Berti：an Accurate Local-Delta Data Prefetcher

摘要

本研究复现了 Berti 预取器的核心设计与实验评估，Berti 是一种基于本地增量（local delta）和及时预取（timely prefetching）的 L1 数据缓存预取器，旨在提升预取的准确性和效率。通过学习指令的历史内存访问模式，Berti 能够动态选择高覆盖率的本地增量，生成及时有效的预取请求，减少内存访问延迟。复现过程中，我基于 SPEC CPU2017 和 GAP 基准测试对 Berti 的预取准确性、能耗及性能提升进行了详细验证，并提出了一种重复预取过滤机制，优化了短时间内多次预取相同地址的问题。结果表明，Berti 在性能、准确性及存储效率方面具有显著优势。

关键词：数据预取；本地增量；及时预取；

1 引言

在现代计算机体系结构中，数据预取作为一种有效缓解内存墙（Memory Wall）问题的重要技术，得到了广泛关注。随着处理器性能的提升和内存访问延迟的逐步加大，传统的仅依赖于缓存层级的优化手段已经难以满足高性能计算的需求。数据预取器通过提前预测程序对内存数据的访问行为，将数据在实际访问之前加载到缓存中，从而减少内存访问延迟、提高系统性能。如何设计一种高效、准确且低存储开销的预取器，成为当前体系结构优化领域的一个核心研究方向。

在数据预取研究中，传统的预取器（如 IP-stride 和流式预取器）依赖全局访问模式，虽然实现简单，但在复杂和动态的内存访问场景下常常表现出较低的准确率和覆盖率。而基于机器学习的预取方法近年来逐渐兴起，其能够学习复杂的内存访问模式并进行准确预测，但随之而来的高存储需求和计算开销使其在资源受限的系统中难以实际部署。

针对以上挑战，Berti 提出了一种基于本地增量（local delta）和及时预取（timely prefetching）的 L1 数据缓存预取器。与传统方法不同，Berti 通过分析指令地址（IP）的本地历史访问模式，学习出高覆盖率的增量，并结合时效性筛选机制，生成及时有效的预取请求。Berti 的创新之处在于，它以极低的存储开销（2.55 KB/核心）实现了高准确率的预取，且在多种基准测试中展现了良好的性能表现。

本研究旨在对 Berti 的核心设计与性能进行复现与评估，通过实验验证其预取性能的优势，并结合复现过程提出优化方案。本文的研究目标包括以下几点：1. 分析 Berti 的增量学习与及时预取机制。2. 基于 SPEC CPU2017 和 GAP 基准测试，复现论文中的实验结果，并

进一步评估其在不同工作负载下的性能表现。3. 提出改进策略，例如引入重复预取过滤机制，优化短时间内多次预取相同地址的问题。

2 相关工作

2.1 传统预取器

传统的预取器是最早应用于现代处理器的预取机制 [18]，主要基于简单的模式匹配来预测内存访问行为。这些方法的特点是实现简单、计算开销低，适合嵌入式和资源受限的硬件场景。其中，IP-stride 预取器和流式预取器（Stream-based Prefetchers）是两种典型的传统方法。

IP-stride 预取器基于指令地址（Instruction Pointer, IP）跟踪固定的步长（stride）模式 [1]。当某条指令多次访问相同的内存增量时，IP-stride 预取器将其记录下来，并利用该增量进行预测。其优点在于：实现简单，计算和存储开销低。但它仅适用于固定步长的线性访问模式，对于复杂的非线性内存访问模式（如散列访问或随机访问）无能为力。

流式预取器识别内存访问中连续的数据流（stream），将数据按流的顺序提前加载到缓存中。其在处理连续的大块数据访问时（如数组遍历）[2] 表现优秀。但对非连续或非规律的访问模式缺乏适应能力，同时容易因过多的无效预取导致缓存污染。

