

Computer Architecture

Lecture 18b:

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

Rahul Bera


ETH Zürich

Fall 2022

25 November 2022

The (Memory) Latency Problem

Conventional Latency Tolerance Techniques

- Out-of-order execution [initially by Tomasulo, 1967]
 - Tolerates cache misses that cannot be prefetched
 - Requires extensive hardware resources for tolerating long latencies
- Multithreading [initially in CDC 6600, 1964]
 - Works well if there are multiple threads
 - Improving single thread performance using multithreading hardware is an ongoing research effort
- Caching [initially by Wilkes, 1965]
 - Widely used, simple, effective, but inefficient, passive
 - Not all applications/phases exhibit temporal or spatial locality
- Prefetching [initially in IBM 360/91, 1967]  **Pythia**
 - Works well for regular memory access patterns
 - Prefetching irregular access patterns is difficult, inaccurate, and hardware-intensive

Conventional Latency Tolerance Techniques

- **Out-of-order execution [initially by Tomasulo, 1967]**
 - ❑ Tolerates cache misses that cannot be prefetched
 - ❑ Requires extensive hardware resources for tolerating long latencies
- Multithreading [initially in CDC 6600, 1964]
 - ❑ Works well if there are multiple threads
 - ❑ Improving single thread performance using multithreading hardware is an ongoing research effort
- **Caching [initially by Wilkes, 1965]**
 - ❑ Widely used, simple, effective, but inefficient, passive
 - ❑ Not all applications/phases exhibit temporal or spatial locality
- **Prefetching [initially in IBM 360/91, 1967]**
 - ❑ Works well for regular memory access patterns
 - ❑ Prefetching irregular access patterns is difficult, inaccurate, and hardware-intensive



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>



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

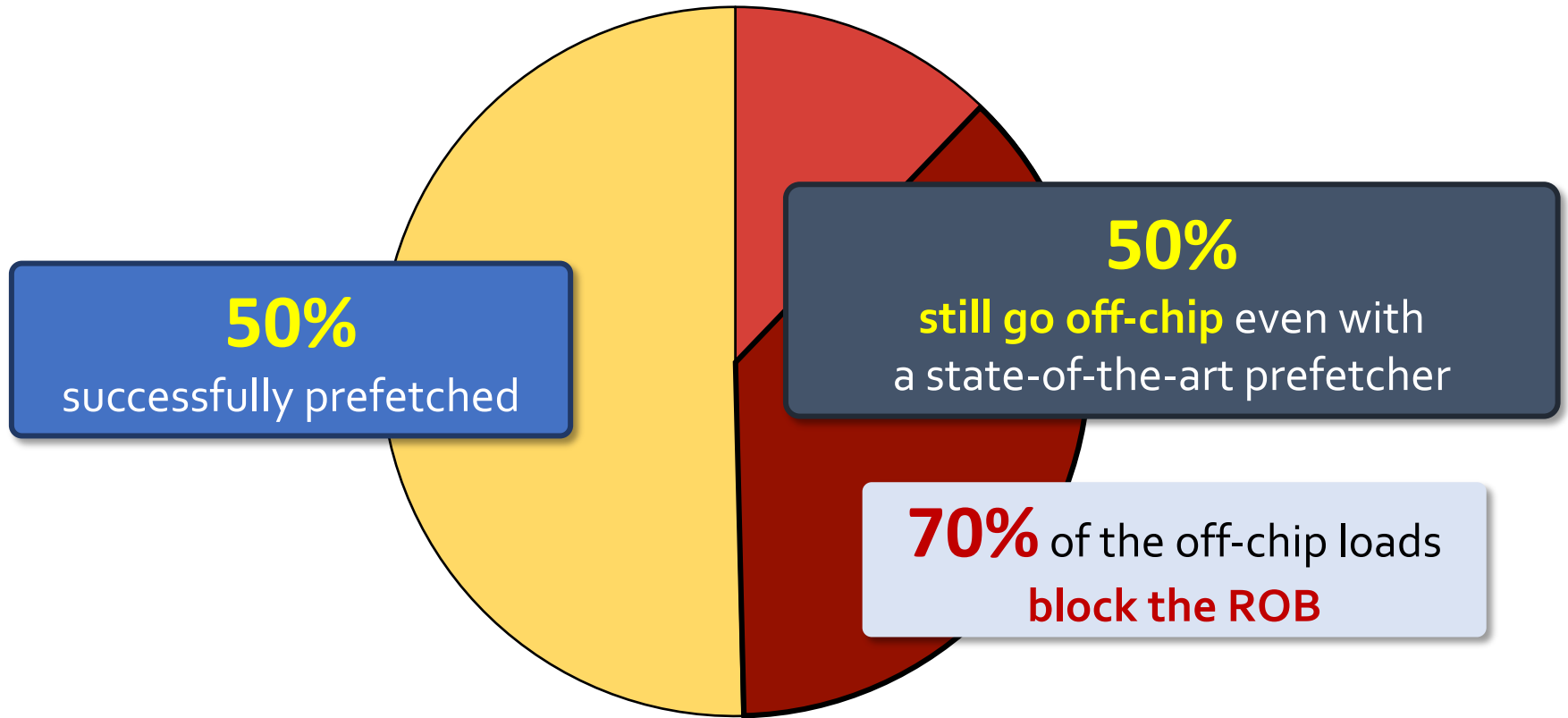
Employ sophisticated **prefetchers**

2

Increase size of on-chip caches

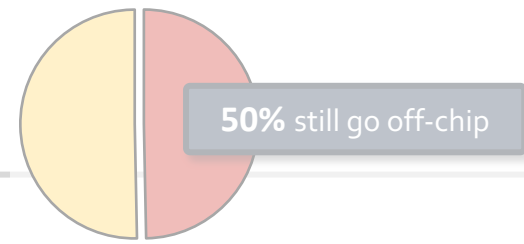
Key Observation 1

Many loads still go off-chip

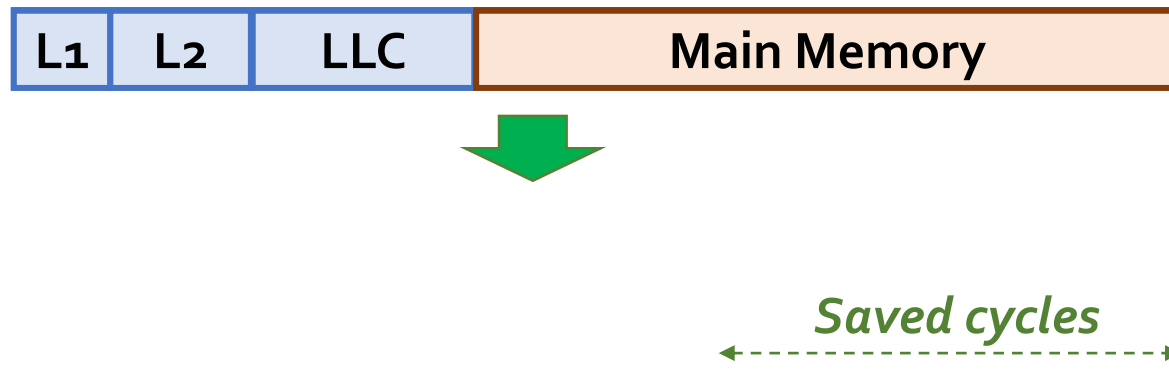


off-chip loads without any prefetcher

Key Observation 2

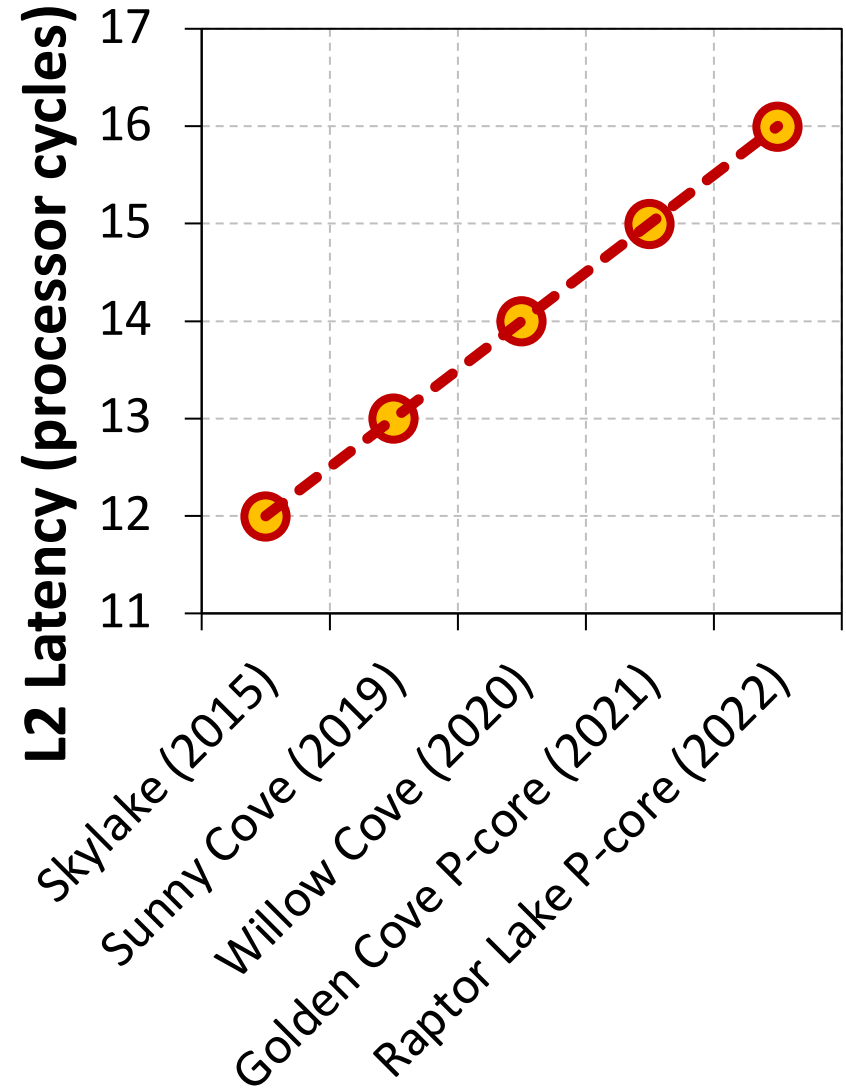
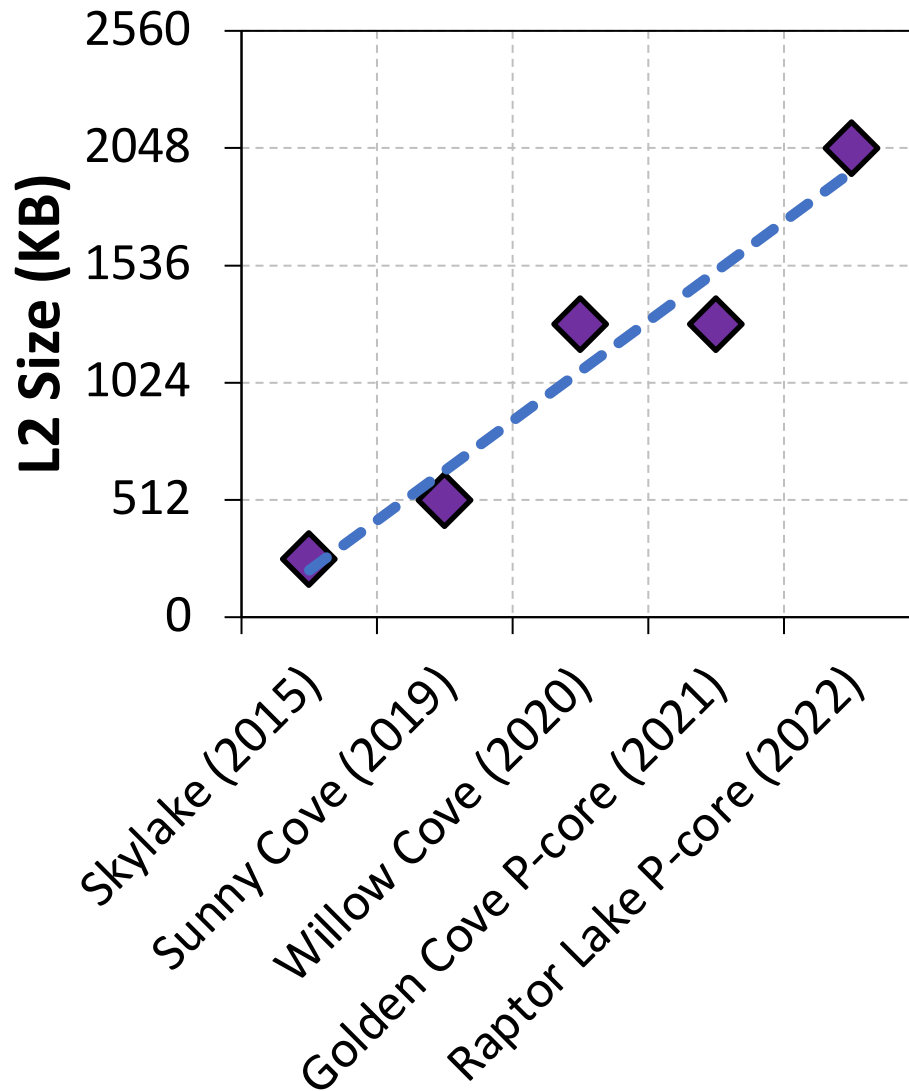


