Pythia & Hermes: ML-Driven Prefetching

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
20 September 2023
VMware

SAFARI ETH zürich Carnegie Mellon

Data-Driven (Self-Optimizing) Architectures

System Architecture Design Today

- Human-driven
  - Humans design the policies (how to do things)
- Many (too) simple, short-sighted policies all over the system
- No automatic data-driven policy learning
- (Almost) no learning: cannot take lessons from past actions

Can we design fundamentally intelligent architectures?

An Intelligent Architecture

- Data-driven
  - Machine learns the "best" policies (how to do things)
- Sophisticated, workload-driven, changing, far-sighted policies
- Automatic data-driven policy learning
- All controllers are intelligent data-driven agents

We need to rethink design (of all controllers)

Self-Optimizing Memory Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana, **"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"**
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 39-50, Beijing, China, June 2008.
  Selected to the ISCA-50 25-Year Retrospective Issue covering 1996-2020 in 2023 (Retrospective (pdf) Full Issue).

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek[1,2] **Onur Mutlu**[2] **José F. Martínez**[1] **Rich Caruana**[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (***MICRO***), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,
  "Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Talk Video (12 minutes)]
  [Lecture Video (25 minutes)]
  [arXiv version]
  [Source Code (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best paper award at MICRO 2022.

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2209.00188.pdf

Self-Optimizing Hybrid SSD Controllers

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu,
"Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"
*Proceedings of the 49th International Symposium on Computer Architecture (**ISCA**)*, New York, June 2022.
[Slides (pptx) (pdf)]
[arXiv version]
[Sibyl Source Code]
[Talk Video (16 minutes)]

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh[1] Rakesh Nadig[1] Jisung Park[1] Rahul Bera[1] Nastaran Hajinazar[1]
David Novo[3] Juan Gómez-Luna[1] Sander Stuijk[2] Henk Corporaal[2] Onur Mutlu[1]
[1]ETH Zürich [2]Eindhoven University of Technology [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2205.07394.pdf

Pythia: Prefetching using Reinforcement Learning

Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (***MICRO***), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Pythia

A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori,
Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

https://github.com/CMU-SAFARI/Pythia

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Executive Summary

- **Background**: Prefetchers predict addresses of future memory requests by associating memory access patterns with program context (called **feature**)
- **Problem**: Three key shortcomings of prior prefetchers:
  - Predict mainly using a **single program feature**
  - Lack **inherent system awareness** (e.g., memory bandwidth usage)
  - Lack **in-silicon customizability**
- **Goal**: Design a prefetching framework that:
  - Learns from **multiple features** and **inherent system-level feedback**
  - Can be **customized in silicon** to use different features and/or prefetching objectives
- **Contribution**: Pythia, which formulates prefetching as reinforcement learning problem
  - Takes **adaptive** prefetch decisions using multiple features and system-level feedback
  - Can be **customized in silicon** for target workloads via simple configuration registers
  - Proposes **a realistic and practical** implementation of RL algorithm in hardware
- **Key Results**:
  - Evaluated using a wide range of workloads from SPEC CPU, PARSEC, Ligra, Cloudsuite
  - Outperforms best prefetcher (in 1-core config.) by **3.4%, 7.7% and 17%** in 1/4/bw-constrained cores
  - Up to **7.8% more performance** over basic Pythia across Ligra workloads via simple customization

https://github.com/CMU-SAFARI/Pythia

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Prefetching Basics

- Predicts addresses of **long-latency memory requests** and fetches data before the program demands it

- Associates access patterns from past memory requests with program context information

Program Feature → Access Pattern

- **Example program features**
  - Program counter (PC)
  - Page number
  - Page offset
  - Cacheline delta
  - …
  - Or a combination of these attributes

Key Shortcomings in Prior Prefetchers

- We observe **three key shortcomings** that significantly limit performance benefits of prior prefetchers

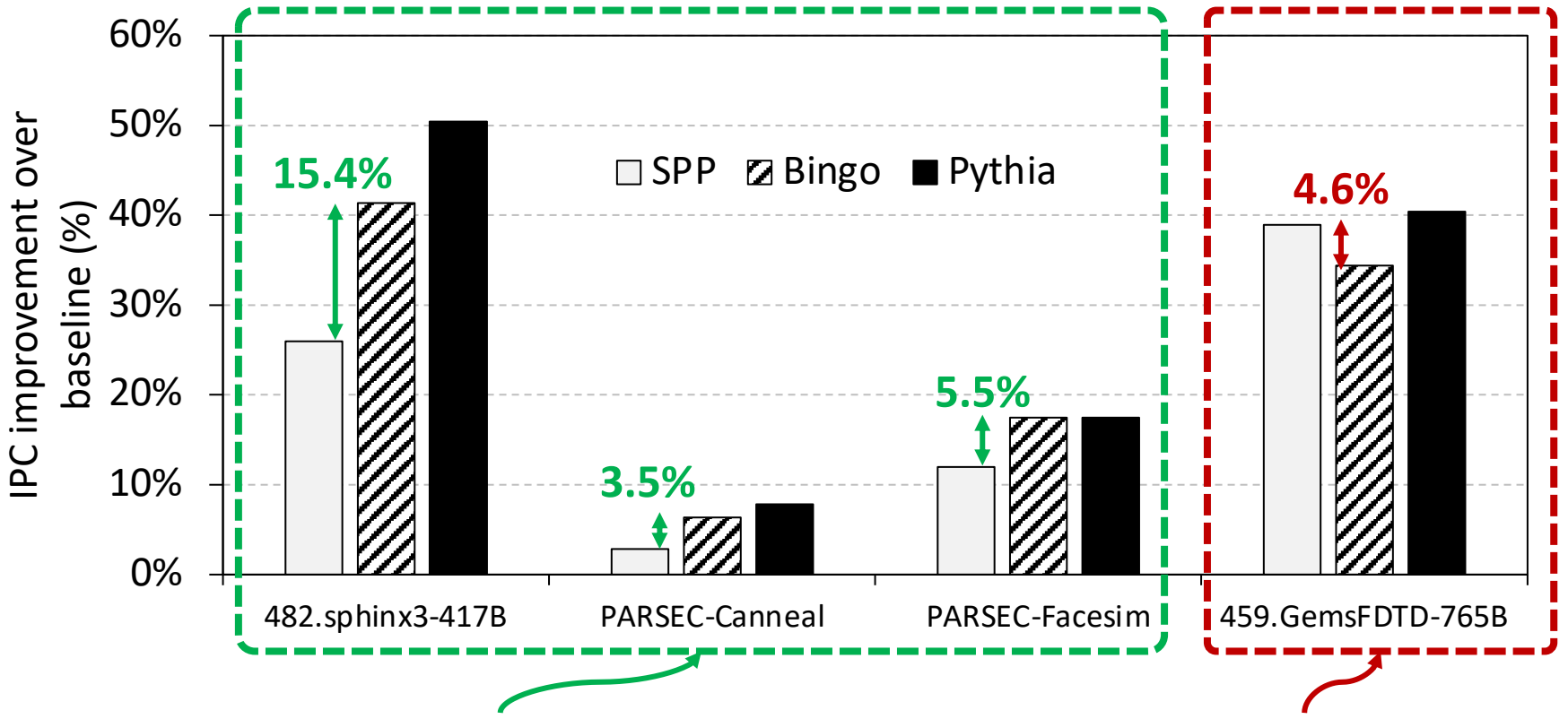
1 Predict mainly using a single program feature

2 Lack inherent system awareness

3 Lack in-silicon customizability

(1) Single-Feature Prefetch Prediction

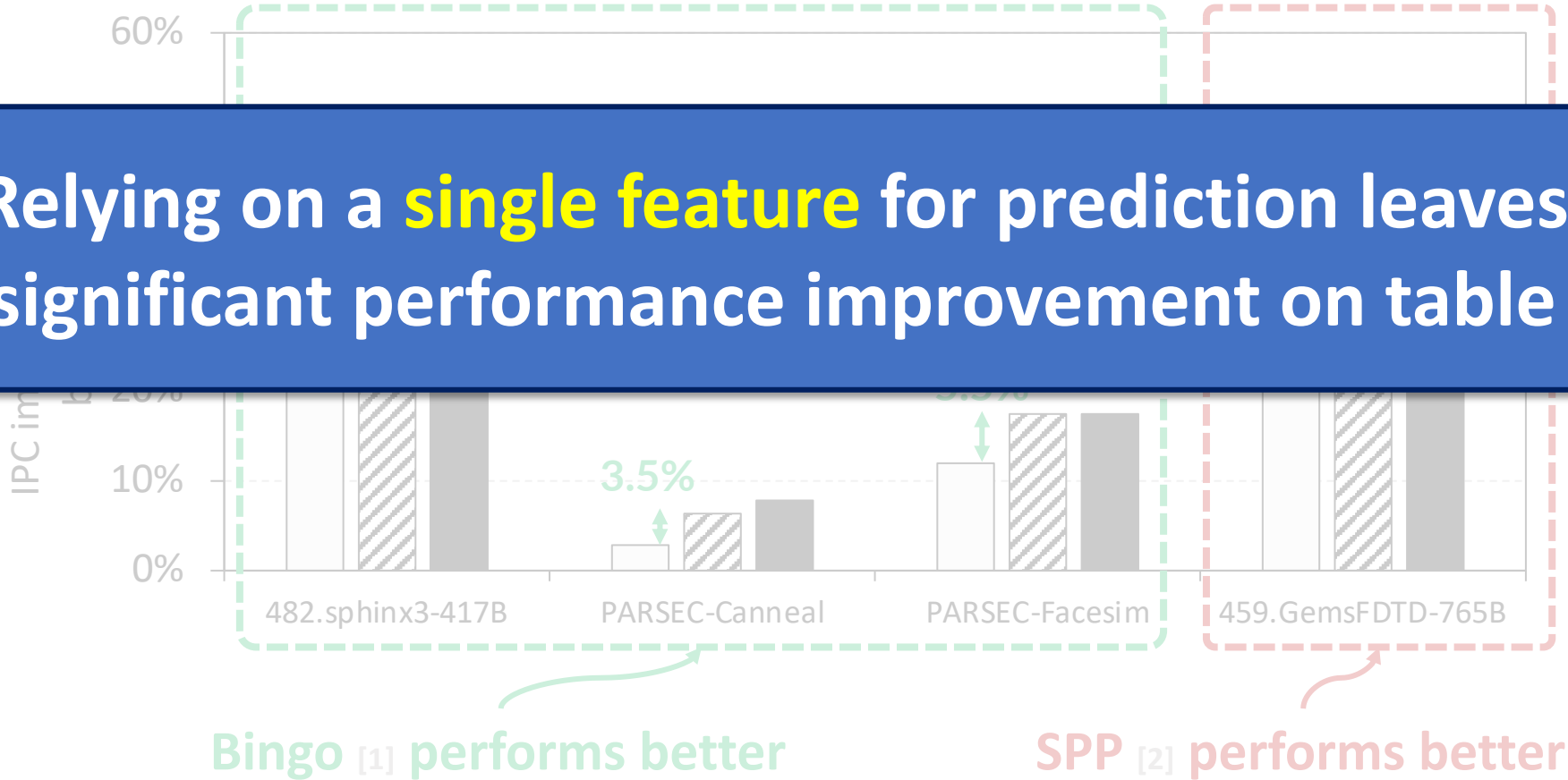
- Provides **good** performance **gains** mainly on workloads where the **feature-to-pattern correlation exists**



[1] Bakshalipour et al., HPCA'19 [2] Kim et al., MICRO'16

(1) Single-Feature Prefetch Prediction

- Provides **good** performance **gains** mainly on workloads where the **feature-to-pattern correlation exists**

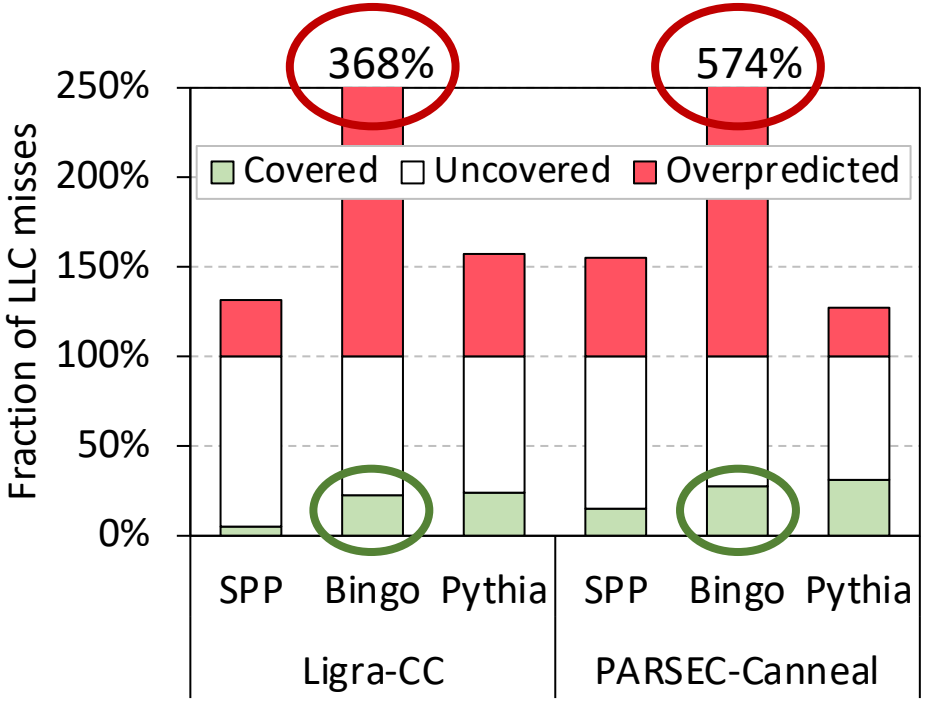


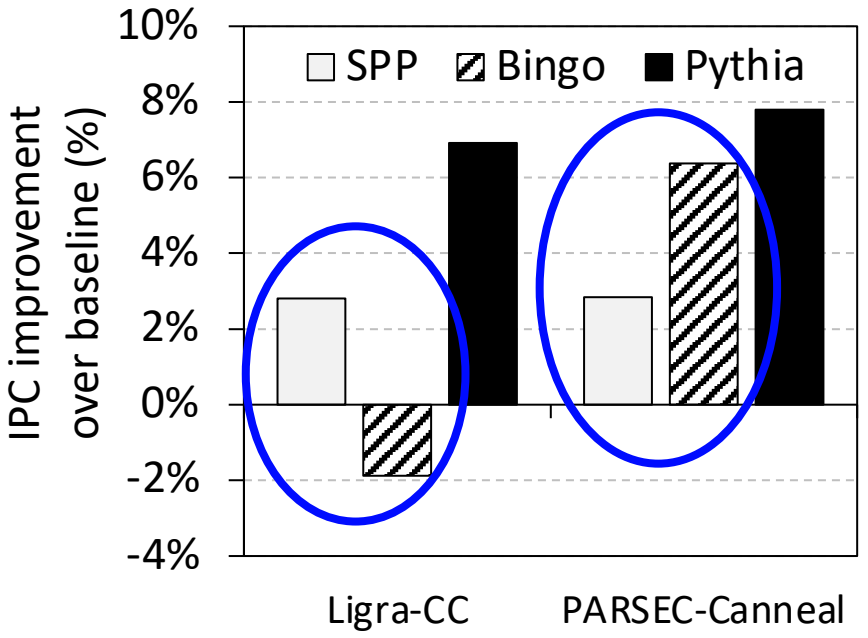
Relying on a single feature for prediction leaves significant performance improvement on table

[1] Bakshalipour et al., HPCA'19 [2] Kim et al., MICRO'16

(2) Lack of Inherent System Awareness

- Little understanding of **undesirable effects** (e.g., memory bandwidth usage, cache pollution, …)
  - Performance loss in **resource-constrained** configurations





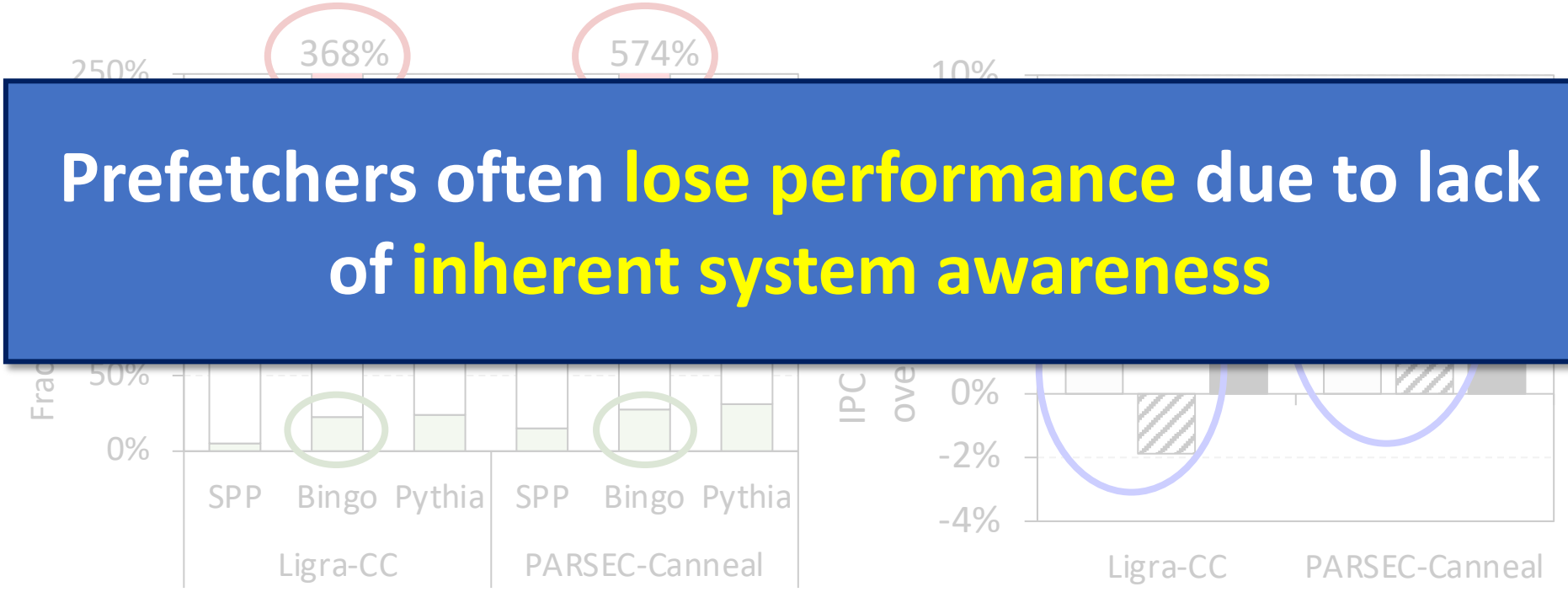
Similar coverage

Lower overpredictions

Yet, lower performance

(2) Lack of Inherent System Awareness

- Little understanding of **undesirable effects** (e.g., memory bandwidth usage, cache pollution, …)
  - Performance loss in **resource-constrained** configurations

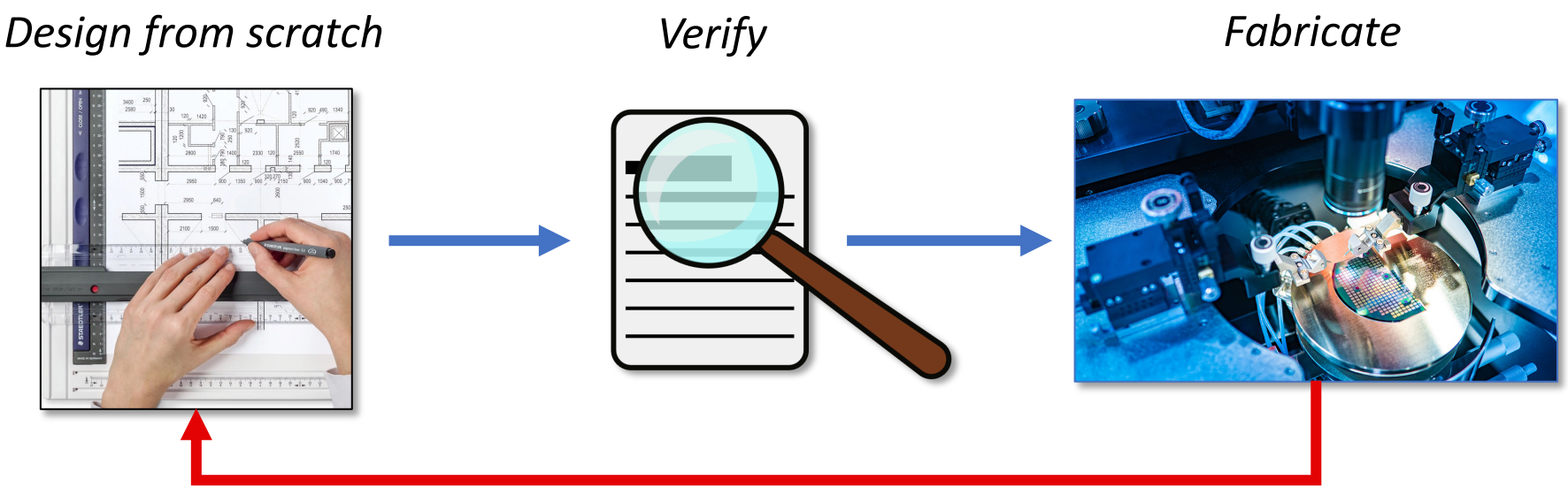


Prefetchers often lose performance due to lack of inherent system awareness

Similar coverage | **Lower overpredictions** | **Yet, lower performance**

(3) Lack of In-silicon Customizability

- Feature **statically** selected at design time
  - **Rigid hardware** designed specifically to exploit that feature
- **No way to change** program feature and/or change prefetcher's objective **in silicon**
  - **Cannot adapt** to a wide range of workload demands



Our Goal

A **prefetching framework** that can:

1. Learn to prefetch using **multiple features** and **inherent system-level feedback** information

2. Be **easily customized in silicon** to use different features and/or change prefetcher's objectives

Our Proposal

Pythia

Formulates prefetching as a
reinforcement learning problem

Pythia is named after the oracle of Delphi, who is known for her accurate prophecies
https://en.wikipedia.org/wiki/Pythia

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

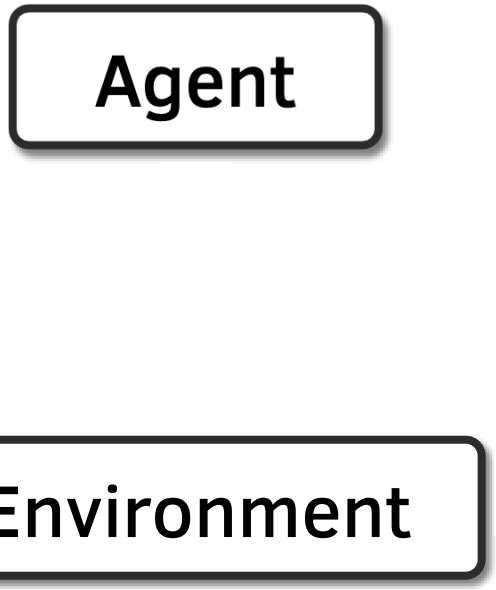
Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Basics of Reinforcement Learning (RL)

- Algorithmic approach to learn to take an **action** in a given **situation** to maximize a numerical **reward**



- Agent stores **Q-values** for *every* state-action pair
  - **Expected return** for taking an action in a state
  - Given a state, selects action that provides **highest** Q-value

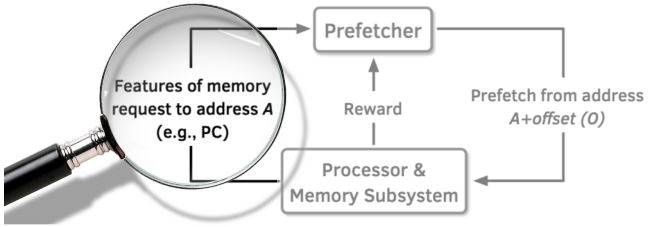
Formulating Prefetching as RL

What is State?

- ***k*-dimensional** vector of features

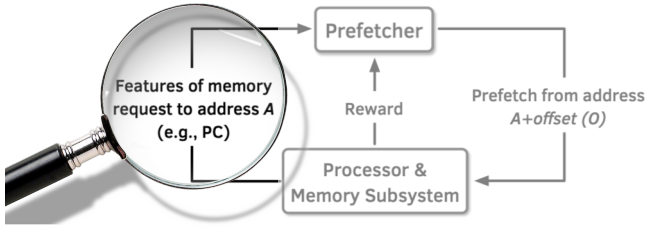
$$S \equiv \{\phi_S^1, \phi_S^2, \ldots, \phi_S^k\}$$

- Feature = control-flow + data-flow



- **Control-flow examples**
  - PC
  - Branch PC
  - Last-3 PCs, …
- **Data-flow examples**
  - Cacheline address
  - Physical page number
  - Delta between two cacheline addresses
  - Last 4 deltas, …

What is State?



Example of a state information

S = {PC+Delta, Sequence of last-4 deltas}

Feature-1 (ϕ_1)

Feature-2 (ϕ_2)

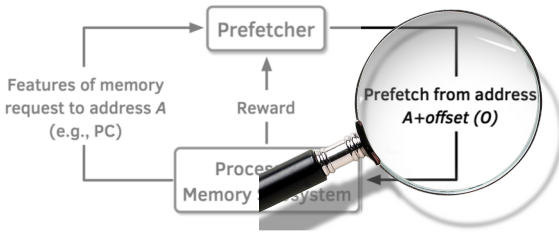
PC
(Control-flow info.)

Cacheline Delta
(Data-flow info.)

Seq. of last-4 deltas
(Data-flow info.)

What is Action?

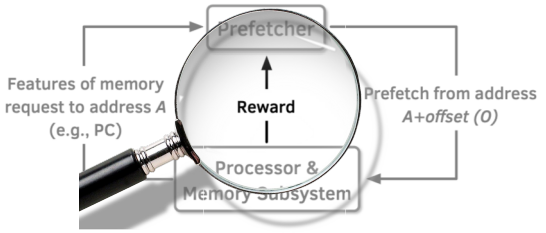
Given a demand access to address A
the action is to **select prefetch offset "O"**



- **Action-space**: 127 actions in the range [-63, +63]
  - For a machine with 4KB page and 64B cacheline
- Upper and lower limits ensure prefetches do not cross **physical page boundary**
- A **zero offset** means **no prefetch** is generated
- We further **prune** action-space by design-space exploration

What is Reward?

- Defines the **objective** of Pythia

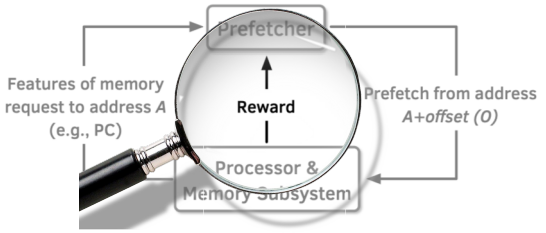


- Encapsulates two metrics:
  - **Prefetch usefulness** (e.g., accurate, late, out-of-page, …)
  - **System-level feedback** (e.g., mem. b/w usage, cache pollution, energy, …)

- We demonstrate Pythia with **memory bandwidth usage** as the system-level feedback in the paper

What is Reward?

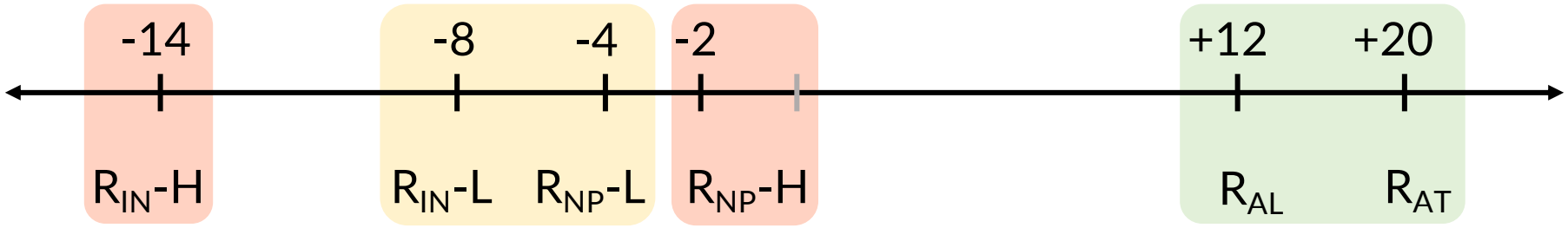
- **Seven** distinct reward levels
  - *Accurate and timely* (R_{AT})
  - *Accurate but late* (R_{AL})
  - *Loss of coverage* (R_{CL})
  - *Inaccurate*
    - With low memory b/w usage (R_{IN}-L)
    - With high memory b/w usage (R_{IN}-H)
  - *No-prefetch*
    - With low memory b/w usage (R_{NP}-L)
    - With high memory b/w usage(R_{NP}-H)



- Values are set at design time via **automatic design-space exploration**
  - Can be **customized** further in silicon for higher performance

Steering Pythia's Objective via Reward Values

- Example reward configuration for
  - Generating **accurate prefetches**
  - Making **bandwidth-aware** prefetch decisions

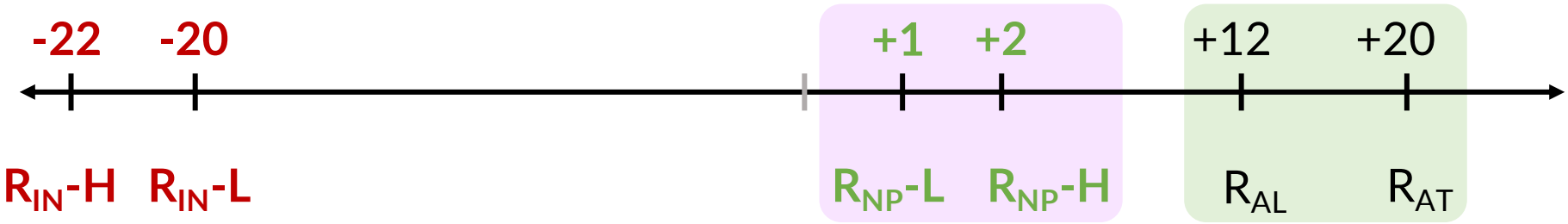


AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

1. **Highly prefers to generate accurate prefetches**
2. **Prefers not to prefetch if memory bandwidth usage is low**
3. **Strongly prefers not to prefetch if memory bandwidth usage is high**

Steering Pythia's Objective via Reward Values

- Customizing reward values to make Pythia **conservative** towards prefetching



AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

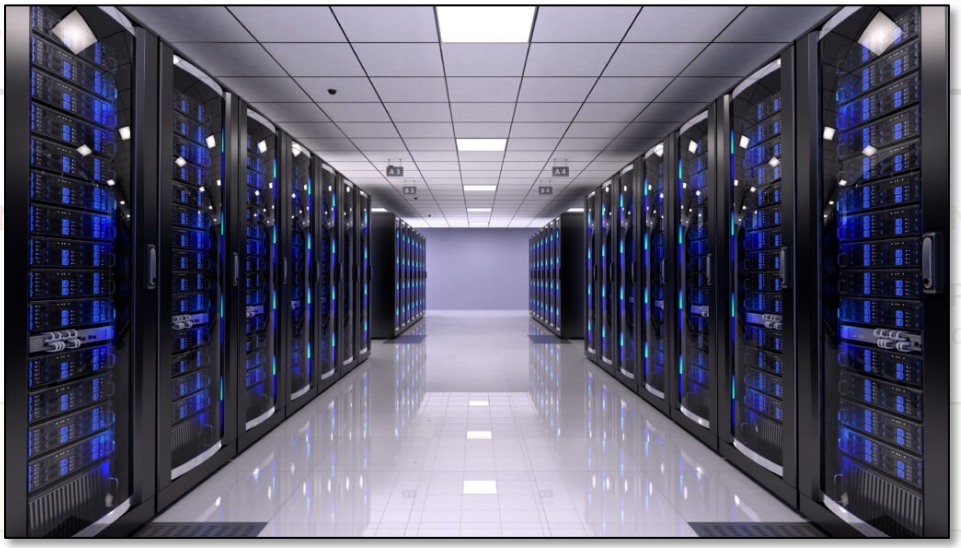
1 **Highly prefers to generate accurate prefetches**

2 **Otherwise prefers not to prefetch**

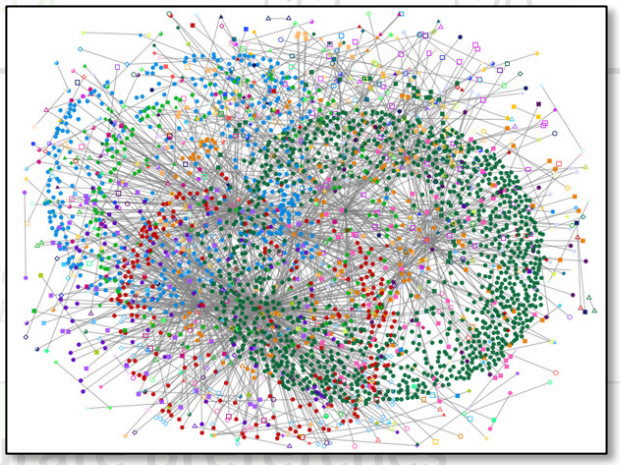
Steering Pythia's Objective via Reward Values