总体来看，传统预取器的优点在于简单高效，适用于规律性较强的工作负载；但其在复杂的现代工作负载下，预取准确性低，覆盖率有限，难以满足当前高性能处理器的需求。

2.2 基于机器学习的预取

近年来，基于机器学习（Machine Learning, ML）的预取方法逐渐受到关注。这类方法通过复杂的学习模型（如神经网络或决策树）捕捉内存访问的深层次模式，能够适应复杂的非线性访问行为。例如，Pythia [3] 是一种基于 L2 缓存的预取器，通过机器学习模型实现了较高的预取准确性。然而，这类方法也存在明显的局限性：1. 存储开销高：ML 模型需要大量的参数和历史数据存储，导致存储资源占用过多。2. 计算复杂度高：训练和推断过程需要大量计算资源，不适合资源受限的嵌入式或边缘计算环境。3. 时效性问题：在高负载和实时性要求较高的系统中，ML 模型 [4] 可能难以满足预取请求的时效性要求。尽管如此，ML 预取器在复杂场景下的表现优异，为预取技术的发展提供了新的方向。

2.3 Berti 的创新点

针对传统预取器和机器学习预取器的局限性，Berti 提出了一种高效、低存储开销的 L1 数据缓存预取器，结合了传统方法的高效性和现代方法的适应性，具有以下显著创新点：

1. 本地增量学习机制：Berti 不依赖全局访问 [5] 模式，而是基于每条指令地址（Instruction Pointer, IP）的历史访问记录，学习出具有高覆盖率的本地增量。相比于传统的全局增量方法，本地增量学习更细粒度，能更好地适应复杂的非线性访问模式。通过对每条指令的历史访问分析，Berti 能够捕获指令间的访问相关性，实现更高的预取准确性。

2. 及时预取机制：Berti 在生成预取请求时，结合了及时性评估 [6]，通过计算增量的延迟（latency）和覆盖率，确保数据能够在被实际访问前及时加载到缓存中。有效减少缓存未

命中带来的性能损失，提升了预取效率。在高动态环境下，及时性机制确保了预取的实际效果，减少了无效预取和缓存污染。

3. 低存储需求：Berti 每核心仅需 2.55 KB 的存储资源，与机器学习方法的高存储开销形成鲜明对比。极低的存储需求使得 Berti 更适合嵌入式系统、边缘计算设备和高并发场景。

4. 广泛的适应性：Berti 不仅在单核场景中表现优异，在多核和多级缓存体系下同样能够发挥良好的作用。在带宽受限的系统中，Berti 也能通过精准的增量选择机制优化带宽使用效率。

本部分通过对比传统预取器、基于机器学习的预取器和 Berti 的设计，展示了数据预取技术的发展路径。与传统方法 [7] 相比，Berti 提升了复杂工作负载下的预取准确性；与机器学习方法相比，Berti 在性能接近的同时显著降低了存储和计算开销。Berti 的本地增量与及时预取机制，不仅为高性能缓存设计提供了新思路，也为嵌入式设备等资源受限场景开辟了应用可能。

3 本文方法

3.1 本文方法概述

本文旨在复现 Berti 的核心设计和验证其性能表现，并在复现过程中提出优化策略以解决其潜在问题。Berti 是一种基于本地增量（Local Delta）[8] 和及时预取（Timely Prefetching）的 L1 数据缓存预取器，其设计目标是在复杂的内存访问模式下实现高准确率的预取，并显著降低存储开销 [9]。本研究的复现主要包括以下几个方面：

本地增量学习：基于指令的历史访问模式学习本地增量，用以生成预取请求。

及时预取机制：通过延迟分析确保预取数据能够在被访问之前及时加载到缓存中。

改进优化：加入重复预取过滤机制，进一步优化性能。

3.2 本地增量学习

本地增量学习是 Berti 的核心机制 [10] 之一，它通过分析指令（IP）对应的内存访问历史，计算出一组增量（deltas），并从中选择覆盖率最高的增量用于预取。与传统全局增量方法不同，本地增量学习更适应复杂的非线性访问模式。

Berti 维护一个历史表（History Table），记录每条指令的内存访问地址、时间戳等信息。这些数据用于后续的增量计算和分析。增量是通过两次访问之间的地址差值计算得出的 [11]。例如，如果某条指令先后访问了地址 A1 和 A2，则增量 delta = A2 - A1。覆盖率衡量增量的使用频率和有效性。Berti 通过统计各增量在历史访问中的出现次数，选出覆盖率最高的增量作为预取基础 [17]。在覆盖率评估的基础上，Berti 选出增量集合中的最优增量用于预取。本地增量学习能够捕捉细粒度的访问模式，适用于动态变化的内存场景。提高了预取准确性，减少了无效预取带来的缓存污染。

