwidth and limit-bandwidth single-core configurations, respectively. Furthermore, our solution (L1D sBerti + L2 Pythia) surpasses the baseline (L1D Berti + L2 Pythia) by 1.8% in the full-bandwidth configuration, while maintaining comparable performance (with a marginal 0.5% regression) under strictly constrained bandwidth conditions.

2 Smart Stride Prefetcher

We implemented the Smart Stride prefetcher by porting the `xs_stride` prefetcher logic from XiangShan [4], a high-performance open-source RISC-V processor. Its architecture is illustrated in Figure 1, and its key characteristics are summarized below.

Stride Training Table. The Smart Stride prefetcher manages stride training through a set-associative metadata table (**Stride Training Table**, SST). To minimize aliasing and maximize set utilization, the table index is derived from the instruction pointer (IP) using an XOR-folding hash function. specifically, the index is computed as:

$$Index = (((IP \gg 10) \oplus (IP \gg 20)) \ll 6) \mid (IP \,\&\, 0x3F) \quad (1)$$

Each entry within the set acts as a stream tracker, storing the `last addr`, the learned `stride`, a confidence counter (`conf`, 2bits), the current prefetch degree (`depth`, 4bits), and a timeliness heuristic counter (`late conf`, 4bits).

Stride Detection with OoO Support. Upon each memory access, the prefetcher calculates the delta (Δ) between the current byte address and the `last addr` stored in the matching entry. The

implementation employs a robust matching logic to handle out of order execution or sparse access streams. A match is declared valid not only if Δ equals the stored stride but also if the absolute delta is a multiple of the stored stride (Δ mod *stride* == 0), provided the stride exceeds the cache block size. If the pattern matches, the `conf` counter is incremented; otherwise, it is decreased. When `conf` reaches zero, the entry resets, adopting the current Δ as the new candidate stride.

Prefetch Issuing. To prevent cache pollution from unstable streams, the prefetcher employs a gating mechanism which requires `conf` ≥ 2 before any prefetch requests are generated. Additionally, the logic includes a noise filter that updates the `conf` only when the calculated Δ is smaller than the cache block size.

Heuristic Depth Adjustment. The prefetch lookahead depth is dynamically tuned using a specific heuristic algorithm governed by the `late conf` counter (initialized to 7). This counter is updated based on the following rules:

- **Late Prefetch:** If a demand miss occurs for an address present in the `recent prefetches` buffer (indicating a late prefetch), `late conf` is incremented by 3 (penalizing latency).
- **Timely Prefetch:** If a useful prefetch hit occurs, `late conf` is decremented by 1.

The depth is adjusted based on specific thresholds: if `late conf` rises to ≥ 12, the `depth` is incremented to cover more latency; if `late conf` drops to ≤ 3, the `depth` is decremented to conserve bandwidth. Upon any depth change, the `late conf` is reset to the midpoint 7.

Unlike the Gem5 implementation of Smart Stride, which identifies late prefetches via the `calculatePrefetch` interface, ChampSim lacks an equivalent mechanism. To bridge this gap, we leverage a `recent prefetch` table to record issued requests for both deduplication and late prefetch identification. Specifically, we classify a cache miss as a late prefetch if its address matches an entry in the `recent prefetch` table.

3 sBerti: A Hybrid Prefetching Framework Combining Berti and Smart Stride

We present sBerti, a decoupled hybrid prefetching framework designed to exploit the complementary strengths of Smart Stride and Berti. The framework employs a dual-engine architecture, integrating a Berti component for complex pattern recognition and a Smart Stride component for robust linear stream detection.

The baseline Berti is a page-localized, latency-aware prefetcher that excels at capturing intra-page, repeating Delta patterns (e.g., +1, +3, +7). However, for simple, long-distance linear streams, Berti's history construction process can be computationally intensive and slow to converge. To address this, sBerti employs Smart Stride as a lightweight, decoupled engine. By leveraging the Instruction Pointer (IP), this component efficiently identifies stable linear strides and aggressively issues prefetches across page boundaries when confidence is high. The Smart Stride component further incorporates heuristic depth adjustment to ensure timely prefetching.

To coordinate these two engines, sBerti implements a unified `Recent Prefetches` buffer to filter redundant requests. Furthermore, Smart Stride utilizes a separate, lightweight PC-based stride table to prevent pollution of Berti's global history structures.