On-chip cache access latency
significantly contributes to off-chip load latency



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

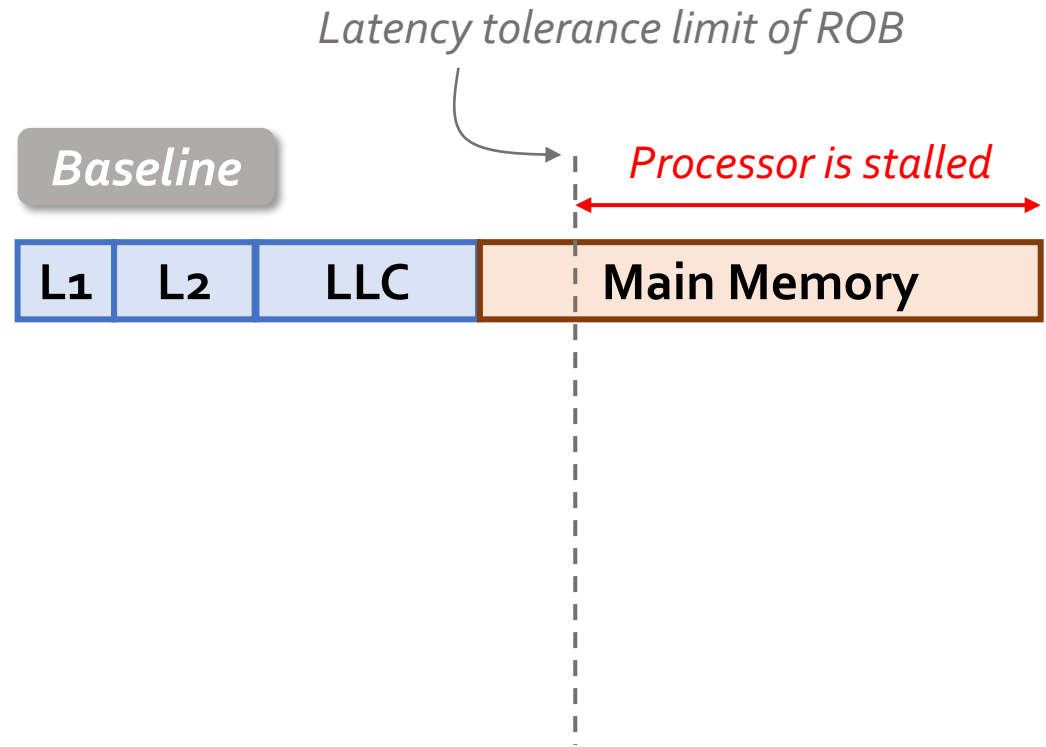
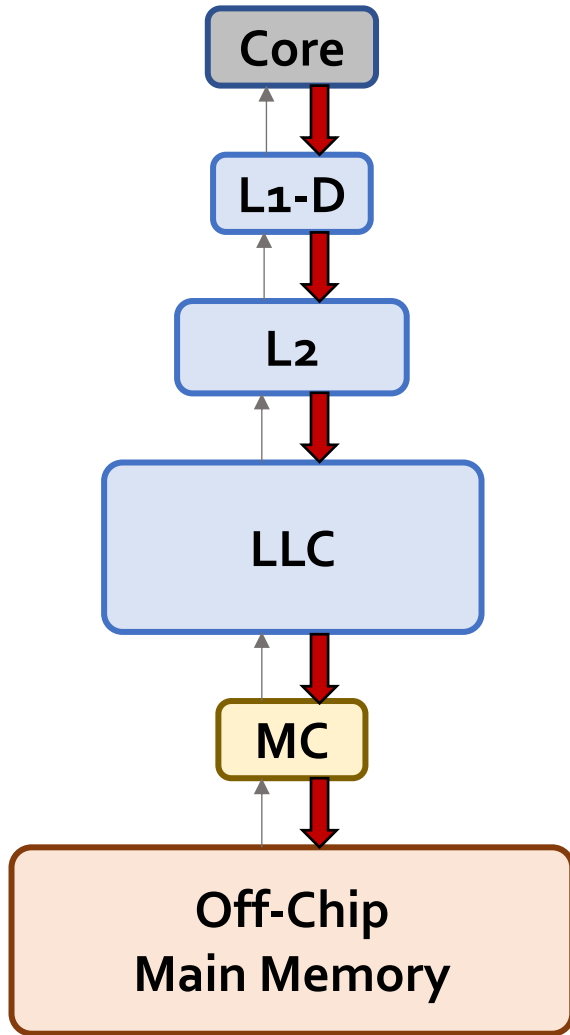


That predicts which loads are likely to **go off-chip**

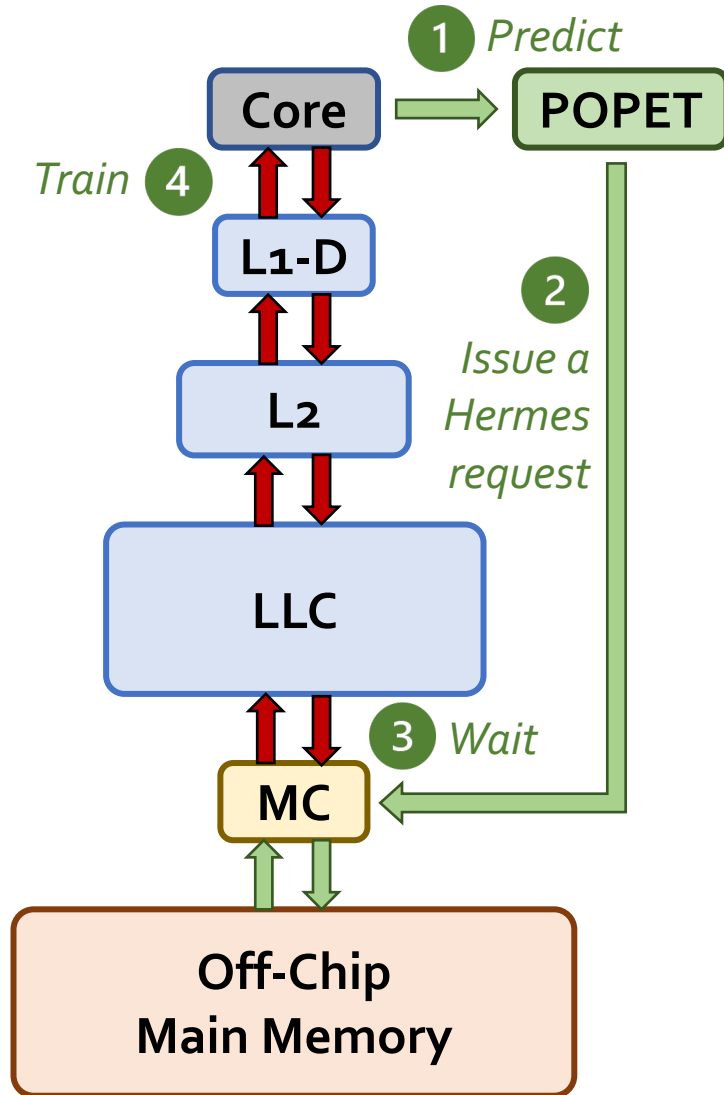


By **learning** from
multiple program context information

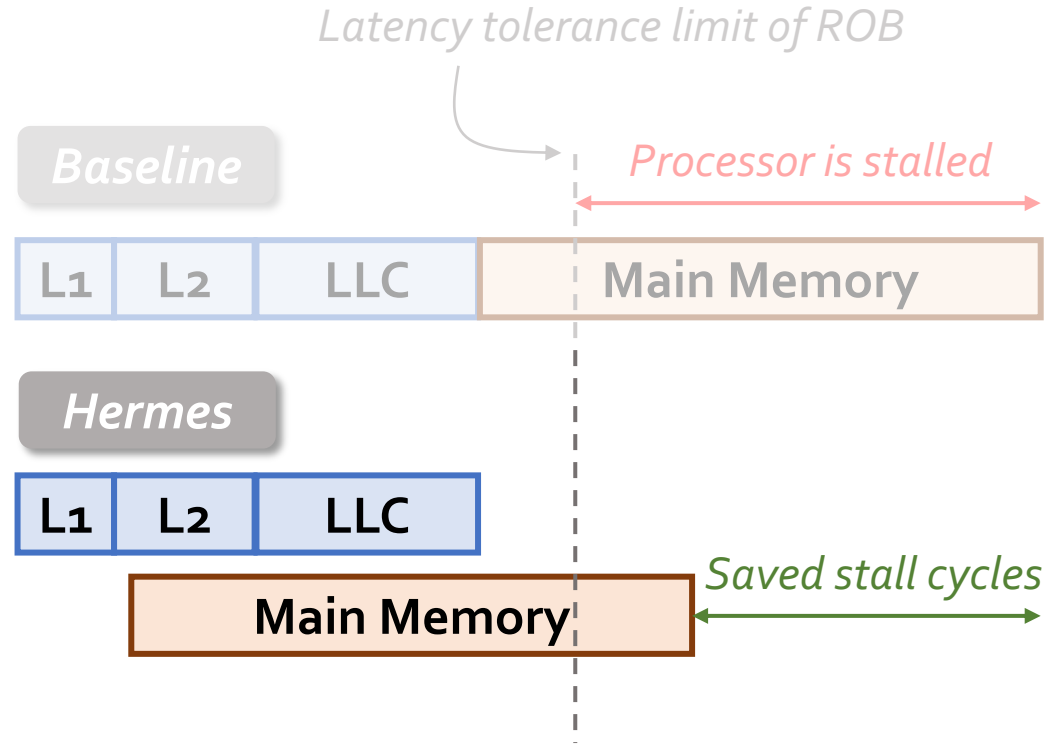
Hermes Overview



Hermes Overview



Perceptron-based
off-chip load predictor



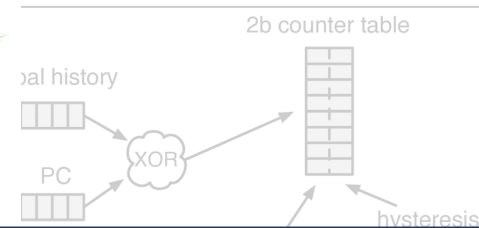
Designing the Off-Chip Load Predictor

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

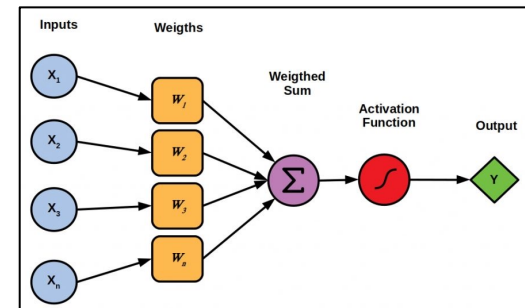
Correlate different program features with off-chip loads