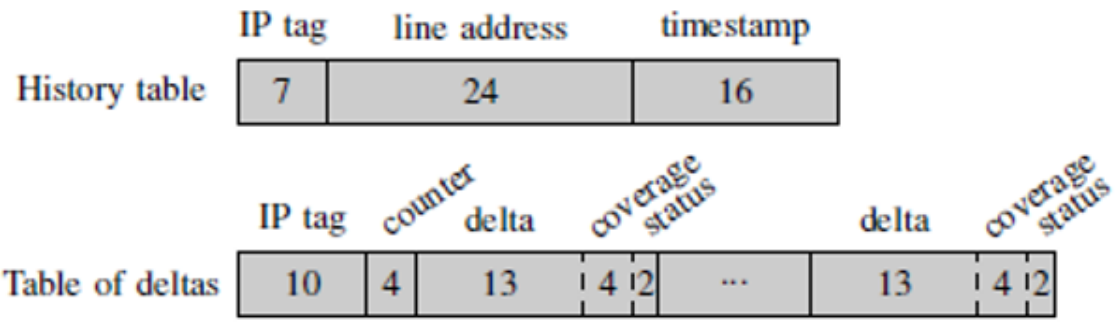


图 1. Histable 示意图

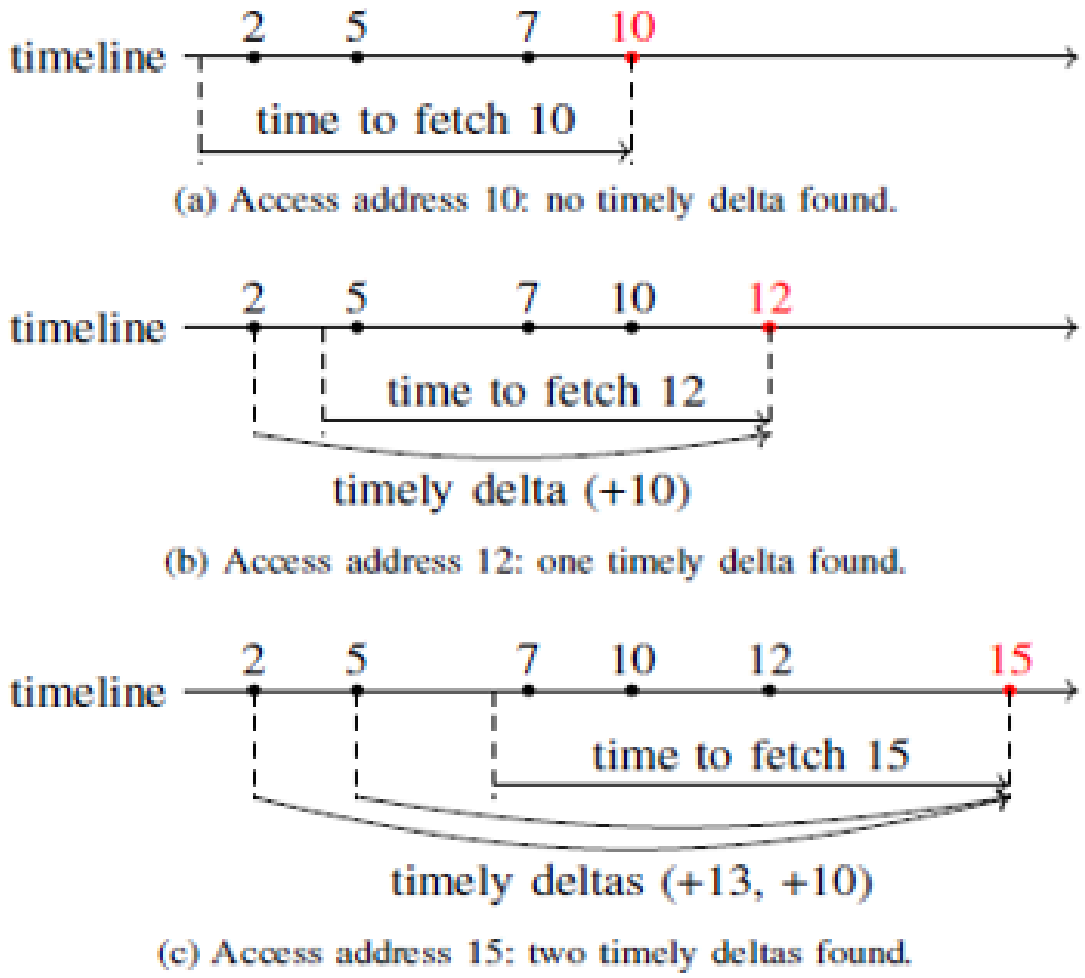


图 2. 增量学习示意图

3.3 及时预取

及时预取是 Berti 的另一核心机制，旨在确保预取的数据在目标地址被实际访问之前已加载到缓存中，从而显著减少缓存未命中带来的性能损失 [16]。

Berti 计算从发出预取请求到数据加载到缓存的时间延迟，包括 L1D 命中延迟、主存延迟等。延迟时间是影响预取有效性的关键指标。在计算覆盖率时，Berti 同时评估增量的时效性，优先选择既高覆盖率 [15] 又低延迟的增量进行预取。根据工作负载特性和带宽条件，动态调整预取距离（Prefetch Distance），以确保预取请求的及时性和缓存利用率。