- Customizing reward values to make Pythia conservative towards p

Strict Pythia configuration



Server-class processors



Bandwidth-sensitive workloads

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

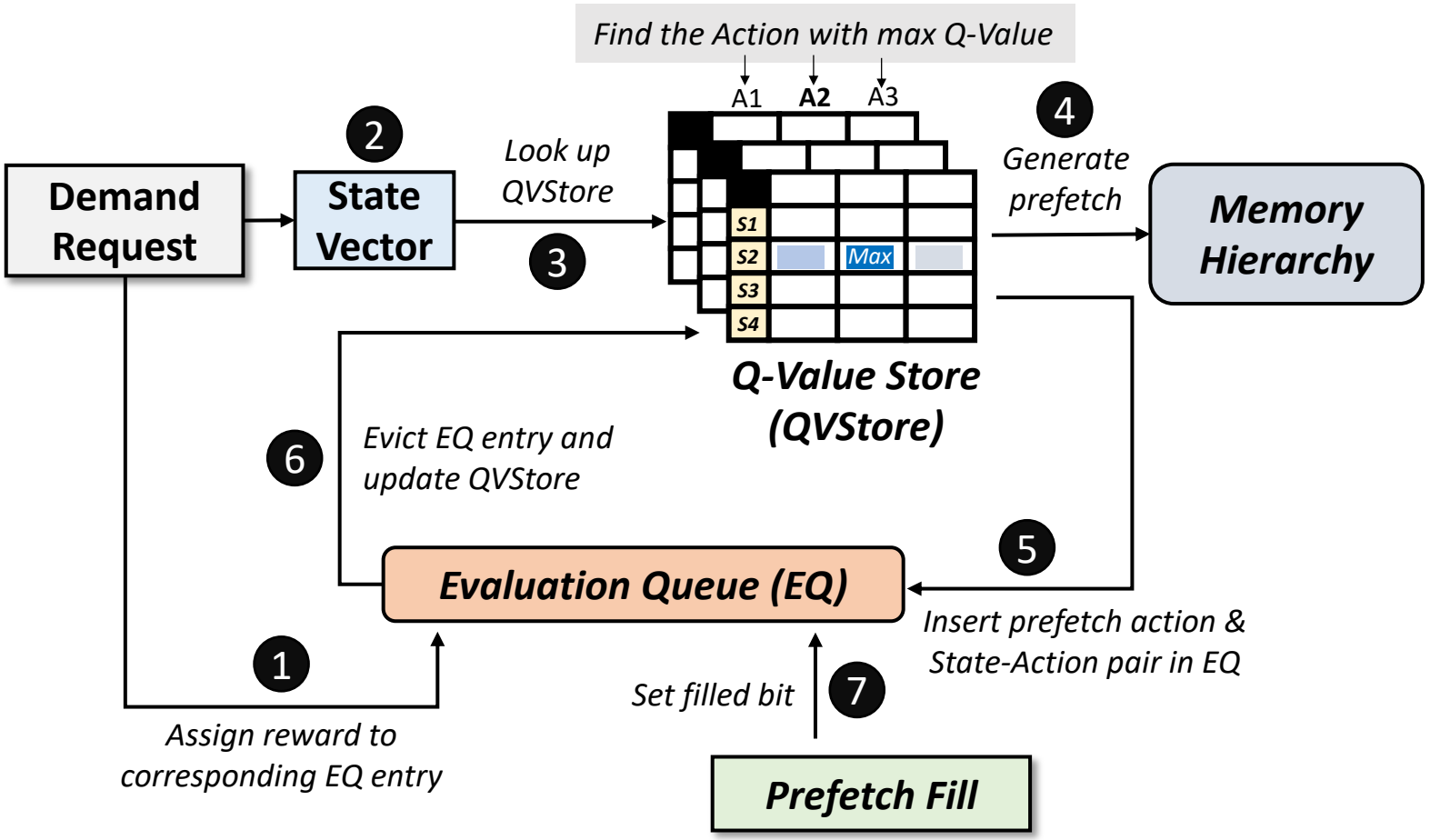
Pythia: Overview

Evaluation of Pythia and Key Results

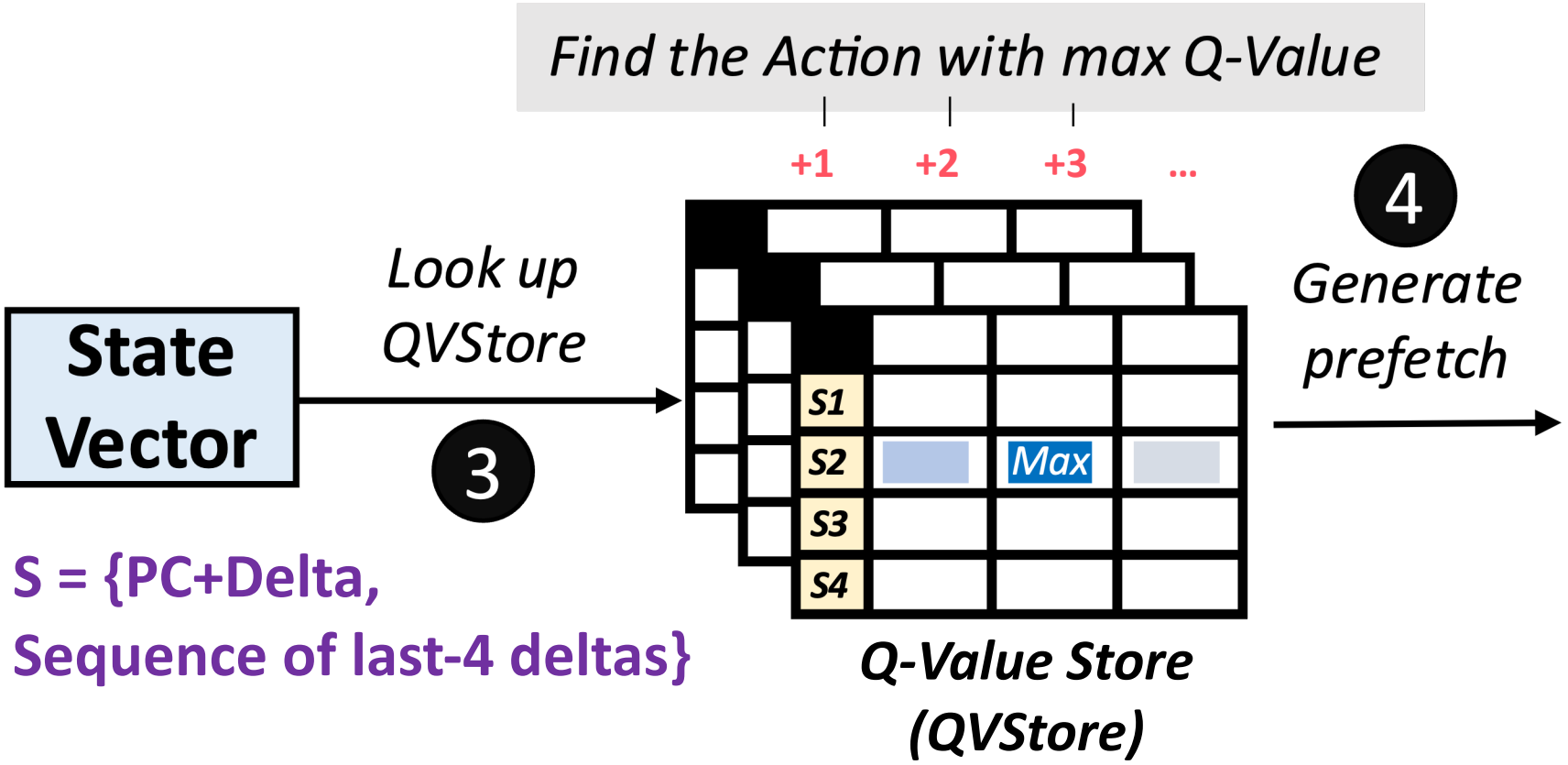
Conclusion

Pythia Overview

- **Q-Value Store**: Records Q-values for *all* state-action pairs
- **Evaluation Queue**: A FIFO queue of recently-taken actions



Architecting QVStore



Architecting the QVStore

Fast prefetch prediction

Fast retrieval of Q-values from QVStore

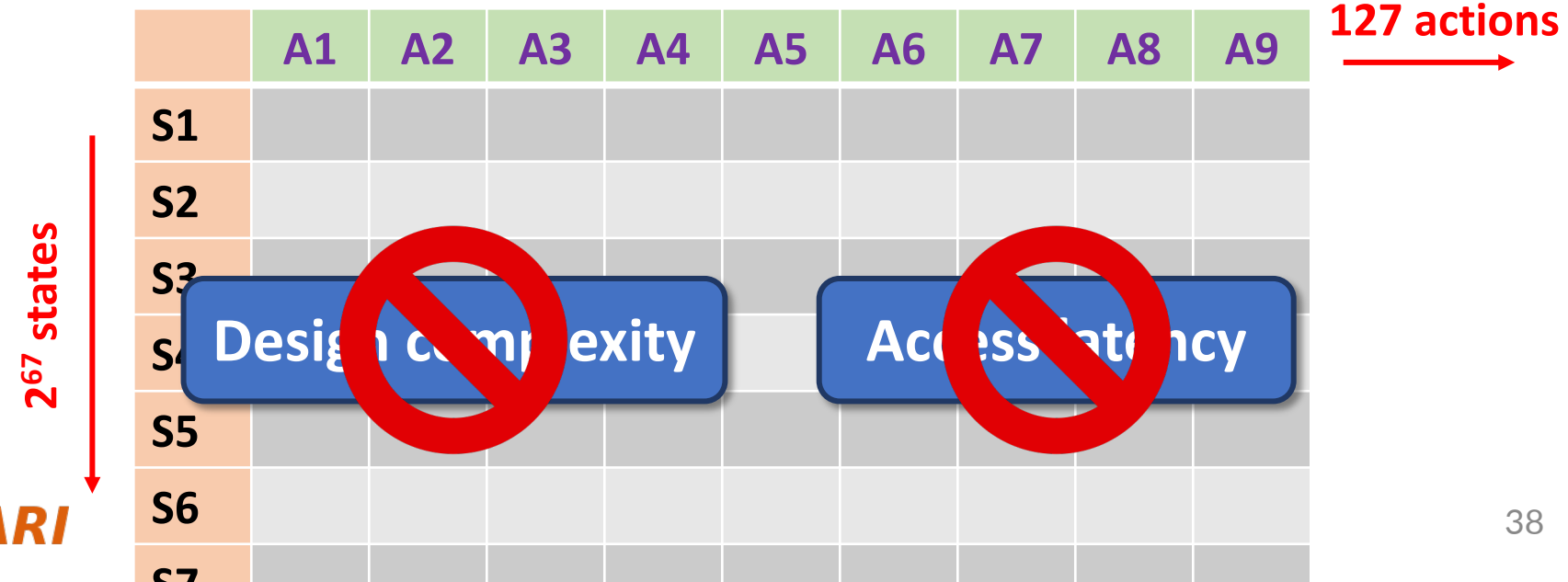
Efficient storage organization of Q-values in QVStore

Organization of QVStore

- A **monolithic** two-dimensional table?
  - Indexed by state and action values
- State-space increases **exponentially** with #bits

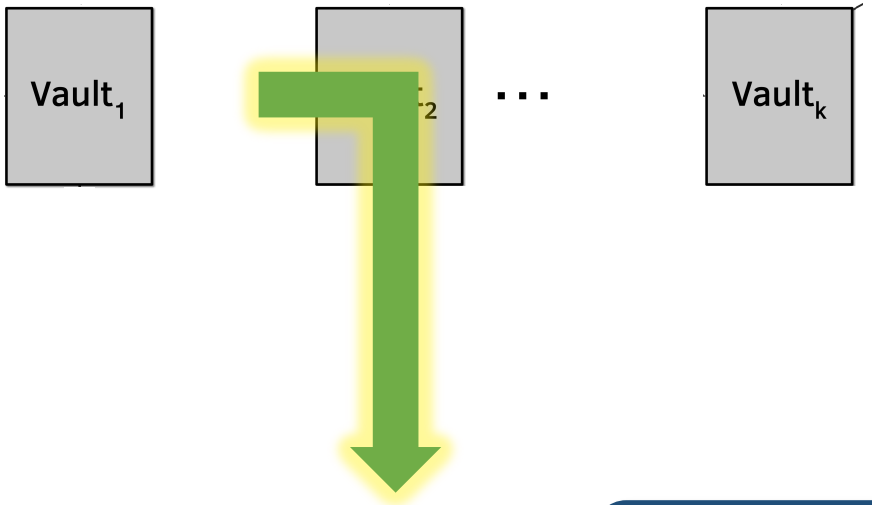
S = {PC+Delta, Sequence of last-4 deltas}

32b + 7b + 4x7b = 67 bits



Organization of QVStore

- We partition QVStore into ***k* vaults** [*k* = number of features in state]
  - Each vault corresponds to one feature and stores the Q-values of **feature-action pairs**



To retrieve Q(S,A) for each action

- Query **each vault in parallel** with feature and action
- **Retrieve feature-action Q-value** from each vault
- Compute **MAX** of all feature-action Q-values

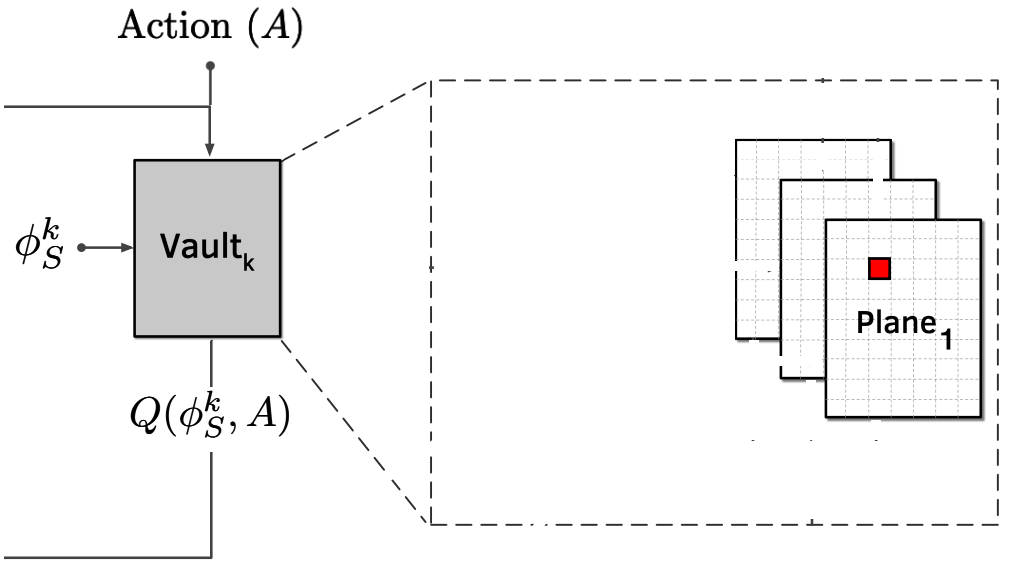
MAX ensures the Q(S,A) is driven by the constituent feature that has highest Q(ϕ,A)

Organization of QVStore

- We further partition each vault into multiple **planes**
  - Each plane stores a **partial** Q-value of a feature-action pair

To retrieve Q(ϕ,A) for each action

- Query **each plane in parallel** with hashed feature and action
- **Retrieve partial feature-action Q-value** from each plane
- Compute **SUM** of all partial feature-action Q-values



Organization of QVStore

- We further partition each vault into multiple **planes**
  - Each plane stores a **partial** Q-value of a feature-action pair

1. Enables sharing of partial Q-values between similar feature values, shortens prefetcher training time

- Query each plane in parallel with hashed feature and action

2. Reduces chances of sharing partial Q-values across widely different feature values

- Compute SUM of all partial feature-action Q-values

More in the Paper

- **Pipelined search** operation for QVStore
- Reward assignment and **QVStore update**
- **Automatic design-space exploration**
  - Feature types
  - Actions
  - Reward and Hyperparameter values

More in the Paper

- **Pipelined search** operation for QVStore
- Reward assignment and QVStore update
  - Reward and Hyperparameter values

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Simulation Methodology

- **Champsim** [3] trace-driven simulator
- **150** single-core memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - PARSEC 2.1
  - Ligra
  - Cloudsuite
- Homogeneous and heterogeneous multi-core mixes
- **Five** state-of-the-art prefetchers
  - SPP [Kim+, MICRO'16]
  - Bingo [Bakhshalipour+, HPCA'19]
  - MLOP [Shakerinava+, 3rd Prefetching Championship, 2019]
  - SPP+DSPatch [Bera+, MICRO'19]
  - SPP+PPF [Bhatia+, ISCA'20]

[3] https://github.com/ChampSim/ChampSim

Basic Pythia Configuration

- Derived from **automatic design-space exploration**

- **State:** 2 features
  - PC+Delta
  - Sequence of last-4 deltas

- **Actions:** 16 prefetch offsets
  - Ranging between -6 to +32. Including 0.

- **Rewards:**
  - R_{AT} = +20; R_{AL} = +12; R_{NP}-H=-2; R_{NP}-L=-4;
  - R_{IN}-H=-14; R_{IN}-L=-8; R_{CL}=-12

List of Evaluated Features

Table 3: List of program control-flow and data-flow components used to derive the list of features for exploration

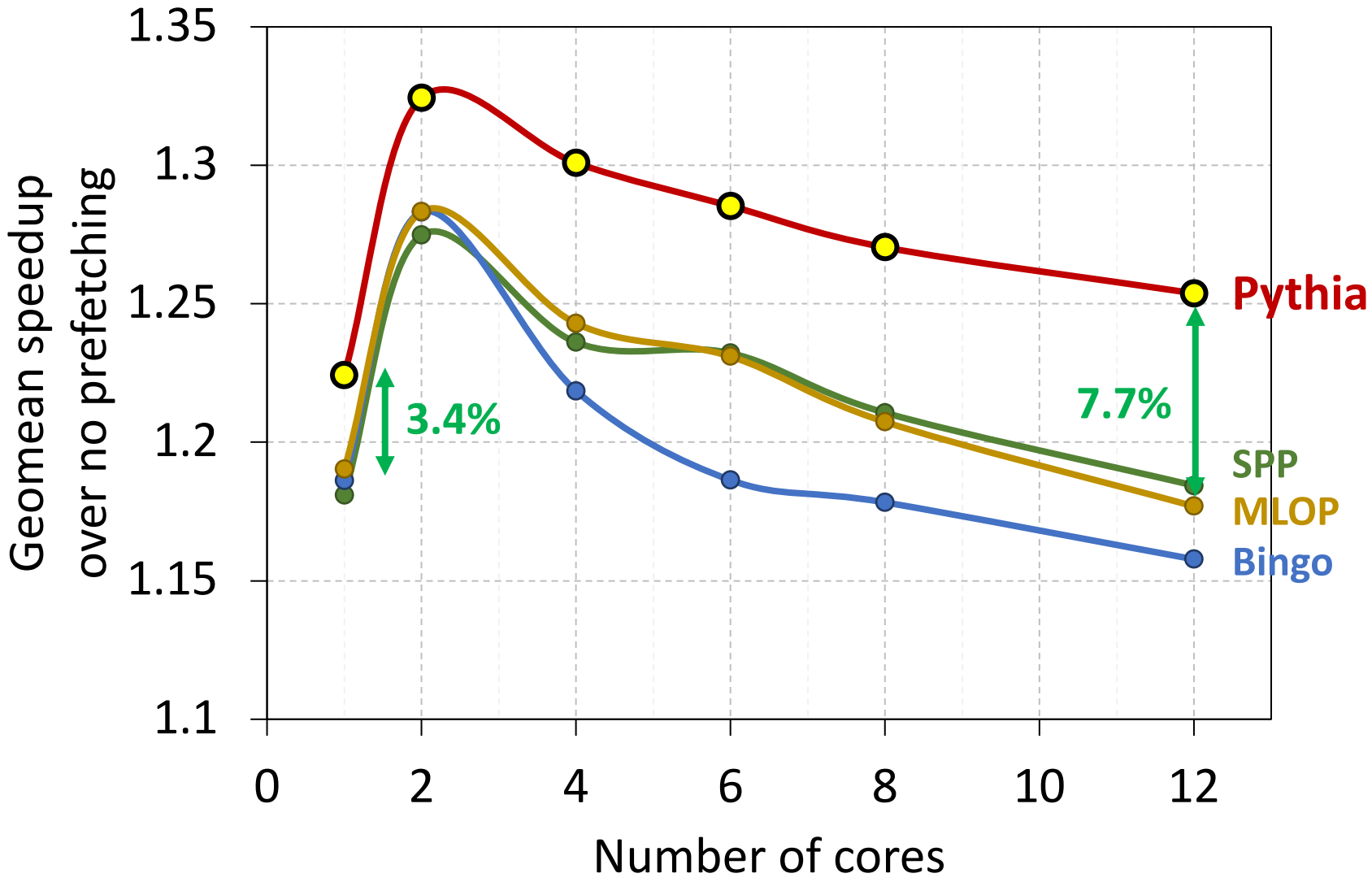
Control-flow Component	Data-flow Component
(1) PC of load request (2) PC-path (XOR-ed last-3 PCs) (3) PC XOR-ed branch-PC (4) None	(1) Load cacheline address (2) Page number (3) Page offset (4) Load address delta (5) Sequence of last-4 offsets (6) Sequence of last-4 deltas (7) Offset XOR-ed with delta (8) None

Basic Pythia Configuration

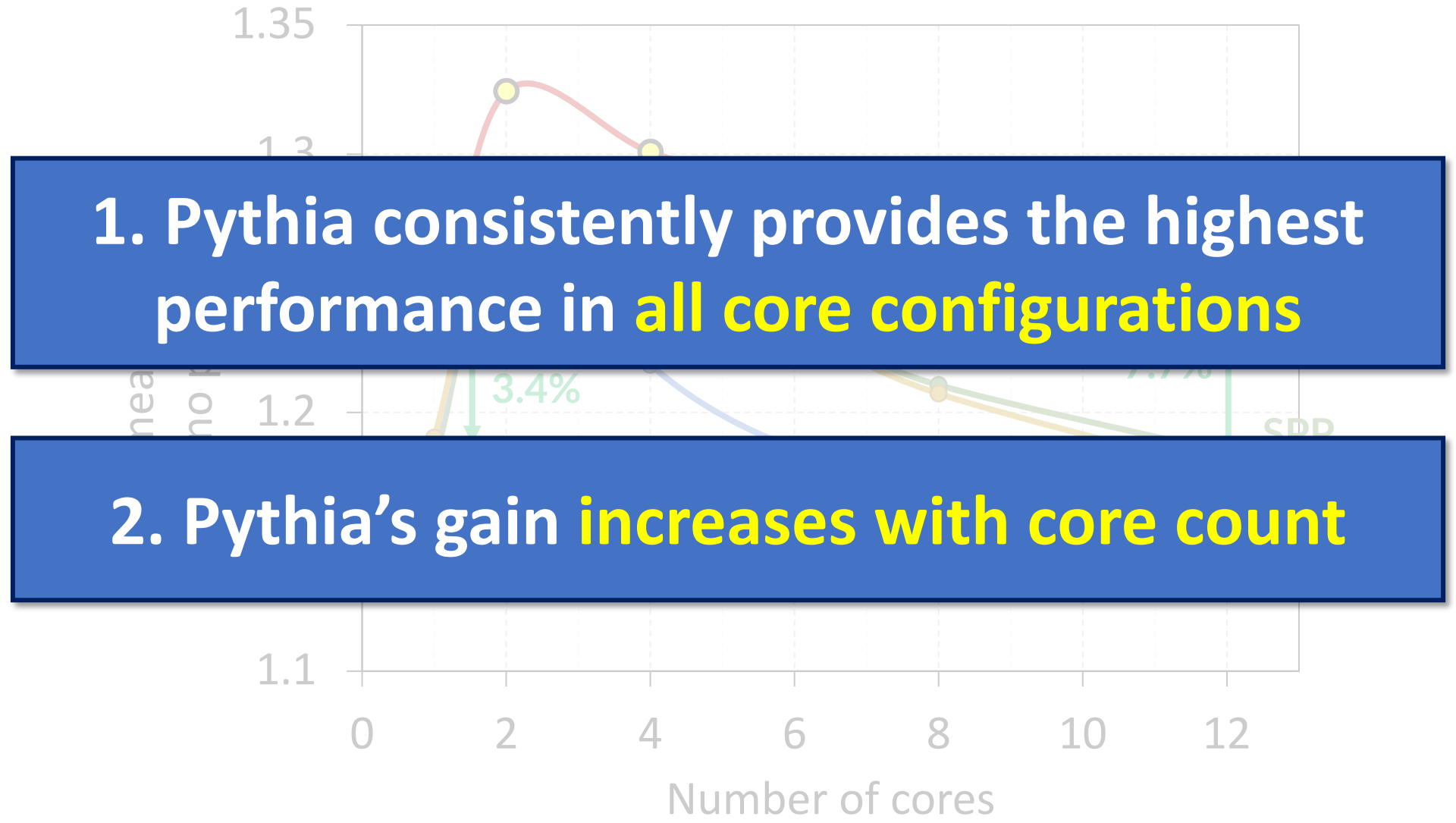
Table 2: Basic Pythia configuration derived from our automated design-space exploration

Features	`PC+Delta, Sequence of last-4 deltas`
Prefetch Action List	{-6,-3,-1,0,1,3,4,5,10,11,12,16,22,23,30,32}
Reward Level Values	$\mathcal{R}_{AT}$=20, $\mathcal{R}_{AL}$=12, $\mathcal{R}_{CL}$=−12, $\mathcal{R}^{H}_{IN}$=−14, $\mathcal{R}^{L}_{IN}$=−8, $\mathcal{R}^{H}_{NP}$=−2, $\mathcal{R}^{L}_{NP}$=−4
Hyperparameters	$\alpha = 0.0065$, $\gamma = 0.556$, $\epsilon = 0.002$

Performance with Varying Core Count



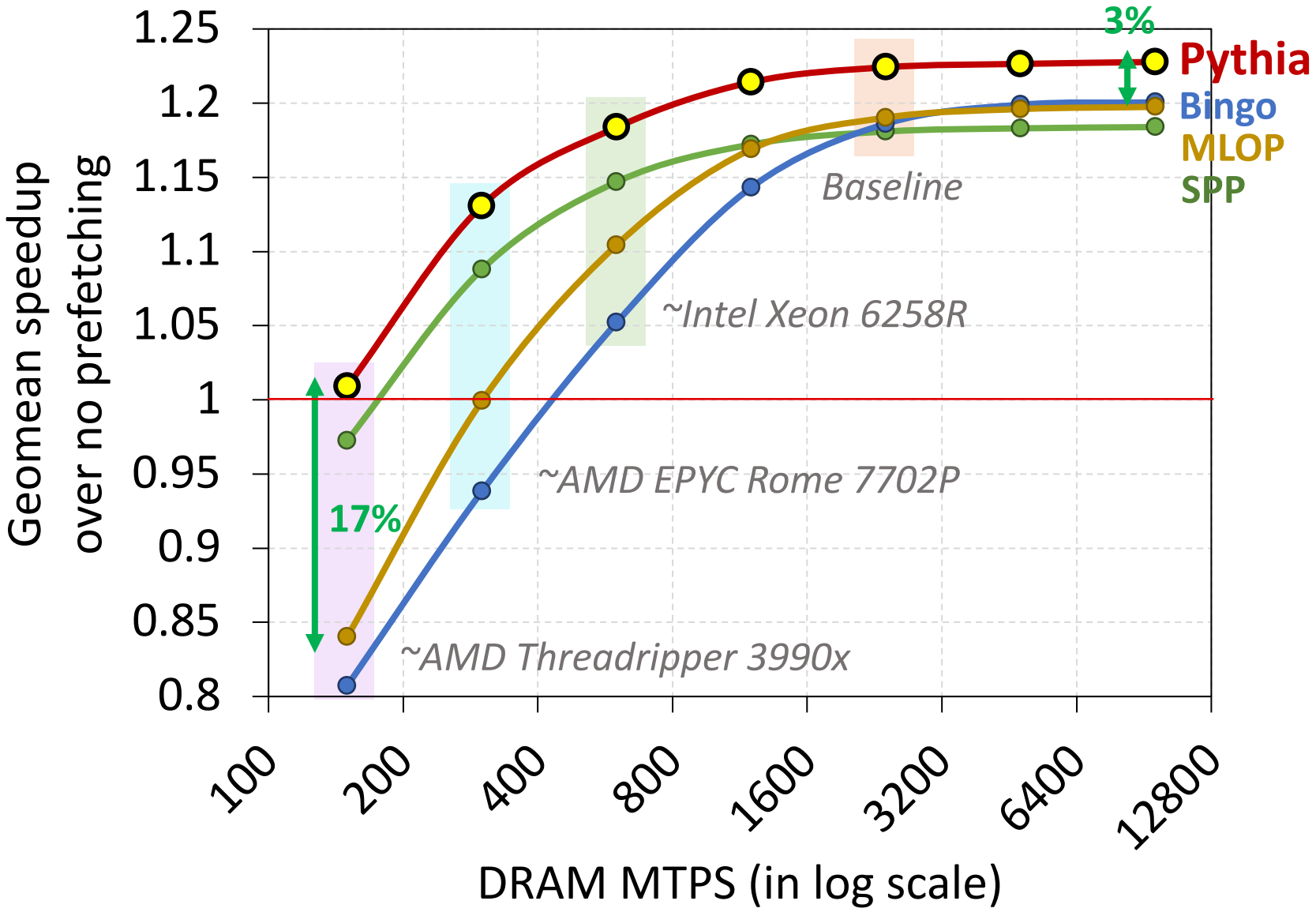
Performance with Varying Core Count



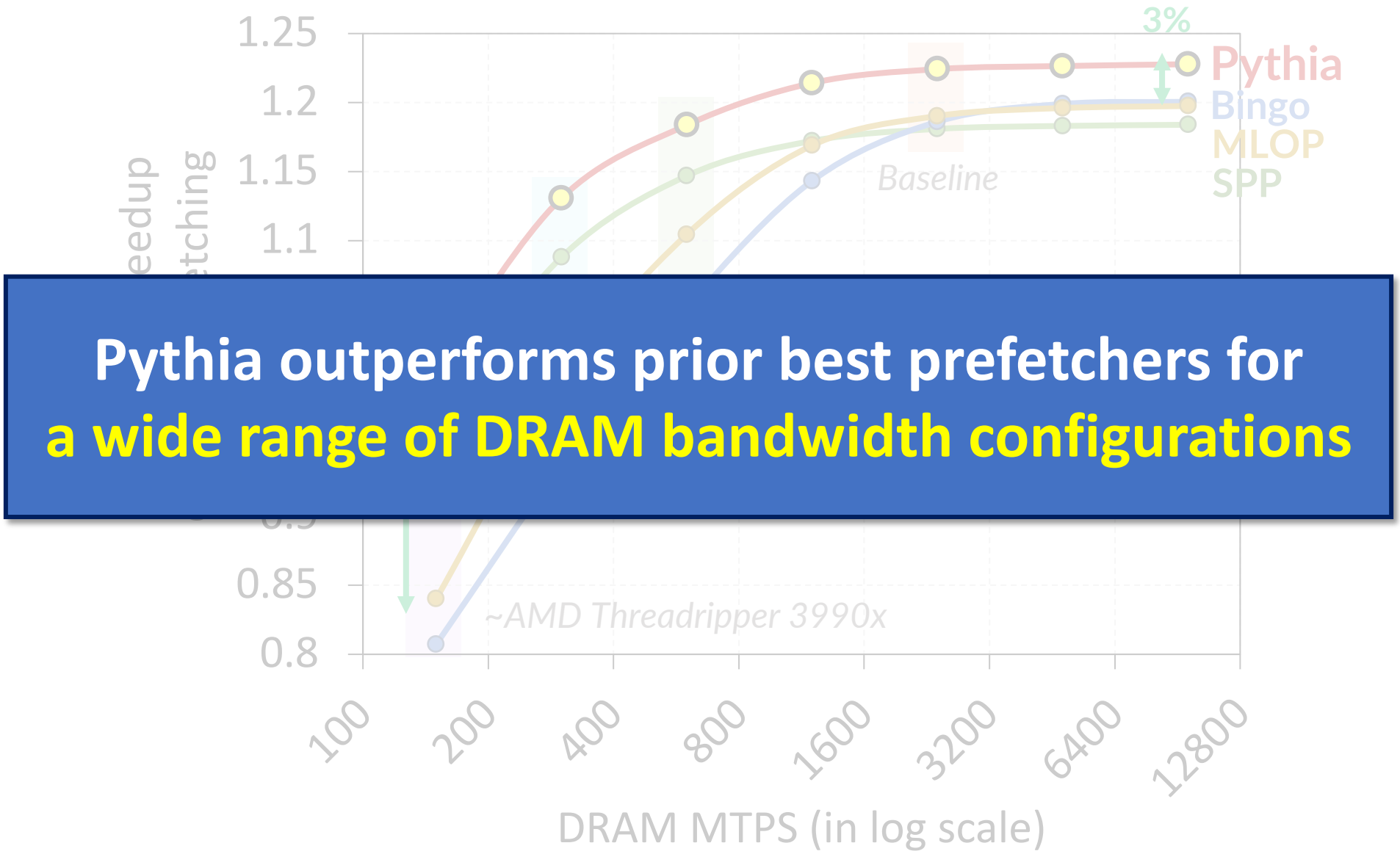
1. Pythia consistently provides the highest performance in all core configurations

2. Pythia's gain increases with core count

Performance with Varying DRAM Bandwidth



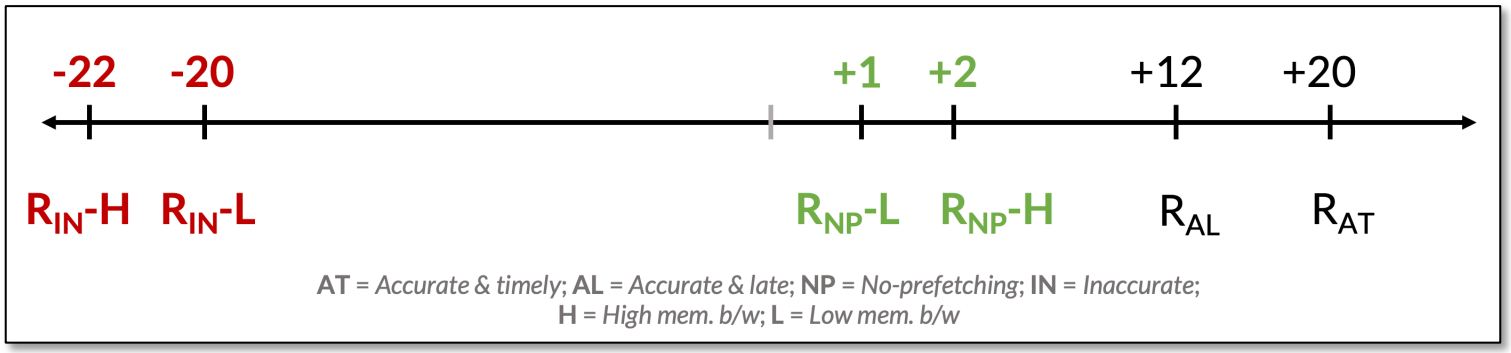
Performance with Varying DRAM Bandwidth



Pythia outperforms prior best prefetchers for a wide range of DRAM bandwidth configurations

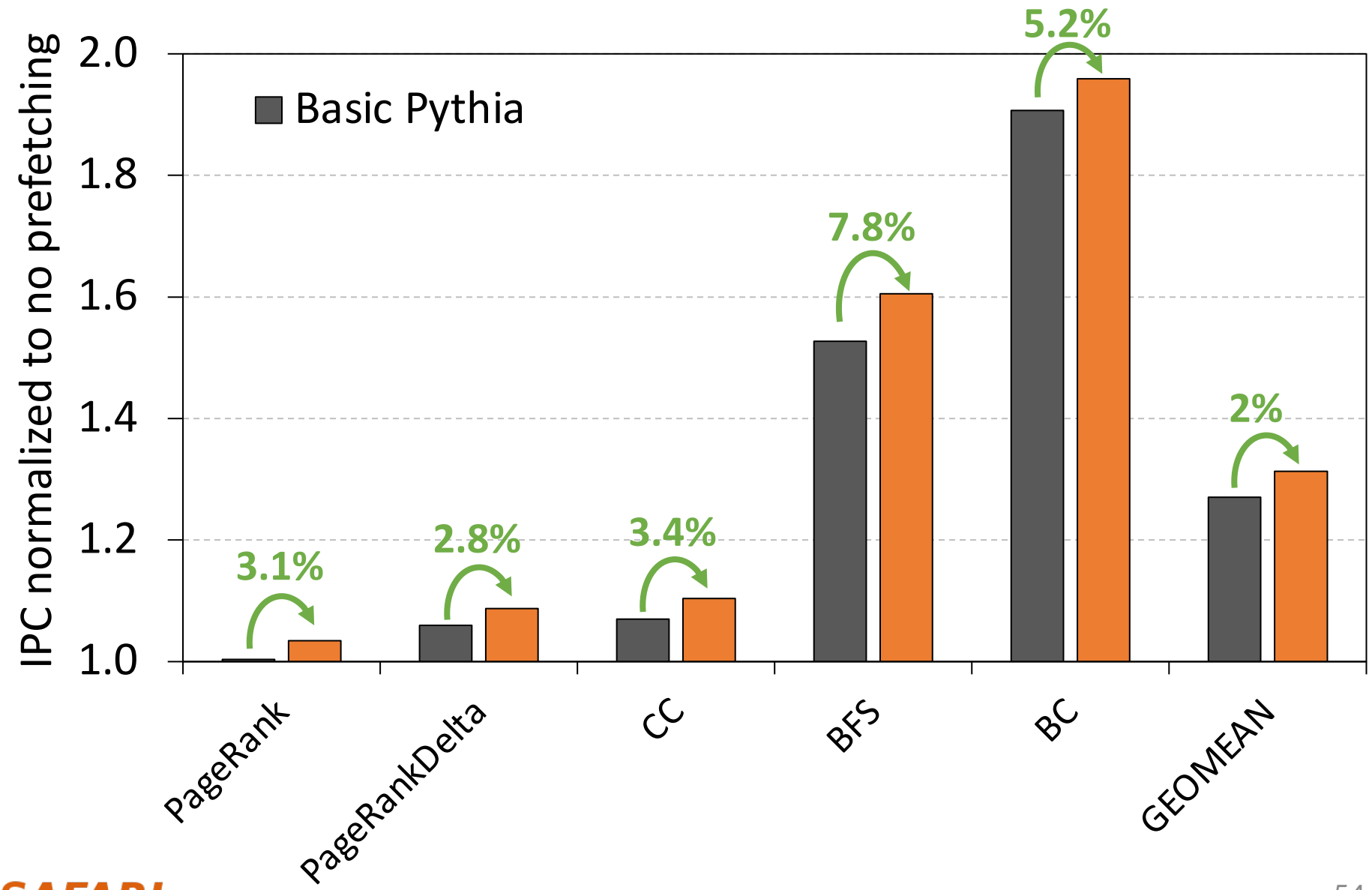
Performance Improvement via Customization

- Reward value customization
- **Strict Pythia configuration**
  - **Increasing** the rewards for **no prefetching**
  - **Decreasing** the rewards for **inaccurate prefetching**