Low storage overhead

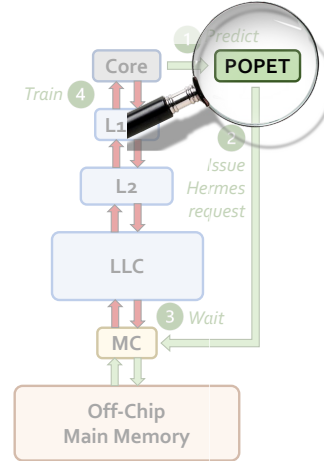
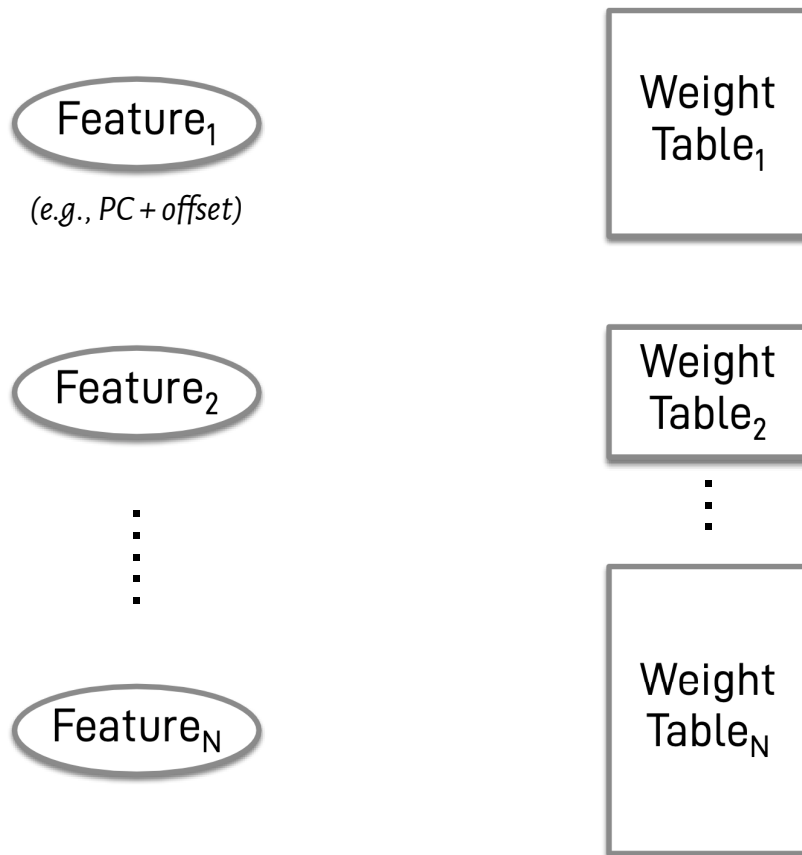


Low design complexity



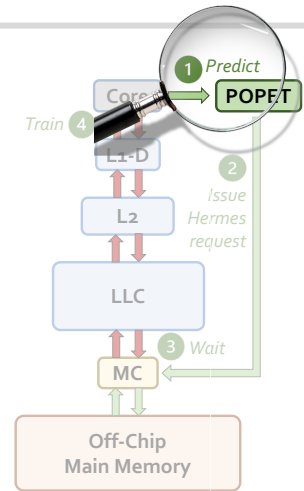
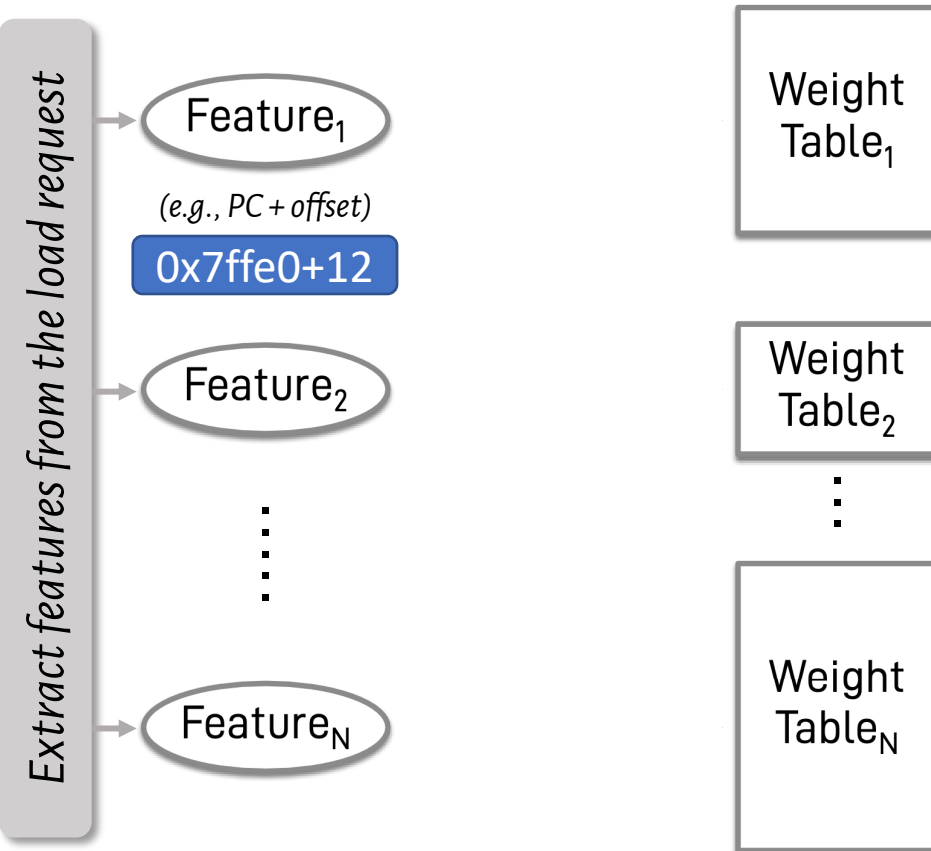
POPET: Perceptron-Based Off-Chip Predictor

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



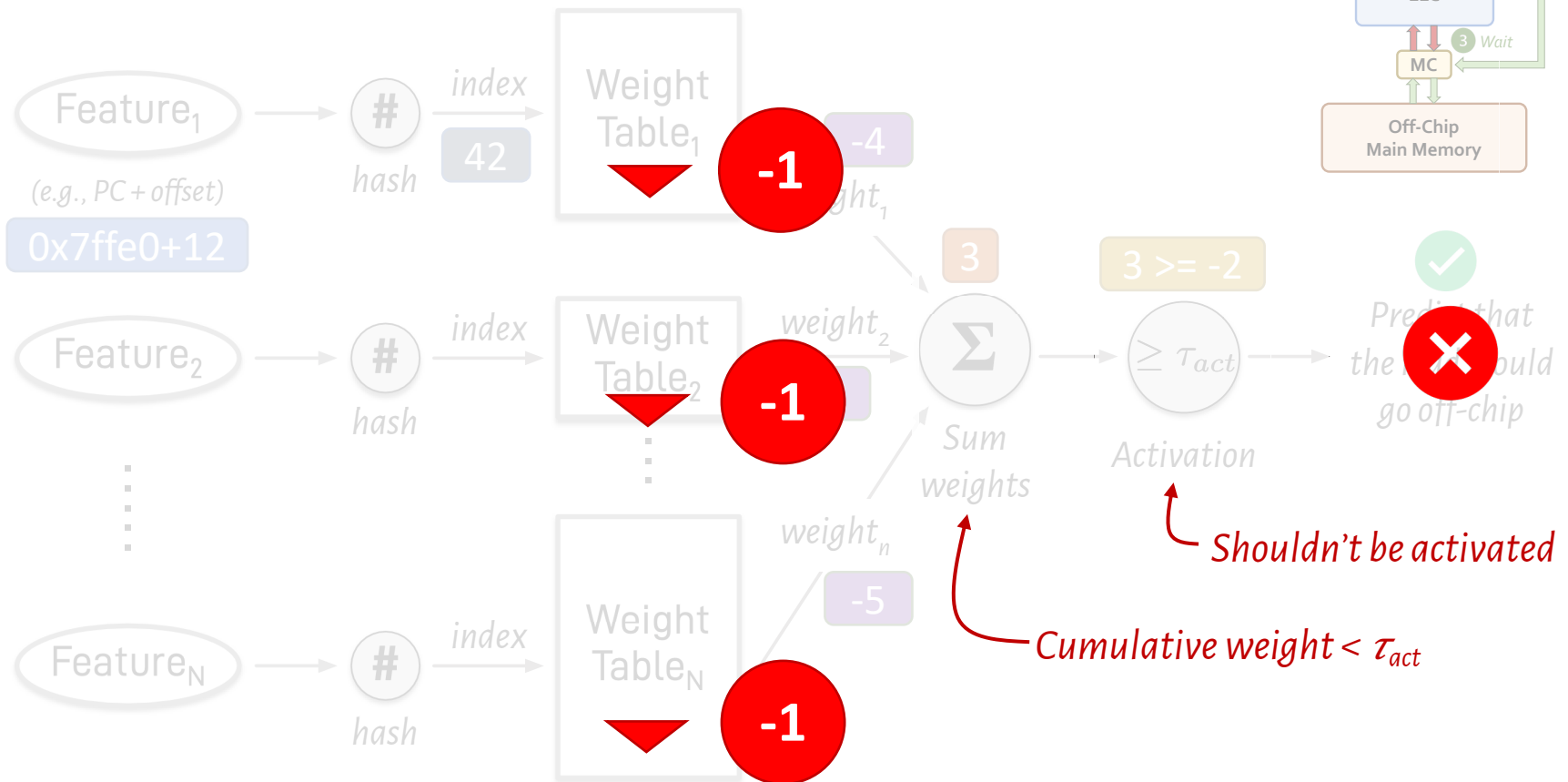
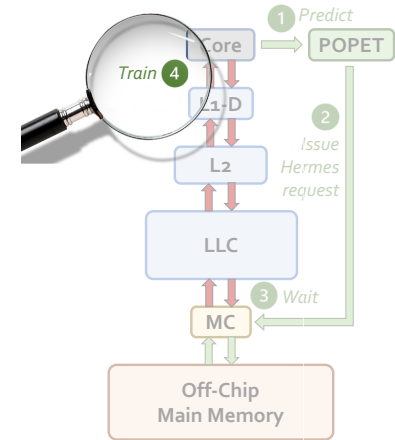
Predicting using POPET