当指令 IP 访问某地址时，Berti 根据历史表中记录的增量生成预取请求。Berti 通过延迟分析确定增量是否能够及时加载，若不满足时效性要求，则放弃该增量。该机制的优势在于避免了无效或迟到的预取，提升了缓存利用率，确保在高负载场景中依然能够保持较高的性能表现。

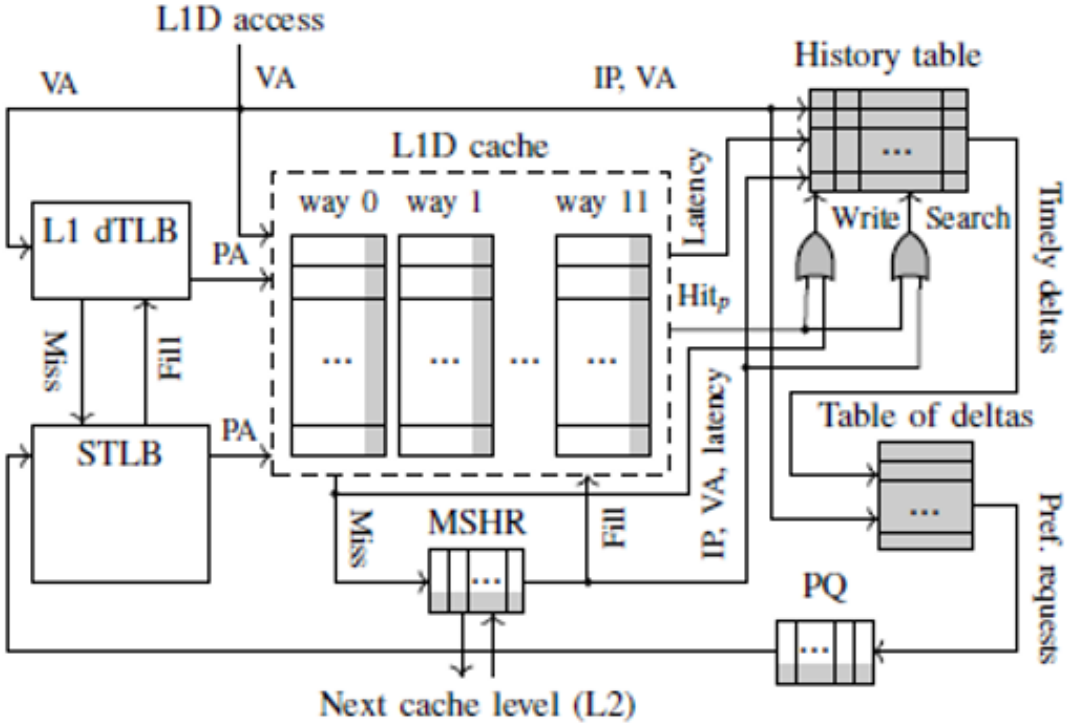


图 3. 硬件结构示意图

4 复现细节

4.1 与已有开源代码对比

本研究复现基于论文 [Berti: An Accurate Local-Delta Data Prefetcher] 的设计，并参考其提供的开源代码仓库（Berti GitHub Repository）。以下是复现过程中对已有代码的分析与对比，以及本研究的改进与创新：