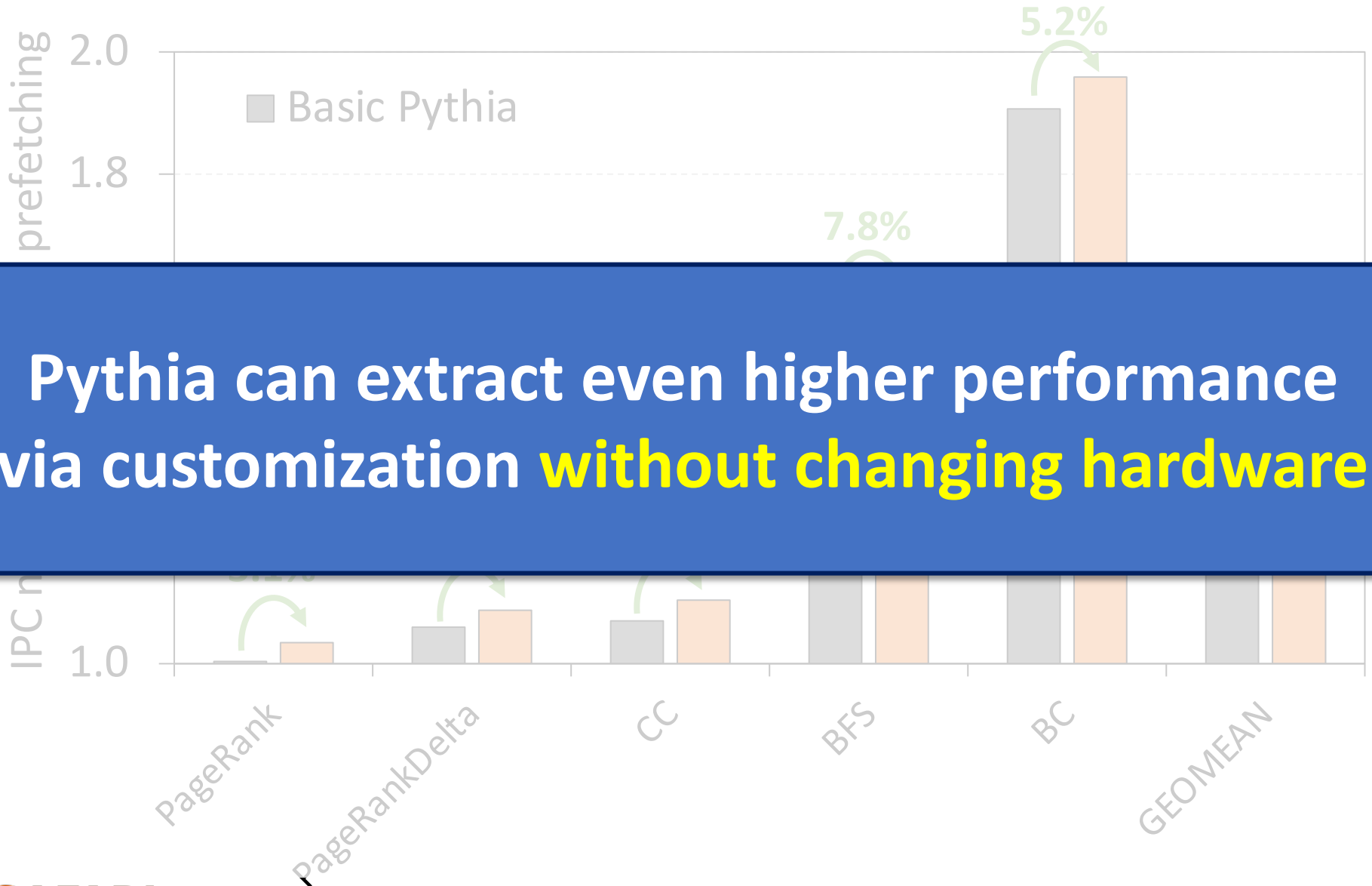


- Strict Pythia is **more conservative** in generating prefetch requests than the basic Pythia
- Evaluate on all **Ligra graph processing workloads**

Performance Improvement via Customization



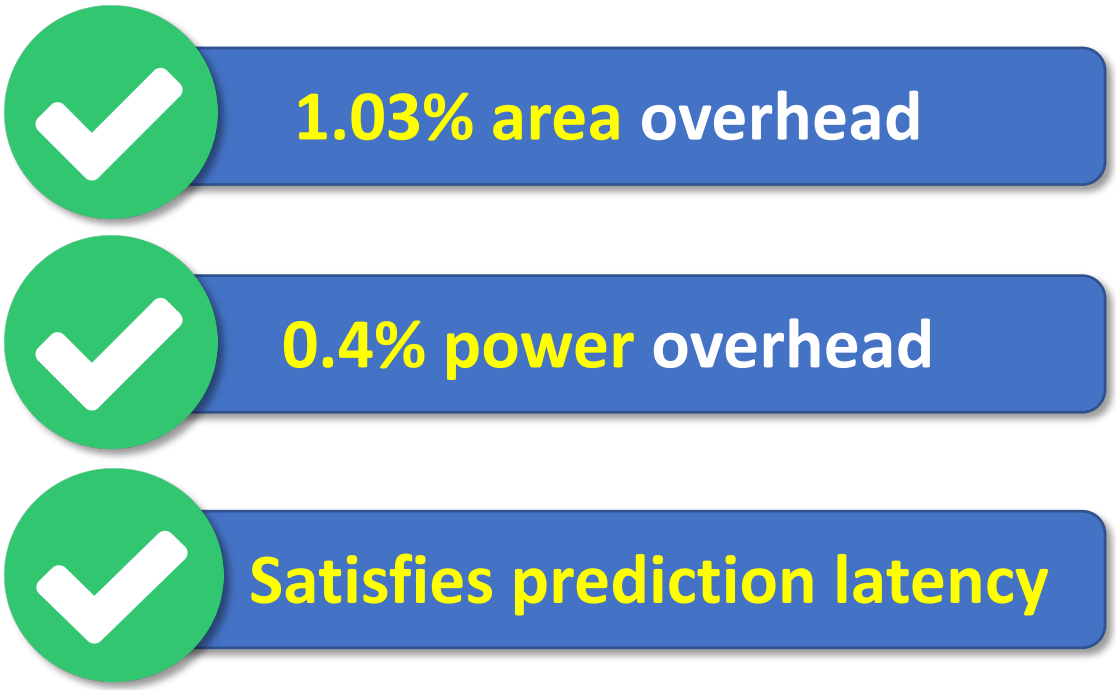
Performance Improvement via Customization



Pythia can extract even higher performance via customization without changing hardware

Pythia's Overhead

- **25.5 KB** of total metadata storage **per core**
  - Only simple tables
- We also model functionally-accurate Pythia with full complexity in **Chisel** [4] HDL



of a desktop-class 4-core Skylake processor (Xeon D2132IT, 60W)

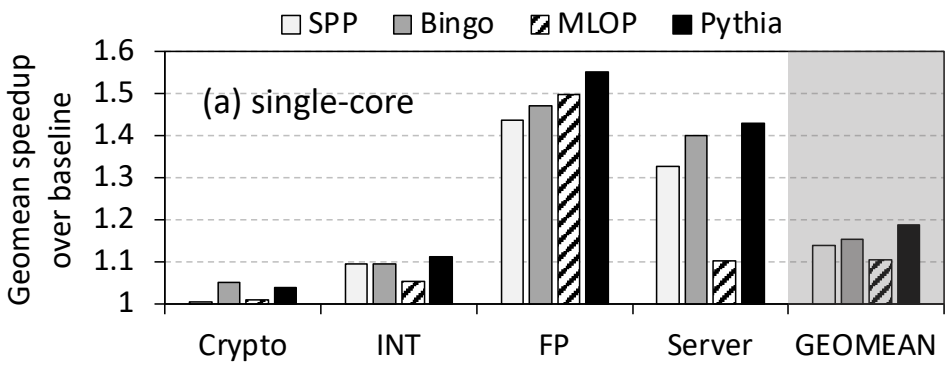
[4] https://www.chisel-lang.org

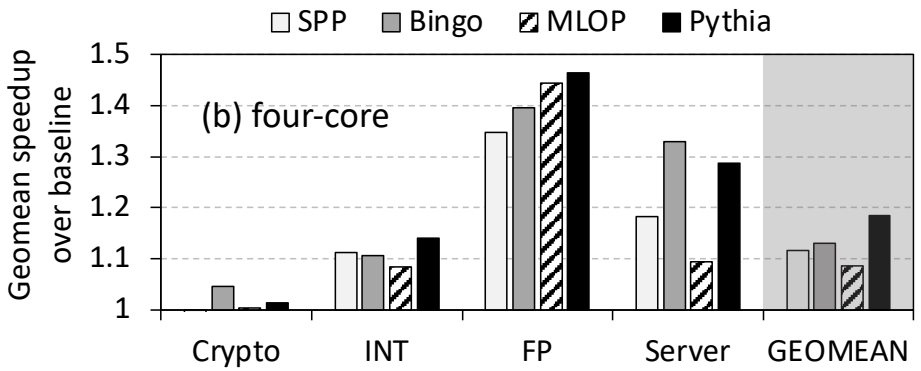
More in the Paper

- Performance comparison with **unseen traces**
  - Pythia provides **equally high** performance benefits
- Comparison against **multi-level prefetchers**
  - Pythia **outperforms** prior best multi-level prefetchers
- Understanding Pythia's learning with **a case study**
  - We reason towards **the correctness** of Pythia's decision
- **Performance sensitivity** towards different features and hyperparameter values
- Detailed single-core and four-core performance

Performance on Previously-Unseen Workloads

- Evaluated with 500 traces from value prediction championship
  - **No prefetcher has been trained on these traces**





Pythia outperforms MLOP and Bingo by 8.3% and 3.5% in single-core

And 9.7% and 5.4% in four-core

More in the Paper

- Performance comparison with unseen traces
  - Pythia provides equally high performance benefits
- Comparison against multi-level prefetchers

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

- Performance sensitivity towards different features and hyperparameter values

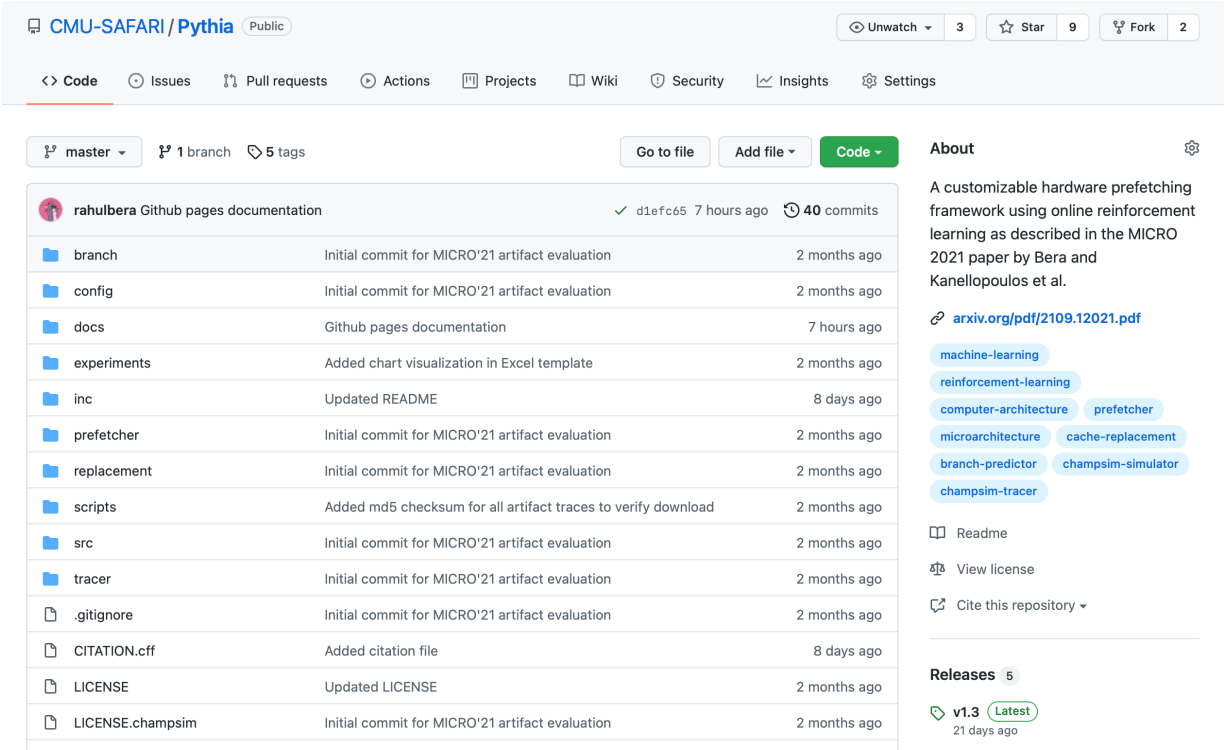
https://arxiv.org/pdf/2109.12021.pdf

- Detailed single-core and four-core performance

Pythia is Open Source

https://github.com/CMU-SAFARI/Pythia

- MICRO'21 **artifact evaluated**
- **Champsim source** code + **Chisel** modeling code
- **All traces** used for evaluation



Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Executive Summary

- **Background**: Prefetchers predict addresses of future memory requests by associating memory access patterns with program context (called **feature**)
- **Problem**: Three key shortcomings of prior prefetchers:
  - Predict mainly using a **single program feature**
  - Lack **inherent system awareness** (e.g., memory bandwidth usage)
  - Lack **in-silicon customizability**
- **Goal**: Design a prefetching framework that:
  - Learns from **multiple features** and **inherent system-level feedback**
  - Can be **customized in silicon** to use different features and/or prefetching objectives
- **Contribution**: Pythia, which formulates prefetching as reinforcement learning problem
  - Takes **adaptive** prefetch decisions using multiple features and system-level feedback
  - Can be **customized in silicon** for target workloads via simple configuration registers
  - Proposes **a realistic and practical** implementation of RL algorithm in hardware
- **Key Results**:
  - Evaluated using a wide range of workloads from SPEC CPU, PARSEC, Ligra, Cloudsuite
  - Outperforms best prefetcher (in 1-core config.) by **3.4%, 7.7% and 17%** in 1/4/bw-constrained cores
  - Up to **7.8% more performance** over basic Pythia across Ligra workloads via simple customization

https://github.com/CMU-SAFARI/Pythia

Pythia

A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori,
Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

https://github.com/CMU-SAFARI/Pythia

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Pythia Discussion

- **FAQs**
  - Why RL?
  - What about large page?
  - What's the prefetch degree?
  - Can customization happen during workload execution?
  - Can runtime mixing create problem?
- **Simulation and Methodology**
  - Basic Pythia configuration
  - System parameters
  - Configuration of prefetchers
  - Evaluated workloads
  - Feature selection
- **Detailed Design**
  - Reward structure
  - Design overview
  - QVStore Organization
- **More Results**
  - Comparison against other adaptive prefetchers
  - Comparison against Context prefetcher
  - Feature combination sensitivity
  - Hyperparameter sensitivity
  - Comparison with multi-level prefetchers
  - Performance in unseen workloads
  - Single-core s-curve
  - Four-core s-curve
  - Detailed performance analysis
  - Benefit of bandwidth awareness
  - Case study
  - Customizing rewards
  - Customizing features

Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (***MICRO***), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Hermes: Perceptron-Based Off-Chip Load Prediction

Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,
  "Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Talk Video (12 minutes)]
  [Lecture Video (25 minutes)]
  [arXiv version]
  [Source Code (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best paper award at MICRO 2022.

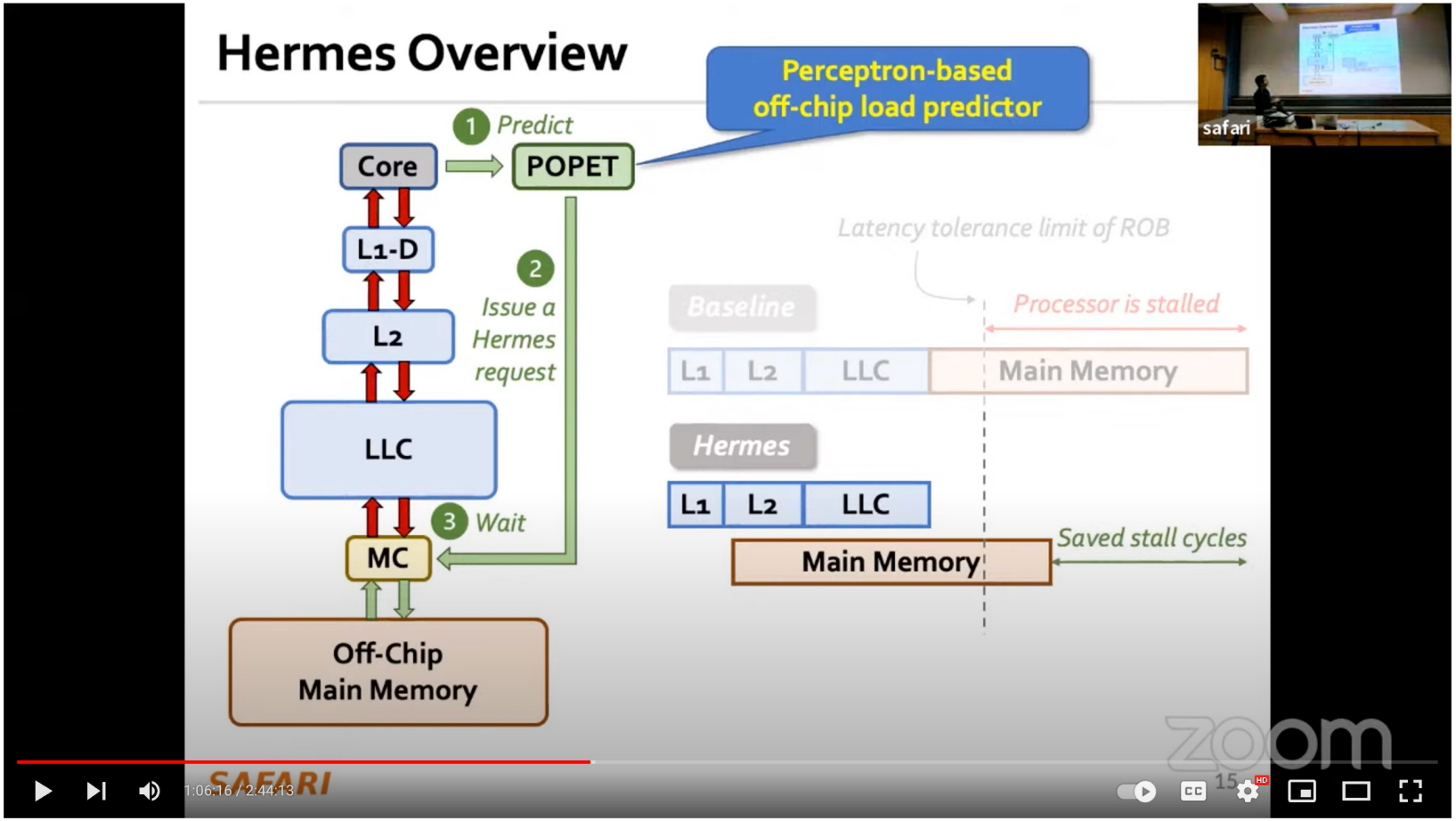
Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2209.00188.pdf

Hermes Talk Video



Computer Architecture - Lecture 18: Cutting-Edge Research in Computer Architecture (Fall 2022)

Onur Mutlu Lectures
32.9K subscribers
Analytics | Edit video
23 | Share | Download | Clip | Save

2.4K views Streamed 5 months ago Livestream - Computer Architecture - ETH Zürich (Fall 2022)
Computer Architecture, ETH Zürich, Fall 2022 (https://safari.ethz.ch/architecture/f...)

https://www.youtube.com/watch?v=PWWBtrL60dQ&t=3609s

HERMES

Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

https://github.com/CMU-SAFARI/Hermes

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

LIRMM

https://arxiv.org/pdf/2209.00188.pdf

The Key Problem

Long-latency off-chip load requests

Often **stall** processor by
blocking instruction retirement from
Reorder Buffer (ROB)

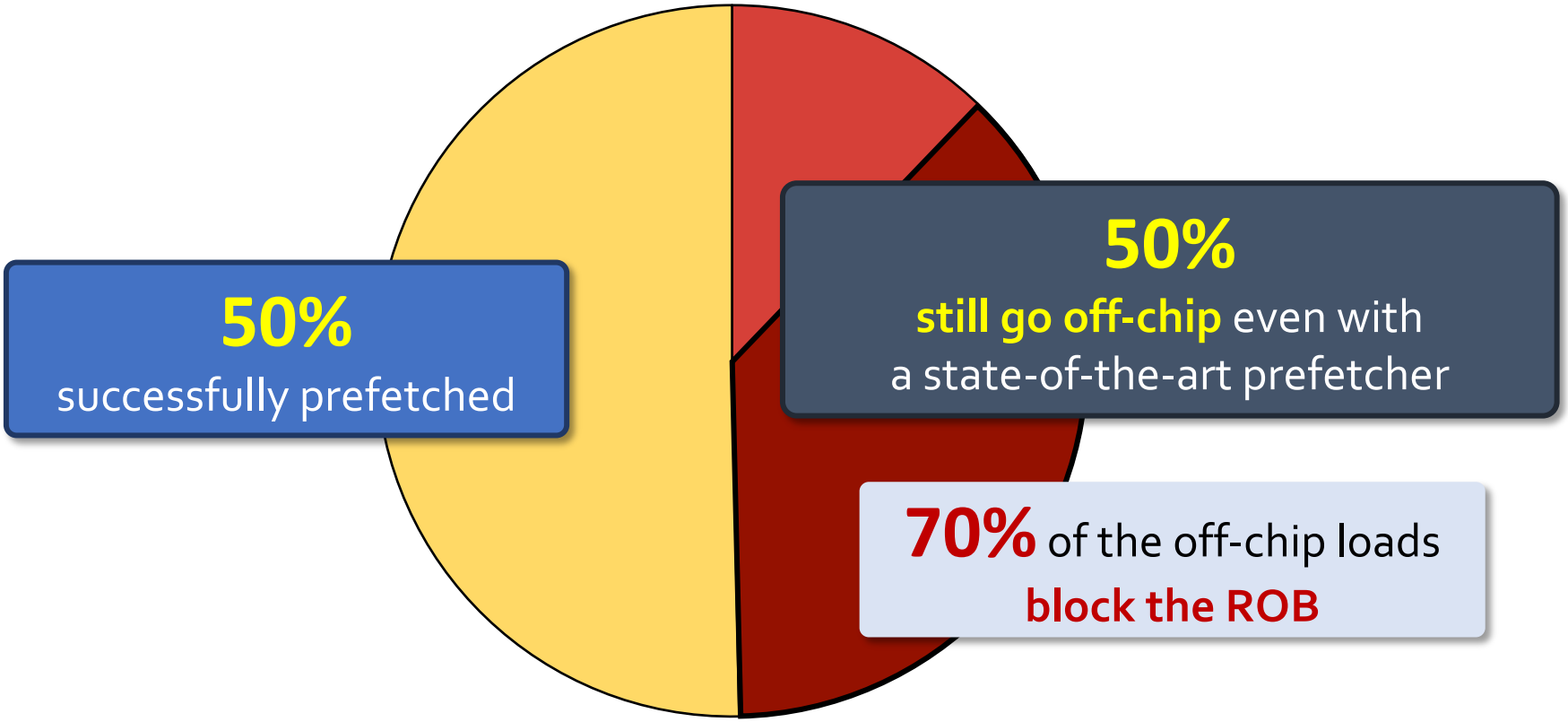
Limit performance

Traditional Solutions

1. Employ sophisticated **prefetchers**
2. **Increase size** of on-chip caches

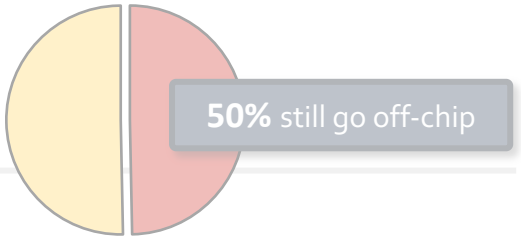
Key Observation 1

Many loads still go off-chip

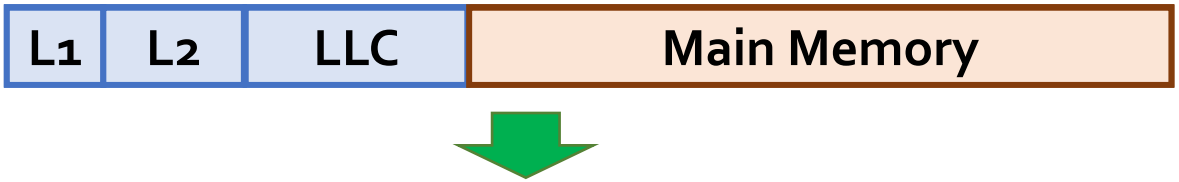


 # off-chip loads without any prefetcher

Key Observation 2

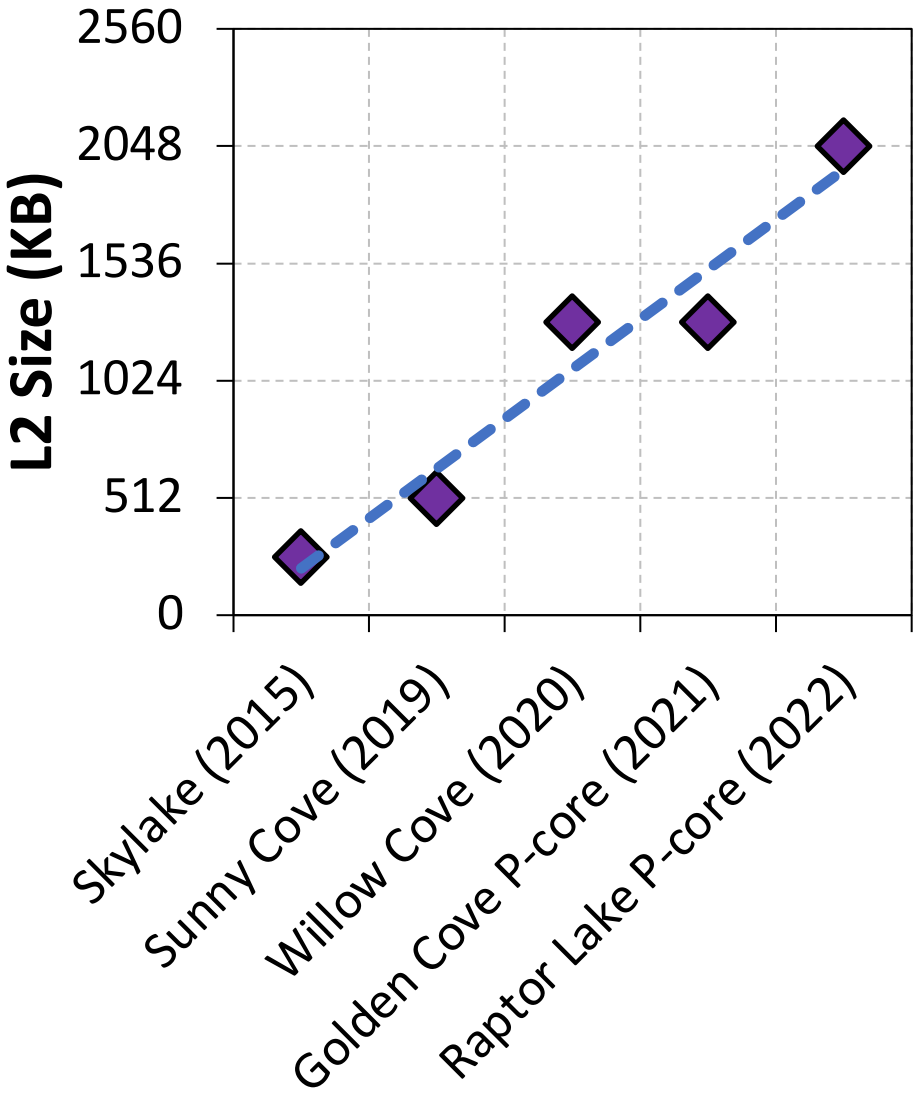


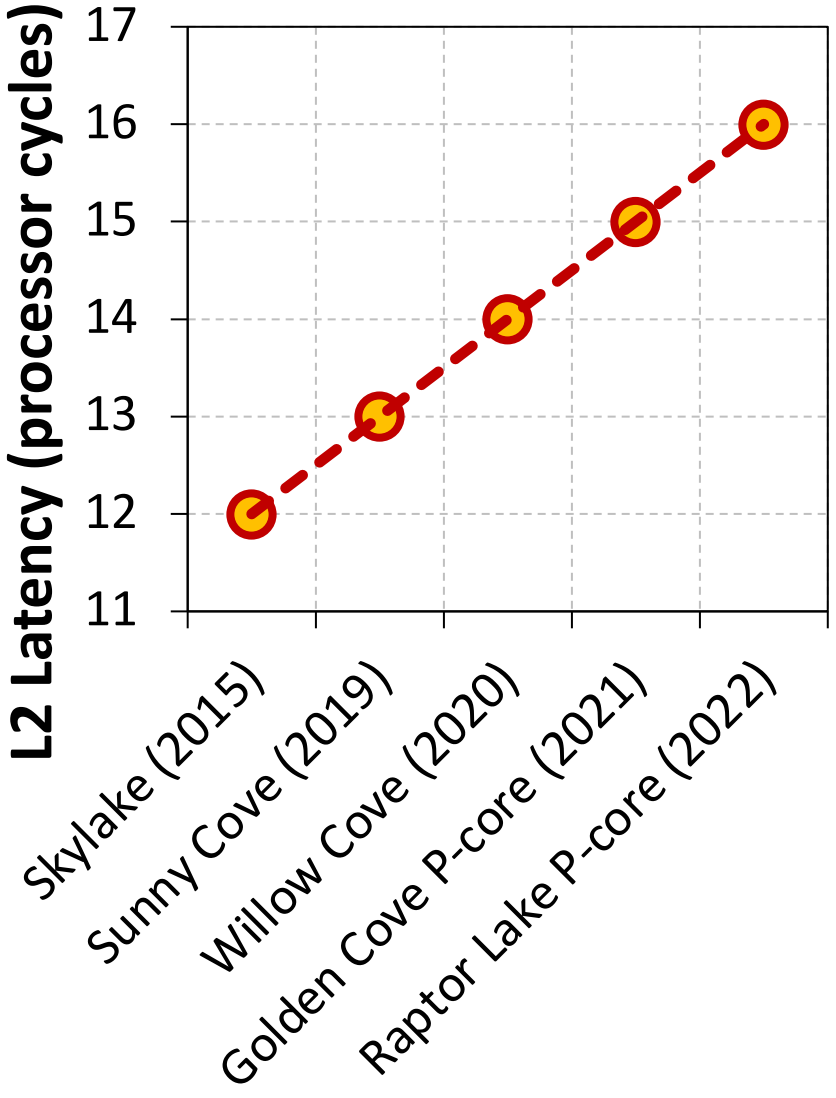
On-chip cache access latency
significantly contributes to off-chip load latency



40% of the **stalls** can be eliminated by **removing on-chip cache access latency from critical path**

Caches are Getting Bigger and Slower…





Our Goal

Improve processor performance
by **removing on-chip cache access latency**
from the **critical path of off-chip loads**

HERMES

Predicts which load requests
are likely to **go off-chip**

Starts **fetching** data **directly** from **main memory**
while concurrently accessing the cache hierarchy

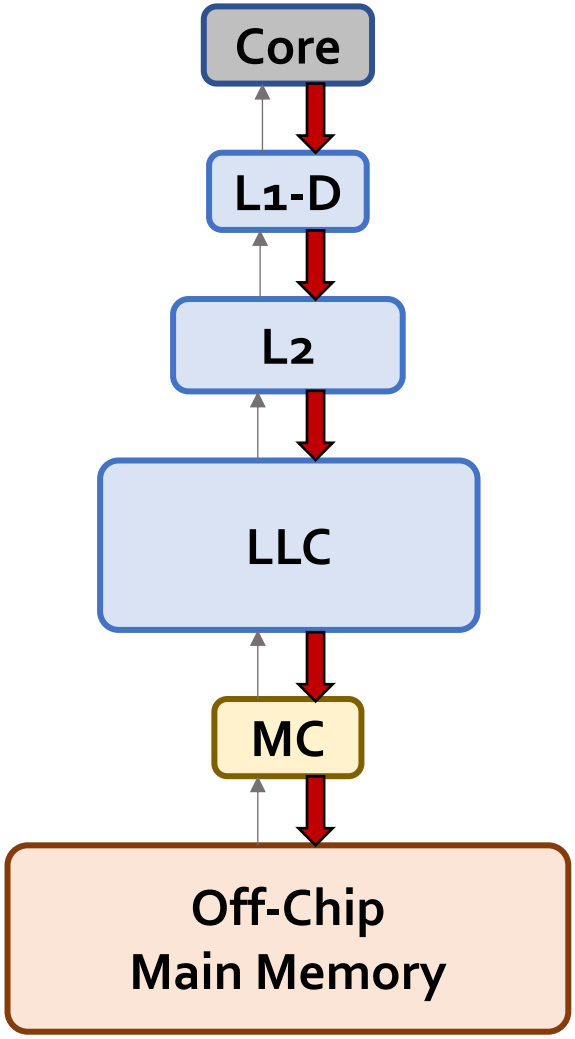
Key Contribution

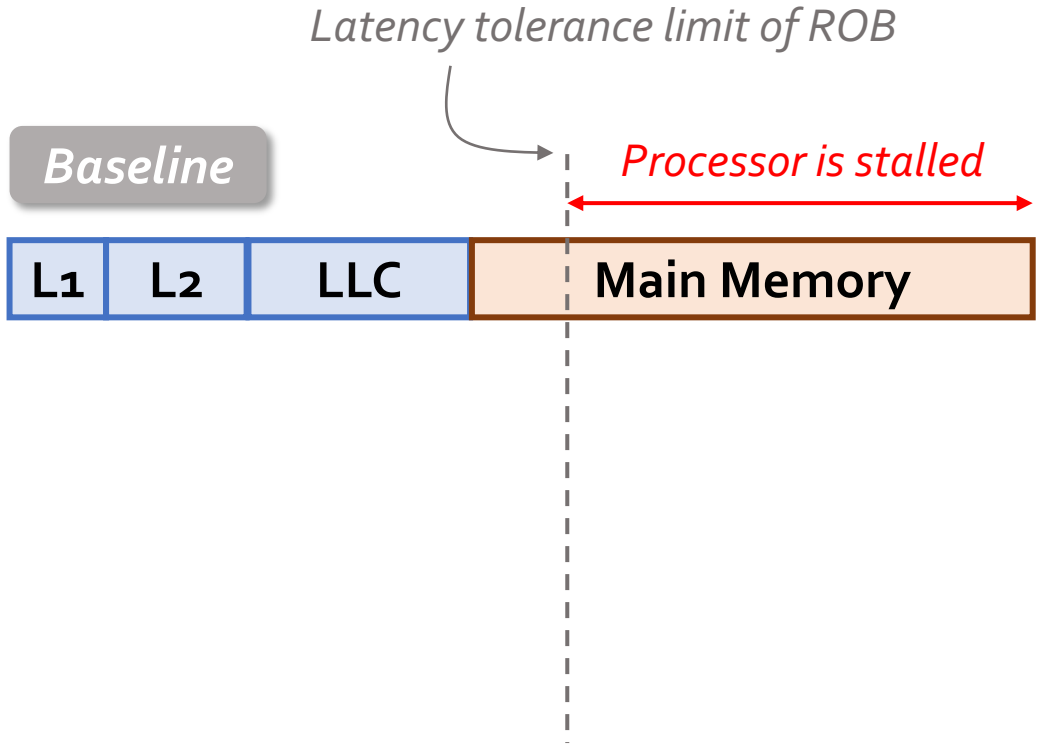
Hermes employs **the first perceptron-based** off-chip load predictor

That predicts which loads are likely to **go off-chip**

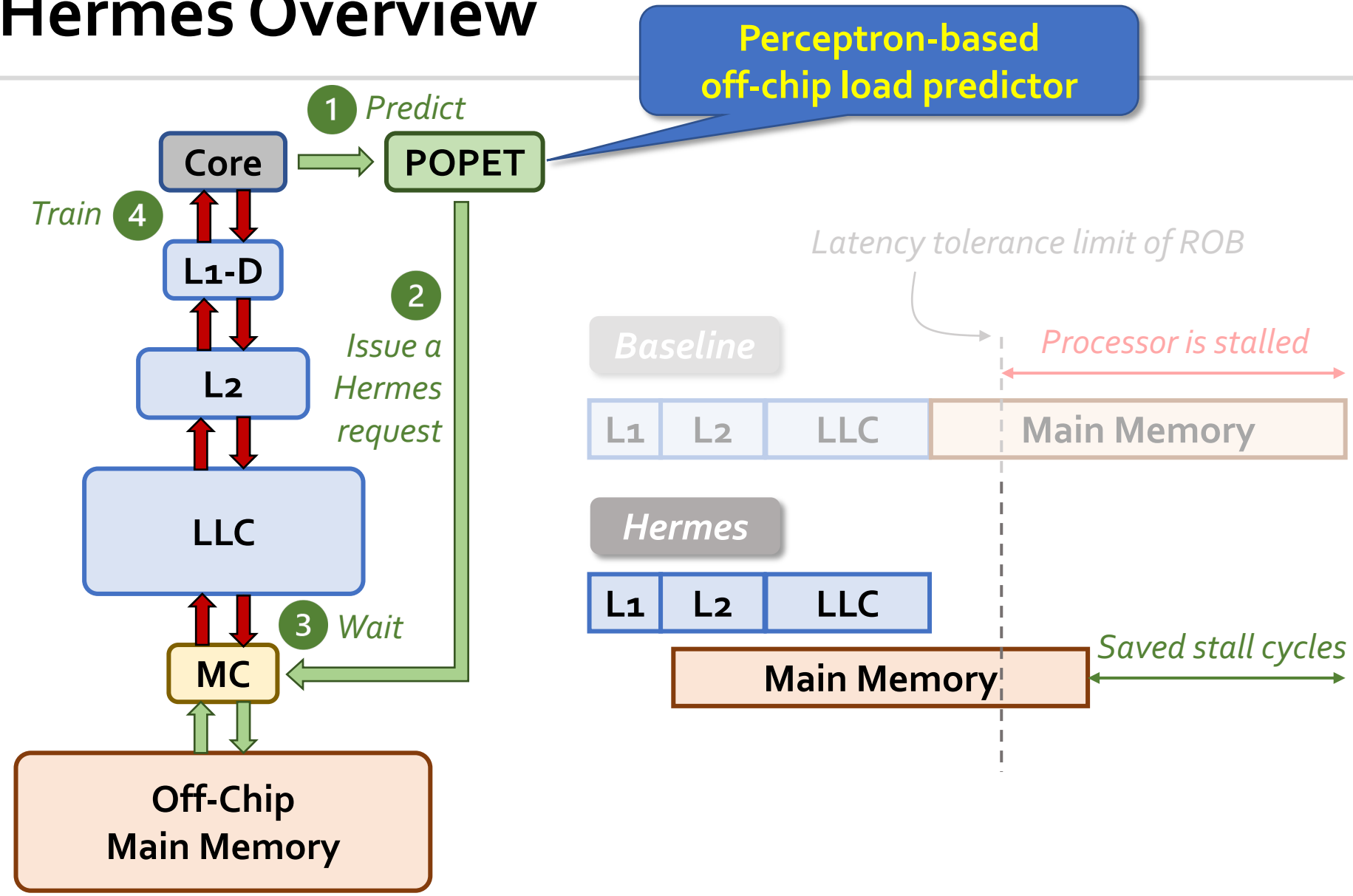
By **learning** from **multiple program context information**

Hermes Overview





Hermes Overview



Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the L1-D cache

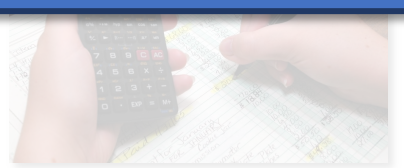
Using branch-predictor-like hybrid predictor:
Global, Gshare, and GSkew

POPET provides
both **higher accuracy** and **higher performance**
than predictors inspired from these previous works

- Metadata size increases with cache hierarchy size

May need to track **all** cache operations

- Gets complex depending on the cache hierarchy configuration (e.g., inclusivity, bypassing,...)