Compared to Berti, sBerti offers the following key advantages:

① **Continuity Across Page Boundaries.** sBerti effectively mitigates the "Cold-start Penalty" inherent to page-based prefetchers. The baseline Berti often suffers from prefetching interruptions at 4KB physical page boundaries, as prefetching across pages incurs low accuracy and low prefetch degrees. Consequently, prefetching for a new page is generally triggered only upon a demand access to that page. This reactive behavior proves insufficient for dense stride patterns, as the prefetch issuance often occurs too late to effectively hide the memory latency. By leveraging Smart Stride, which tracks access patterns in the virtual address space, sBerti maintains predictive continuity across page frames. This ensures that data prefetching remains uninterrupted, enabling seamless traversal across page boundaries with high confidence and aggressive prefetch degrees.

② **Balance of Aggressiveness and Accuracy.** To ensure high accuracy, Berti relies on strict confidence checks and a low prefetch degree defined by `L1D_MAX_NUM_BURST_PREFETCHES`. sBerti incorporates Smart Stride's dynamic depth adjustment, allowing the system to aggressively increase the prefetch degree for high-confidence linear streams while maintaining precision for irregular patterns. This enables dynamic modulation of prefetching aggressiveness based on stream access patterns, especially for AI-ML workloads.

4 Storage Overhead

The sBerti prefetcher deployed at L1D levels consumes a total storage of 30.03 KB for each core. As depicted in Table 1, sBerti involves three major structures of storage: Berti, Smart Stride (SS) and Recent Prefetch (RP).

①**Berti.** The largest component is the Berti structures composed of Current Pages (2110 B), Previous Request (3584 B), Previous Prefetch (1856 B), Recorded Page (17409 B) and IP (1408 B). These tables incur 23.86 KB of storage. Note that while the size of each entry is identical with the origin Berti, we tune the amounts of entries to spare space for the other structures.

②**Smart Stride.** The Smart Stride table accounts for 2880 bytes as a 4 ways set-associative structure of 64 sets. For each entry, it stores tags (partial PC), last address (48 bits), stride (13 bits of signed number), confidence (2 bits), depth (4 bits), timeliness counter (4 bits), LRU counter (2 bits) and valid bit, 90 bits in sum.

③**Recent Prefetch.** Recent prefetch table tracks address bits of recent prefetch histories (48 bits) with 256 entries of records, additionally with a head entry (8 bit). Thus, this table occupies 1537 B of storage.

Since our solution employs the same Pythia L2 prefetcher as the baseline, the storage overhead at this level remains identical. A detailed breakdown is provided in Table 1.

5 Experimental Setup & Results

We evaluate sBerti using the Champsim simulator and traces provided by DPC4. As shown in Table 2, our configuration deploys sBerti as the L1D prefetcher and Pythia as the L2 prefetcher. Notably,

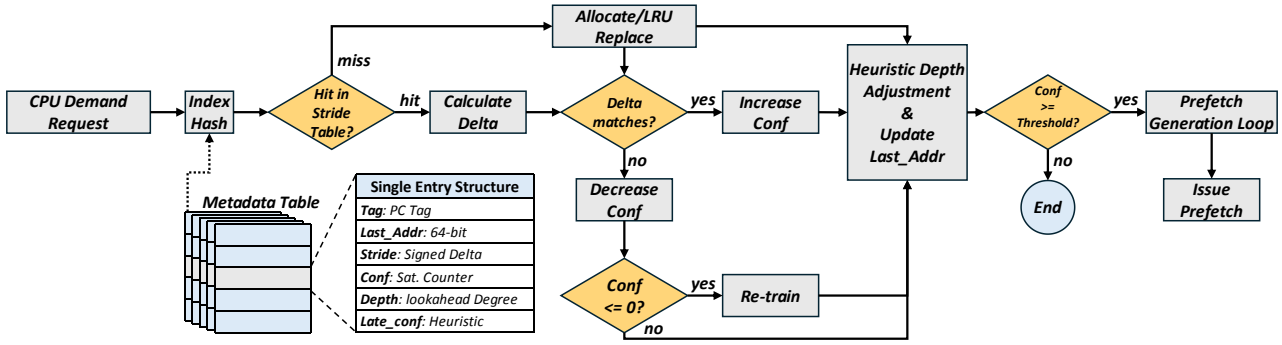

Figure 1: Block diagram of the Smart Stride Prefetcher.