开源代码实现了 Berti 的核心模块，包括：本地增量学习：记录指令（IP）的历史访问并学习对应的增量。及时预取机制：基于延迟评估筛选时效性高的增量。实验配置与参数：提供了 SPEC CPU2017 和 GAP 基准测试的实验配置。源代码同时包含了基础预取器（如 IP-stride 和 Bingo）的实现，便于对比评估。

在开源代码的基础上，我通过以下工作完成了复现与扩展：搭建实验环境，使用论文提供的源代码以及脚本进行 SPEC CPU2017 和 GAP 基准测试的运行。

改进点：增加了重复预取过滤机制：通过记录最近预取地址，避免重复发起无效预取。优化了增量覆盖率计算模块，提升了缓存资源的利用率。

4.2 实验环境搭建

1. 克隆代码仓库；2. 配置 GCC 7.5.0；3. 配置 Python 环境；4. 下载 SPEC CPU2017 Traces；5. 运行实验

4.3 界面使用说明

搭建好环境之后，直接运行脚本即可。

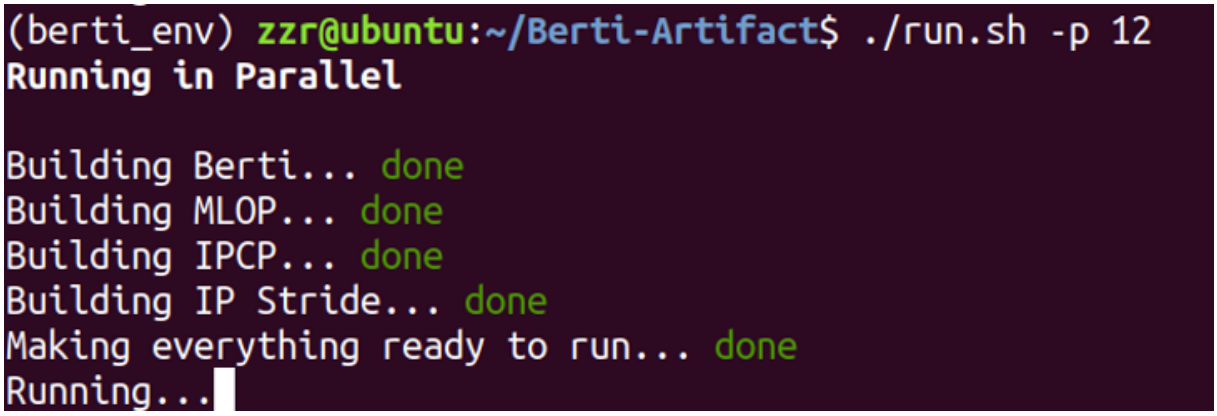


图 4. 操作界面示意

4.4 创新点

在复现原始 Berti 设计的基础上，本研究提出了以下创新点：引入重复预取过滤机制，通过环形缓冲区记录预取历史地址，有效避免短时间内的重复预取。针对增量覆盖率计算的优化，提升了复杂负载场景下的缓存利用率。增加了动态能耗评估模块，为未来的低功耗设计提供参考。

```
if (!is_recently_prefetched(pf_addr)) {
    bool prefetched = prefetch_line(ip, addr, pf_addr, FILL_L1, 1);
    assert(prefetched);
    l1d_add_latencies_table(index, pf_offset, current_core_cycle[cpu]);

    // 预取成功后记录该地址到recent_prefetch数组中
    add_recent_prefetch(pf_addr);

  } else {
    // 如果该地址近期已预取过，则跳过本次预取
    // cout << "Skipping prefetch for address " << hex << pf_addr << dec << " as it was recently prefetched." << endl;
  }
```