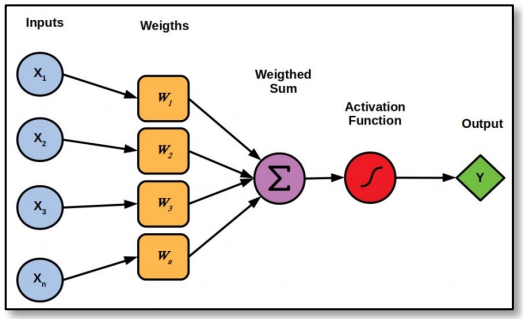


Learning from program behavior

Correlate different program features with off-chip loads

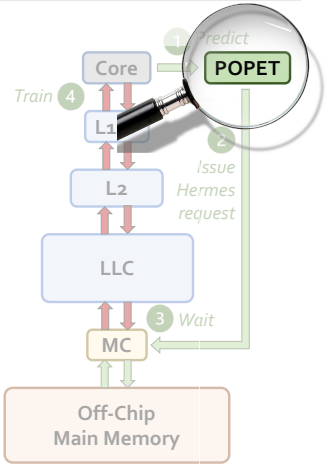
Low storage overhead

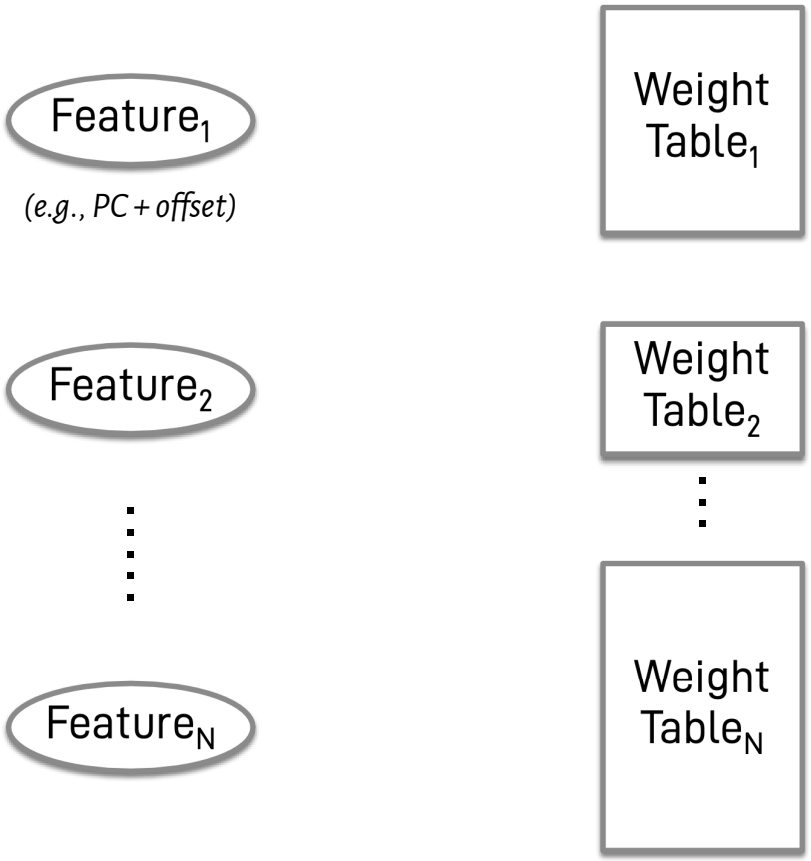
Low design complexity



POPET: Perceptron-Based Off-Chip Predictor

- Multi-feature hashed perceptron model[1]
  - Each feature has its own *weight table*
    - Stores correlation between feature value and off-chip prediction

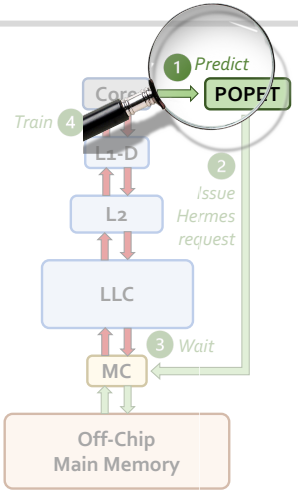


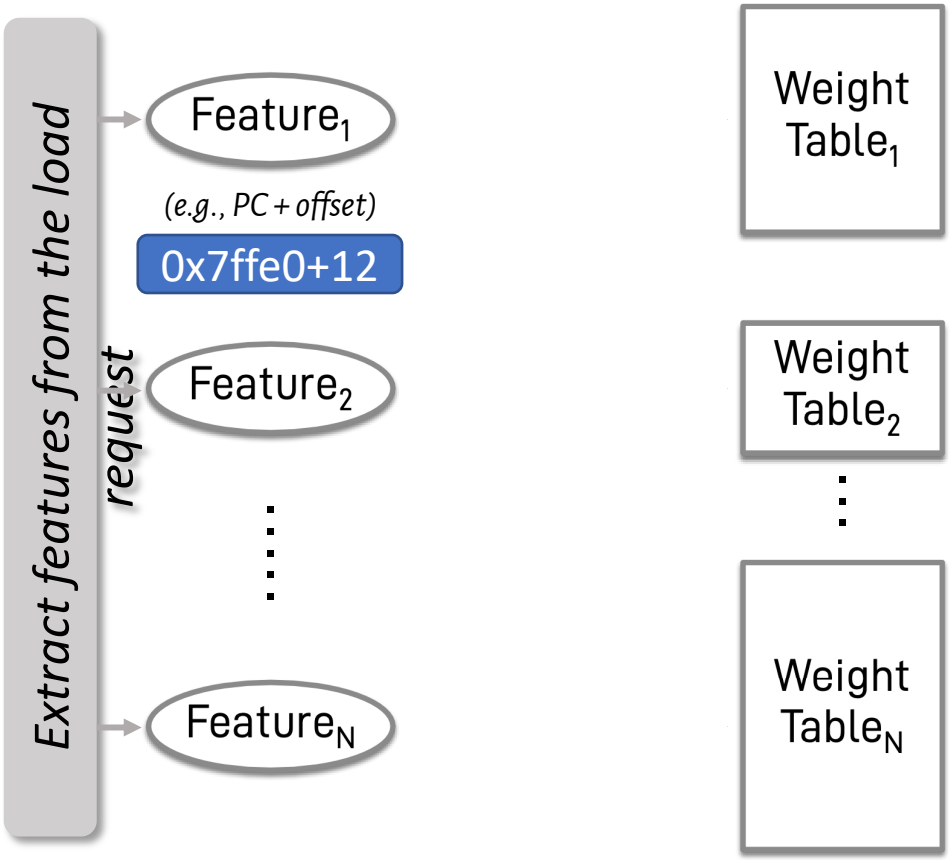


[1] D. Tarjan and K. Skadron, "Merging Path and Gshare Indexing in Perceptron Branch Prediction," TACO, 2005

Predicting using POPET

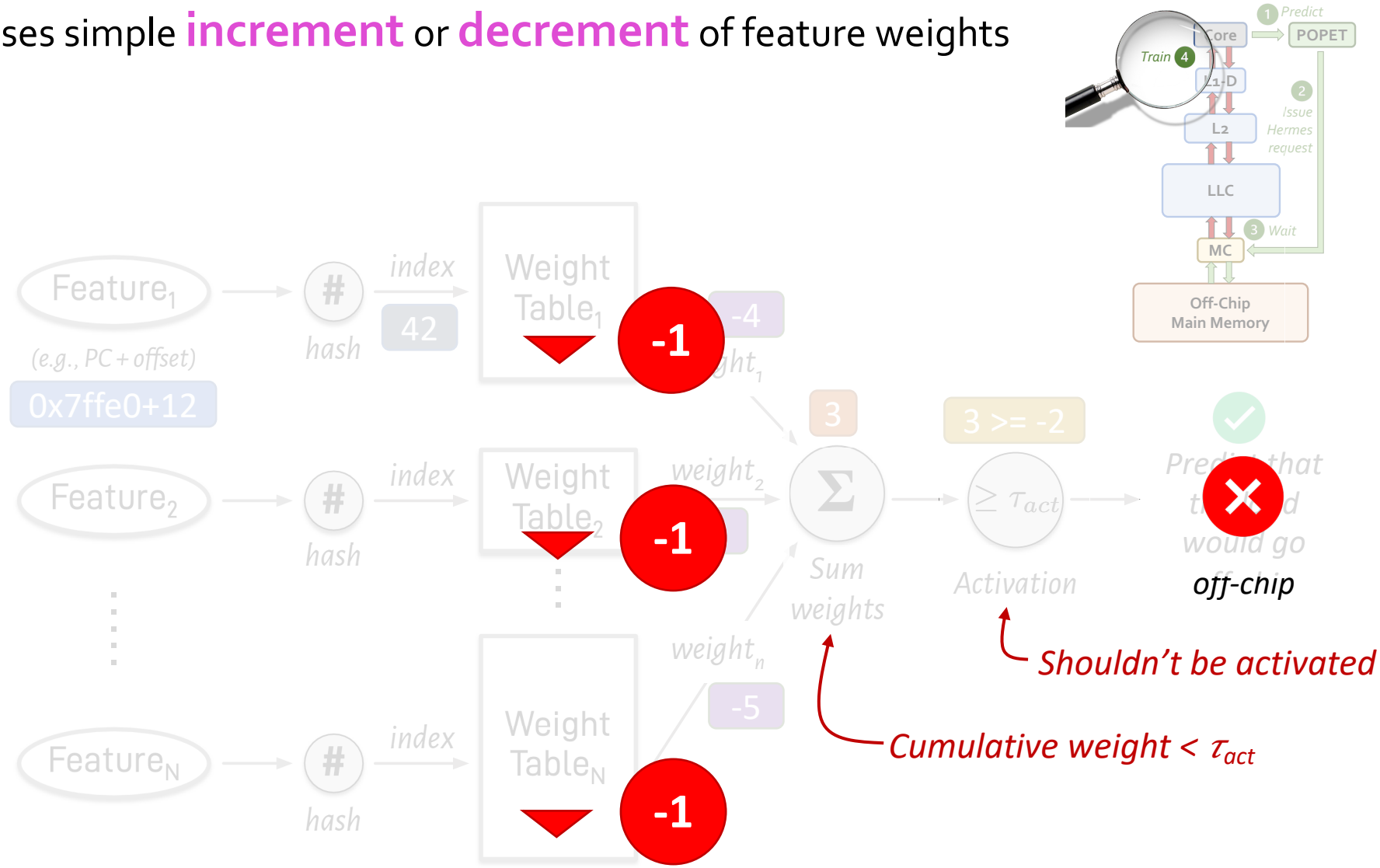
- Uses simple **table lookups**, **addition**, and **comparison**





Training POPET

- Uses simple **increment** or **decrement** of feature weights



Features Used in Hermes

Table 1: The initial set of program features used for automated feature selection. ⊕ represents a bitwise XOR operation.

Features without control-flow information	Features with control-flow information
	8. Load PC
1. Load virtual address	9. PC ⊕ load virtual address
2. Virtual page number	10. PC ⊕ virtual page number
3. Cacheline offset in page	11. PC ⊕ cacheline offset
4. First access	12. PC + first access
5. Cacheline offset + first access	13. PC ⊕ byte offset
6. Byte offset in cacheline	14. PC ⊕ word offset
7. Word offset in cacheline	15. Last-4 load PCs
	16. Last-4 PCs

Table 2: POPET configuration parameters

Selected features	• PC ⊕ cacheline offset • PC ⊕ byte offset • PC + first access • Cacheline offset + first access • Last-4 load PCs
Threshold values	$\tau_{act} = -18, T_N = -35, T_P = 40$

Evaluation

Simulation Methodology

- **ChampSim** trace driven simulator
- **110 single-core** memory-intensive traces
  - SPEC CPU 2006 and 2017
  - PARSEC 2.1
  - Ligra
  - Real-world applications
- **220 eight-core** memory-intensive trace mixes

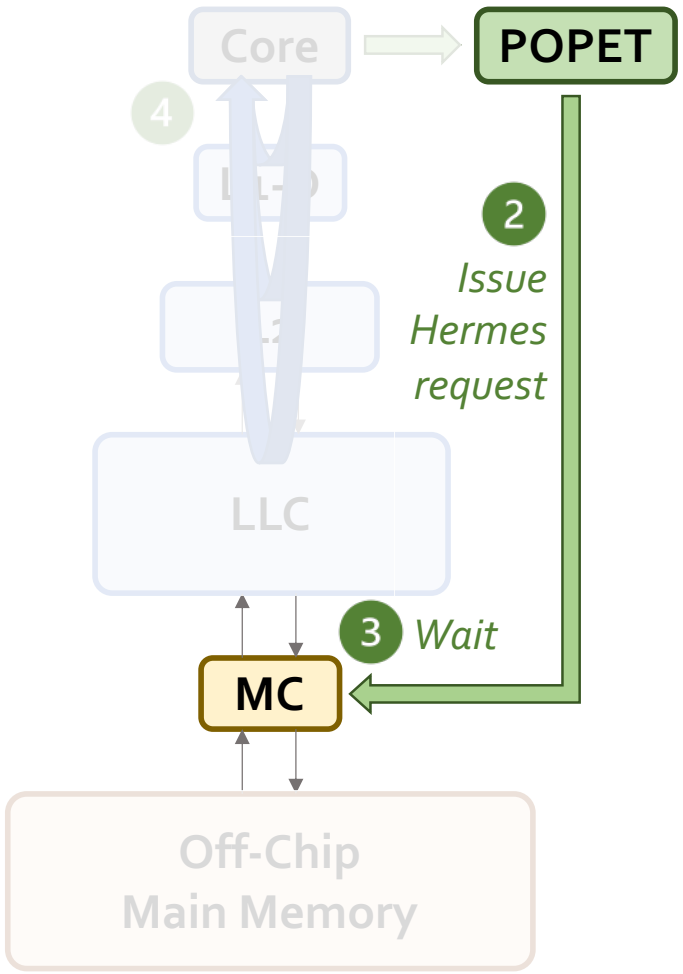
LLC Prefetchers

- Pythia [Bera+, MICRO'21]
- Bingo [Bakshalipour+, HPCA'19]
- MLOP [Shakerinava+, 3rd Prefetching Championship'19]
- SPP + Perceptron filter [Bhatia+, ISCA'20]
- SMS [Somogyi+, ISCA'06]

Off-Chip Predictors

- **History-based**: **HMP** [Yoaz+, ISCA'99]
- **Tracking-based**: Address Tag-Tracking based Predictor (**TTP**)
- **Ideal Off-chip Predictor**

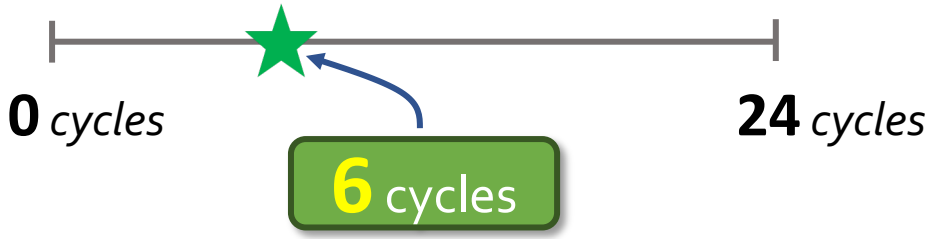
Latency Configuration



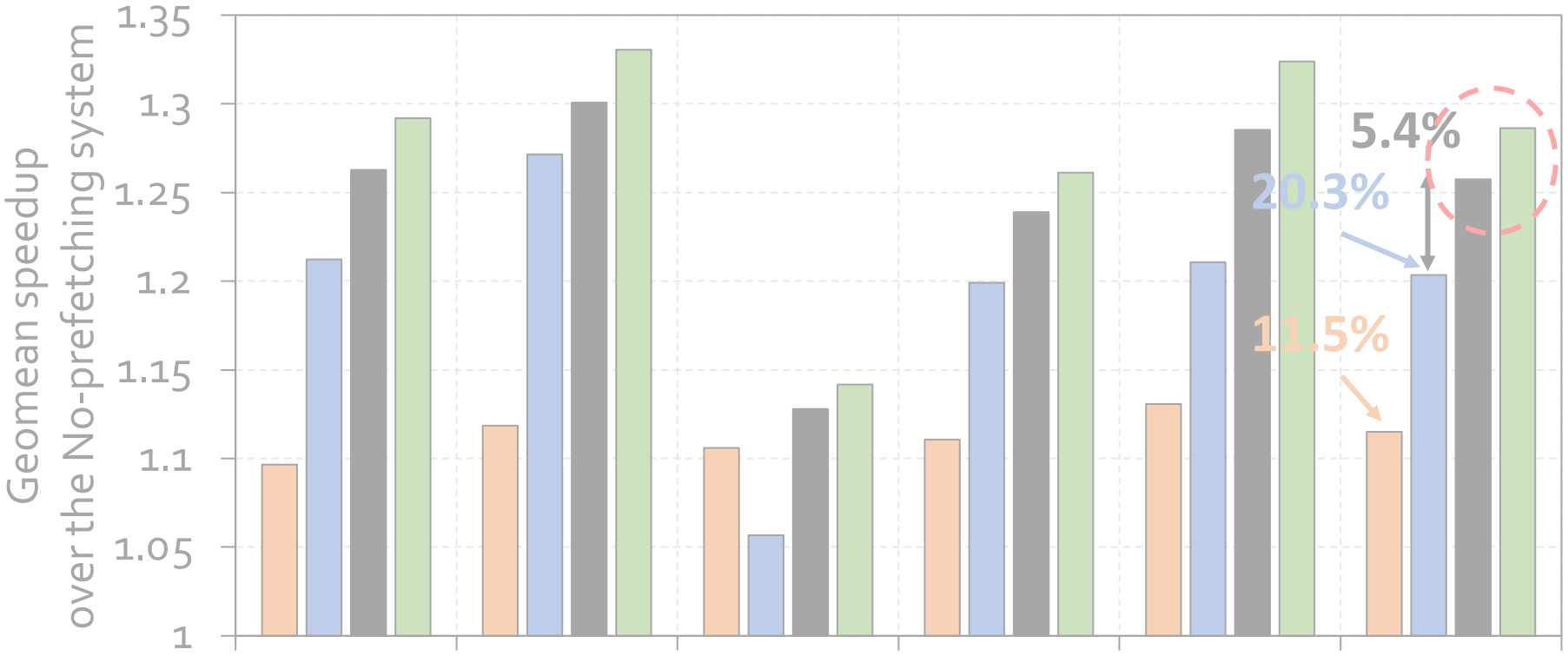
- **Cache *round-trip* latency**
  - L1-D: **5** cycles
  - L2: **15** cycles
  - LLC: **55** cycles

- **Hermes request issue latency**
  (incurred after address translation)

  Depends on
  - **Interconnect** between POPET and MC



Single-Core Performance Improvement

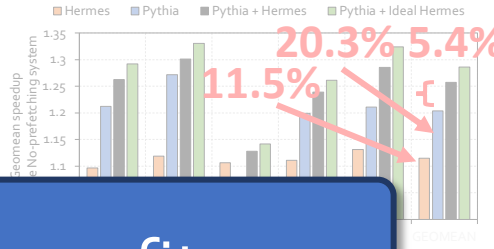


Hermes alone provides nearly

Hermes provides nearly **90%** performance benefit of **Ideal Hermes** that has an **ideal off-chip load predictor**

with only **1/3rd** storage overhead

Increase in Main Memory Requests

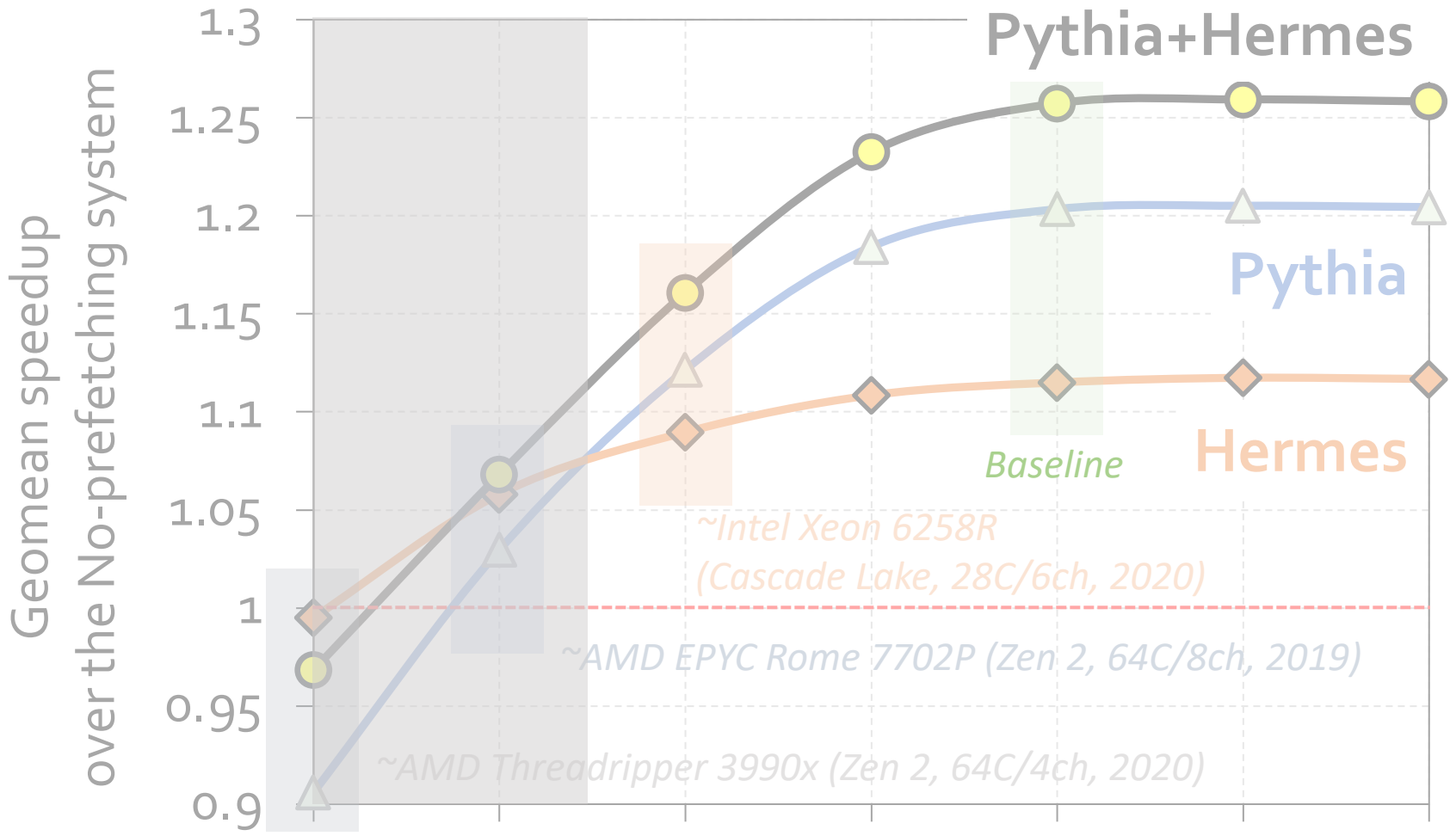


For **every 1% performance** benefit,
increase in main memory requests

Pythia	**2%**
Hermes on top of Pythia	**1%**
Hermes alone	**0.5%**

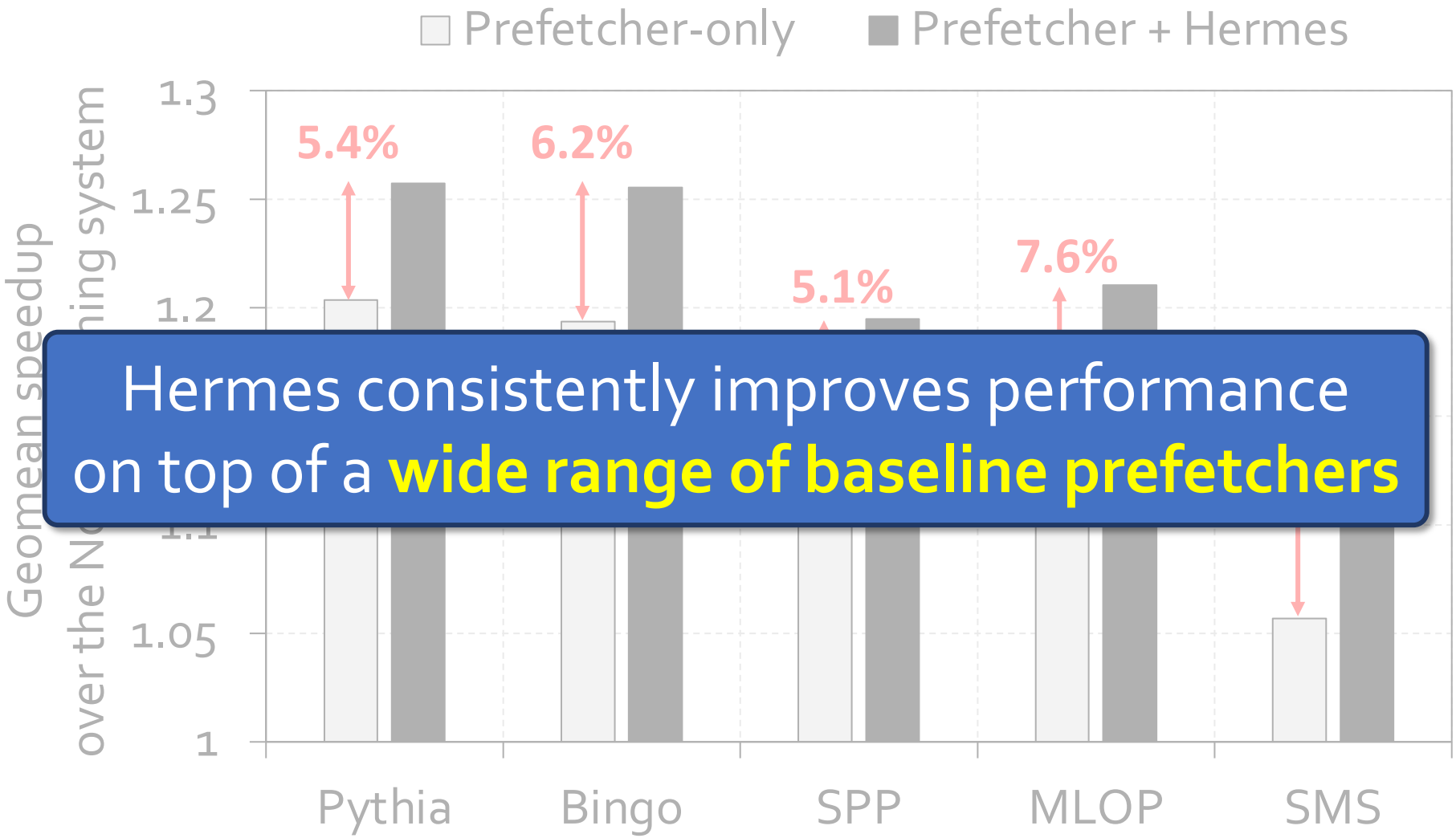
Hermes is more **bandwidth-efficient**
than even an efficient prefetcher like Pythia

Performance with Varying Memory Bandwidth



Hermes+Pythia outperforms Pythia
across all bandwidth configurations

Performance with Varying Baseline Prefetcher



Hermes consistently improves performance on top of a **wide range of baseline prefetchers**

Overhead of Hermes

4 KB storage overhead

1.5% power overhead*

**On top of an Intel Alder Lake-like performance-core [2] configuration*

[2] https://www.anandtech.com/show/16881/a-deep-dive-into-intels-alder-lake-microarchitectures/3

More in the Paper

- Performance sensitivity to:
  - Cache hierarchy access latency
  - Hermes request issue latency
  - Activation threshold
  - ROB size (in extended version on arXiv)
  - LLC size (in extended version on arXiv)

- **Accuracy, coverage, and performance analysis** against **HMP** and **TTP**

- Understanding **usefulness of each program feature**

- Effect on **stall cycle reduction**

- **Performance** analysis on an **eight-core** system

More in the Paper

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

Long-latency load requests continue to limit the performance of modern high-performance processors. To increase the latency tolerance of a processor, architects have primarily relied on two key techniques: sophisticated data prefetchers and large on-chip caches. In this work, we show that: (1) even a sophisticated state-of-the-art prefetcher can only predict half of the off-chip load requests on average across a wide range of workloads, and (2) due to the increasing size and complexity of on-chip caches, a large fraction of the latency of an off-chip load request is spent accessing the on-chip cache hierarchy to solely determine that it needs to go off-chip.

The goal of this work is to accelerate off-chip load requests by removing the on-chip cache access latency from their critical path. To this end, we propose a new technique called Hermes, whose key idea is to: (1) accurately predict which load requests

off-chip main memory (i.e., an *off-chip load*) often stalls the processor core by blocking the instruction retirement from the reorder buffer (ROB), thus limiting the core's performance [88, 91, 92]. To increase the latency tolerance of a core, computer architects primarily rely on two key techniques. First, they employ increasingly sophisticated hardware prefetchers that can learn complex memory address patterns and fetch data required by future load requests before the core demands them [28, 32, 33, 35, 75]. Second, they significantly scale up the size of the on-chip cache hierarchy with each new generation of processors [10, 11, 16].

Key problem. Despite recent advances in processor core design, we observe two key trends in new processor designs that leave a significant opportunity for performance improvement on the table. First, even a sophisticated state-of-the-art

https://arxiv.org/pdf/2209.00188.pdf

To Summarize…

Summary

Hermes advocates for **off-chip load prediction**,
a **different** form of speculation than
load address prediction employed by prefetchers

Off-chip load prediction can be applied **by itself**
or **combined with load address prediction**
to provide performance improvement

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

High coverage
(74%)

Low storage overhead
(4KB/core)

High performance improvement
over best prior baseline
(5.4%)

High performance per bandwidth

Hermes is Open Sourced

All workload traces

13 prefetchers

- Stride [Fu+, MICRO'92]
- Streamer [Chen and Baer, IEEE TC'95]
- SMS [Somogyi+, ISCA'06]
- AMPM [Ishii+, ICS'09]
- Sandbox [Pugsley+, HPCA'14]
- BOP [Michaud, HPCA'16]
- SPP [Kim+, MICRO'16]
- Bingo [Bakshalipour+, HPCA'19]
- SPP+PPF [Bhatia+, ISCA'19]
- DSPatch [Bera+, MICRO'19]
- MLOP [Shakerinava+, DPC-3'19]
- IPCP [Pakalapati+, ISCA'20]
- Pythia [Bera+, MICRO'21]

9 off-chip predictors

Predictor type	Description
Base	Always NO
Basic	Simple confidence counter-based threshold
Random	Random Hit-miss predictor with a given positive probability
HMP-Local	Hit-miss predictor [Yoaz+, ISCA'99] with local prediction
HMP-GShare	Hit-miss predictor with GShare prediction
HMP-GSkew	Hit-miss predictor with GSkew prediction
HMP-Ensemble	Hit-miss predictor with all three types combined
TTP	Tag-tracking based predictor
Perc	Perceptron-based OCP used in this paper

https://github.com/CMU-SAFARI/Hermes

Easy To Define Your Own Off-Chip Predictor

- Just extend the OffchipPredBase class

```
8   class OffchipPredBase
9   {
10  public:
11      uint32_t cpu;
12      string type;
13      uint64_t seed;
14      uint8_t dram_bw; // current DRAM bandwidth bucket
15
16      OffchipPredBase(uint32_t _cpu, string _type, uint64_t _seed) : cpu(_cpu), type(_type), seed(_seed)
17      {
18          srand(seed);
19          dram_bw = 0;
20      }
21      ~OffchipPredBase() {}
22      void update_dram_bw(uint8_t _dram_bw) { dram_bw = _dram_bw; }
23
24      virtual void print_config();
25      virtual void dump_stats();
26      virtual void reset_stats();
27      virtual void train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
28      virtual bool predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
29  };
30
31  #endif /* OFFCHIP_PRED_BASE_H */
32
```

Easy To Define Your Own Off-Chip Predictor

- Define your own `train()` and `predict()` functions

```
19  void OffchipPredBase::train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
20  {
21      // nothing to train
22  }
23
24  bool OffchipPredBase::predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
25  {
26      // predict randomly
27      // return (rand() % 2) ? true : false;
28      return false;
29  }
```

- Get statistics like accuracy (stat name *precision*) and coverage (stat name *recall*) out of the box

```
Core_0_offchip_pred_true_pos 2358716
Core_0_offchip_pred_false_pos 276883
Core_0_offchip_pred_false_neg 132145
Core_0_offchip_pred_precision 89.49
Core_0_offchip_pred_recall 94.69
```

Off-Chip Prediction Can Further Enable…

Prioritizing loads that are likely go off-chip in **cache queues** and **on-chip network routing**

Better instruction scheduling of data-dependent instructions

Other ideas to improve **performance** and **fairness** in multi-core system design…

HERMES

Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

https://github.com/CMU-SAFARI/Hermes

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

LIRMM

https://arxiv.org/pdf/2209.00188.pdf

Hermes Discussion

- **FAQs**
  - What are the selected set of program features?
  - Can you provide some intuition on why these features work?
  - What happens in case of a misprediction?
  - What's the performance headroom for off-chip prediction?
  - Do you see a variance of different features in final prediction accuracy?
- **Simulation Methodology**
  - System parameters
  - Evaluated workloads
- **More Results**
  - Percentage of off-chip requests
  - Reduction in stall cycles by reducing the critical path
  - Fraction of off-chip load requests
  - Accuracy and coverage of POPET
  - Effect of different features
  - Are all features required?
  - 1C performance
  - 1C performance line graph
  - 1C performance against prior predictors
  - Effect on stall cycles
  - 8C performance
  - Sensitivity:
    - Hermes request issue latency
    - Cache hierarchy access latency
    - Activation threshold
    - ROB size
    - LLC size
  - Power overhead
  - Accuracy without prefetcher
  - Main memory request overhead with different prefetchers

Hermes Paper [MICRO 2022]

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,
  "Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Talk Video (12 minutes)]
  [Lecture Video (25 minutes)]
  [arXiv version]
  [Source Code (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best paper award at MICRO 2022.