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



Training POPET

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



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

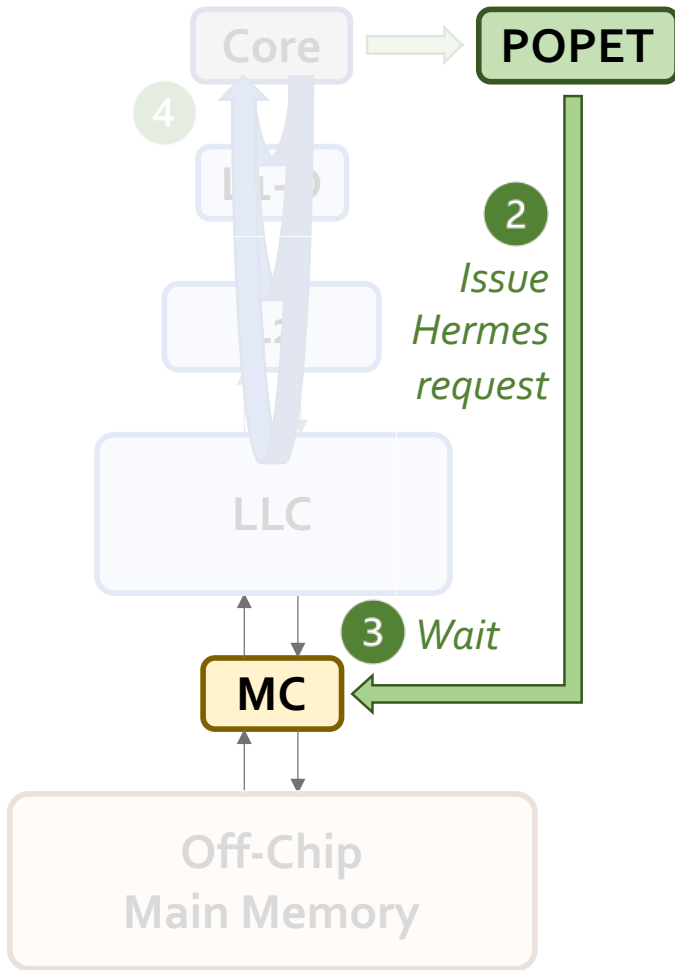
LLC Prefetchers

- 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**

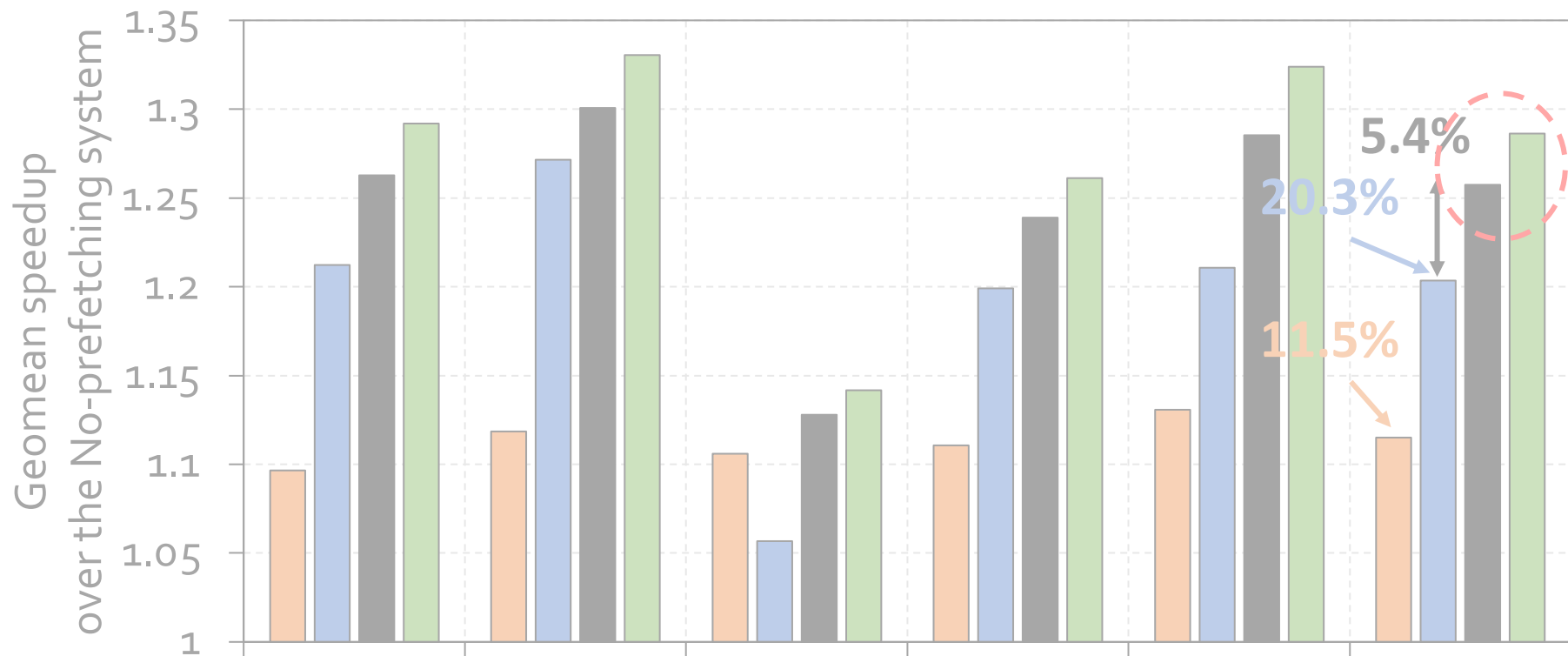
(incurred after address translation)

Depends on

- **Interconnect** between POPET and MC



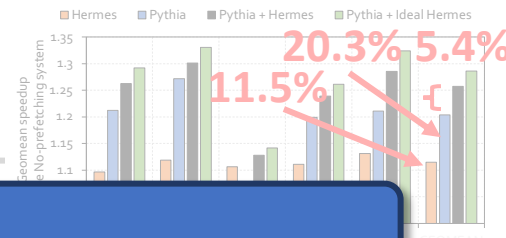
Single-Core Performance Improvement



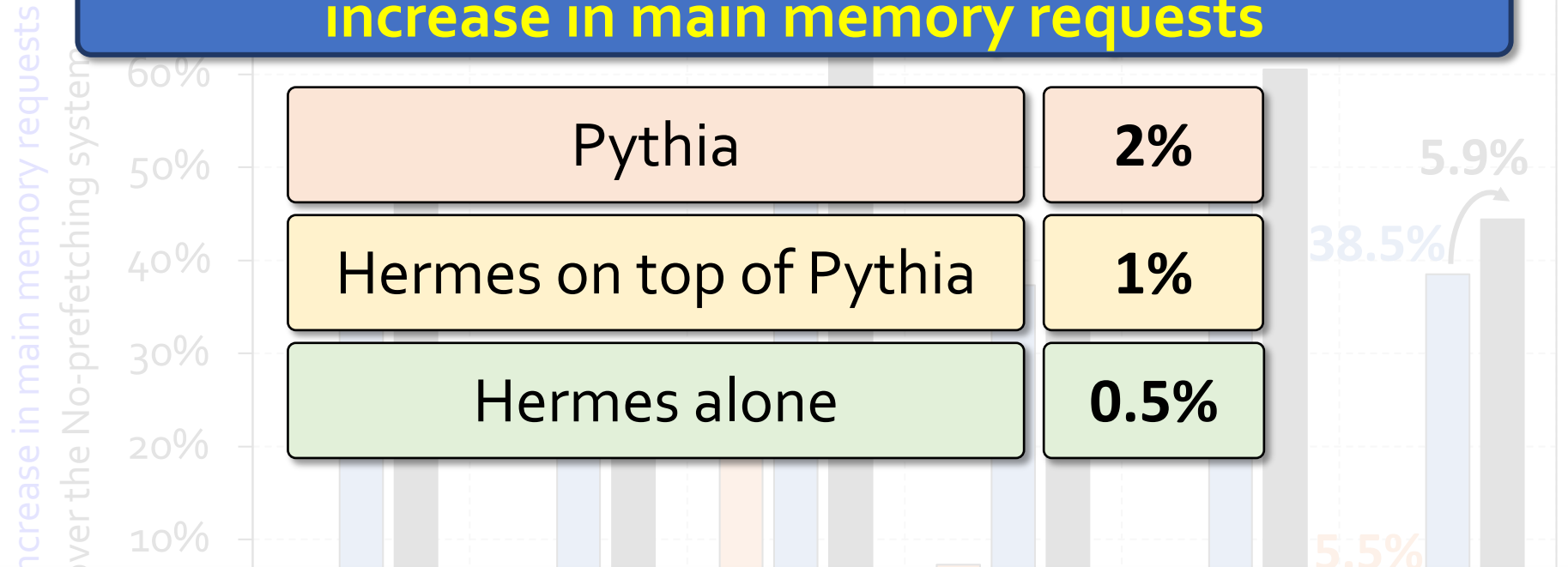
Hermes alone provides nearly

Hermes provides nearly **90%** performance benefit of **Ideal Hermes** that has an **ideal off-chip load predictor** with only **2.5%** storage overhead