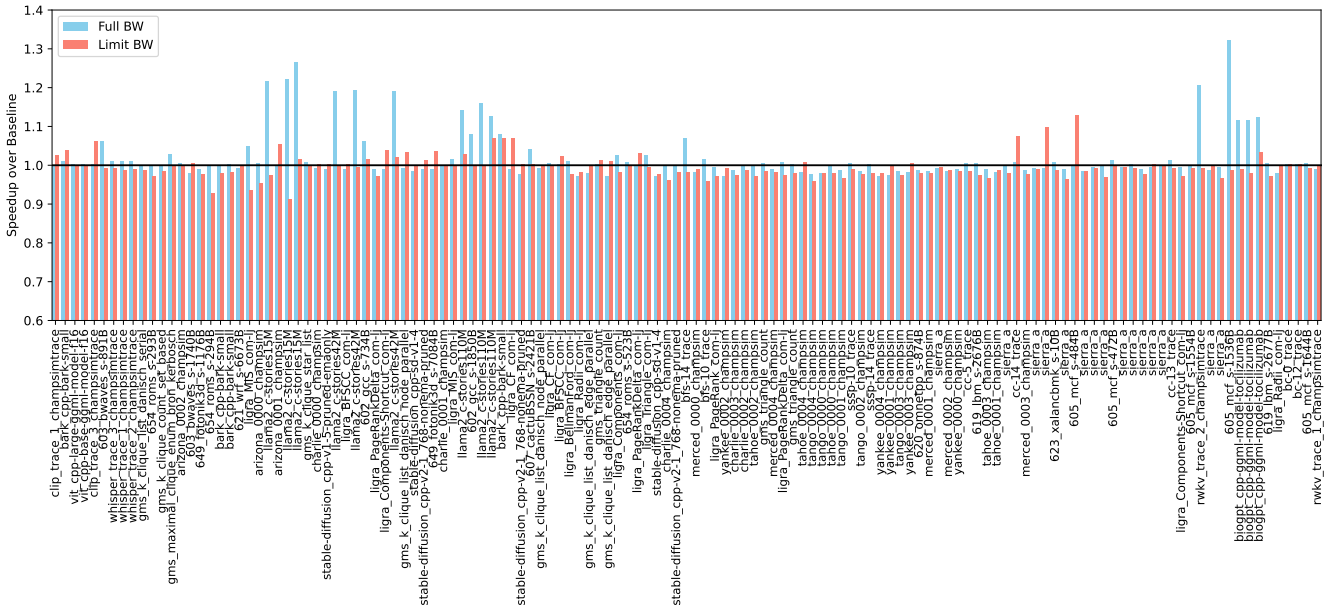

Figure 2: IPC speedup of our design (L1D sBerti + L2 Pythia) normalized to the corresponding baseline (L1D Berti + L2 Pythia) under full and limited bandwidth configurations.

we disable prefetching at the Last Level Cache (LLC) to avoid performance regression caused by bandwidth saturation in bandwidth-constrained configurations.

Performance Comparison with Berti. We evaluate sBerti against the standalone Berti prefetcher across 131 traces. The results demonstrate that sBerti consistently outperforms Berti, achieving geometric mean IPC improvements of 3.9% and 1.4% under full-bandwidth and bandwidth-constrained single-core configurations, respectively.

To further verify the source of the gains, specifically regarding dense stride patterns, we analyzed the distribution of memory access deltas and calculated the proportion of +1/-1 strides relative to all non-zero deltas for each workload. Results in Figure 3 reveal a clear trend: for the 39 workloads dominated by dense stride patterns (where prevalence exceeds 50%), sBerti delivers speedups in 34 cases, achieving an average improvement of 12.14% across

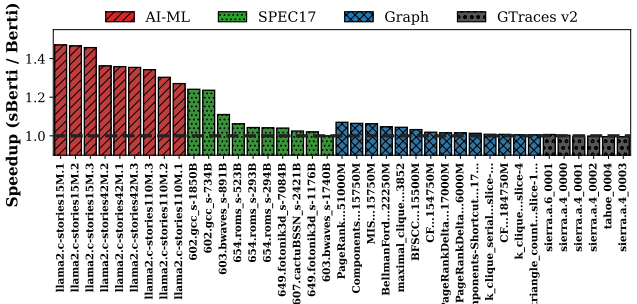

Figure 3: IPC speedup of sBerti over standalone Berti under the full-bandwidth configuration for workloads with high streaming intensity (>50% stream accesses)