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2209.00188.pdf

Pythia & Hermes: ML-Driven Prefetching

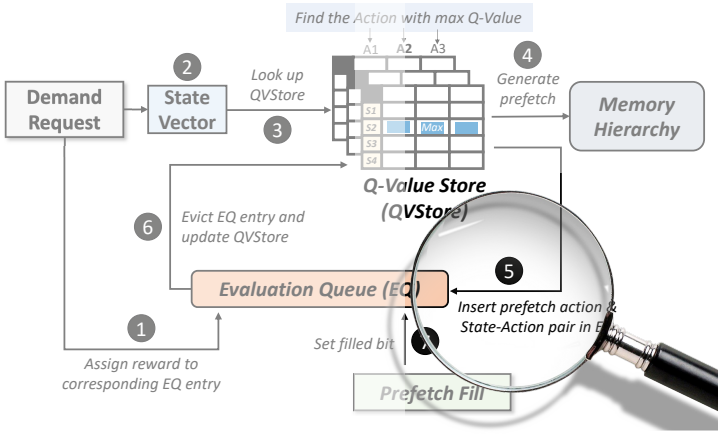
Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
20 September 2023
VMware

SAFARI ETH zürich Carnegie Mellon

PYTHIA BACKUP

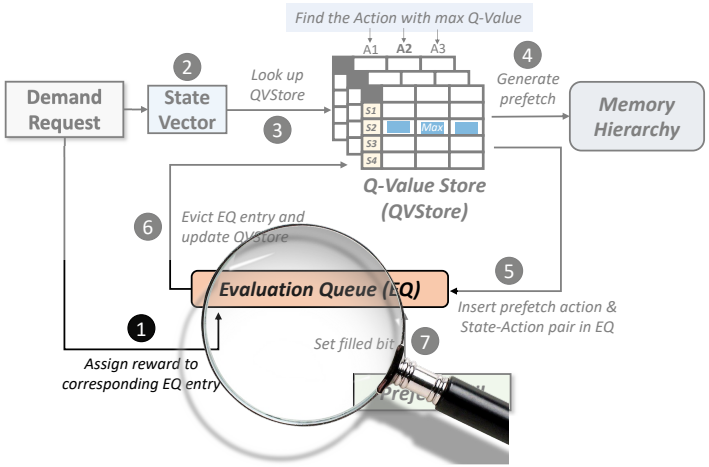
Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction
- **During EQ insertion**: for actions
  - Not to prefetch
  - Out-of-page prefetch



Reward Assignment to EQ Entry

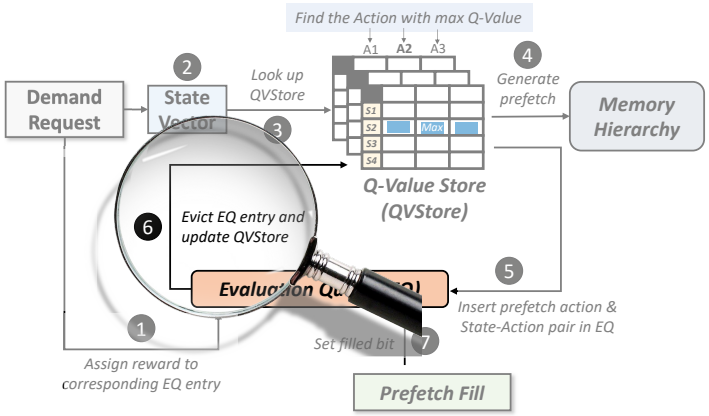
- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction



- **During EQ insertion**: for actions
  - Not to prefetch
  - Out-of-page prefetch
- **During EQ residency**:
  - In case address of a demand matches with address in EQ (*signifies accurate prefetch*)

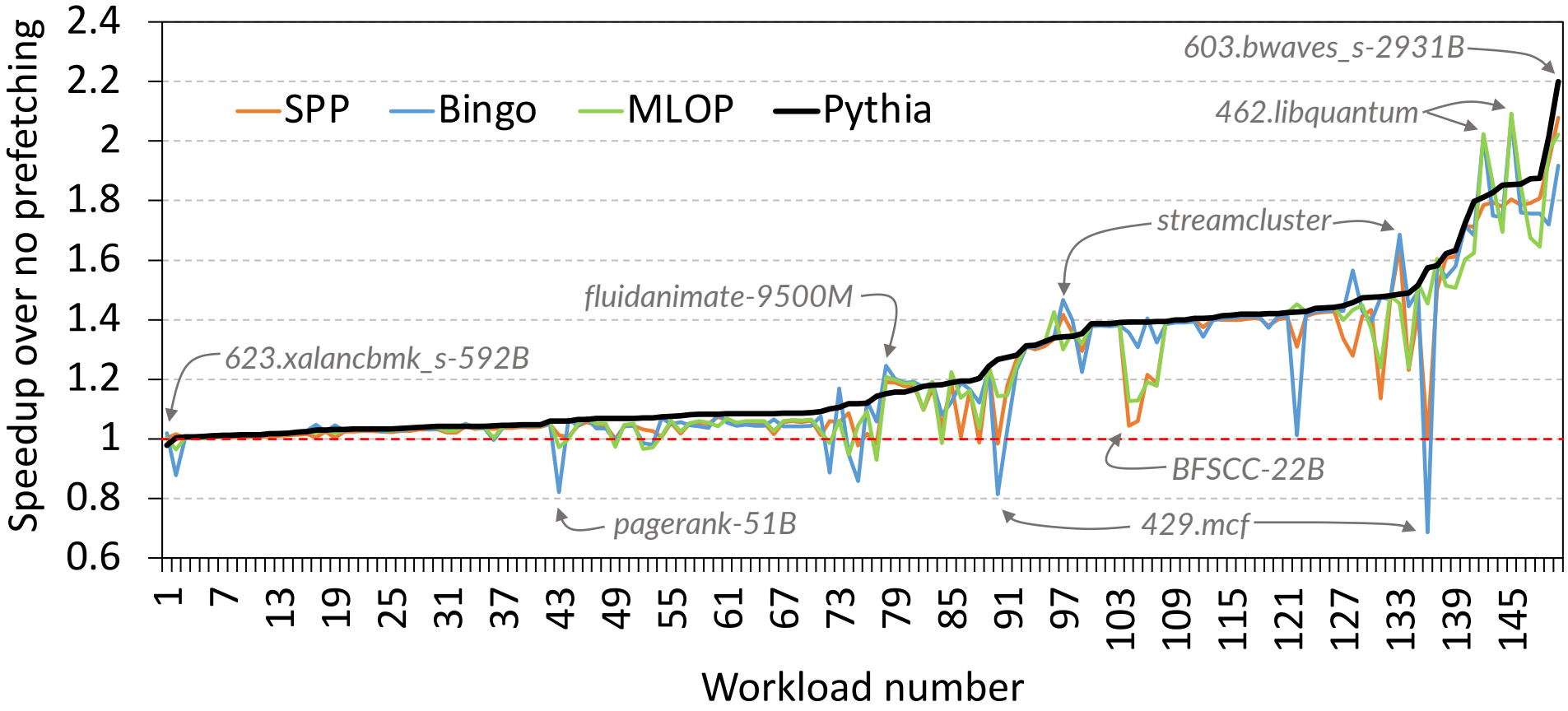
Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction

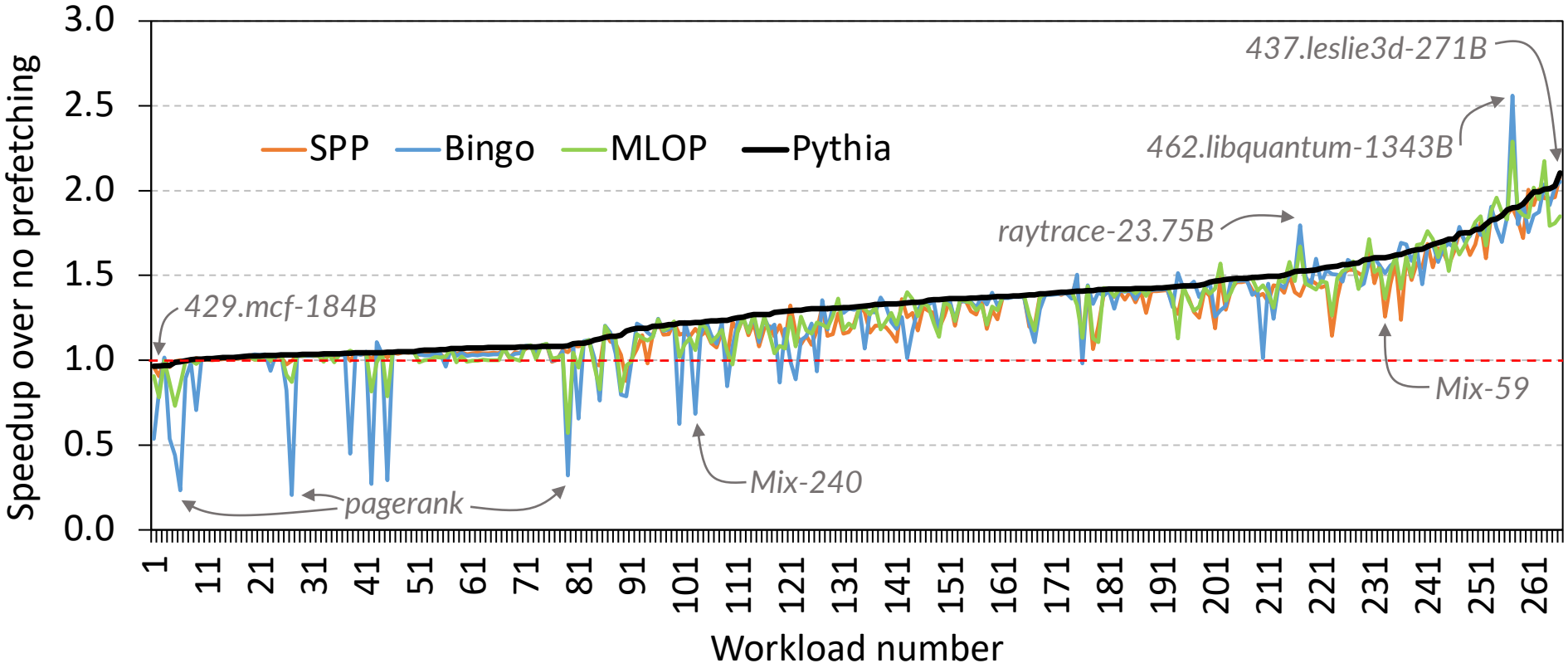


- **During EQ insertion:** for actions
  - Not to prefetch
  - Out-of-page prefetch
- **During EQ residency:**
  - In case address of a demand matches with address in EQ (*signifies accurate prefetch*)
- **During EQ eviction:**
  - In case no reward is assigned till eviction (*signifies inaccurate prefetch*)

Performance S-curve: Single-core



Performance S-curve: Four-core



FAQs

Pythia Discussion

- **FAQs**
  - Why RL?
  - What about large page?
  - What's the prefetch degree?
  - Can customization happen during workload execution?
  - Can runtime mixing create problem?
- **Simulation and Methodology**
  - Basic Pythia configuration
  - System parameters
  - Configuration of prefetchers
  - Evaluated workloads
  - Feature selection
- **Detailed Design**
  - Reward structure
  - Design overview
  - QVStore Organization
- **More Results**
  - Comparison against other adaptive prefetchers
  - Comparison against Context prefetcher
  - Feature combination sensitivity
  - Hyperparameter sensitivity
  - Comparison with multi-level prefetchers
  - Performance in unseen workloads
  - Single-core s-curve
  - Four-core s-curve
  - Detailed performance analysis
  - Benefit of bandwidth awareness
  - Case study
  - Customizing rewards
  - Customizing features

Why RL? Why Not Supervised Learning?

- Determining the **benefits of prefetching** (i.e., whether a decision was good for performance or not) is **not easy**
  - **Depends on a complex set of metrics**
    - Coverage, accuracy, timeliness
    - Effects on system: b/w usage, pollution, cross-application interference, …
  - **Dynamically-changing environmental conditions** change the benefit
  - **Delayed feedback due to long latency** (might not receive feedback at all for inaccurate prefetches!)

- Differs from classification tasks (e.g., branch prediction)
  - Performance strongly correlates mainly to accuracy
  - Does not depend on environment
  - Bounded feedback delay

What About Large Pages?

- Pythia's framework can be **easily extended** to incorporate additional prefetch actions (i.e., possible prefetch offsets for the page size)

- To decrease the storage overhead
  - **Prune action space** via automatic design-space exploration
  - **Hash action values** to retrieve Q-values

What is the Prefetch Degree? Is It Managed by the RL Agent?

- Pythia employs **a simple degree selector**, separate from the RL agent
  - If the agent has selected the same prefetch action (O) multiple times in a row, Pythia increases the degree (A+2O, A+3O, …)
  - At most degree 4

- Future works on managing degree by the RL agent

Can the Customization Be Done While the Workload is Running?

- Certainly.
- Pythia, being an **online learning** technique, will autonomously adapt (and optimize) its policy to use the new program features or the modified reward values

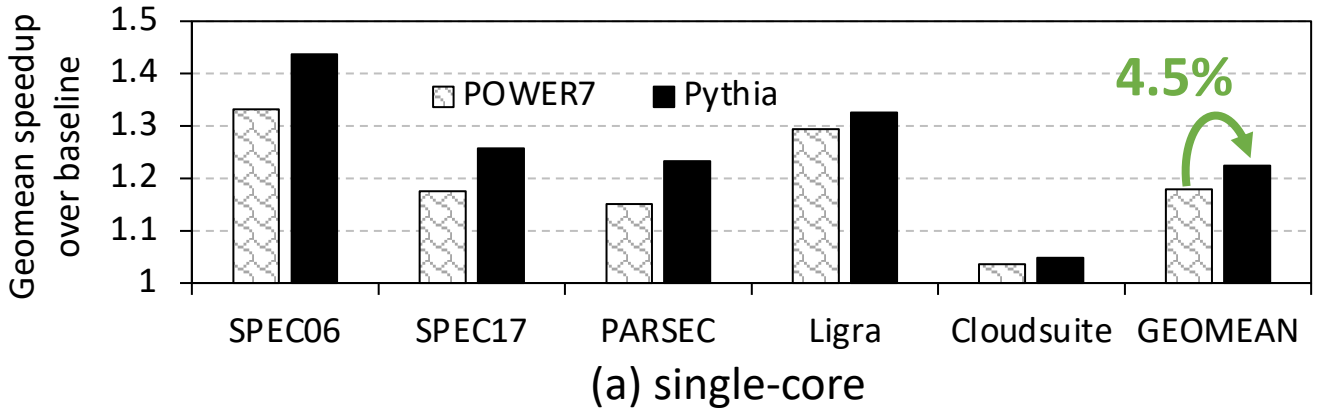
Can Runtime Workload Mix Create an Issue?

- We implement the bandwidth usage feedback using a counter in the memory controller. Thus Pythia already has a **global view of the memory bandwidth usage** that incorporates all workloads running on a multi-core system

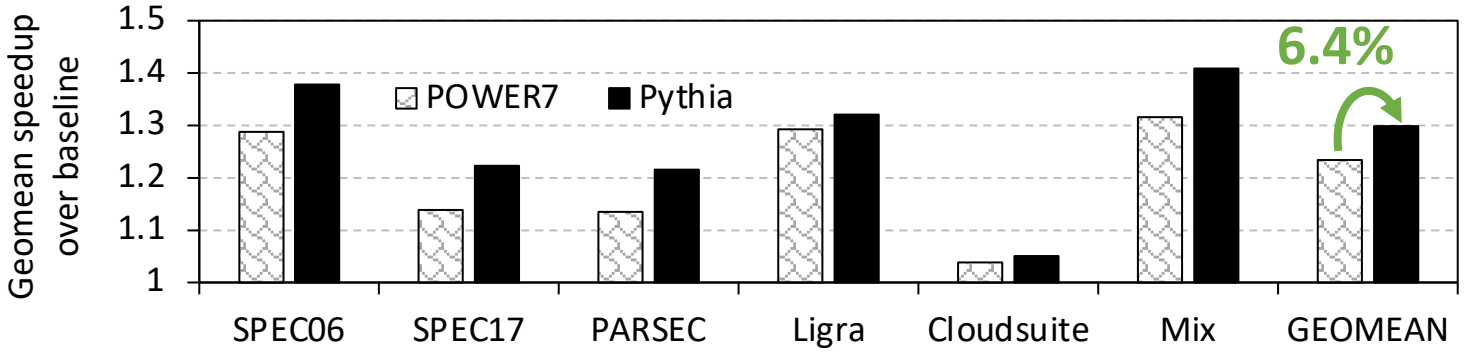
- We evaluate a diverse set (300 of each category) of four-core, eight-core, twelve-core random workload mixes
- Based on our evaluation, we observe that **Pythia dynamically adapts** itself to varying workload demands

How does Pythia Compare Against Other Adaptive Prefetching Solutions?

- We compare Pythia against **IBM POWER7**[5] prefetcher
  - Adaptively selects prefetcher degree/configuration by monitoring program IPC



(a) single-core

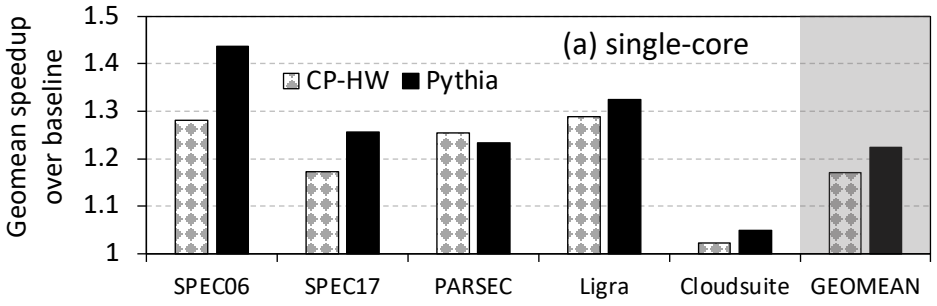


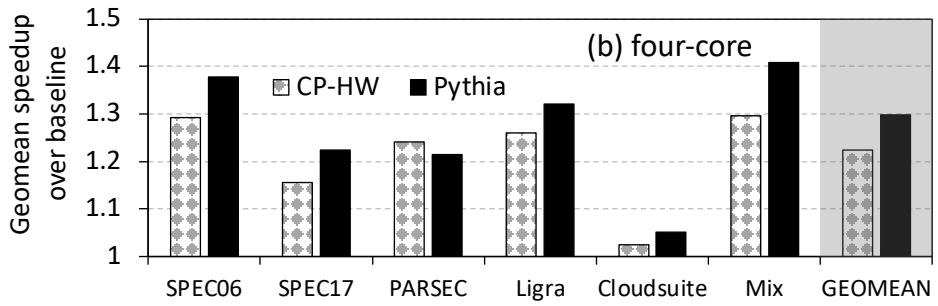
(b) four-core

[5] Jimenez et al., TOPC'14

How Does Pythia Compare Against the Context Prefetcher?

- Pythia widely differs from the Context Prefetcher (CP)[6] in all three aspects: state, action, and reward. The key differences are:
  - CP **does not consider system-level feedback**
  - CP models the agent as a contextual bandit which **takes myopic prefetch decisions** as compared to Pythia
  - CP **requires compiler support** to extract software-level features



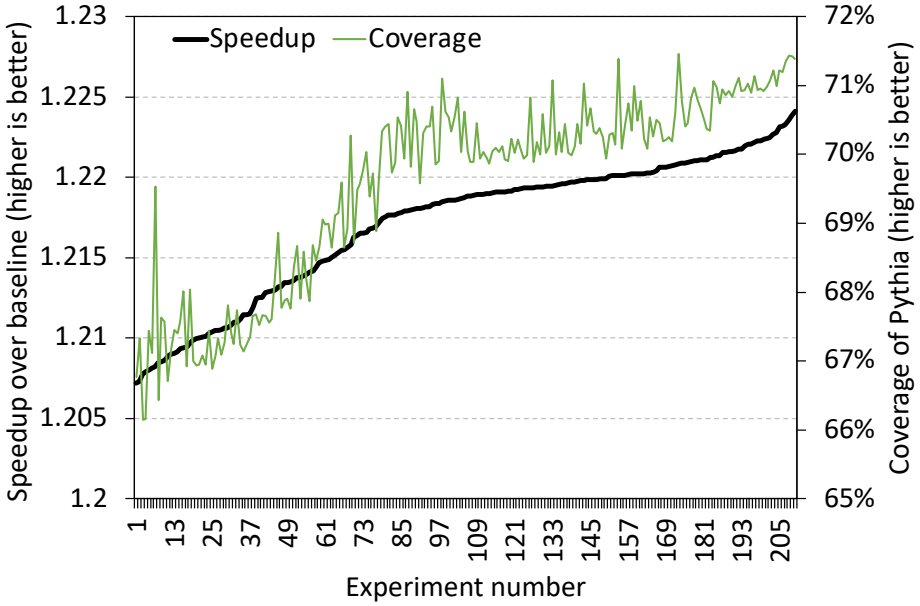


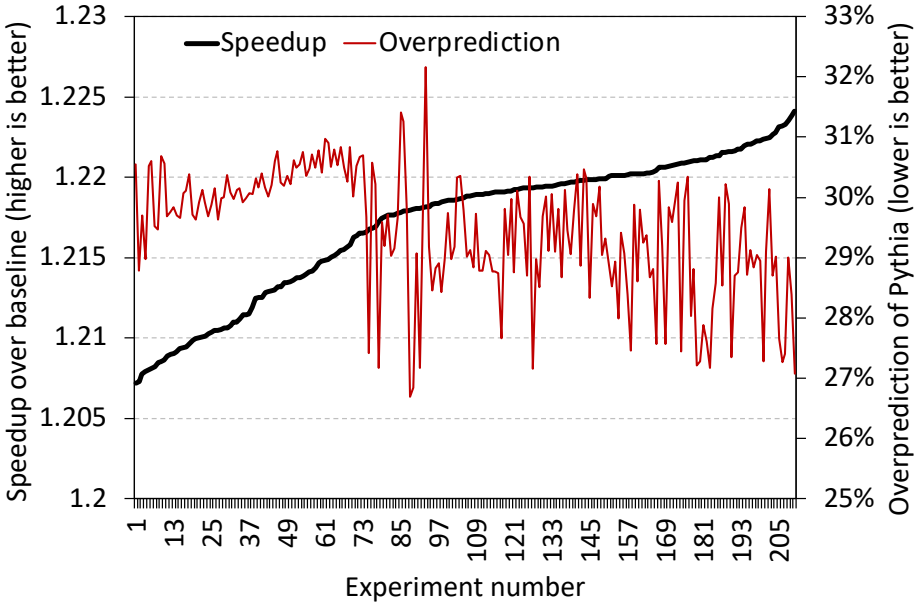
Pythia outperforms CP-HW by 5.3% in single-core and 7.6% in four-core system

[6] Leeor et al., ISCA'15

How Pythia's Performance Changes With Various State Definitions You Have Swept?

- In total we evaluate state defined as any-one, any-two, and any-three combinations of 32 features





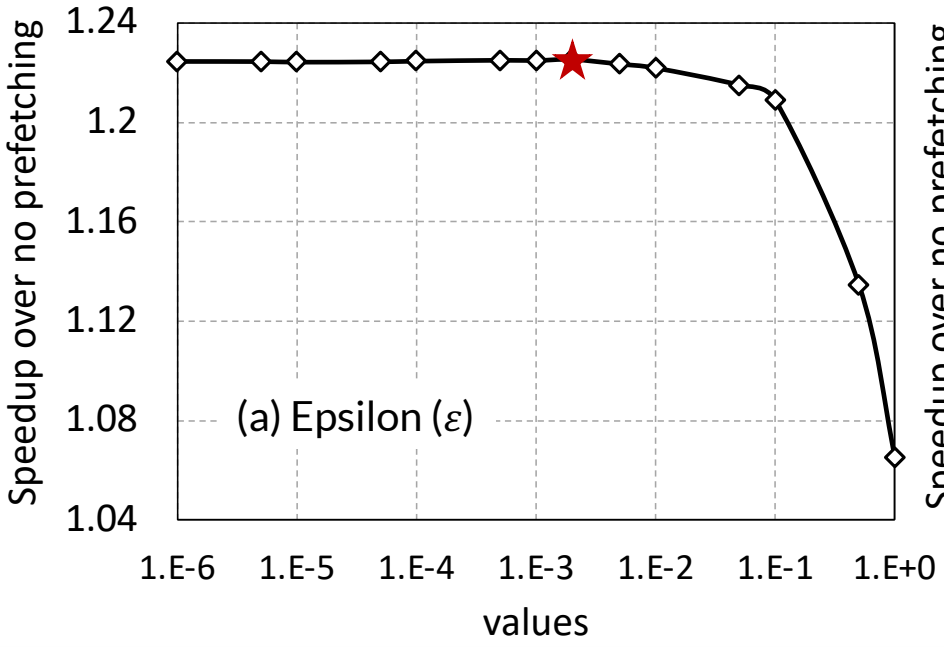
Performance gain ranges from 20.7% to 22.4%

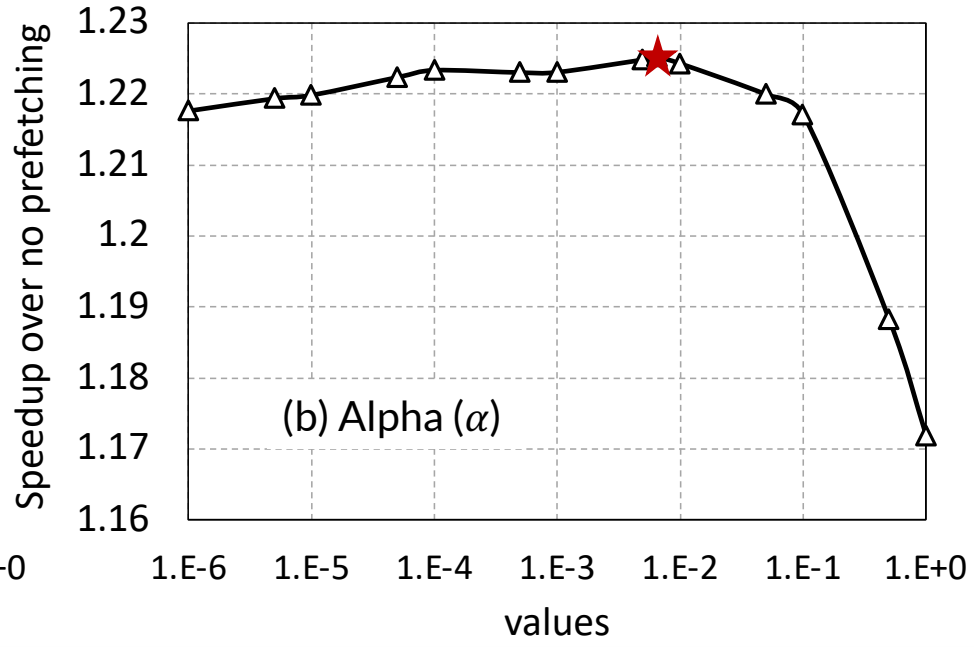
Coverage ranges from 66.2% to 71.5%

Overprediction ranges from 26.7% to 32.2%

Is Pythia Sensitive to Hyperparameters?

- Not setting hyperparameters can significantly impact the overall performance improvement

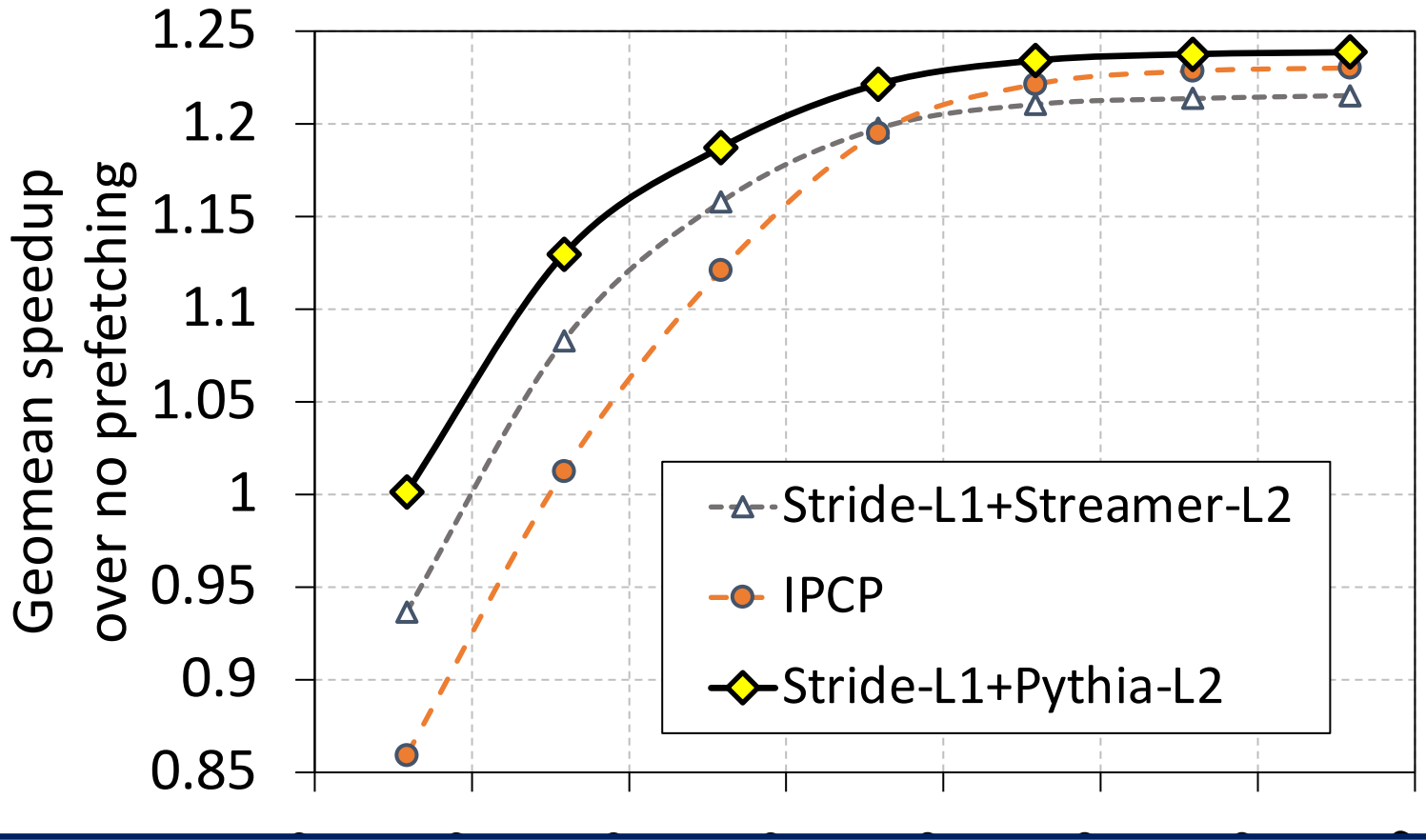




Changing ε from 0.002 to 1.0 drops perf. by 16%

Changing α from 0.0065 to 1.0 drops perf. by 5.4%

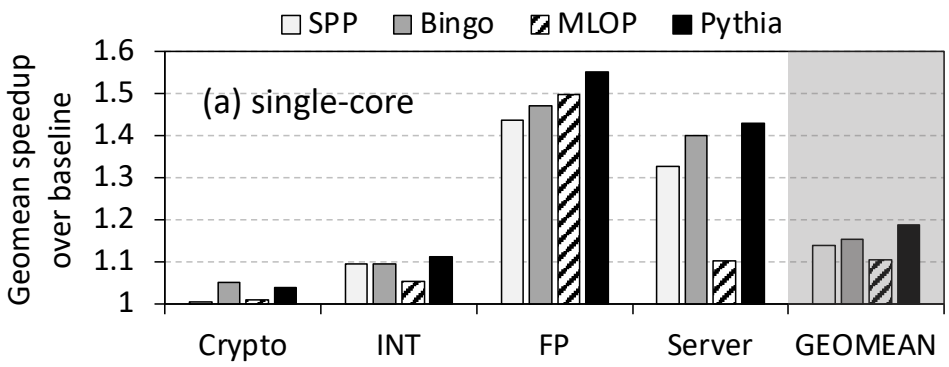
How Does Pythia Compare Against Commercial Multi-level Prefetchers?

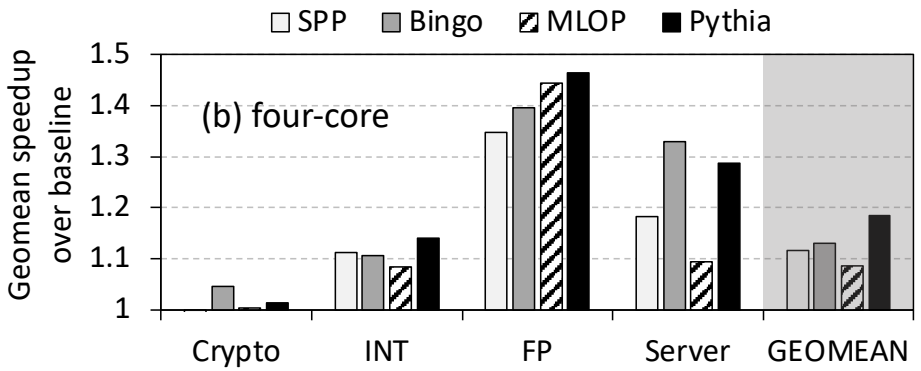


Pythia outperforms IPCP [7] by 14.2% on average in 150-MTPS

Does Pythia Perform Equally Well for Unseen Workloads?

- Evaluated with 500 traces from value prediction championship
  - **No prefetcher has been trained on these traces**





Pythia outperforms MLOP and Bingo by 8.3% and 3.5% in single-core

And 9.7% and 5.4% in four-core

Basic Pythia Configuration

Table 2: Basic Pythia configuration derived from our automated design-space exploration

Features	`PC+Delta, Sequence of last-4 deltas`
Prefetch Action List	{-6,-3,-1,0,1,3,4,5,10,11,12,16,22,23,30,32}
Reward Level Values	$\mathcal{R}_{AT}$=20, $\mathcal{R}_{AL}$=12, $\mathcal{R}_{CL}$=−12, $\mathcal{R}^{H}_{IN}$=−14, $\mathcal{R}^{L}_{IN}$=−8, $\mathcal{R}^{H}_{NP}$=−2, $\mathcal{R}^{L}_{NP}$=−4
Hyperparameters	$\alpha = 0.0065$, $\gamma = 0.556$, $\epsilon = 0.002$

System Parameters

Table 5: Simulated system parameters

Core	1-12 cores, 4-wide OoO, 256-entry ROB, 72/56-entry LQ/SQ
Branch Pred.	Perceptron-based [69], 20-cycle misprediction penalty
L1/L2 Caches	Private, 32KB/256KB, 64B line, 8 way, LRU, 16/32 MSHRs, 4-cycle/14-cycle round-trip latency
LLC	2MB/core, 64B line, 16 way, SHiP [133], 64 MSHRs per LLC Bank, 34-cycle round-trip latency
Main Memory	**1C:** Single channel, 1 rank/channel; **4C:** Dual channel, 2 ranks/channel; **8C:** Quad channel, 2 ranks/channel; 8 banks/rank, 2400 MTPS, 64b data-bus/channel, 2KB row buffer-/bank, tRCD=15ns, tRP=15ns, tCAS=12.5ns

Configuration of Prefetchers

Table 7: Configuration of evaluated prefetchers

SPP [78]	256-entry ST, 512-entry 4-way PT, 8-entry GHR	**6.2 KB**
Bingo [27]	2KB region, 64/128/4K-entry FT/AT/PHT	**46 KB**
MLOP [111]	128-entry AMT, 500-update, 16-degree	**8 KB**
DSPatch [30]	Same configuration as in [30]	**3.6 KB**
PPF [32]	Same configuration as in [32]	**39.3 KB**
Pythia	2 features, 2 vaults, 3 planes, 16 actions	**25.5 KB**

Evaluated Workloads

Table 6: Workloads used for evaluation

Suite	# Workloads	# Traces	Example Workloads
SPEC06	16	28	gcc, mcf, cactusADM, lbm, ...
SPEC17	12	18	gcc, mcf, pop2, fotonik3d, ...
PARSEC	5	11	canneal, facesim, raytrace, ...
Ligra	13	40	BFS, PageRank, Bellman-ford, ...
Cloudsuite	4	53	cassandra, cloud9, nutch, ...