Increase in Main Memory Requests

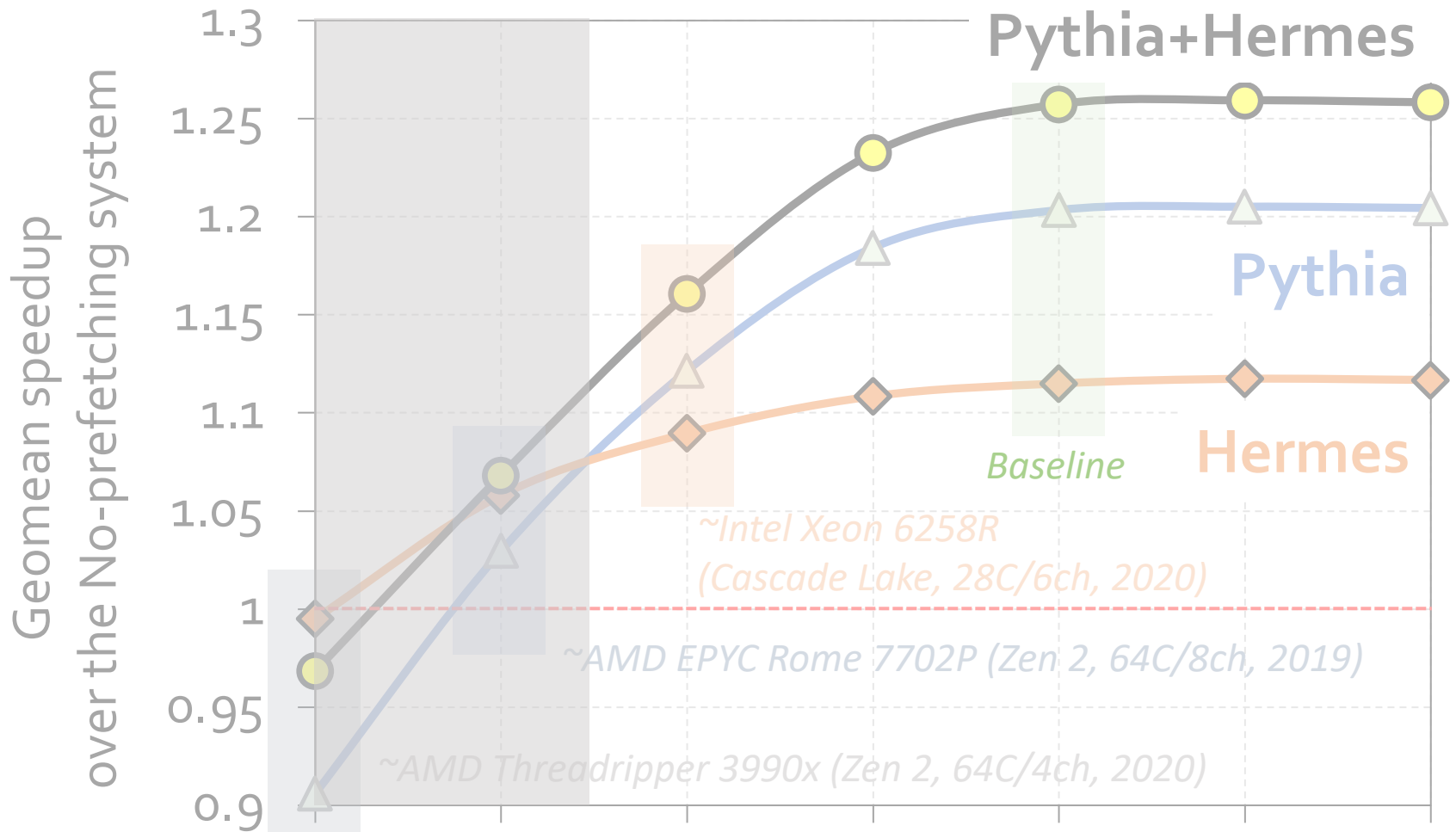


For **every 1% performance** benefit,
increase in main memory requests



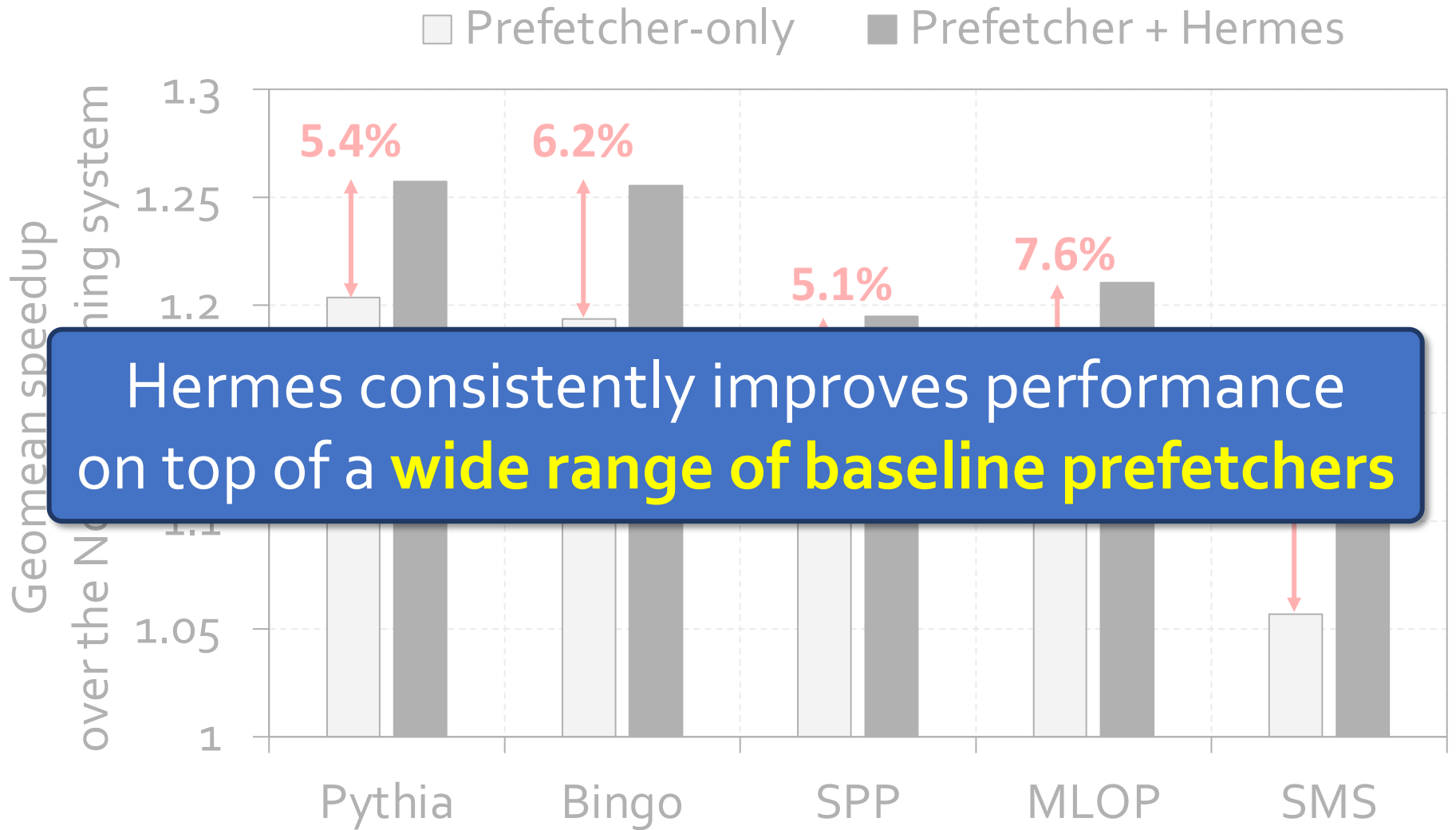
Hermes is more **bandwidth-efficient**
than even an efficient prefetcher like Pythia

Performance with Varying Memory Bandwidth



Hermes+Pythia outperforms Pythia
across all bandwidth configurations

Performance with Varying Baseline Prefetcher



Overhead of Hermes



4 KB storage overhead



1.5% power overhead*

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

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 Bera¹ Konstantinos Kanellopoulos¹ Shankar Balachandran² David Novo³
Ataberk Olgun¹ Mohammad Sadrosadati¹ Onur Mutlu¹

¹ETH Zürich ²Intel Processor Architecture Research Lab ³LIRMM, 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 re-order 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

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>



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](#)

BACKUP

Initial Set of Program Features

Features without control-flow information	Features with control-flow information
<ol style="list-style-type: none">1. Load virtual address2. Virtual page number3. Cacheline offset in page4. First access5. Cacheline offset + first access6. Byte offset in cacheline7. Word offset in cacheline	<ol style="list-style-type: none">8. Load PC9. $PC \oplus$ load virtual address10. $PC \oplus$ virtual page number11. $PC \oplus$ cacheline offset12. $PC +$ first access13. $PC \oplus$ byte offset14. $PC \oplus$ word offset15. Last-4 load PCs16. Last-4 PCs

Selected Set of Program Features

Five features

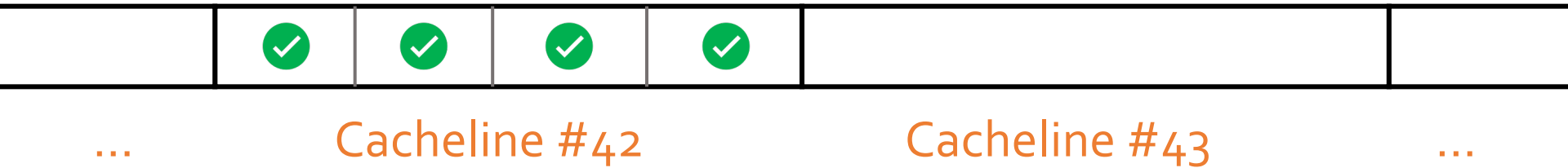
- $PC \oplus$ cacheline offset
- $PC \oplus$ byte offset
- $PC \leftarrow$ first access
- Cacheline offset \leftarrow first access
- Last-4 load PCs

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



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?

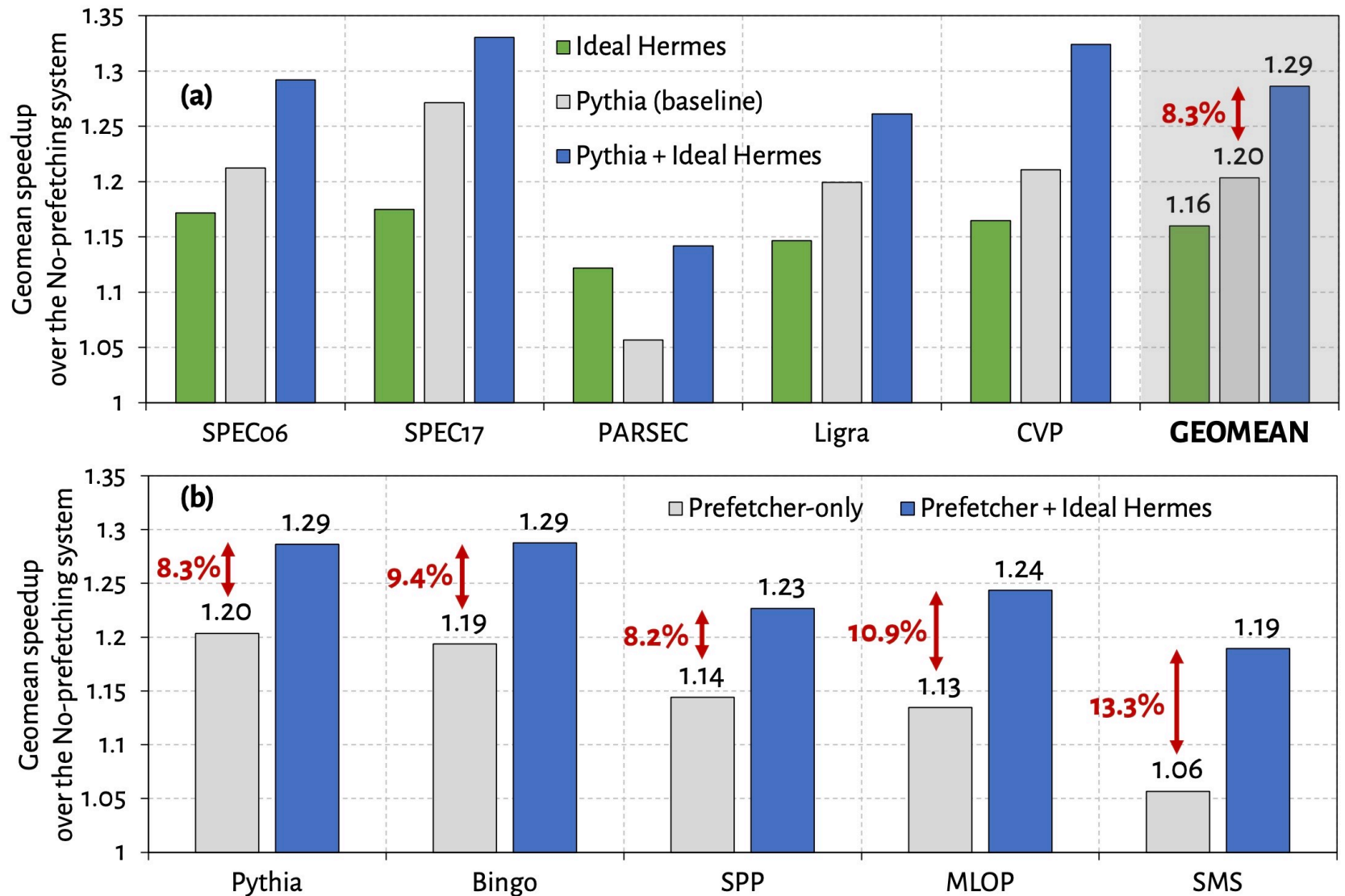
- Two cases of mispredictions:
- Predicted on-chip but actually goes off-chip
 - Loss of performance improvement opportunity