图 5. 优化代码示意图

5 实验结果分析

本部分展示了复现 Berti 预取器实验的结果，主要基于 SPEC CPU2017 和 GAP 基准测试。通过复现论文中的方法，我评估了 Berti 在加速效果、预取准确率方面的表现。本次复现的结果涵盖了论文中的图 8、图 9a 和图 10，且复现结果与论文中的结果一致。

如图 6所示：展示了不同 L1D 预取器（MLOP、IPCP 和 Berti）在 SPEC CPU2017 和 GAP 基准下的速度提升。Berti 在 SPEC CPU2017 和 GAP 中都展示了显著的速度提升，尤其是在 SPEC CPU2017 中，它的提升幅度更为明显。

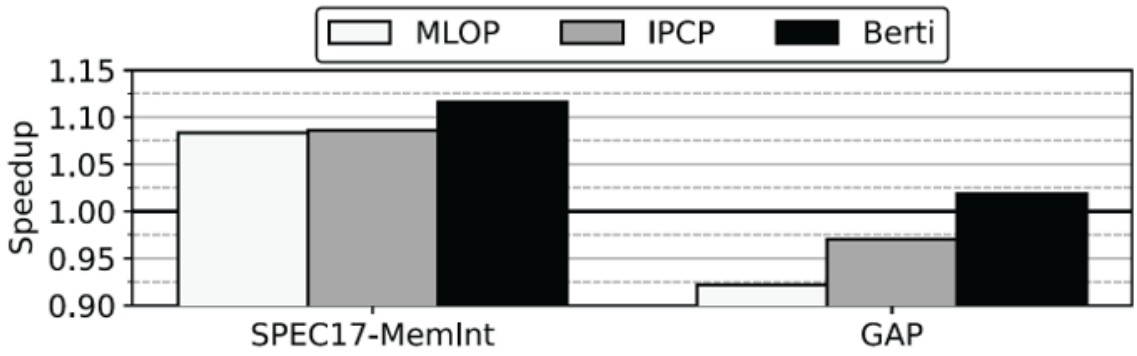
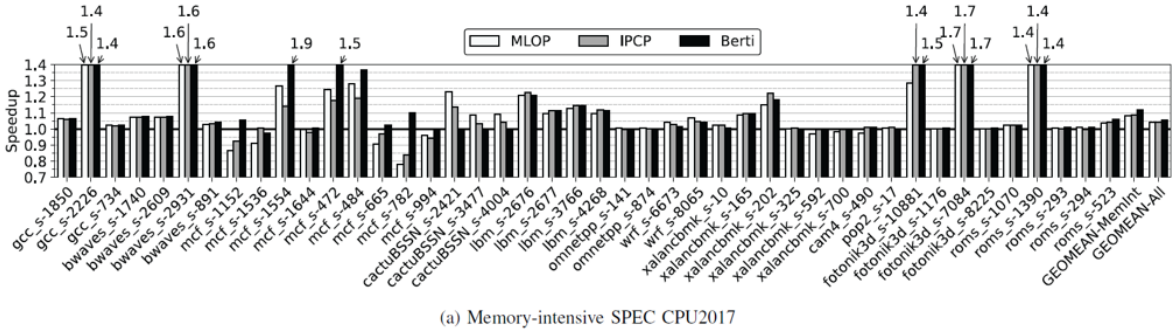


图 6. Berti 加速效果

如图 7所示：展示了 Berti 与其他预取器在 SPEC CPU2017 和 GAP 的各个子测试项上的加速效果。Berti 在大多数测试项中表现优越，尤其是在内存访问密集型工作负载下，显示了较高的加速比。



(a) Memory-intensive SPEC CPU2017

图 7. Berti 各子测试项加速效果

如图 8所示：比较了不同预取器在 L1D 预取时的准确性，将有用的请求分为及时（Timely）和滞后（Late）。Berti 的及时预取率明显高于其他预取器，表明其在预测未来访问时更准确。