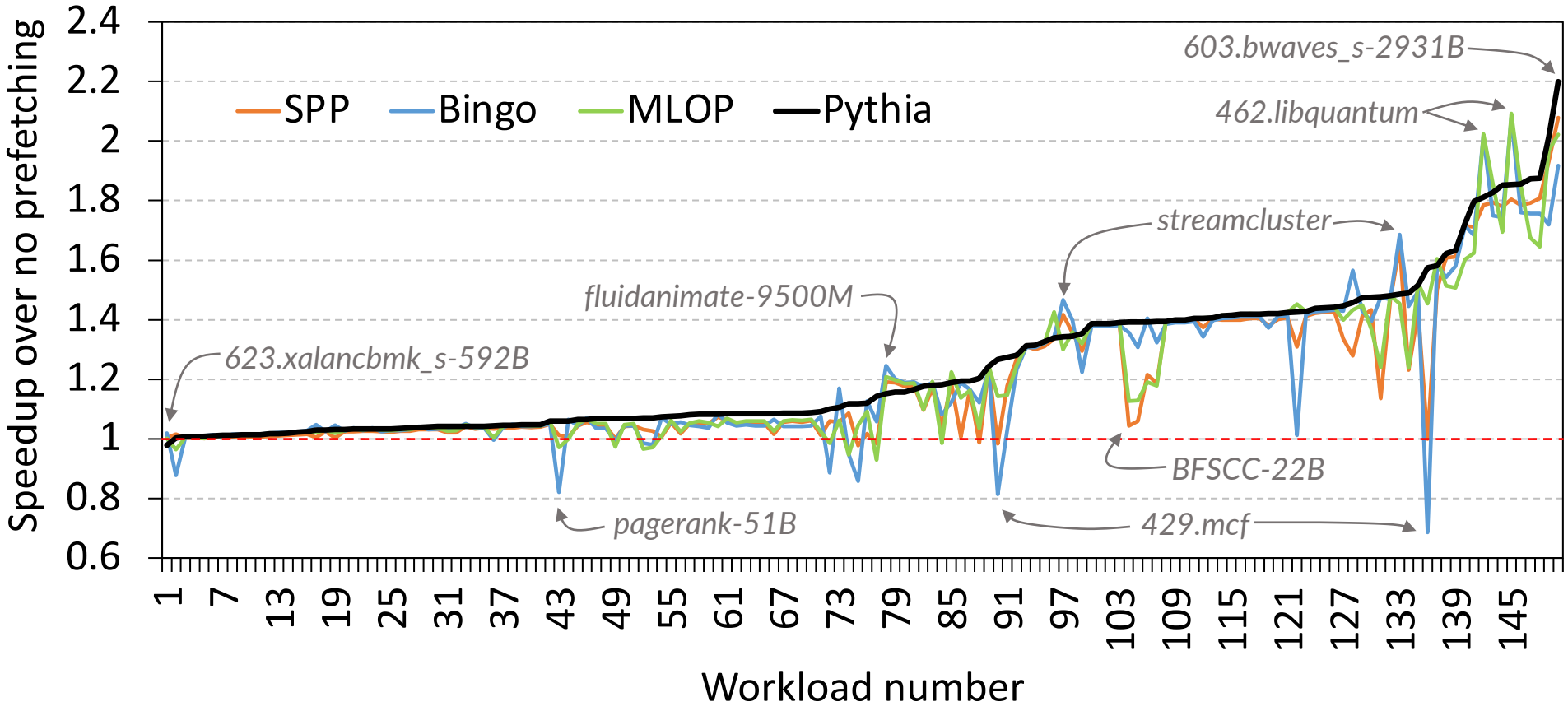
List of Evaluated Features

Table 3: List of program control-flow and data-flow components used to derive the list of features for exploration

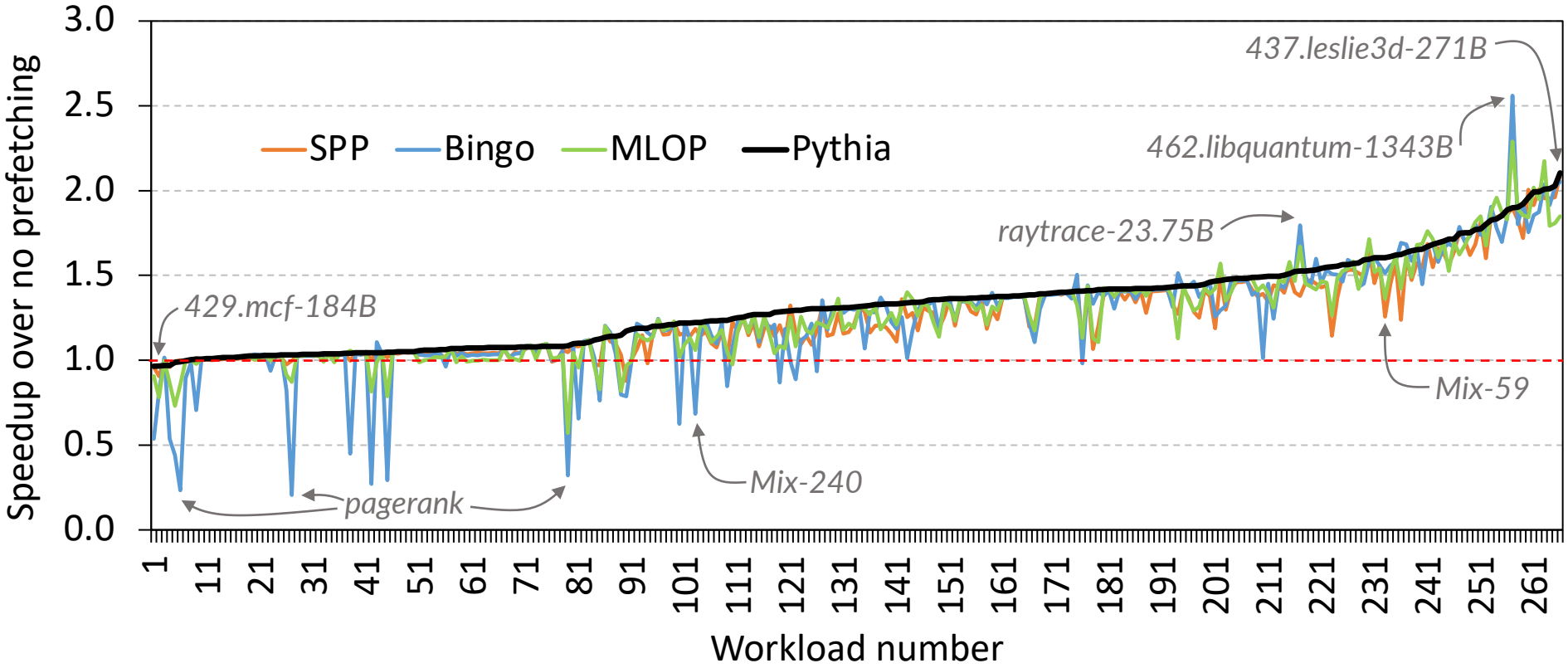
Control-flow Component	Data-flow Component
(1) PC of load request (2) PC-path (XOR-ed last-3 PCs) (3) PC XOR-ed branch-PC (4) None	(1) Load cacheline address (2) Page number (3) Page offset (4) Load address delta (5) Sequence of last-4 offsets (6) Sequence of last-4 deltas (7) Offset XOR-ed with delta (8) None

MORE RESULTS

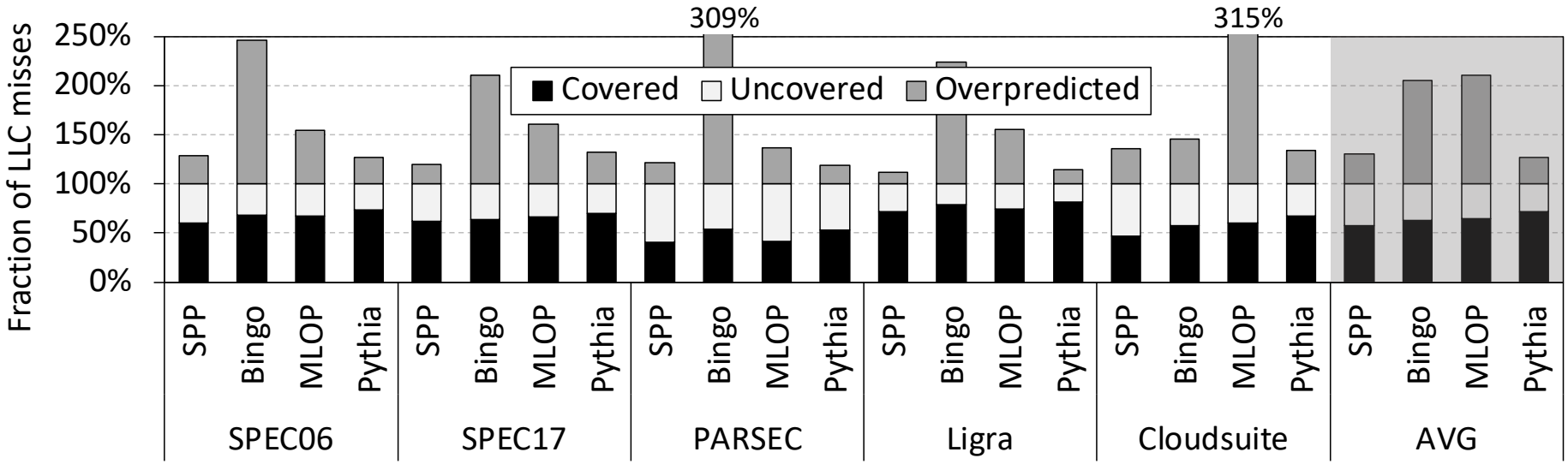
Performance S-curve: Single-core



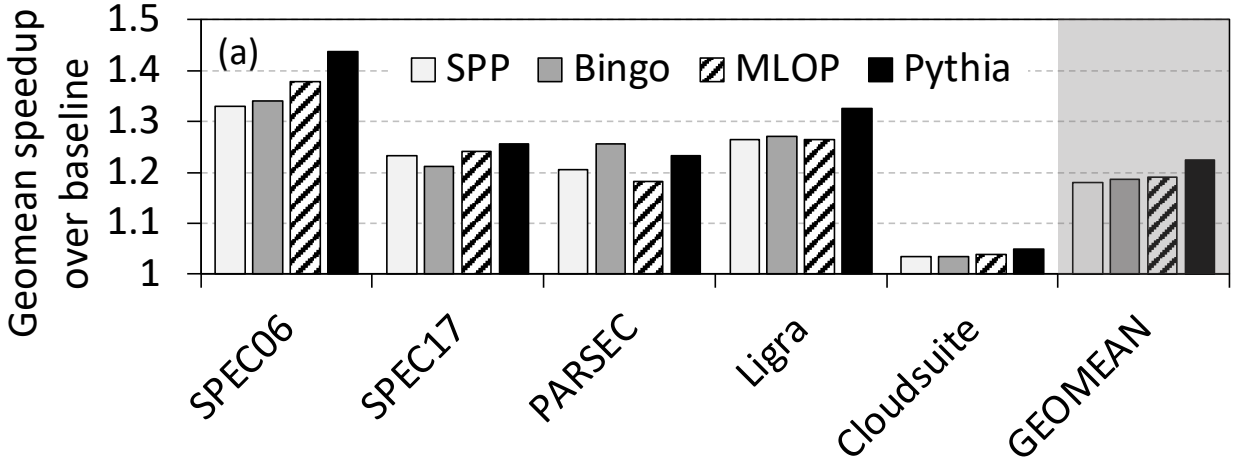
Performance S-curve: Four-core

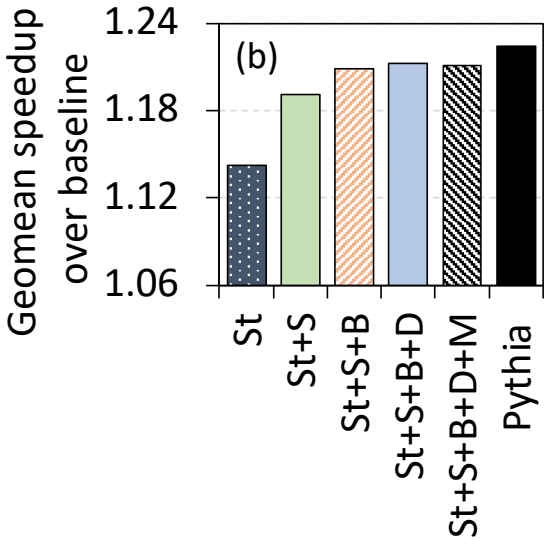


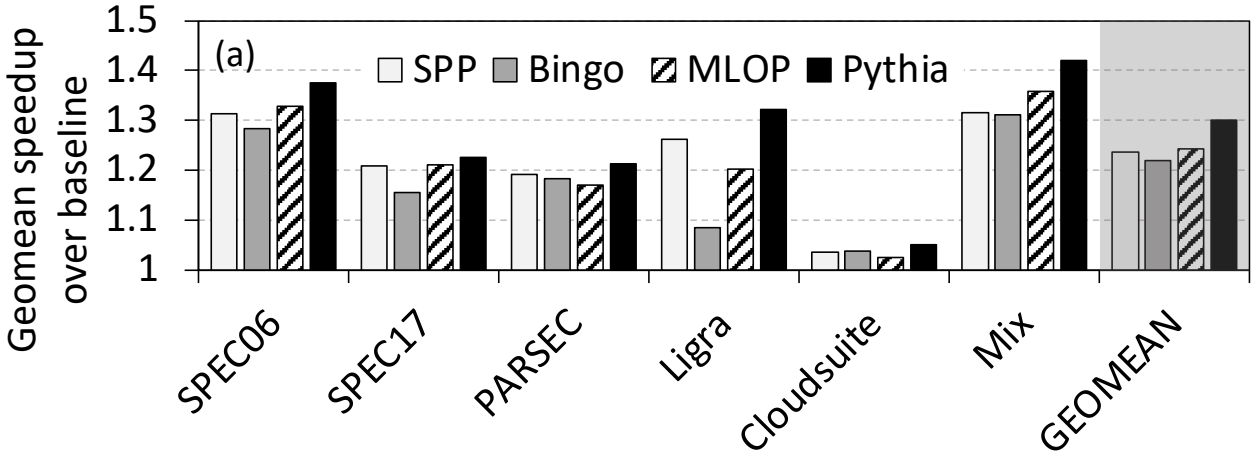
Single-core Coverage & Overprediction

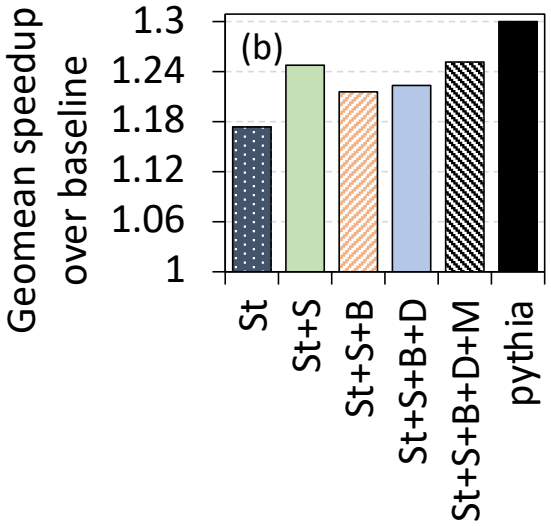


Detailed Performance

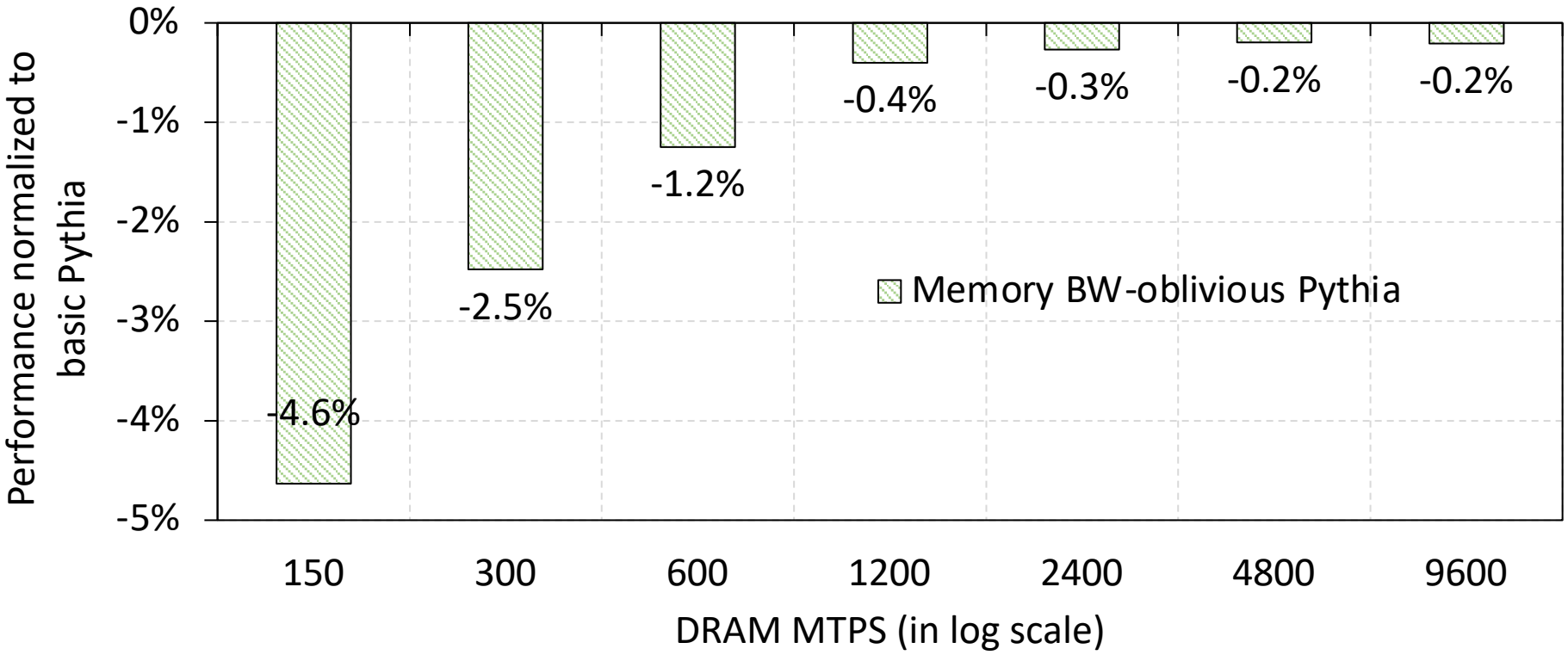








Benefit of Bandwidth Awareness



Case Study

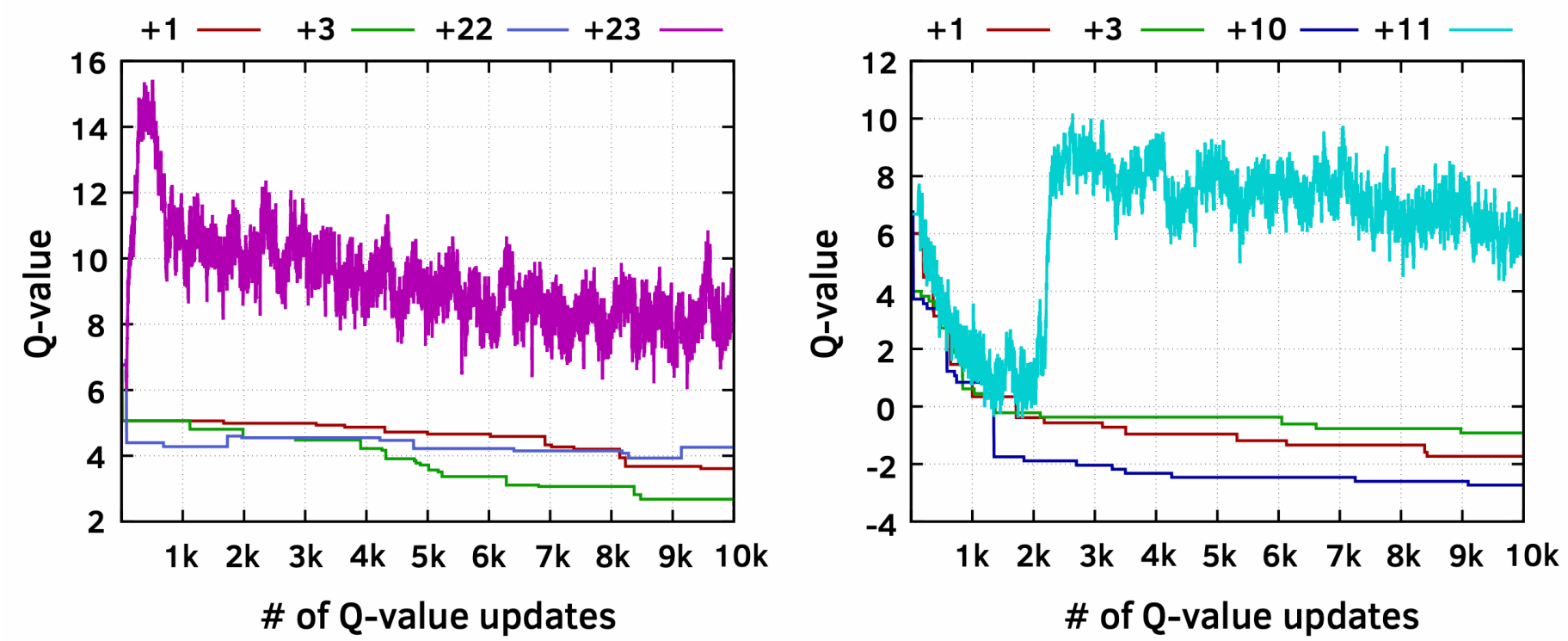


Figure 13: Q-value curves of PC+Delta feature values (a) 0x436a81+0 and (b) 0x4377c5+0 in 459.GemsFDTD-1320B.

Customizing Rewards

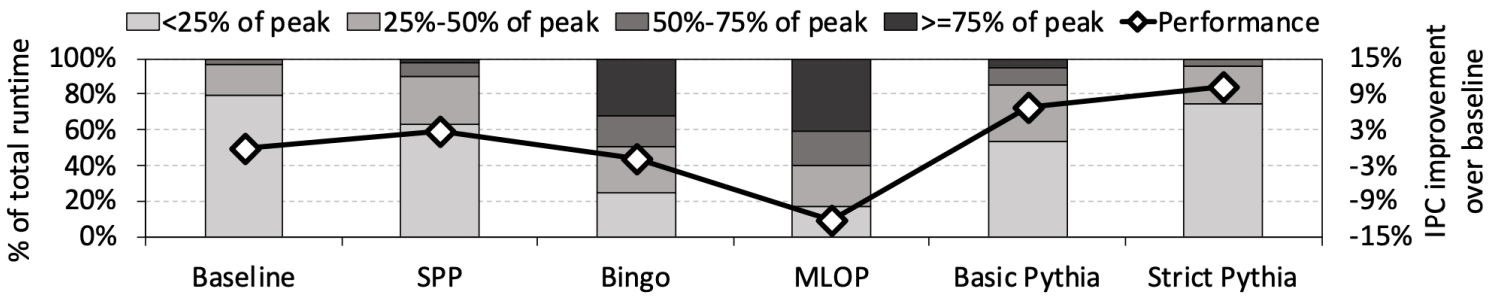


Figure 14: Performance and main memory bandwidth usage of prefetchers in Ligra-CC.

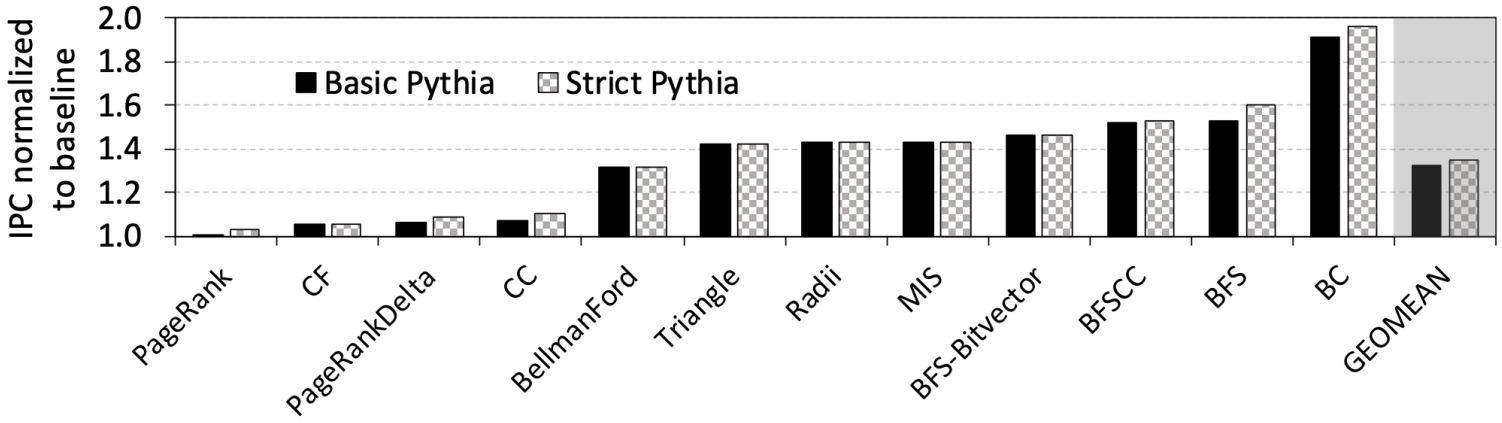


Figure 15: Performance of the basic and strict Pythia configurations on the Ligra workload suite.

Customizing Features

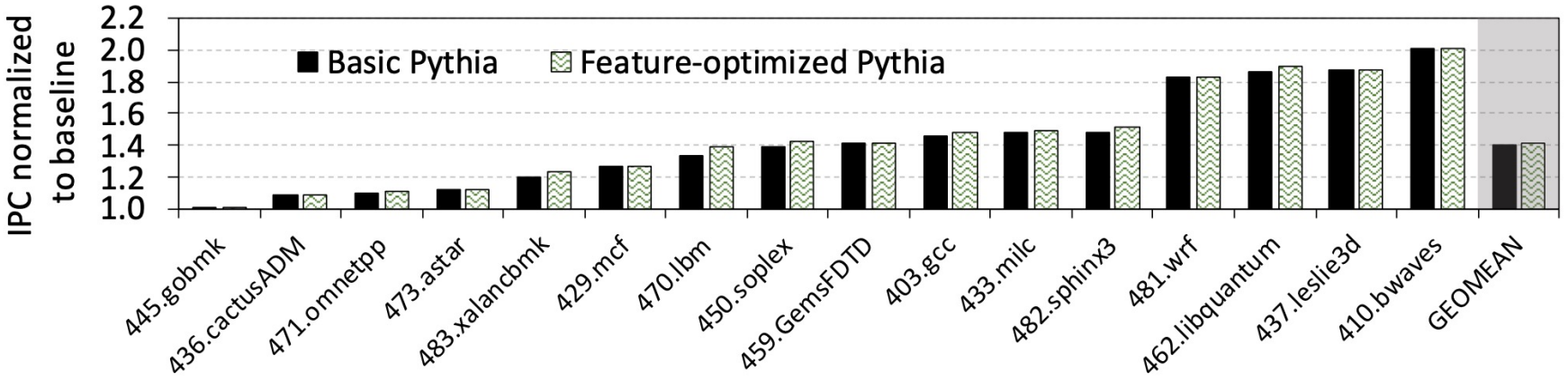


Figure 16: Performance of the basic and feature-optimized Pythia on the SPEC CPU2006 suite.

Hermes Discussion

- **FAQs**
  - What are the selected set of program features?
  - Can you provide some intuition on why these features work?
  - What happens in case of a misprediction?
  - What's the performance headroom for off-chip prediction?
  - Do you see a variance of different features in final prediction accuracy?
- **Simulation Methodology**
  - System parameters
  - Evaluated workloads
- **More Results**
  - Percentage of off-chip requests
  - Reduction in stall cycles by reducing the critical path
  - Fraction of off-chip load requests
  - Accuracy and coverage of POPET
  - Effect of different features
  - Are all features required?
  - 1C performance
  - 1C performance line graph
  - 1C performance against prior predictors
  - Effect on stall cycles
  - 8C performance
  - Sensitivity:
    - Hermes request issue latency
    - Cache hierarchy access latency
    - Activation threshold
    - ROB size
    - LLC size
  - Power overhead
  - Accuracy without prefetcher
  - Main memory request overhead with different prefetchers

HERMES BACKUP

Initial Set of Program Features

Features without control-flow information	Features with control-flow information
1. Load virtual address	8. Load PC
2. Virtual page number	9. PC $\oplus$ load virtual address
3. Cacheline offset in page	10. PC $\oplus$ virtual page number
4. First access	11. PC $\oplus$ cacheline offset
5. Cacheline offset + first access	12. PC + first access
6. Byte offset in cacheline	13. PC $\oplus$ byte offset
7. Word offset in cacheline	14. PC $\oplus$ word offset
	15. Last-4 load PCs
	16. Last-4 PCs

Selected Set of Program Features

Five features

- PC $\oplus$ cacheline offset
- PC $\oplus$ byte offset
- PC + first access
- Cacheline offset + first access
- Last-4 load PCs

A **binary hint** that represents whether or not a cacheblock has been recently touched

When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

... Cacheline #42 Cacheline #43 ...

Without prefetcher

- PC + first access
- Cacheline offset + first access

With a simple stride prefetcher

- Cacheline offset + first access

What Happens in case of a Misprediction?

- Two cases of mispredictions:

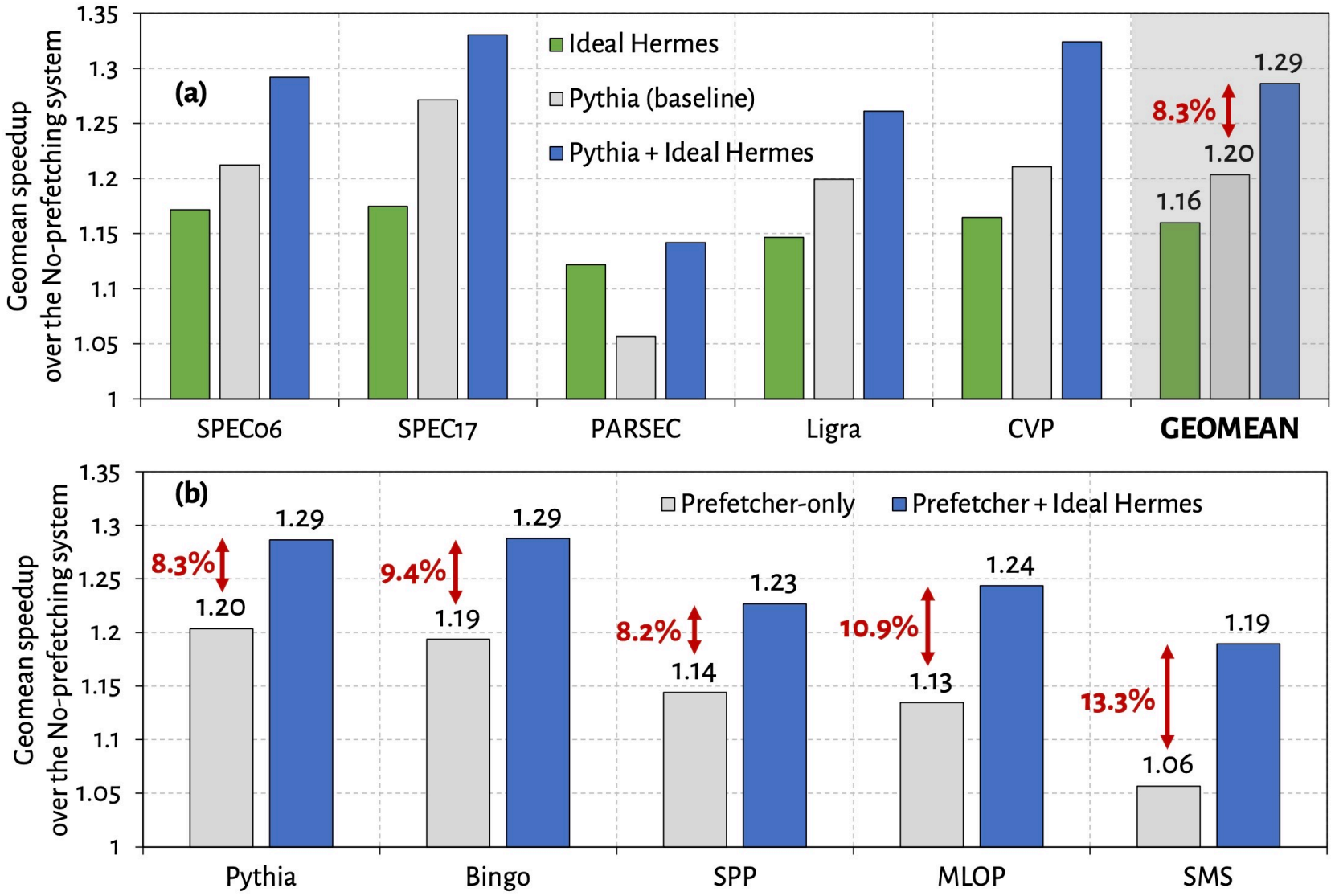
- **Predicted on-chip but actually goes off-chip**
  - Loss of performance improvement opportunity

No need for misprediction detection and recovery

- **Predicted off-chip but actually is on-chip**
  - Memory controller forwards the data to LLC **if and only if** a load to the same address have already missed LLC and arrived at the memory controller

No need for misprediction detection and recovery

Performance Headroom of Off-Chip Prediction



System Parameters

Table 4: Simulated system parameters

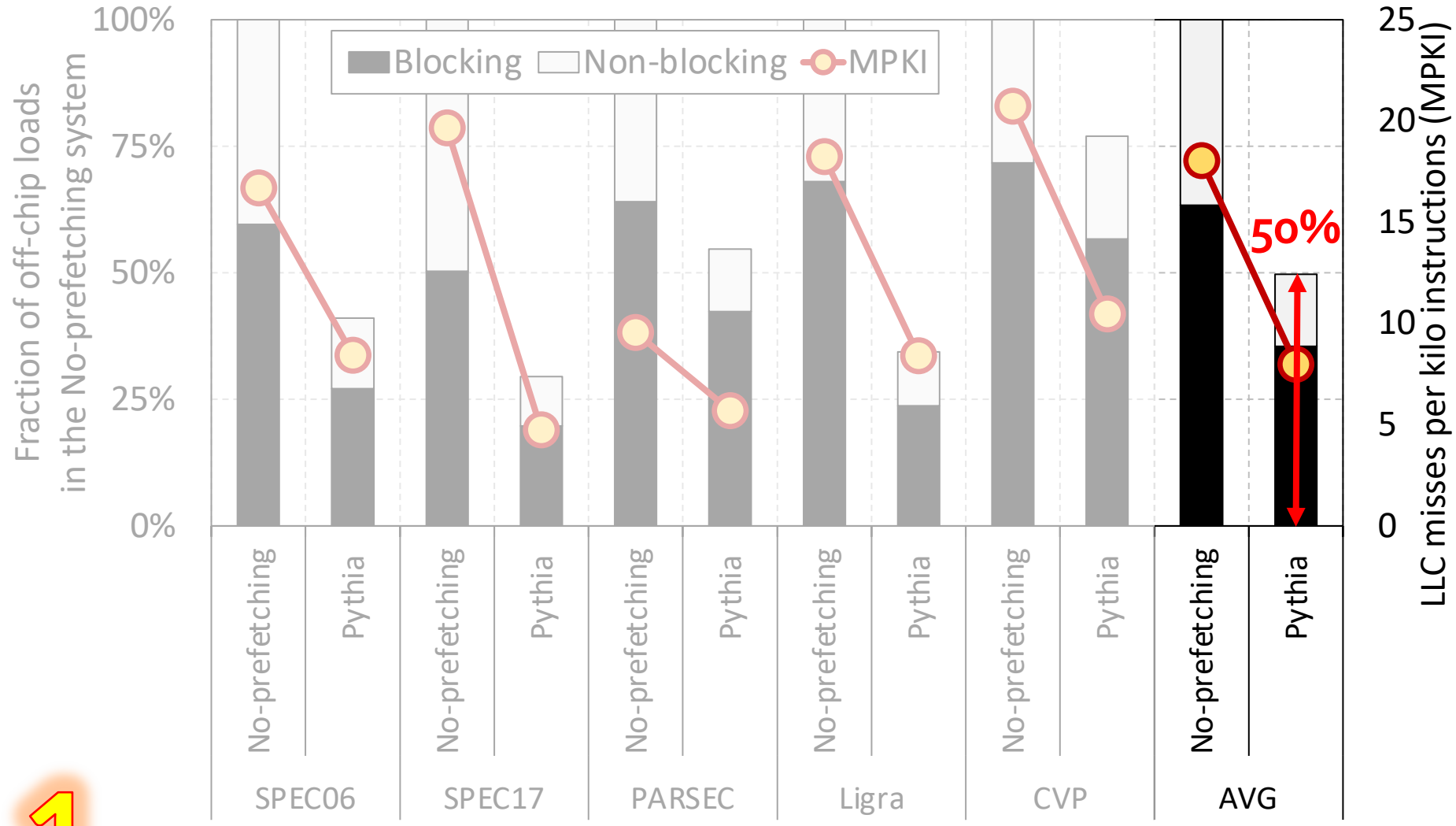
Core	1 and 8 cores, 6-wide fetch/execute/commit, 512-entry ROB, 128/72-entry LQ/SQ, Perceptron branch predictor [61] with 17-cycle misprediction penalty
L1/L2 Caches	Private, 48KB/1.25MB, 64B line, 12/20-way, 16/48 MSHRs, LRU, 5/15-cycle round-trip latency [25]
LLC	3MB/core, 64B line, 12 way, 64 MSHRs/slice, SHiP [122], 55-cycle round-trip latency [24, 25], **Pythia** prefetcher [32]
Main Memory	**1C:** 1 channel, 1 rank per channel; **8C:** 4 channels, 2 ranks per channel; 8 banks per rank, DDR4-3200 MTPS, 64b data-bus per channel, 2KB row buffer per bank, tRCD=12.5ns, tRP=12.5ns, tCAS=12.5ns
Hermes	**Hermes-O/P**: 6/18-cycle Hermes request issue latency

Evaluated Workloads

Table 5: Workloads used for evaluation

Suite	#Workloads	#Traces	Example Workloads
SPEC06	14	22	gcc, mcf, cactusADM, lbm, ...
SPEC17	11	23	gcc, mcf, pop2, fotonik3d, ...
PARSEC	4	12	canneal, facesim, raytrace, ...
Ligra	11	20	BFS, PageRank, Radii, ...
CVP	33	33	integer, floating-point, server, ...