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

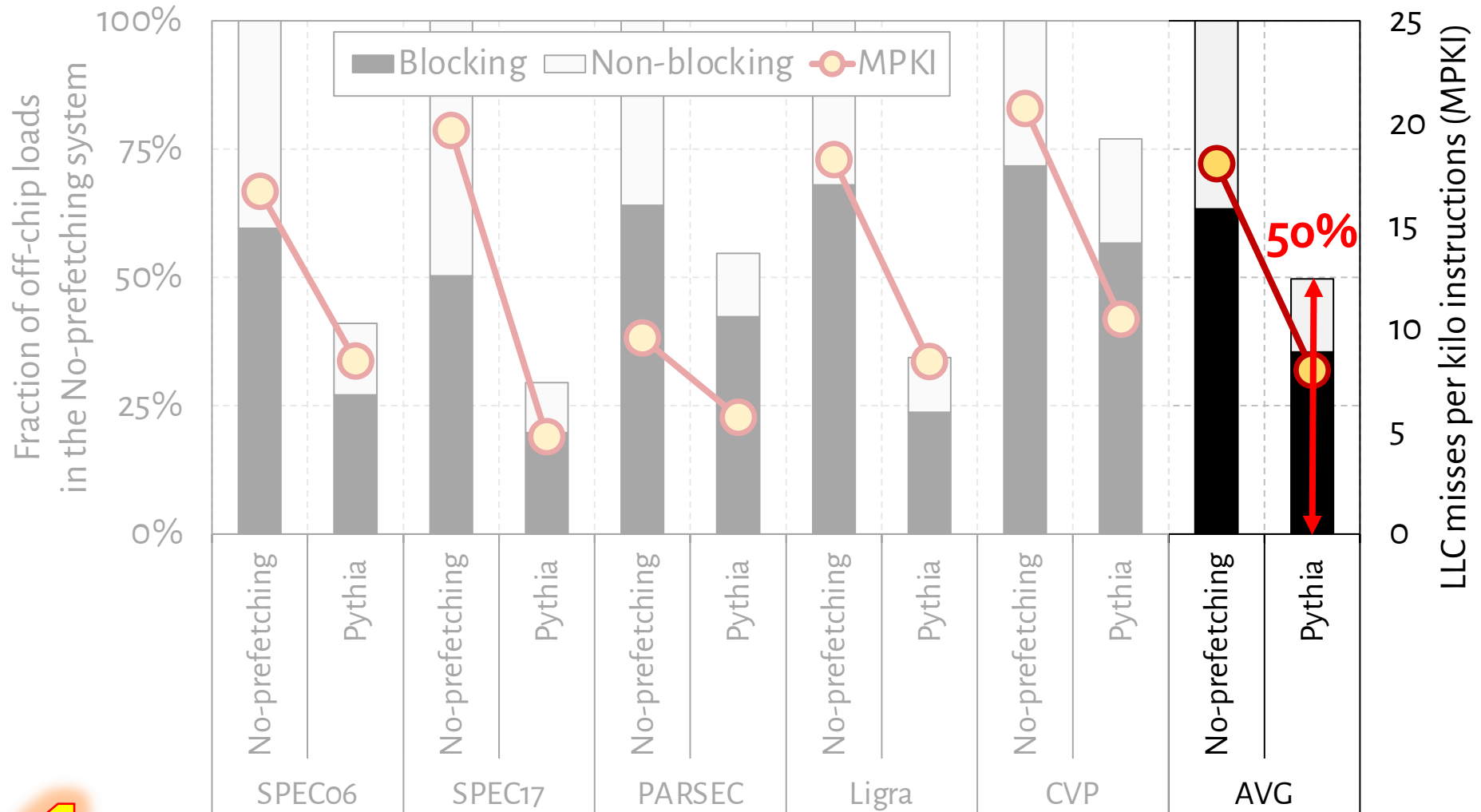
Core	1 and 8 cores, 6-wide fetch/execute/commit, 512-entry ROB, 128/72-entry LQ/SQ, Perceptron branch predictor [61] with 17-cycle misprediction penalty
L1/L2 Caches	Private, 48KB/1.25MB, 64B line, 12/20-way, 16/48 MSHRs, LRU, 5/15-cycle round-trip latency [25]
LLC	3MB/core, 64B line, 12 way, 64 MSHRs/slice, 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, ...

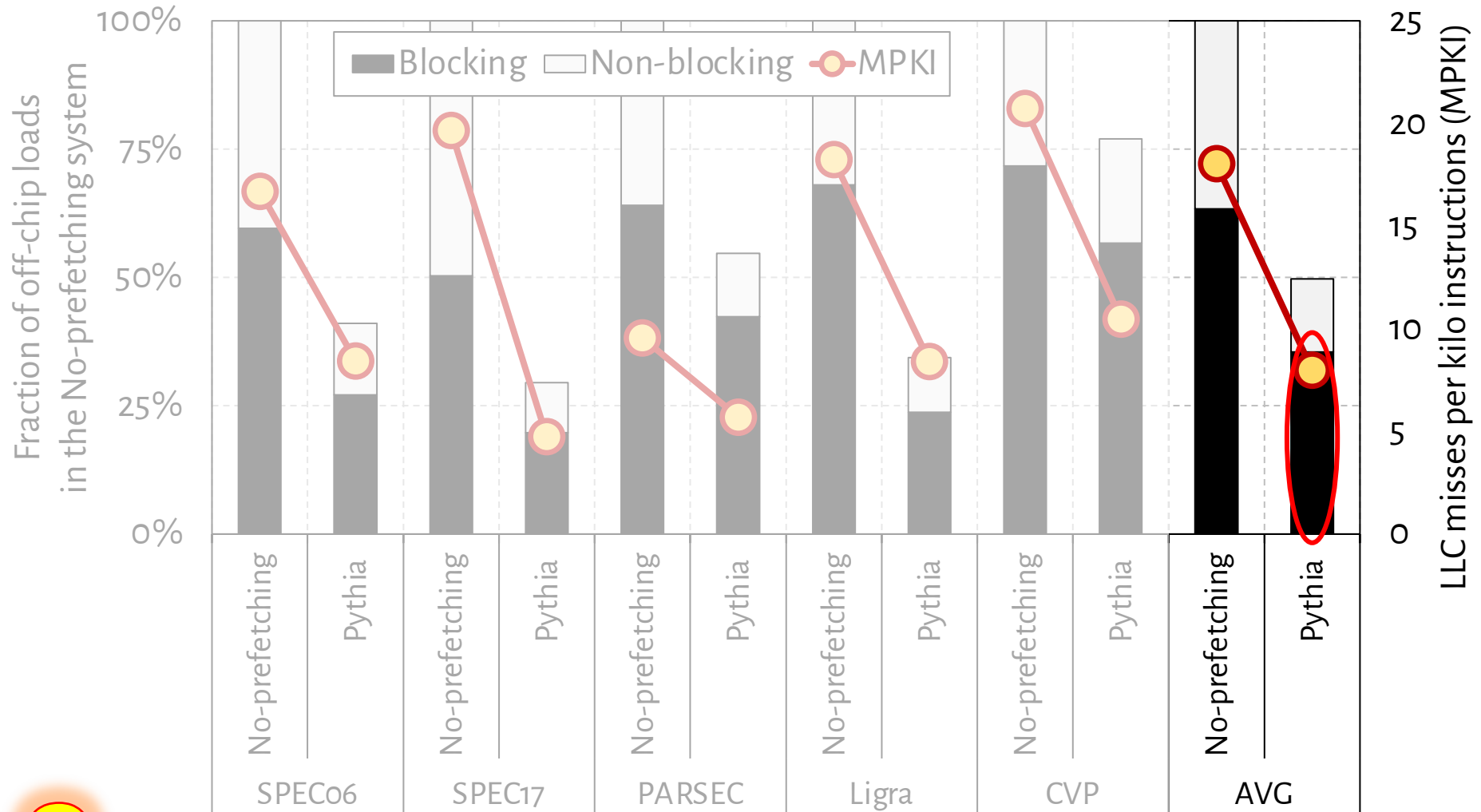
Observation: Not All Off-Chip Loads are Prefetched