Level	Structure	Bits per entry	Number of entries	Total
L1D	Berti Current Pages	$36 + 10 + 64 + (8 + 6) \times 10 + 6 + 6$	$2^6 - 1$	2110 B
L1D	Berti Previous Requests	$6 + 6 + 16$	2^{10}	3584 B
L1D	Berti Previous Prefetches	$6 + 6 + 16 + 1$	2^9	1856 B
L1D	Berti Recorded Pages	$32 + 64 + 6 + 8 + 11$	$2^{10} + 2^7 - 1$	17409 B
L1D	Berti IP	11	2^{10}	1408 B
L1D	Smart Stride	$16 + 48 + 13 + 2 + 4 + 4 + 2 + 1$	$2^2 \times 2^6$	2880 B
L1D	Recent Prefetches	48	2^8	1537 B
Total	sBerti Prefetcher			30.06 KB
L2C	Signature	$36 + 48 + 6 + 7 + 64 + 48 + 6 + 1 + 2 \times (6 + 4) + 64 + 64 + 10$	2^6	4944 B
L2C	Prefetch Tracker	$48 + (48 + 48 + 7 + 12 + 32 + 24 + 1) + 6 + 1 + 1 + 5 + 3 + 1$	2^8	7584 B
L2C	QV Store	$2 \times 3 \times 128 \times 32 \times 16$	/	48 KB
Total	Pythia Prefetcher			60.24 KB

Table 1: Storage consumption breakdown of sBerti and Pythia prefetcher.

Cache Level	Prefetcher	Description
L1D	sBerti	Hybrid prefetcher combining Smart Stride and Berti.
L2	Pythia [1]	Reinforcement-learning (RL)-based prefetcher
LLC	None	Prefetching disabled

Table 2: Our prefetcher configuration for L1, L2, and LLC caches.

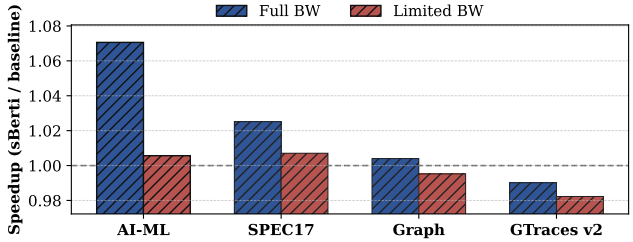

Figure 4: IPC speedup of our design normalized to the baseline (L1D Berti + L2 Pythia), categorized by workload class under full and limited bandwidth configurations.

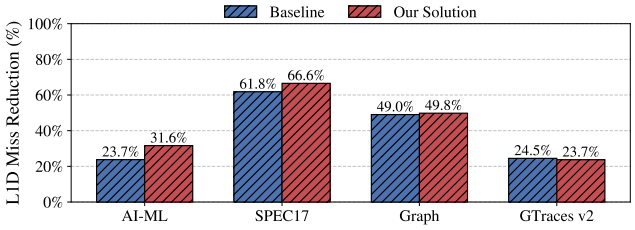

Figure 5: L1D Prefetch coverage (i.e., miss reduction ratio compared to no prefetcher) of our solution versus the baseline (L1D Berti + L2 Pythia), categorized by workload class under the full-bandwidth configuration.

all 39 workloads. Notably, the benefits of sBerti extend beyond pure streaming workloads—performance improvements are also observed in approximately half of the remaining 92 workloads.

Comparison with Baseline (L1D Berti + L2 Pythia). Figure 2 presents the IPC speedup normalized to the baseline (L1D Berti + L2 Pythia). Overall, our solution (L1D sBerti + L2 Pythia) surpasses the baseline by 1.8% under the FullBW configuration, while incurring a negligible regression of only 0.5% under the LimitBW configuration.