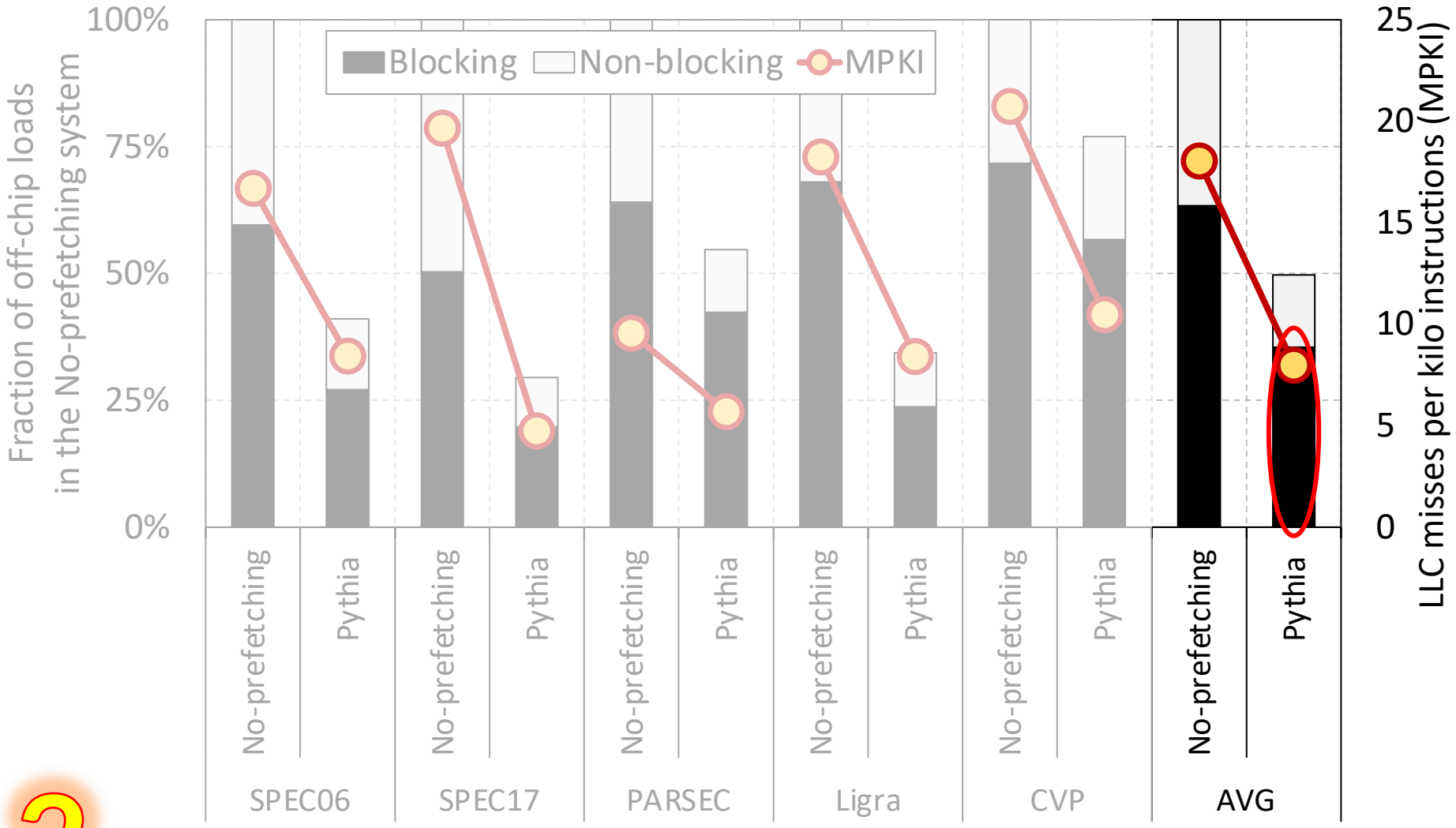
Observation: Not All Off-Chip Loads are Prefetched



1

Nearly **50%** of the loads are still **not prefetched**

Observation: Not All Off-Chip Loads are Prefetched

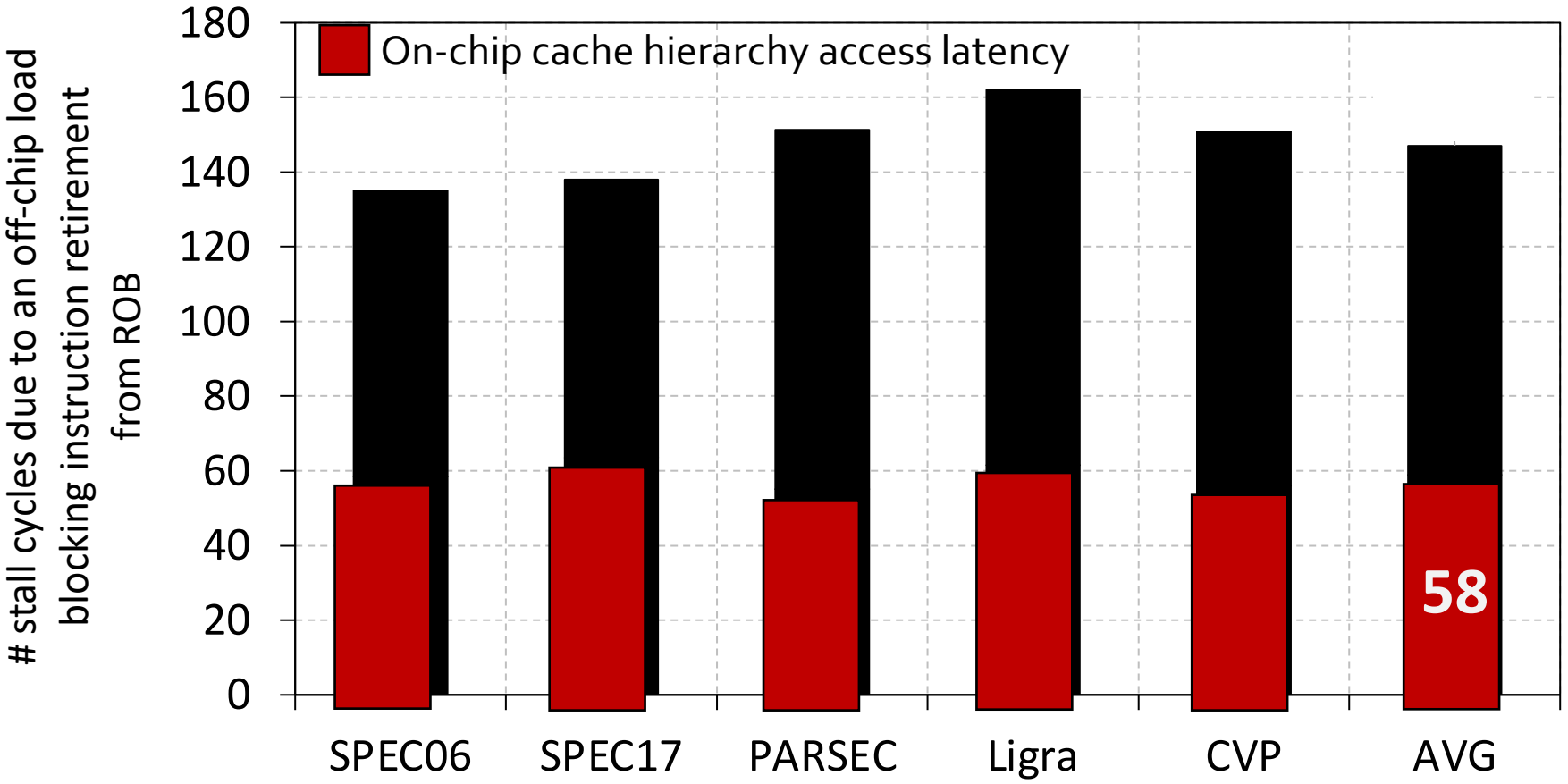


2

70% of these off-chip loads **blocks ROB**

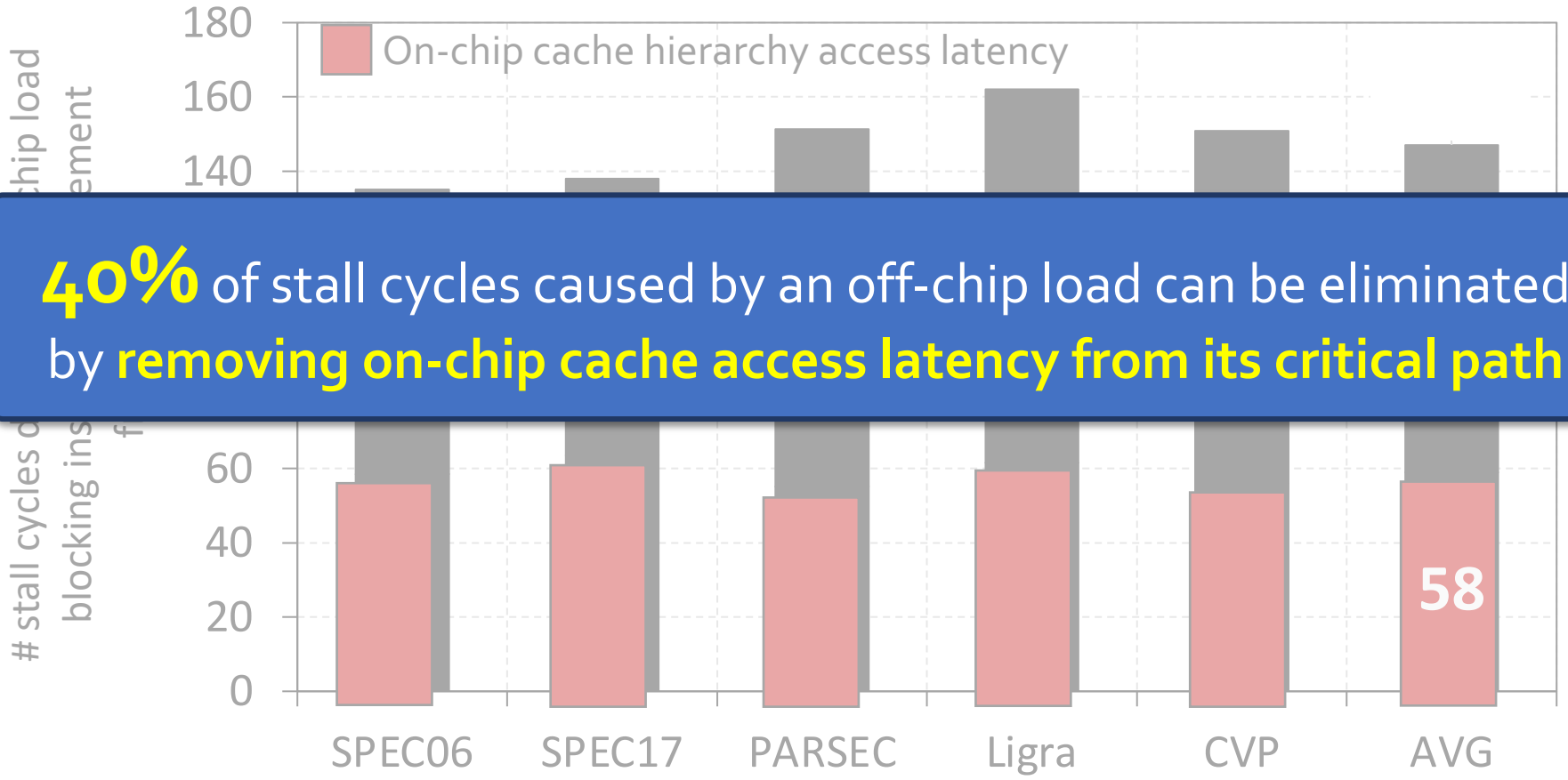
Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load



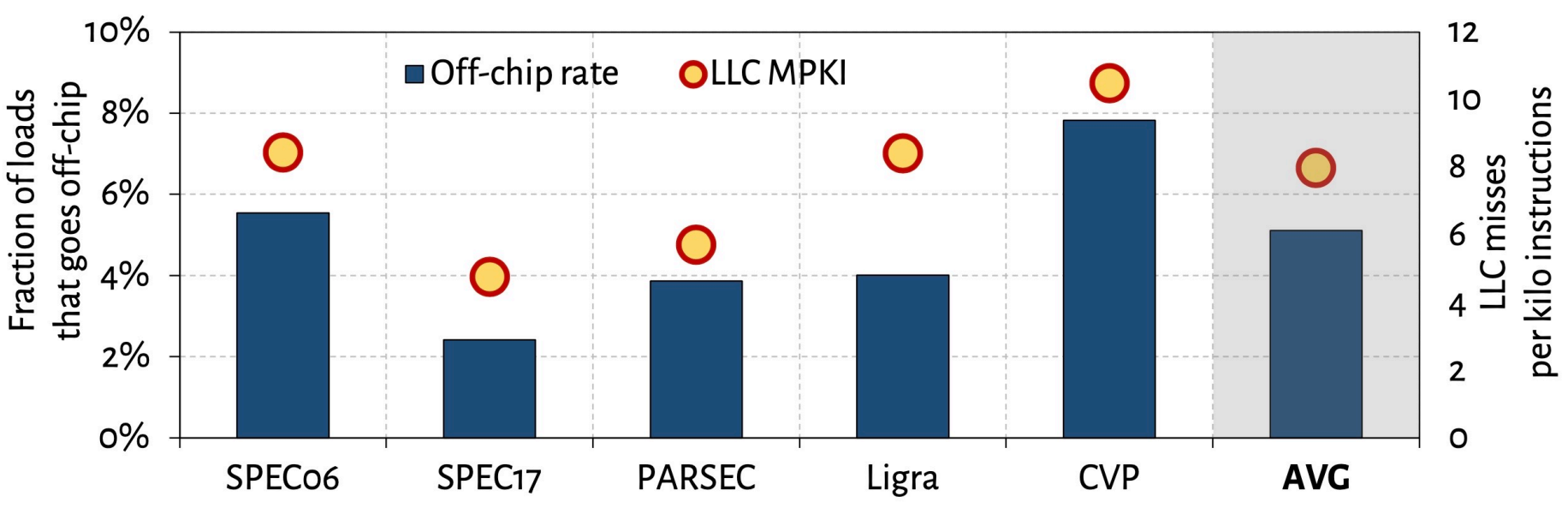
Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load

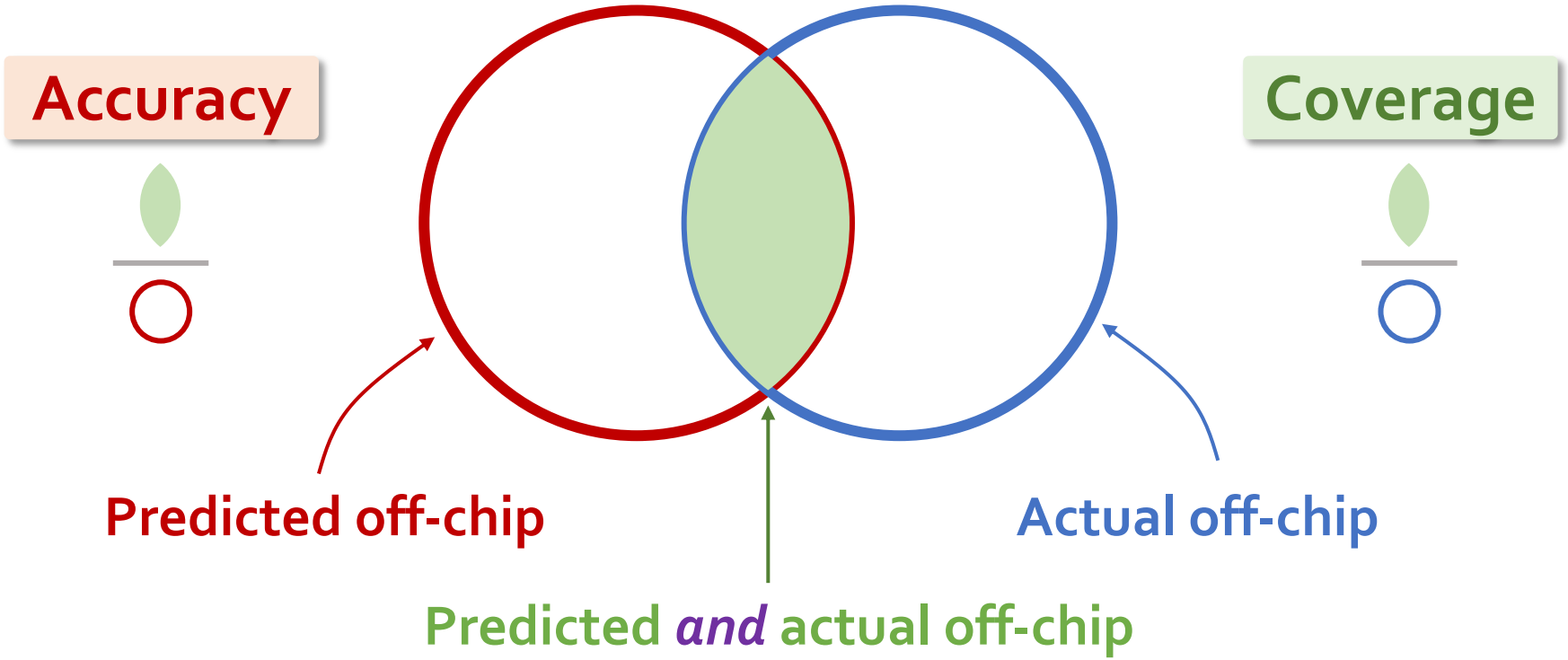


40% of stall cycles caused by an off-chip load can be eliminated by **removing on-chip cache access latency from its critical path**

What Fraction of Load Requests Goes Off-Chip?

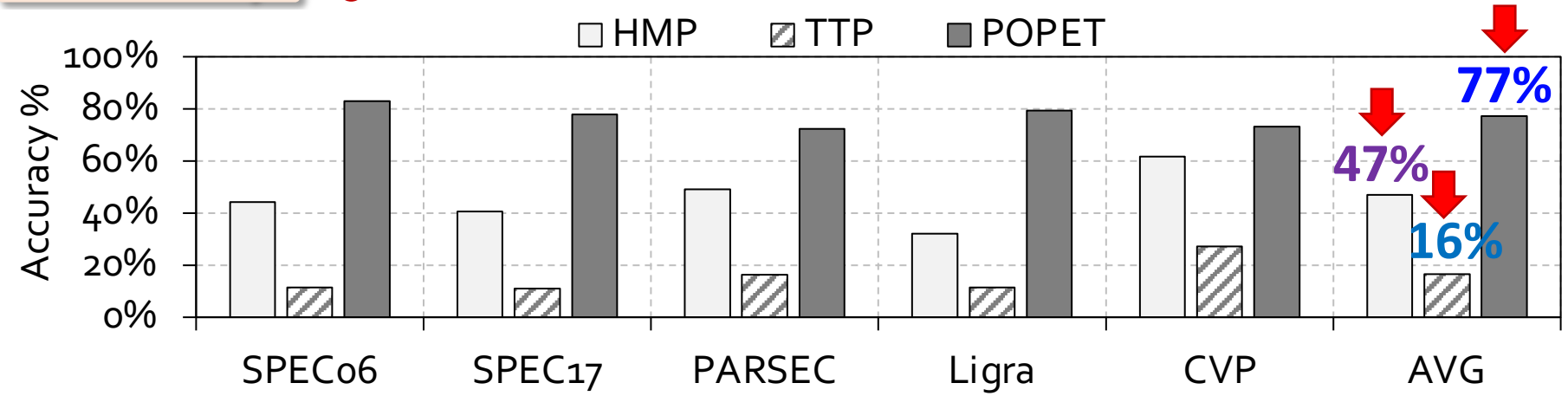


Off-Chip Prediction Quality: *Defining Metrics*

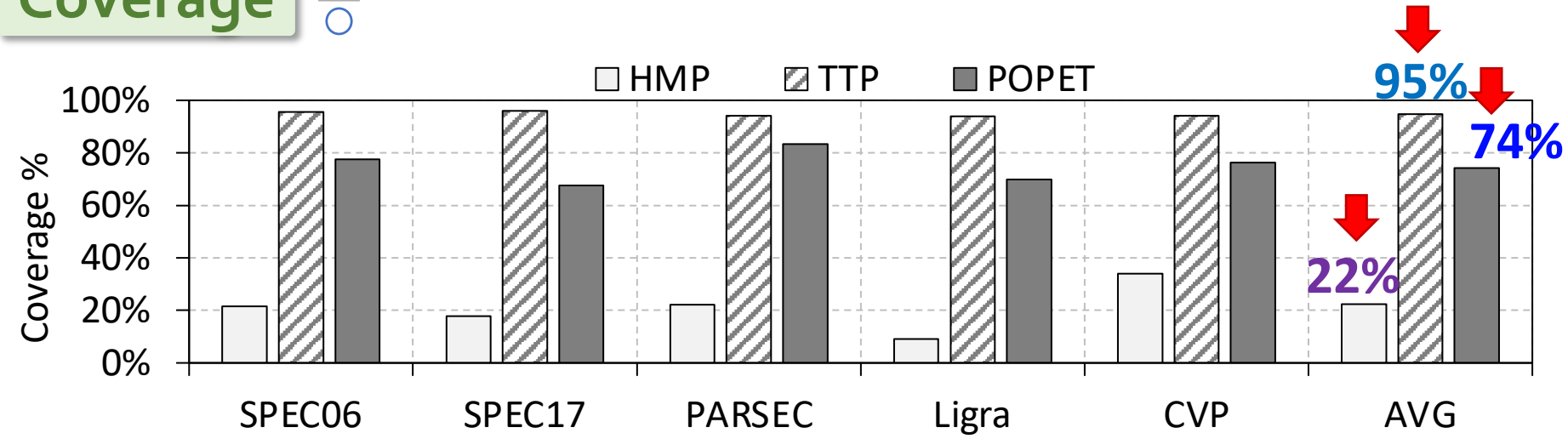


Off-Chip Prediction Quality: *Analysis*

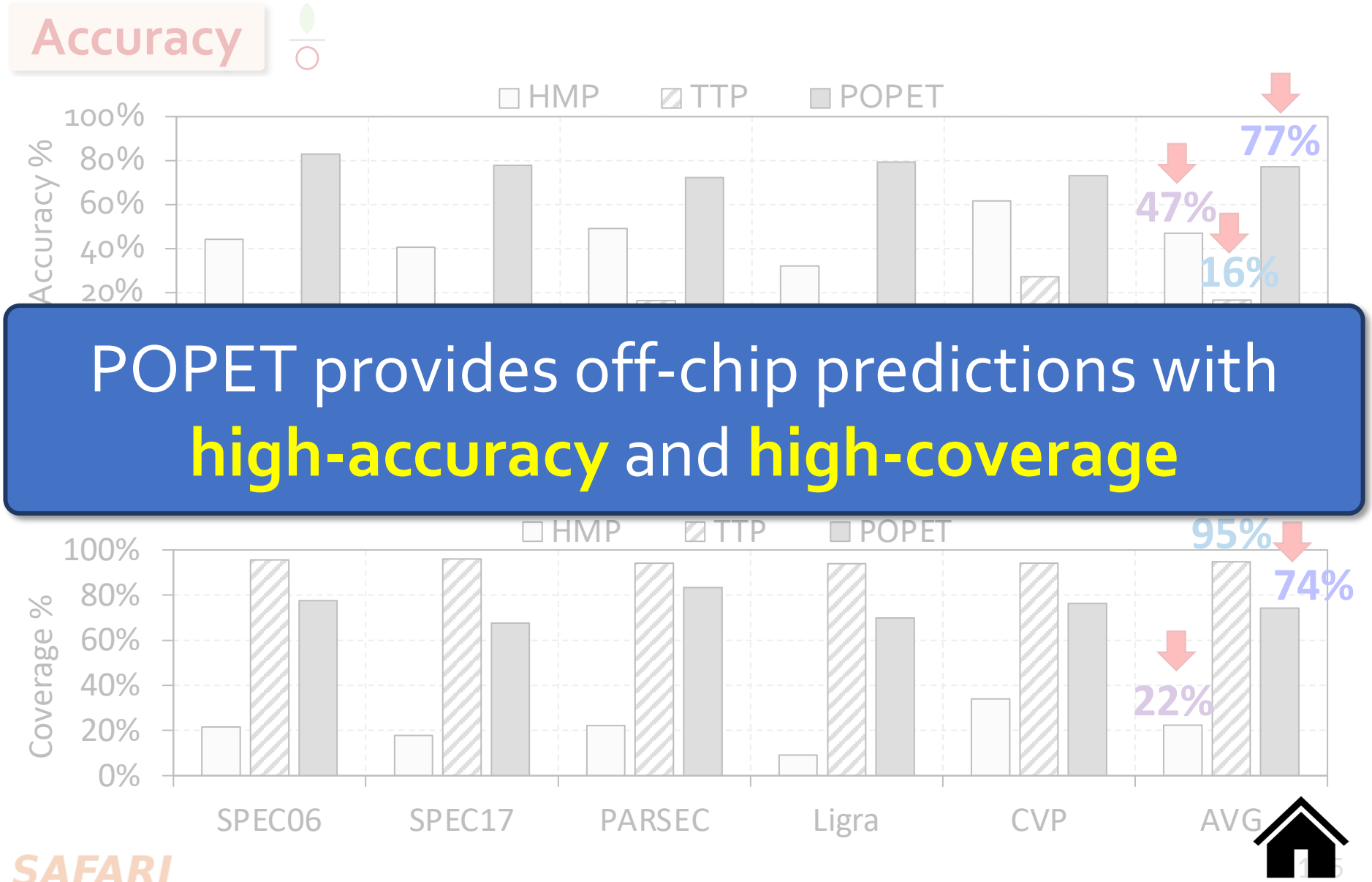
Accuracy



Coverage

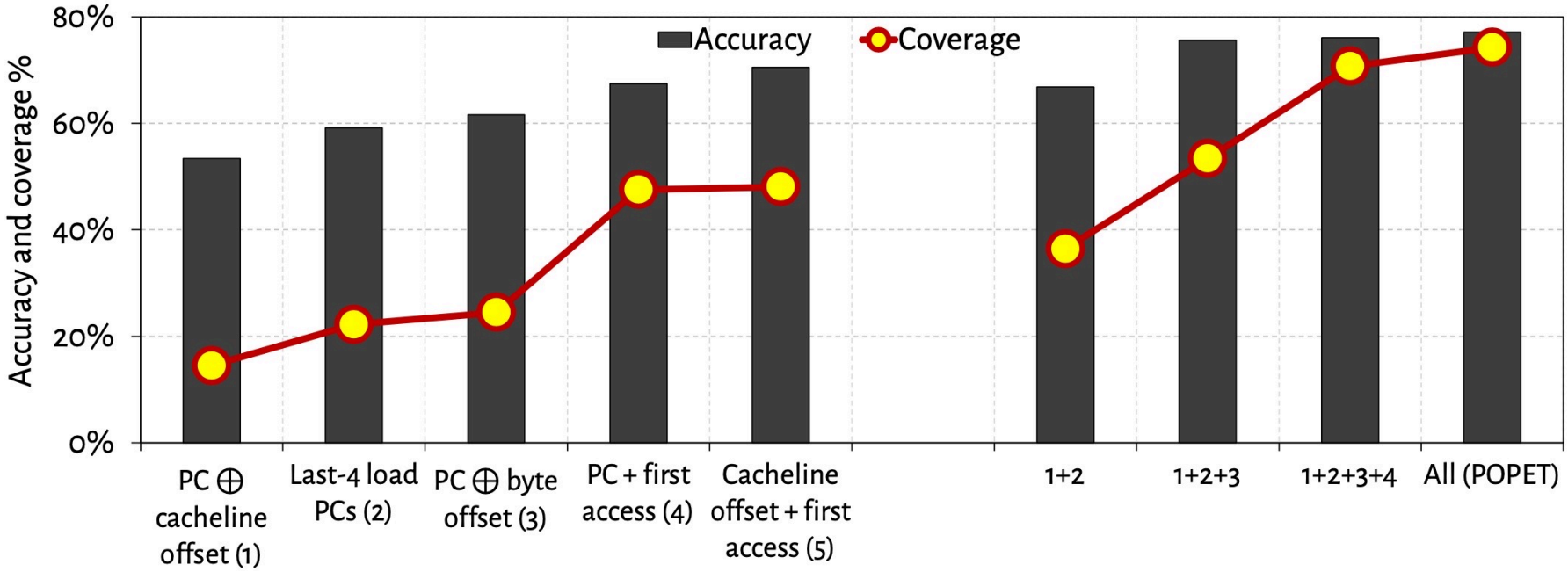


Off-Chip Prediction Quality: *Analysis*



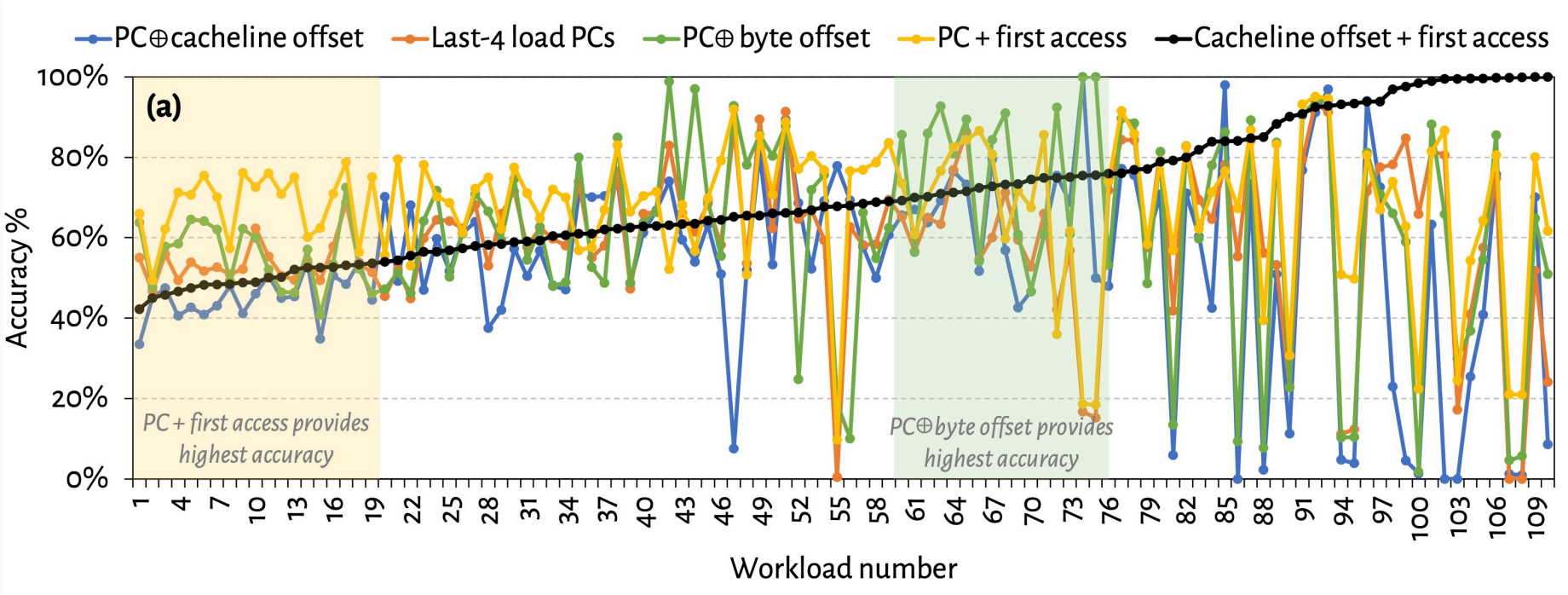
POPET provides off-chip predictions with **high-accuracy** and **high-coverage**

Effect of Different Features



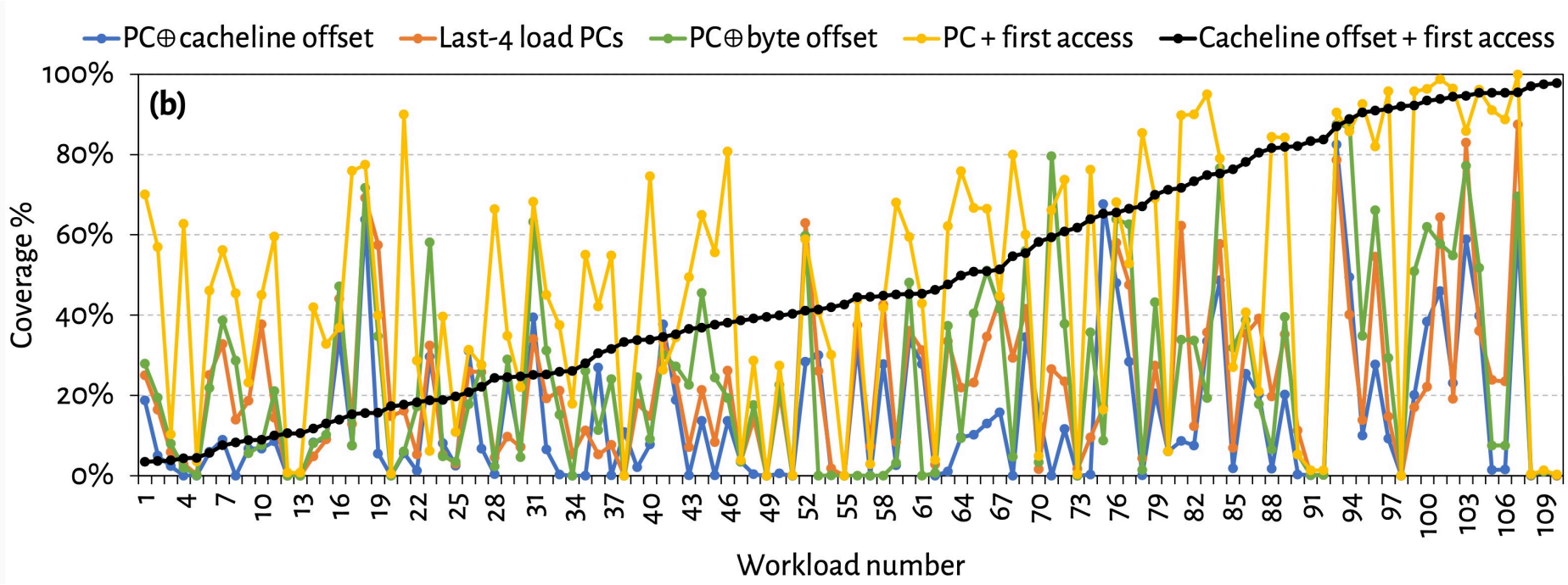
Combination of features provides both **higher accuracy and higher coverage** than any individual feature

Are All Features Required? (1)



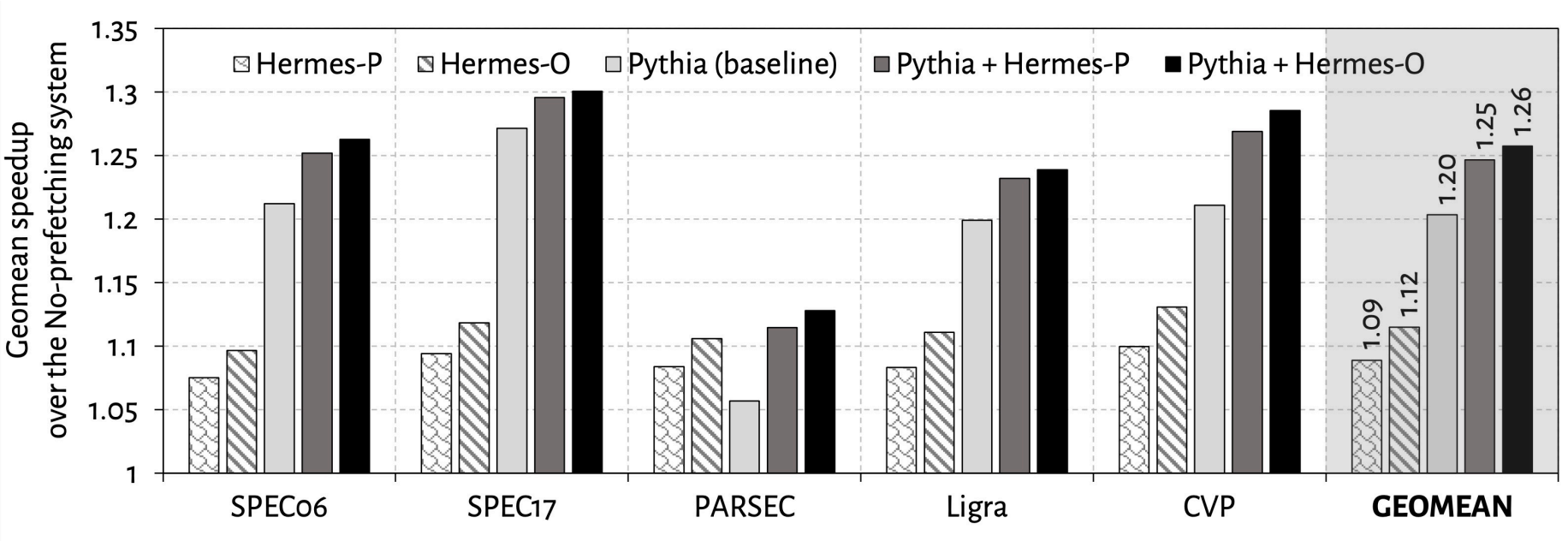
No single feature individually provides
highest prediction accuracy across ***all*** workloads