1

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

Observation: Not All Off-Chip Loads are Prefetched



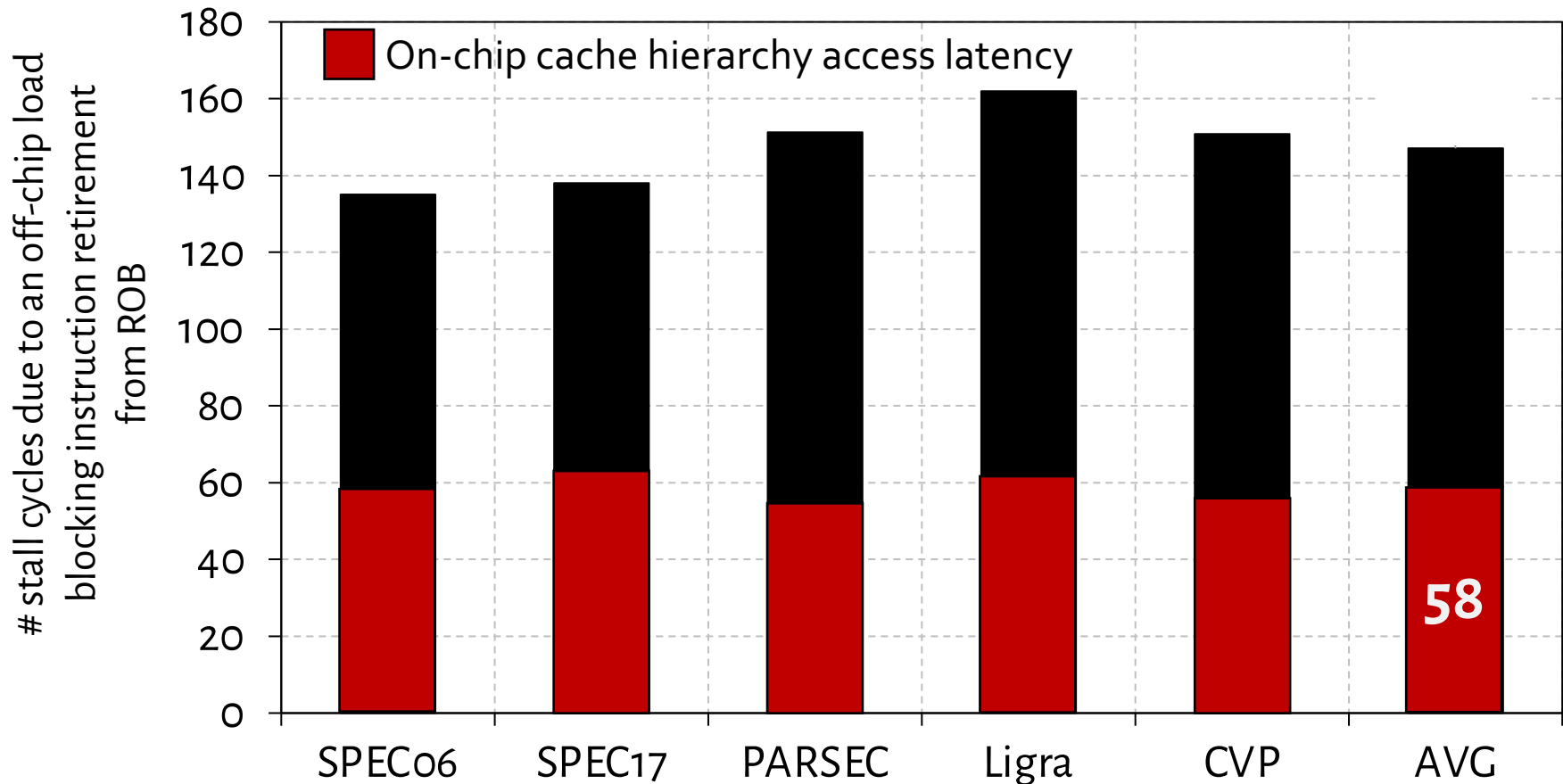
2

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



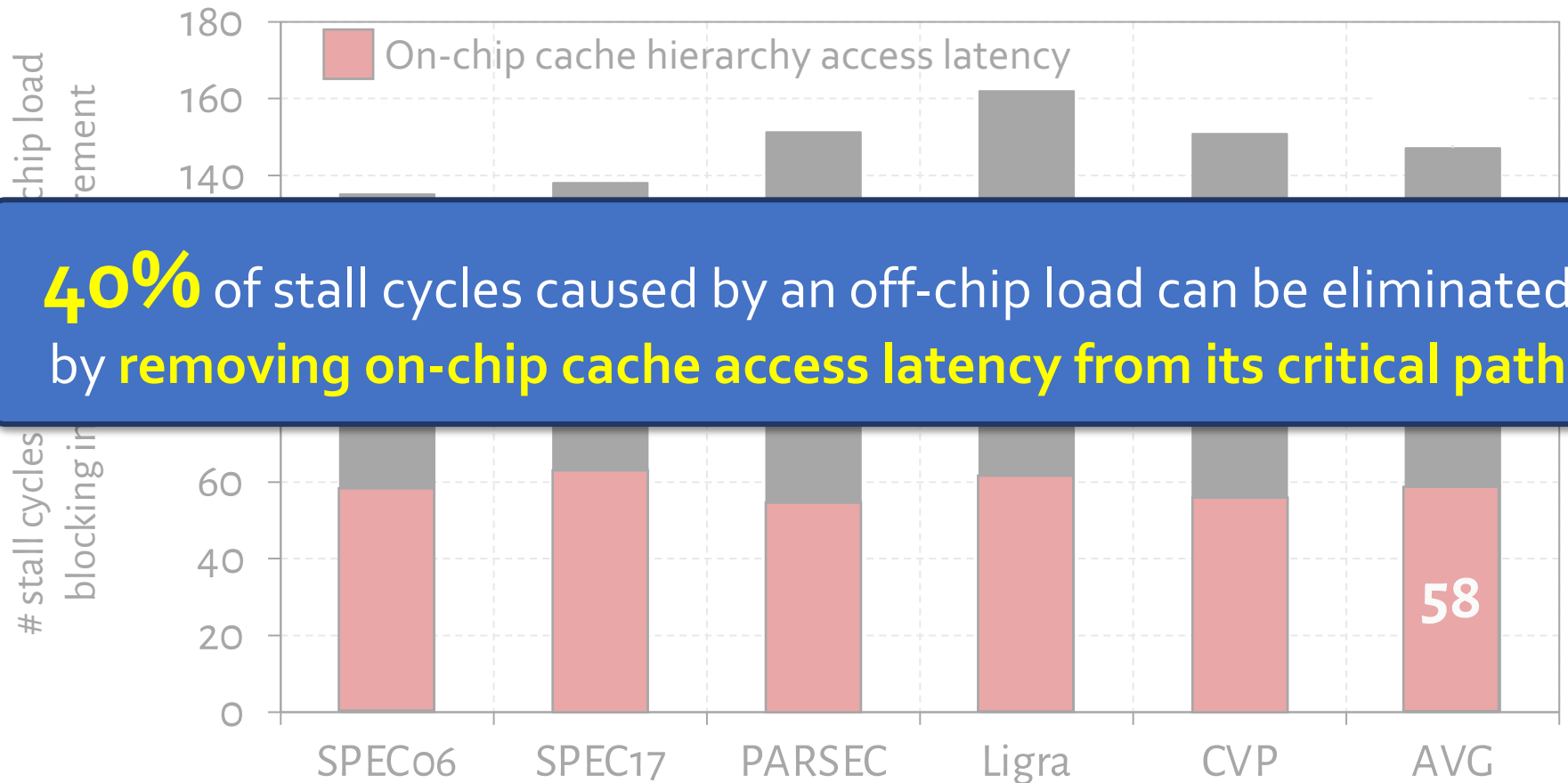
Observation: With Large Cache Comes Longer Latency

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

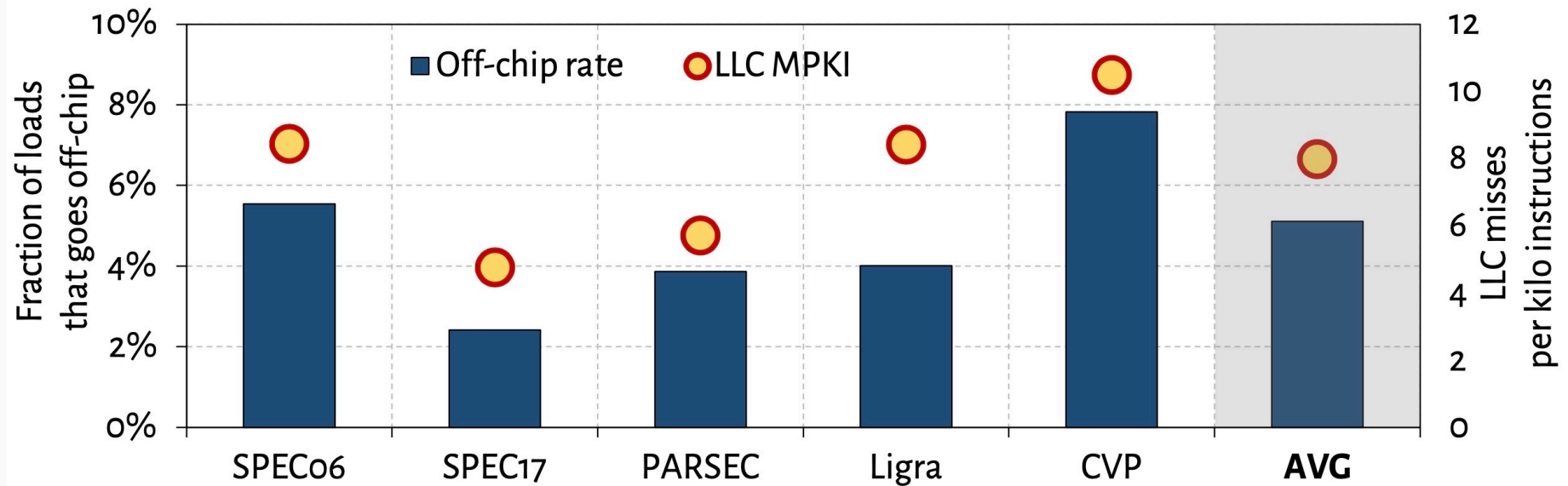


Observation: With Large Cache Comes Longer Latency

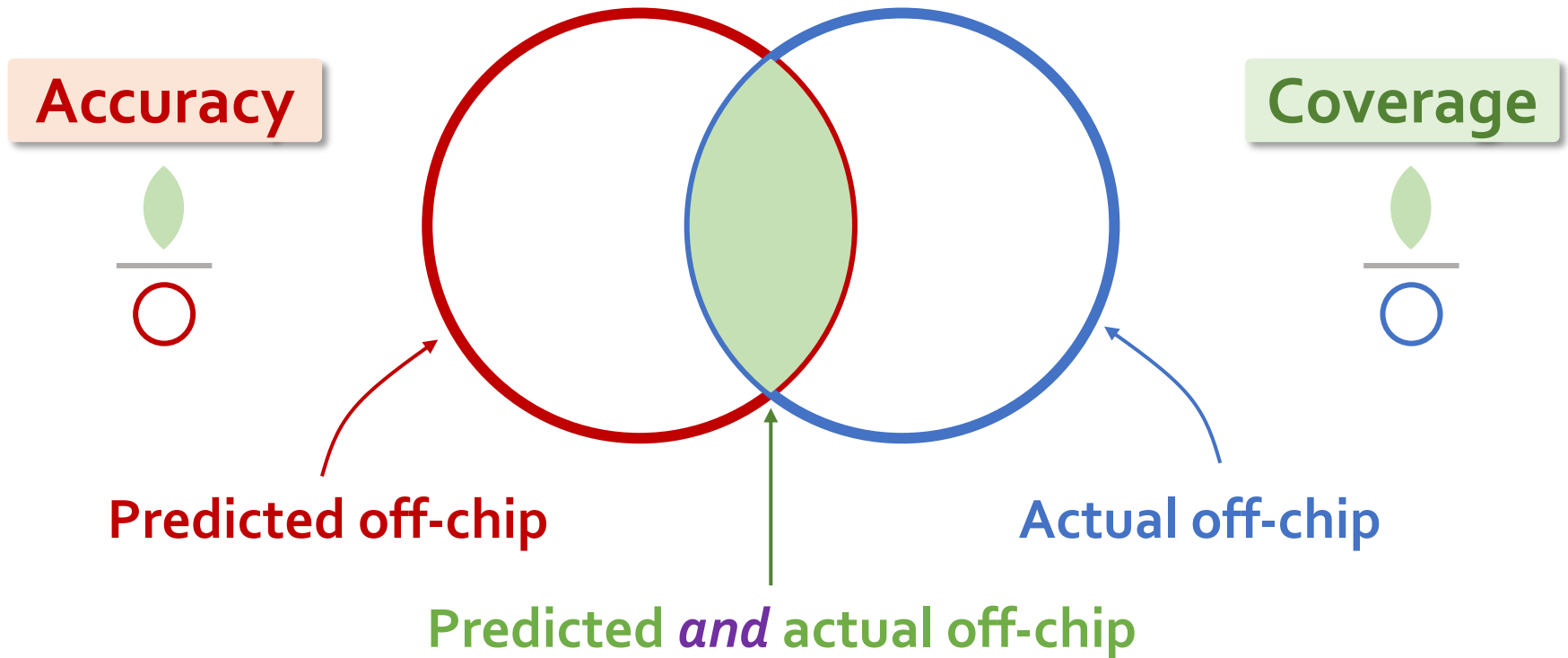
- On-chip cache access latency significantly contributes to the latency of an off-chip load



What Fraction of Load Requests Goes Off-Chip?