Figure 4 provides a breakdown of geometric mean speedups by workload category. Under the FullBW configuration, AI-ML workloads demonstrate the most substantial gains (**+7.1%**), followed by SPEC17 (**+2.5%**) and Graph (+0.4%). Conversely, Google Traces_v2 incurs a slight regression (-1.0%). In the LimitBW configuration, however, these performance benefits are largely attenuated due to bandwidth contention. AI-ML and SPEC17 retain only modest improvements of +0.6% and +0.7%, respectively, while Graph (-0.5%) and Google Traces_v2 (-1.8%) suffer minor degradations.

For workloads characterized by dense stride patterns, such as AI-ML, sBerti delivers substantial performance gains when bandwidth is abundant. However, under constrained bandwidth, the aggressive prefetch requests compete with demand accesses, thereby attenuating the performance benefits.

Figure 5 compares the L1D prefetch coverage of our solution against the baseline under the full-bandwidth configuration, broken down by workload category. Most notably, coverage for AI-ML workloads rises substantially from 23.7% to 31.6%, while SPEC17 also sees a solid improvement from 61.8% to 66.6%.

Figure 6 compares the L1D prefetch accuracy of our solution against the baseline under the full-bandwidth configuration. As shown in the figure, the accuracy exhibits varying degrees of reduction. This indicates that there is still ample headroom for further optimization.

6 Conclusion

This paper addresses the limitations of the state-of-the-art Berti prefetcher in handling dense, cross-page streaming patterns in

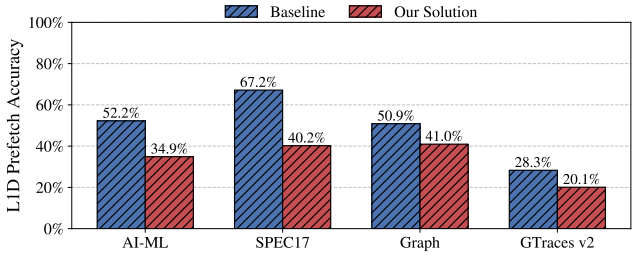

Figure 6: L1D Prefetch accuracy of our solution versus the baseline (L1D Berti + L2 Pythia), categorized by workload class under the full-bandwidth configuration.

AI-ML workloads. We introduce **sBerti**, a hybrid framework synergizing Berti's precision with a robust Smart Stride prefetching engine. By decoupling stream detection from page-based history, sBerti enables aggressive, seamless cross-page stride prefetching while capturing complex page access patterns.

Evaluations using DPC4 traces demonstrate that sBerti outperforms standalone Berti, delivering geometric mean IPC improvements of 3.9% and 1.4% under full and limited bandwidth configurations. Furthermore, alongside Pythia at L2, our design surpasses the baseline (L1D Berti + L2 Pythia) by 1.8% in full bandwidth configuration while maintaining parity under limit bandwidth configuration.

7 Acknowledgments

This work was generously supported by the Guangdong Municipal Science and Technology Project (Nos. 2024QN11X196). We would like to thank the anonymous reviewers for their helpful feedback.

References

[1] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1121–1137. doi:10.1145/3466752.3480114

[2] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros. 2022. Berti: an Accurate Local-Delta Data Prefetcher. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 975–991. doi:10.1109/MICRO56248.2022.00072

[3] Alberto Ros. 2019. Berti: A Per-Page Best-Request-Time Delta Prefetcher. In *The 3rd Data Prefetching Championship (DPC-3)*.

[4] XiangShan. 2025. GEM5 Source Code: xs_stride.cc. https://github.com/OpenXiangShan/GEM5/blob/xs-dev/src/mem/cache/prefetch/xs_stride.cc GitHub repository. Accessed: 2025-12-20.

[5] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, Jiawei Lin, Tong Liu, Zhigang Liu, Jiazhan Tan, Huaqiang Wang, Huizhe Wang, Kaifan Wang, Chuanqi Zhang, Fawang Zhang, Linjuan Zhang, Zifei Zhang, Yangyang Zhao, Yaoyang Zhou, Yike Zhou, Jiangrui Zou, Ye Cai, Dandan Huan, Zusong Li, Jiye Zhao, Zihao Chen, Wei He, Qiyuan Quan, Xingwu Liu, Sa Wang, Kan Shi, Ninghui Sun, and Yungang Bao. 2022. Towards Developing High Performance RISC-V Processors Using Agile Methodology. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1178–1199. doi:10.1109/MICRO56248.2022.00080