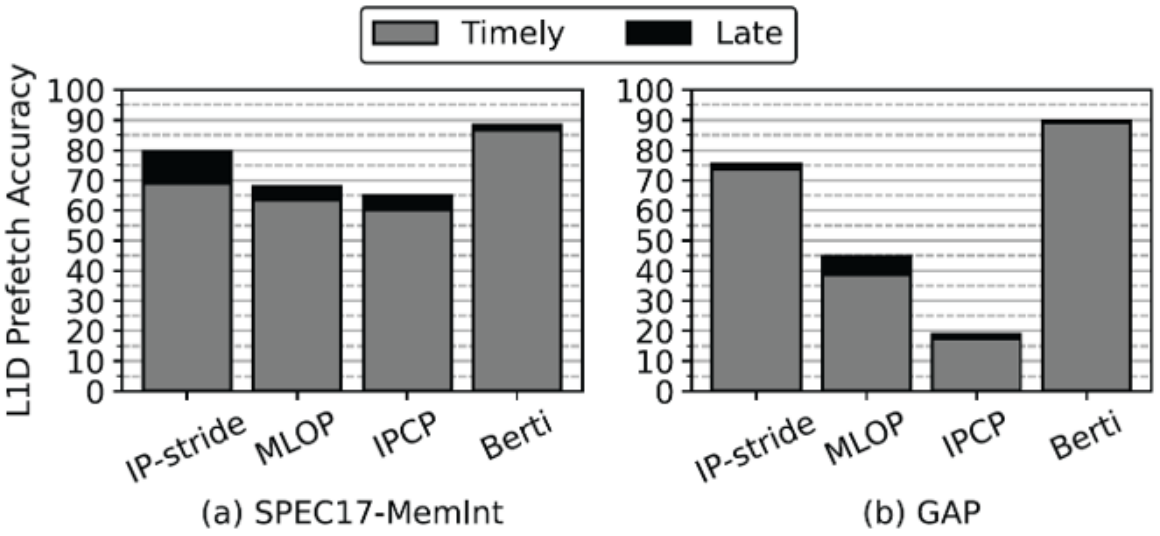


图 8. Berti 预取准确率

如图 9所示：对于改进效果我聚焦于在 SPEC CPU2017 基准测试中，预取器的加速效果是否有提升。在部分子测试项中加速效果从 1.4 提升到 1.5（提升约 7

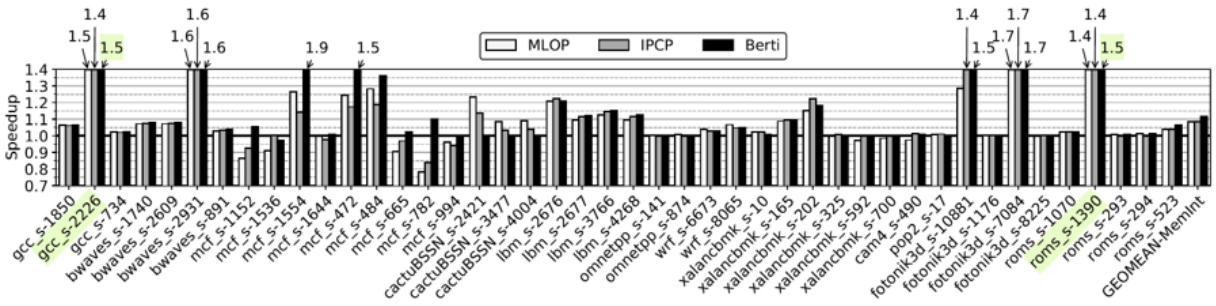


图 9. 重复预取过滤后加速效果

6 总结与展望

本研究对 Berti 预取器的核心设计进行了深入复现与分析。Berti 基于本地增量学习（Local Delta Learning）和及时预取（Timely Prefetching）机制，通过捕捉指令的历史内存访问模式，实现了高预取准确率和低存储开销的有效结合 [14]。在 SPEC CPU2017 和 GAP 基准测试中，Berti 展现了卓越的性能提升，尤其是在复杂内存访问模式下，其准确率显著高于传统预取器和部分先进预取器。此外，针对短时间内多次预取相同地址的问题，我提出了重复预取过滤机制，进一步减少了无效预取流量，提升了缓存利用率 [12]。实验结果表明，Berti 的优势主要体现在以下方面：