Are All Features Required? (2)



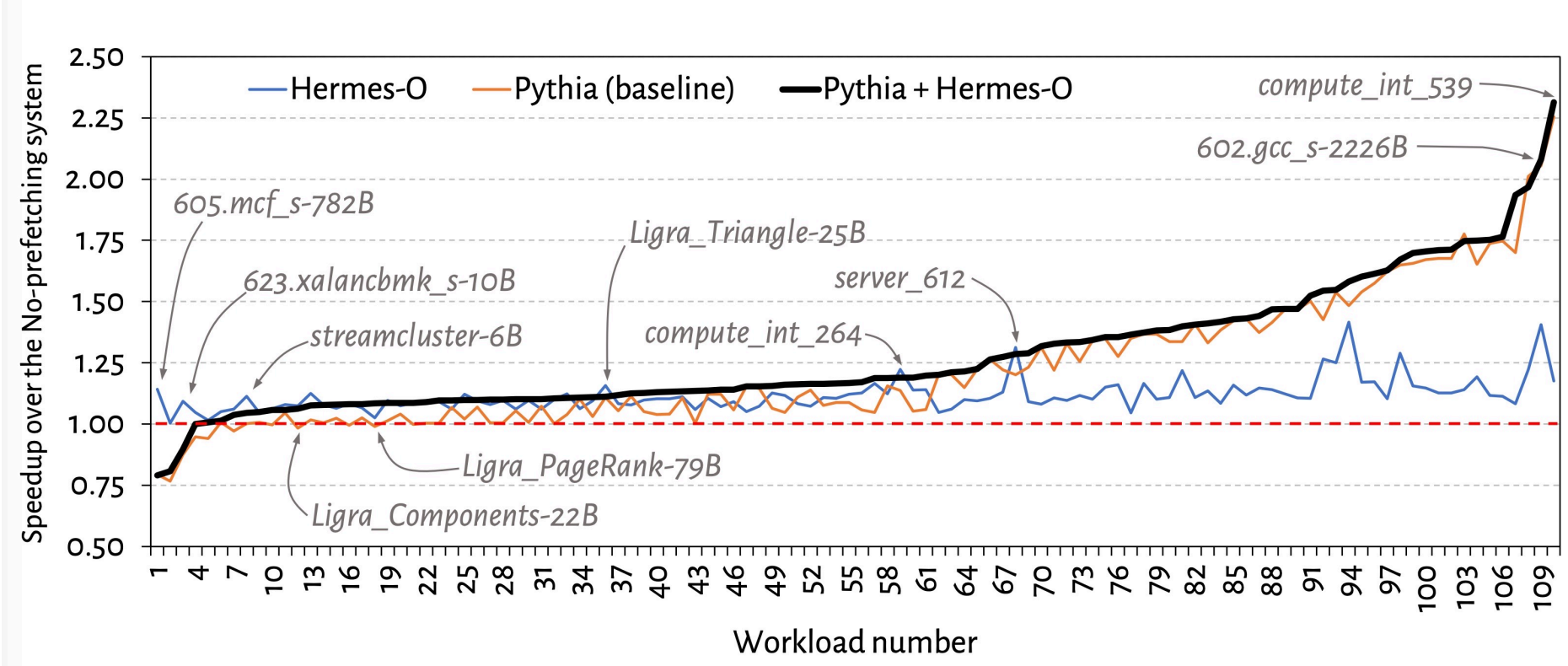
No single feature individually provides
highest prediction coverage also across ***all*** workloads

Single-Core Performance

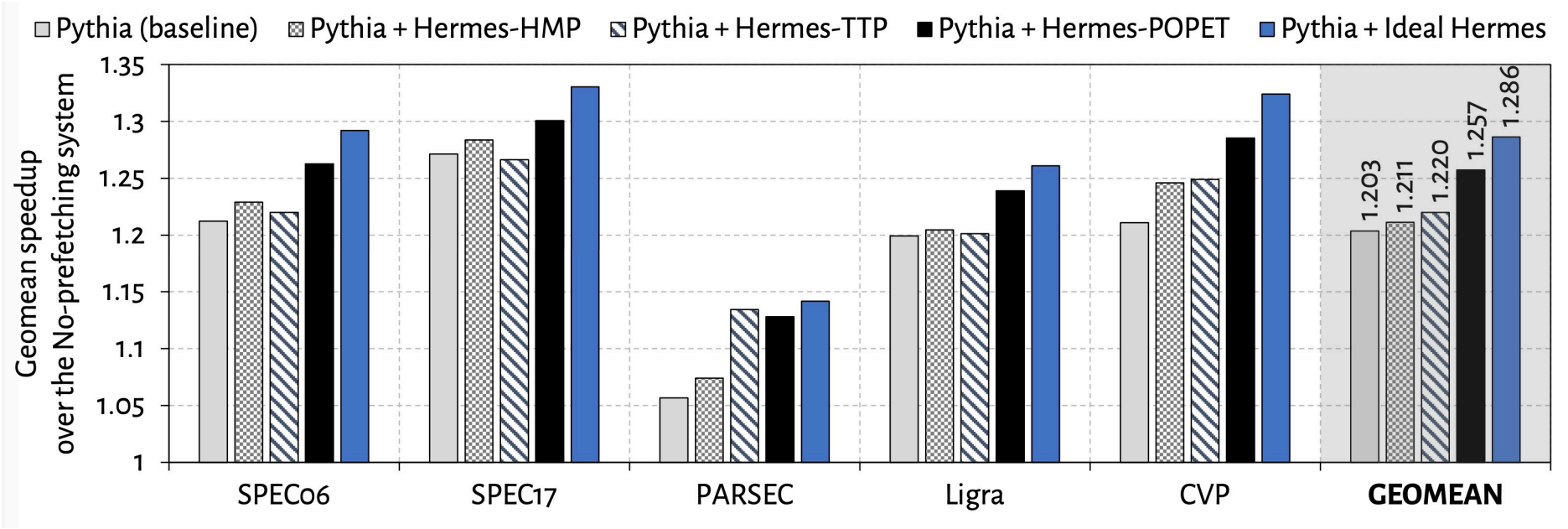


Hermes in combination with Pythia outperforms **Pythia alone** in every workload category

Single-Core Performance Line Graph



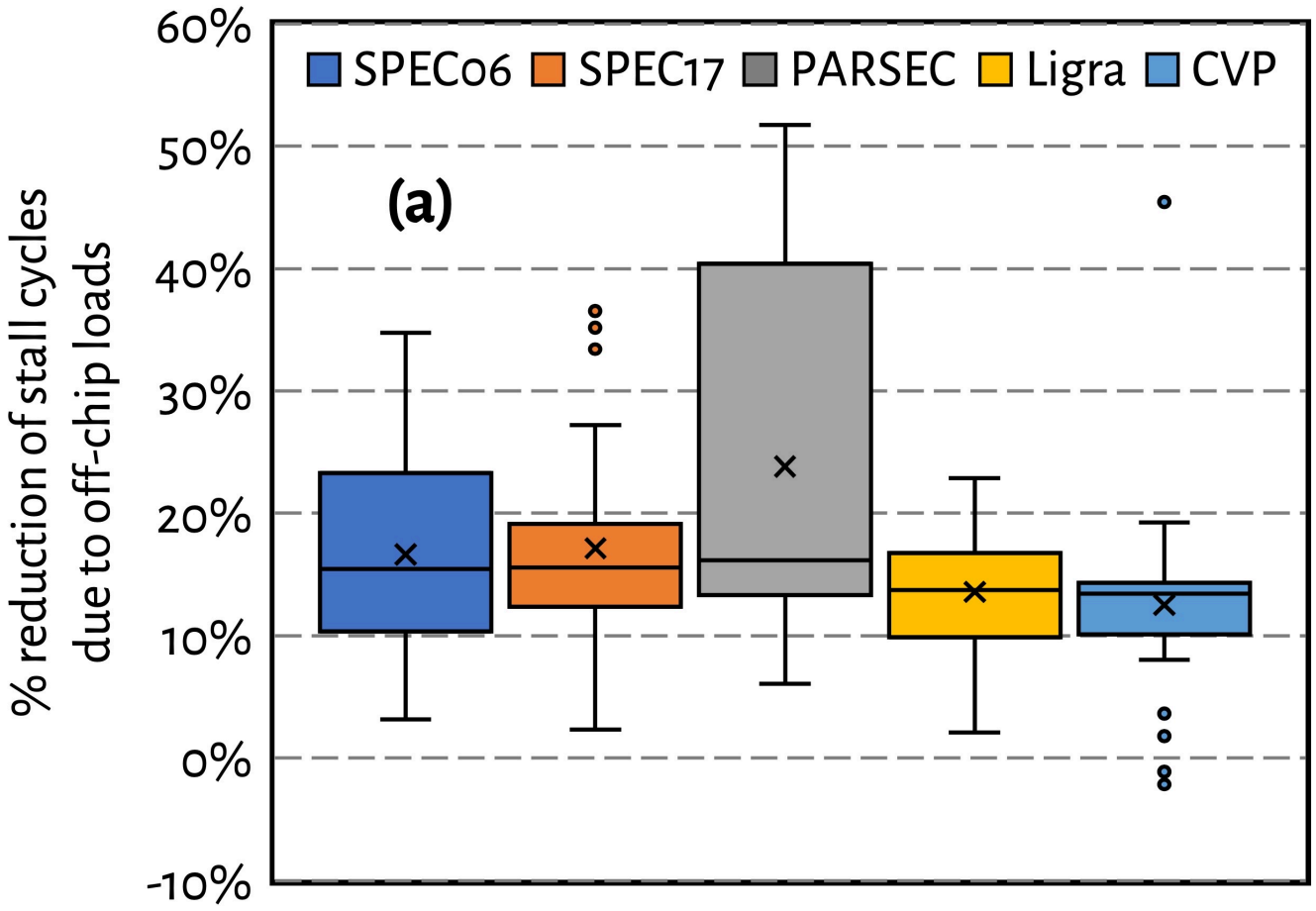
Single-Core Performance Against Prior Predictors



POPET provides higher performance benefit than prior predictors

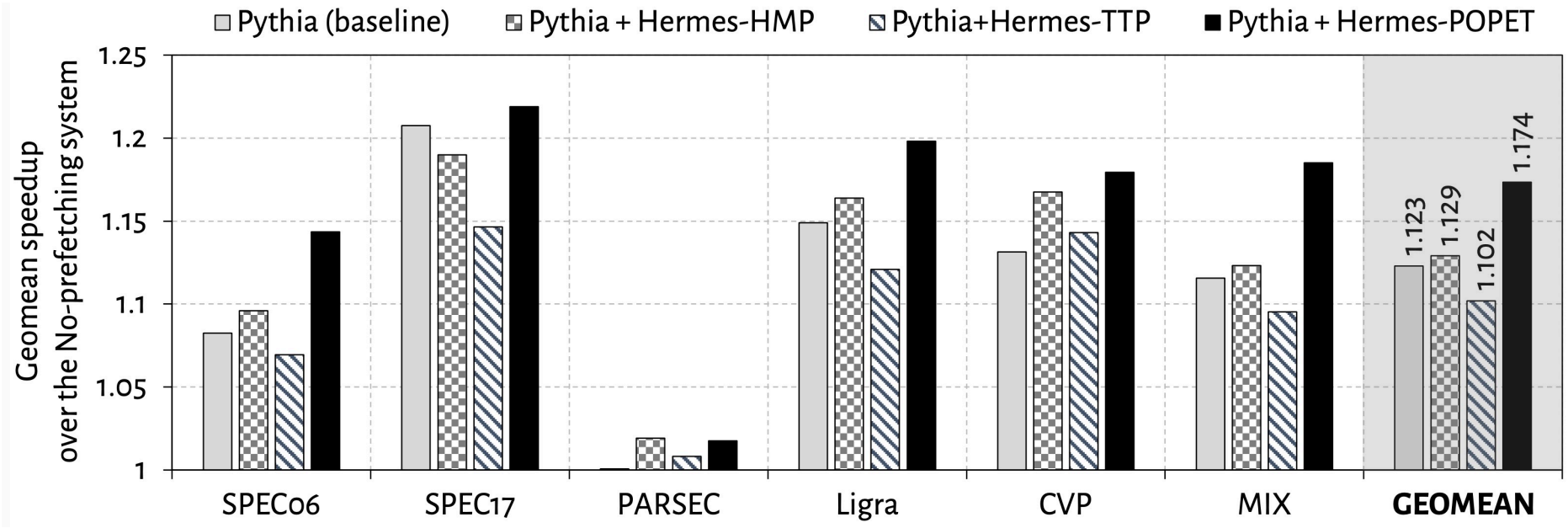
Hermes with POPET achieves nearly **90%** performance improvement of the Ideal Hermes

Effect on Stall Cycles



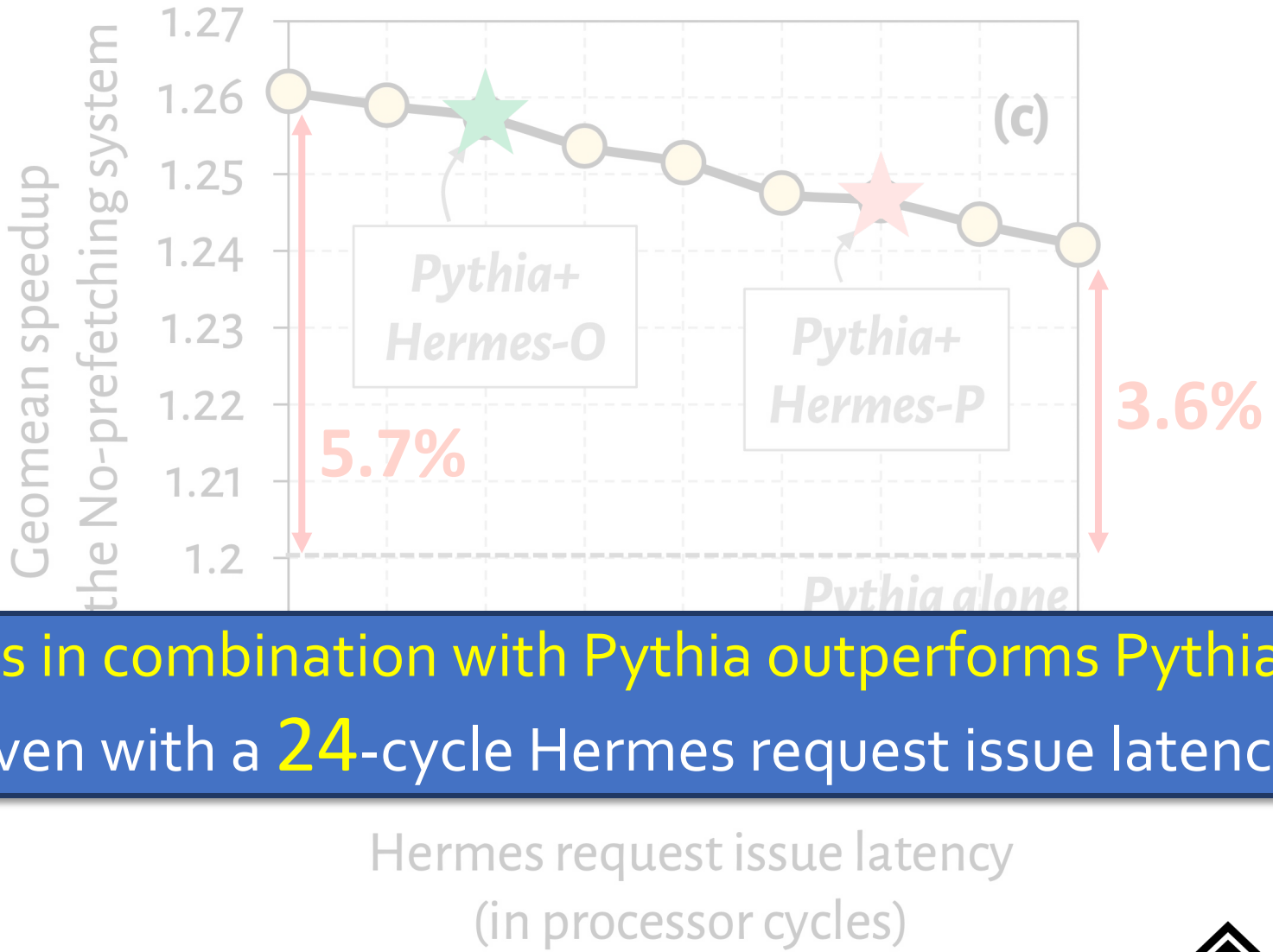
Hermes reduces off-chip load induced stall cycles on average by **16.2%** (up-to **51.8%**)

Eight-Core Performance



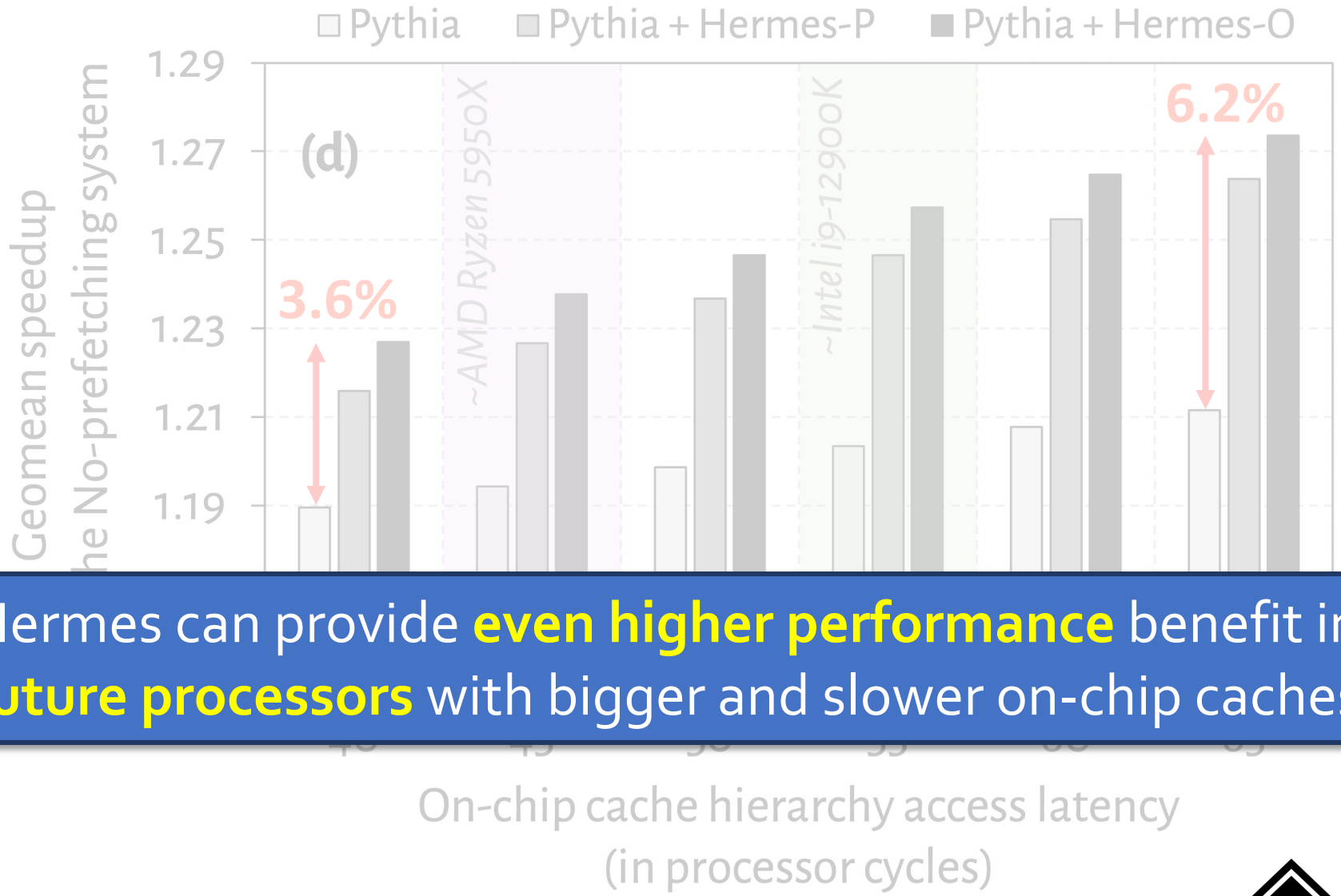
Hermes in combination with Pythia outperforms Pythia alone by **5.1%** on average

Effect of Hermes Request Issue Latency



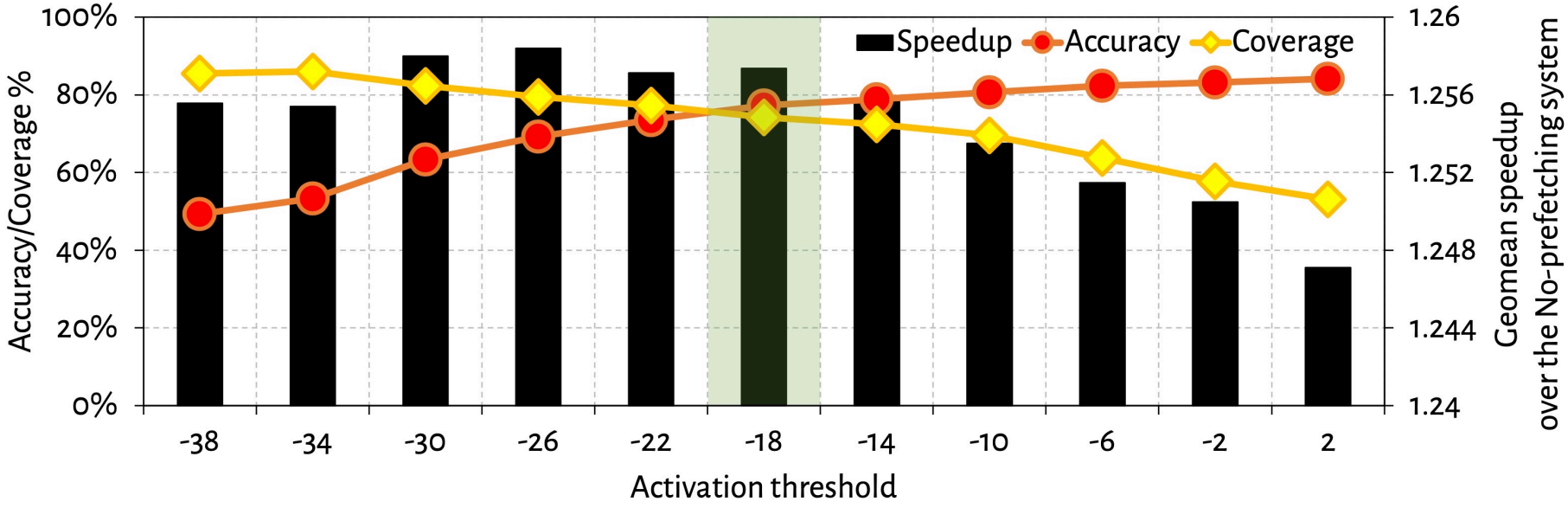
Hermes in combination with Pythia outperforms Pythia alone even with a 24-cycle Hermes request issue latency

Effect of Cache Hierarchy Access Latency



Hermes can provide **even higher performance** benefit in **future processors** with bigger and slower on-chip caches

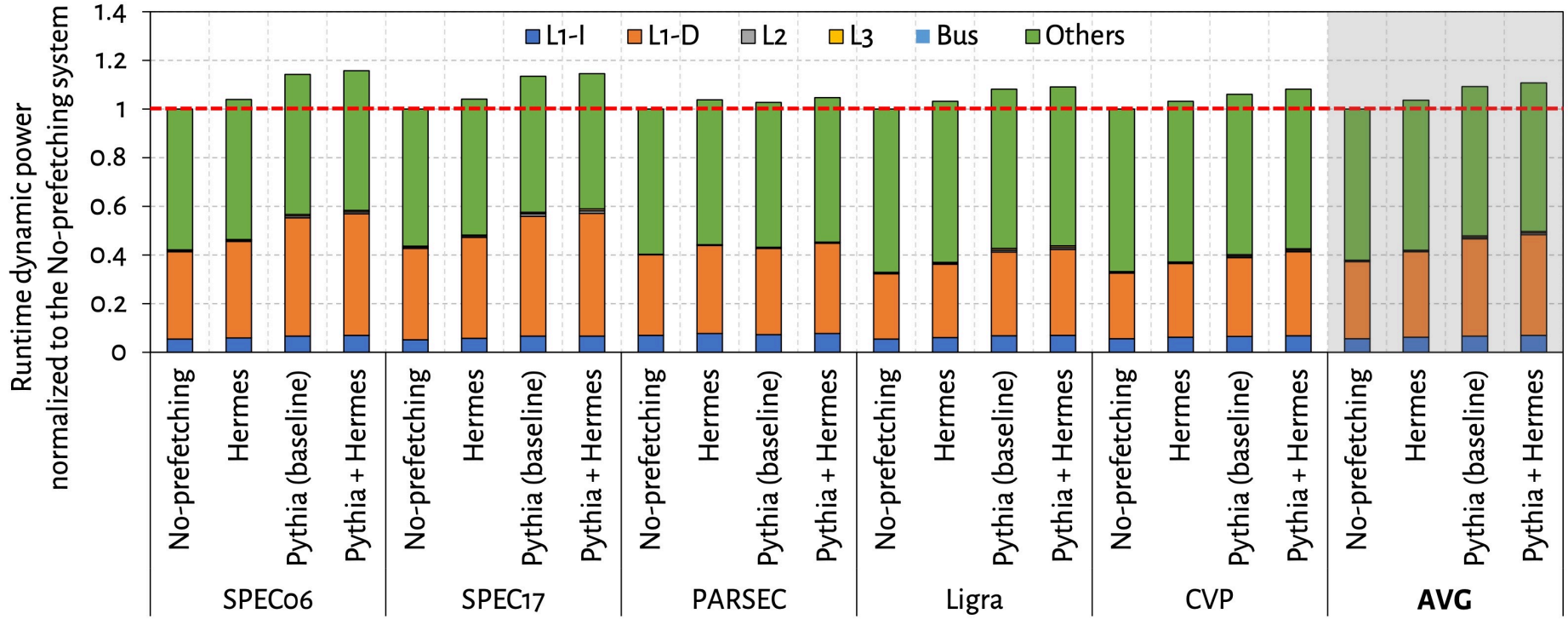
Effect of Activation Threshold



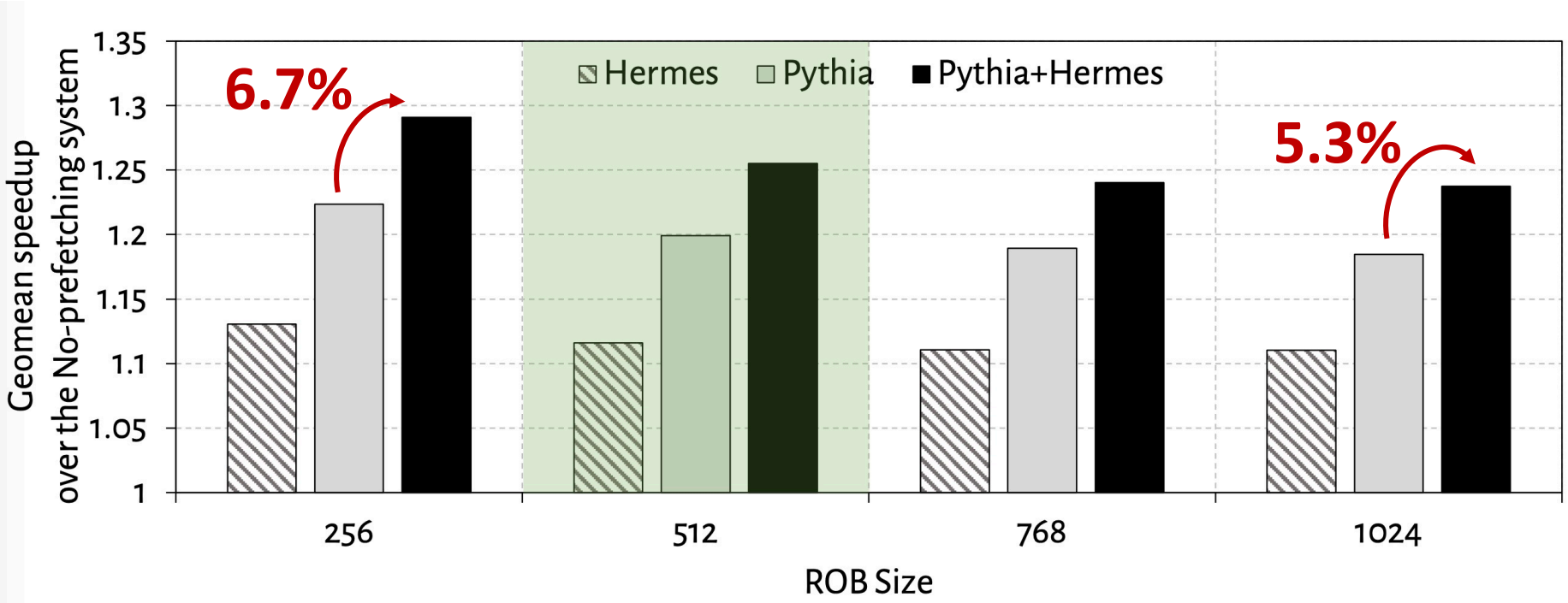
With increase in activation threshold

1. Accuracy increases
2. Coverage decreases

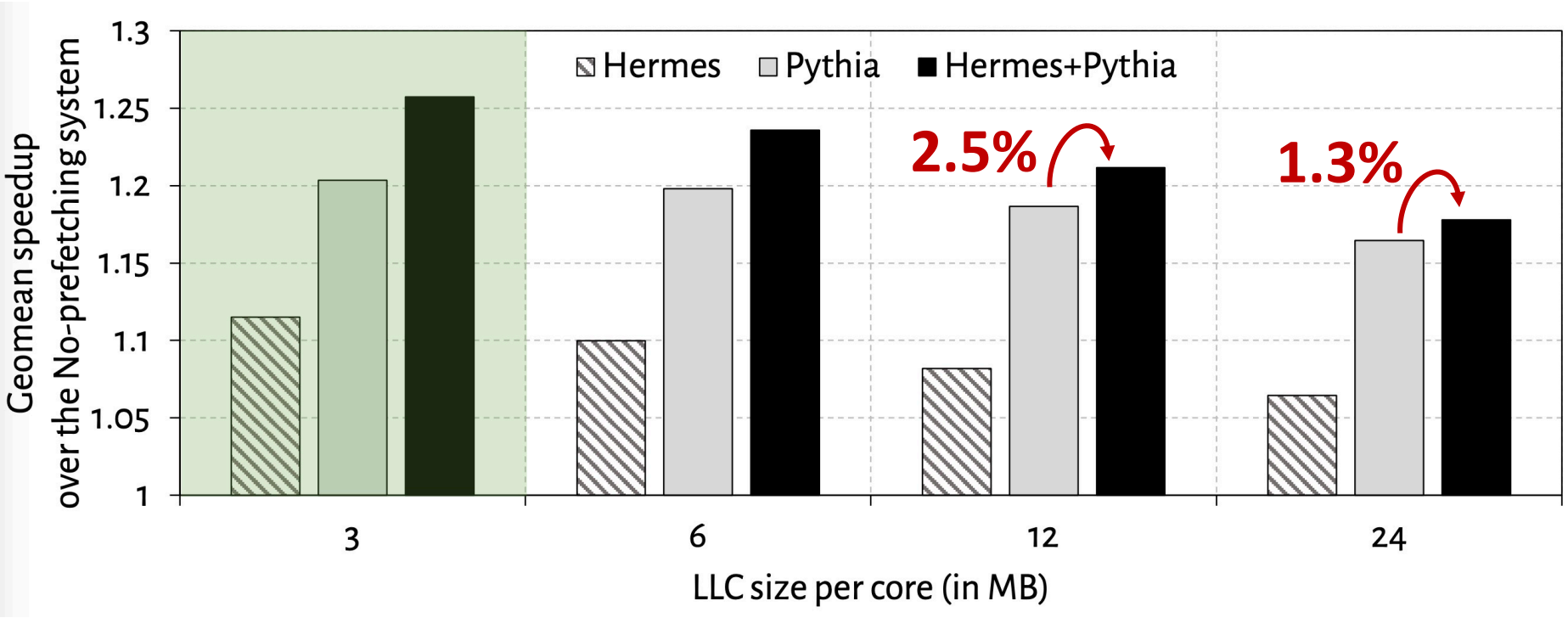
Power Overhead



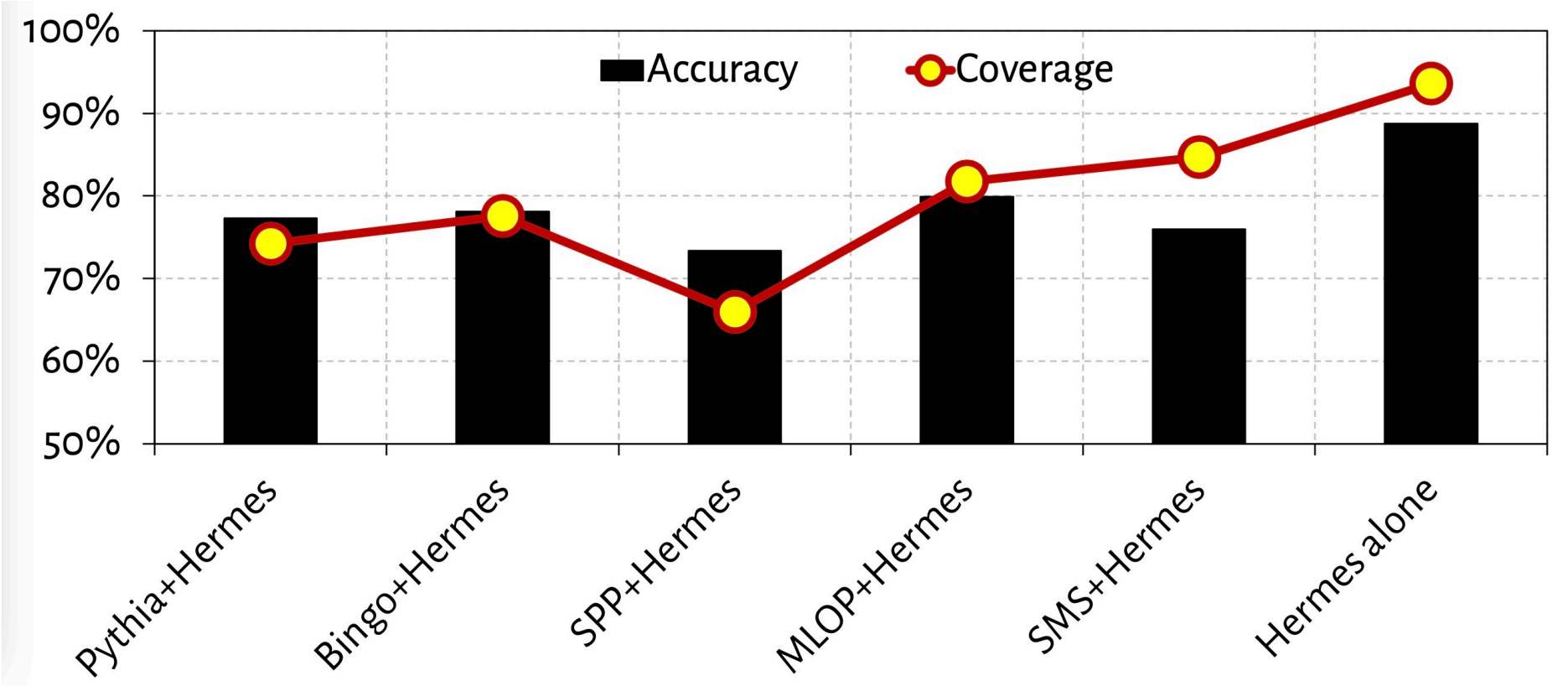
Effect of ROB Size



Effect of LLC Size



Accuracy and Coverage with Different Prefetchers



POPET's **accuracy and coverage increases significantly** in absence of a data prefetcher

Increase in Main Memory Requests