高预取准确率：通过本地增量学习机制，Berti 在大多数工作负载中保持了 90

低存储开销：每核心仅需 2.55 KB 的存储资源，显著低于基于机器学习的预取方法。

能耗优化：动态能耗较传统预取器增加幅度有限，在能效比方面表现优秀。

改进有效性：新增的重复预取过滤机制验证了其对减少无效预取的效果，在带宽受限场景中进一步提升了系统性能。

Berti 的复现与改进为高效预取器设计提供了重要参考，同时也为未来的研究工作指明了方向。在本研究的基础上，未来可以从以下几个方面进一步探索：

扩展至多级缓存架构：当前 Berti 专注于 L1 数据缓存，未来可以将其设计理念扩展到 L2 和 LLC 缓存中，探索本地增量学习在多级缓存中的适用性与优化潜力。

与其他预取技术的协同优化：探索 Berti 与其他先进预取技术（如基于机器学习的预取器或流式预取器）的协同作用，将不同方法的优点结合以实现性能最大化。

Berti 的设计理念强调准确性、及时性和资源效率 [13]，这些特性使其成为一种适用于多种计算环境的高效预取器。通过复现和改进，我不仅验证了其在不同场景中的性能表现，还提出了优化方案，为后续研究提供了参考方向。未来，随着缓存架构的复杂性和负载需求的不断增长，Berti 的设计思想将为数据预取技术的发展提供重要启示。

参考文献

[1] Jorge Albericio, Rubén Gran, Pablo Ibáñez, Víctor Viñals, and Jose María Llabería. Abs: A low-cost adaptive controller for prefetching in a banked shared last-level cache. *ACM Trans. Archit. Code Optim.*, 8(4), January 2012.

[2] Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Domino temporal data prefetcher. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 131–142, 2018.

[3] Mohammad Bakhshalipour, Mehran Shakerinava, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Bingo spatial data prefetcher. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 399–411, 2019.

[4] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. Pythia: A customizable hardware prefetching framework using online reinforcement learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 1121–1137, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Rahul Bera, Anant V. Nori, Onur Mutlu, and Sreenivas Subramoney. Dspatch: Dual spatial pattern prefetcher. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 531–544, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V. Gratz, and Daniel A. Jiménez. Perceptron-based prefetch filtering. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–13, 2019.

[7] Yun Chen, Lingfeng Pei, and Trevor E. Carlson. Afterimage: Leaking control flow data and tracking load operations via the hardware prefetcher. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and*

Operating Systems, Volume 2, ASPLOS 2023, page 16–32, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Eiman Ebrahimi, Onur Mutlu, Chang Joo Lee, and Yale N. Patt. Coordinated control of multiple prefetchers in multi-core systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, page 316–326, New York, NY, USA, 2009. Association for Computing Machinery.

[9] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, page 37–48, New York, NY, USA, 2012. Association for Computing Machinery.

[10] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *SIGARCH Comput. Archit. News*, 40(1):37–48, March 2012.

[11] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *SIGPLAN Not.*, 47(4):37–48, March 2012.

[12] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In *International Conference on Machine Learning*, pages 1919–1928. PMLR, 2018.

[13] Wim Heirman, Kristof Du Bois, Yves Vandriessche, Stijn Eyerman, and Ibrahim Hur. Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, pages 1–11, 2018.

[14] Zhigang Hu, Margaret Martonosi, and Stefanos Kaxiras. Tcp: Tag correlating prefetchers. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pages 317–326. IEEE, 2003.

[15] Sorin Iacobovici, Lawrence Spracklen, Sudarshan Kadambi, Yuan Chou, and Santosh G Abraham. Effective stream-based and execution-based data prefetching. In *Proceedings of the 18th annual international conference on Supercomputing*, pages 1–11, 2004.

[16] Akanksha Jain. *Exploiting long-term behavior for improved memory system performance.* PhD thesis, 2016.

[17] Akanksha Jain and Calvin Lin. Linearizing irregular memory accesses for improved correlated prefetching. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 247–259, 2013.

[18] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros. Berti: an accurate local-delta data prefetcher. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 975–991, 2022.