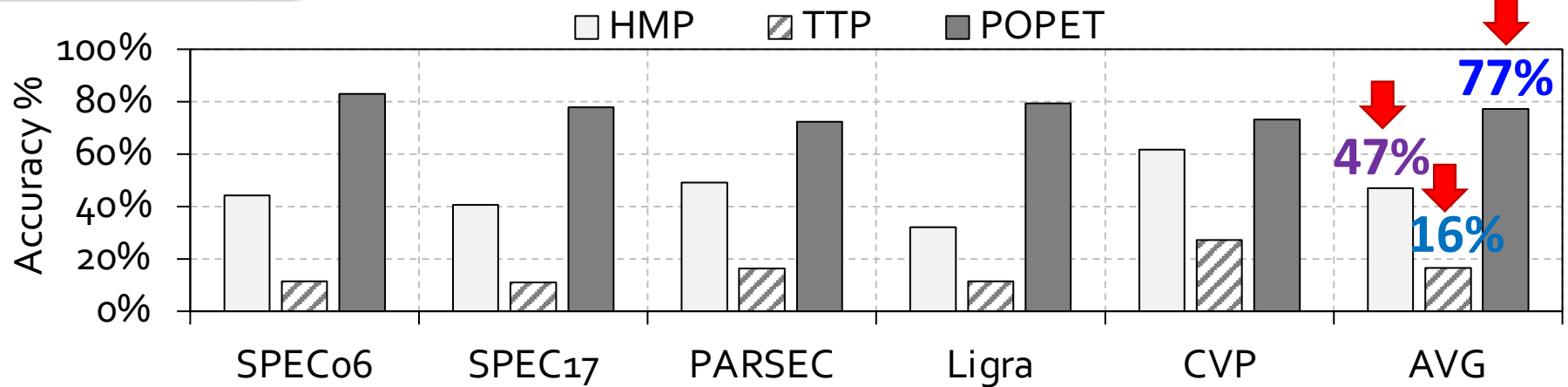


Off-Chip Prediction Quality: *Defining Metrics*

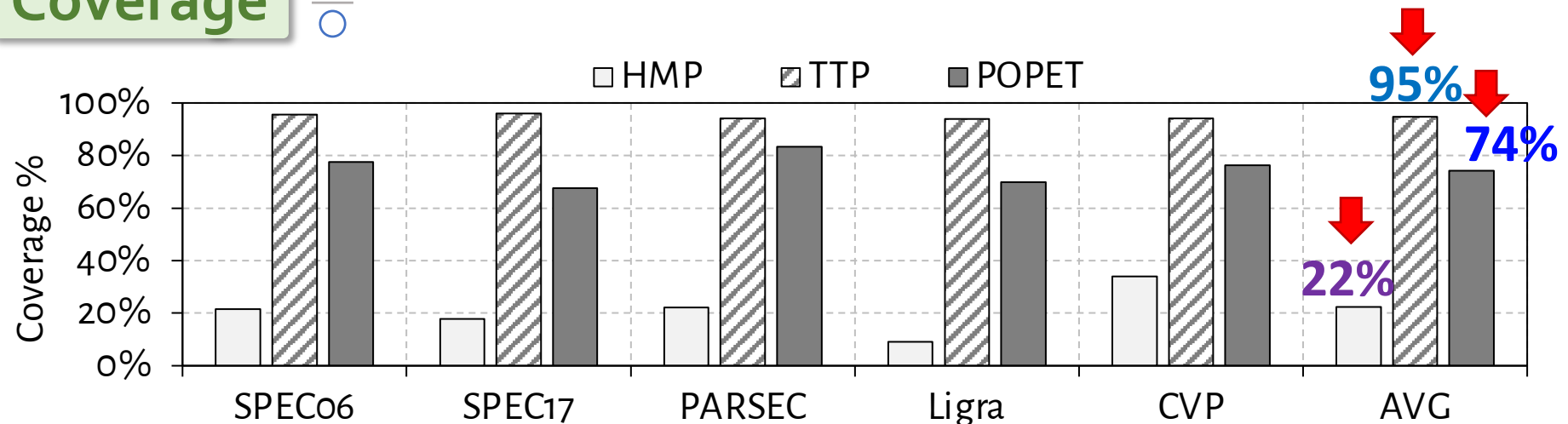


Off-Chip Prediction Quality: *Analysis*

Accuracy

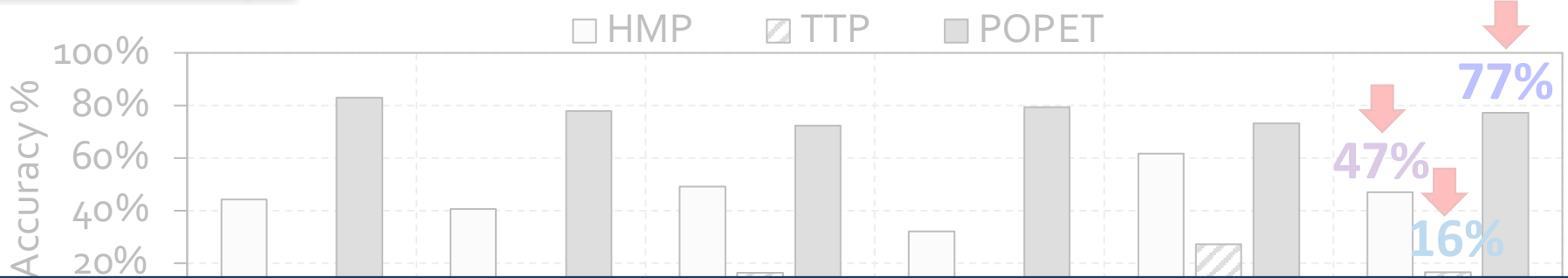


Coverage

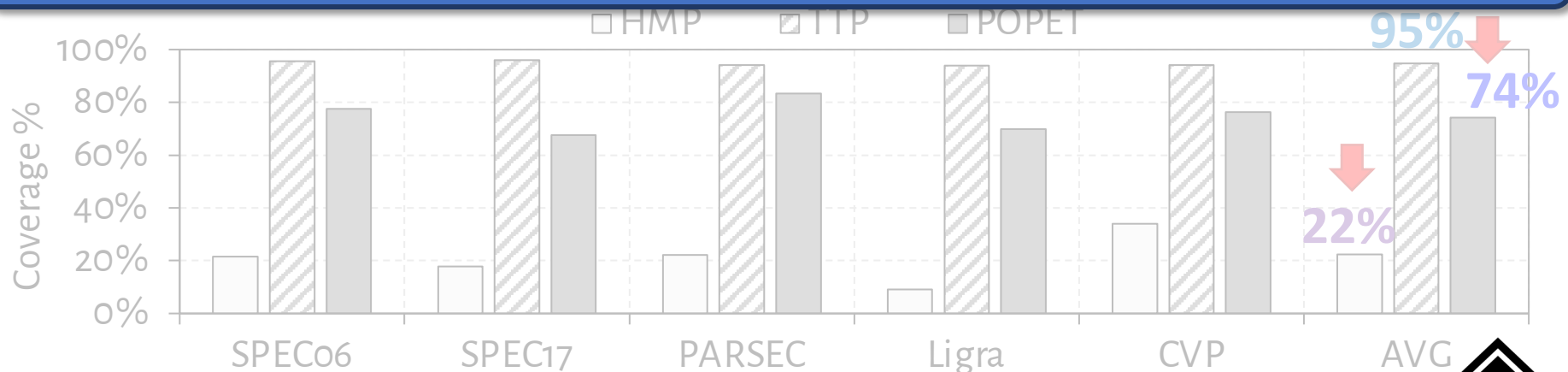


Off-Chip Prediction Quality: Analysis

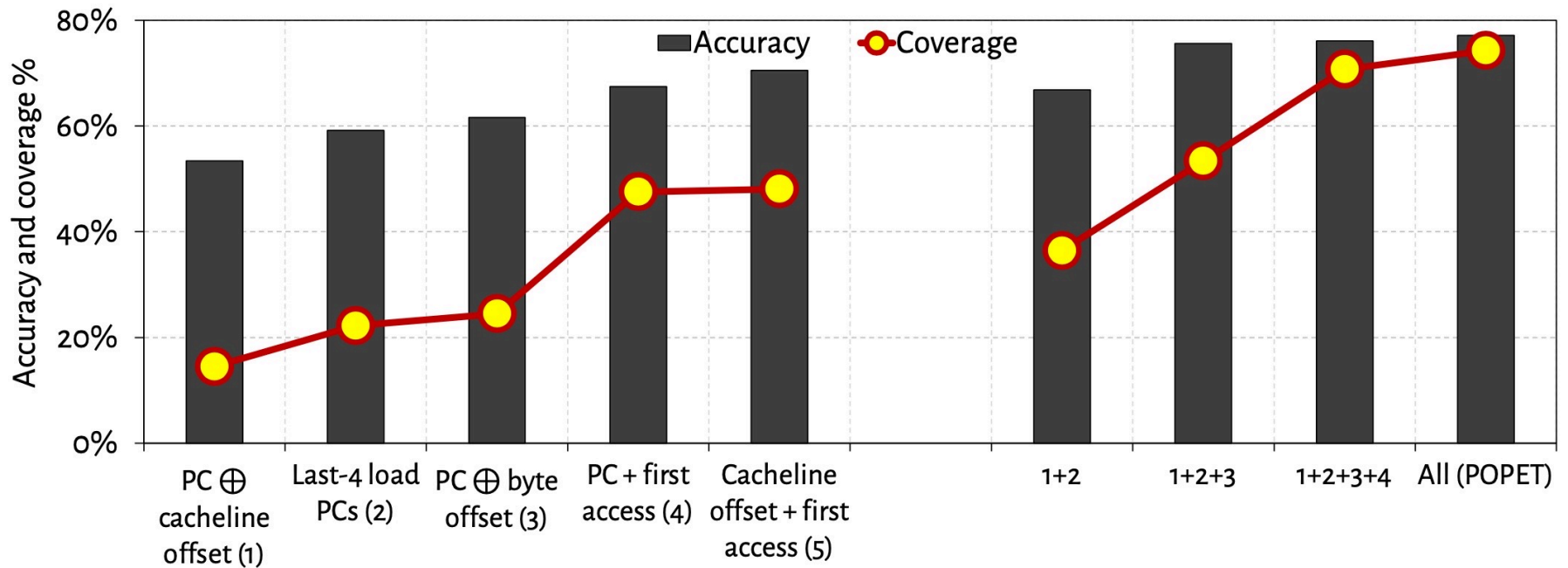
Accuracy



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

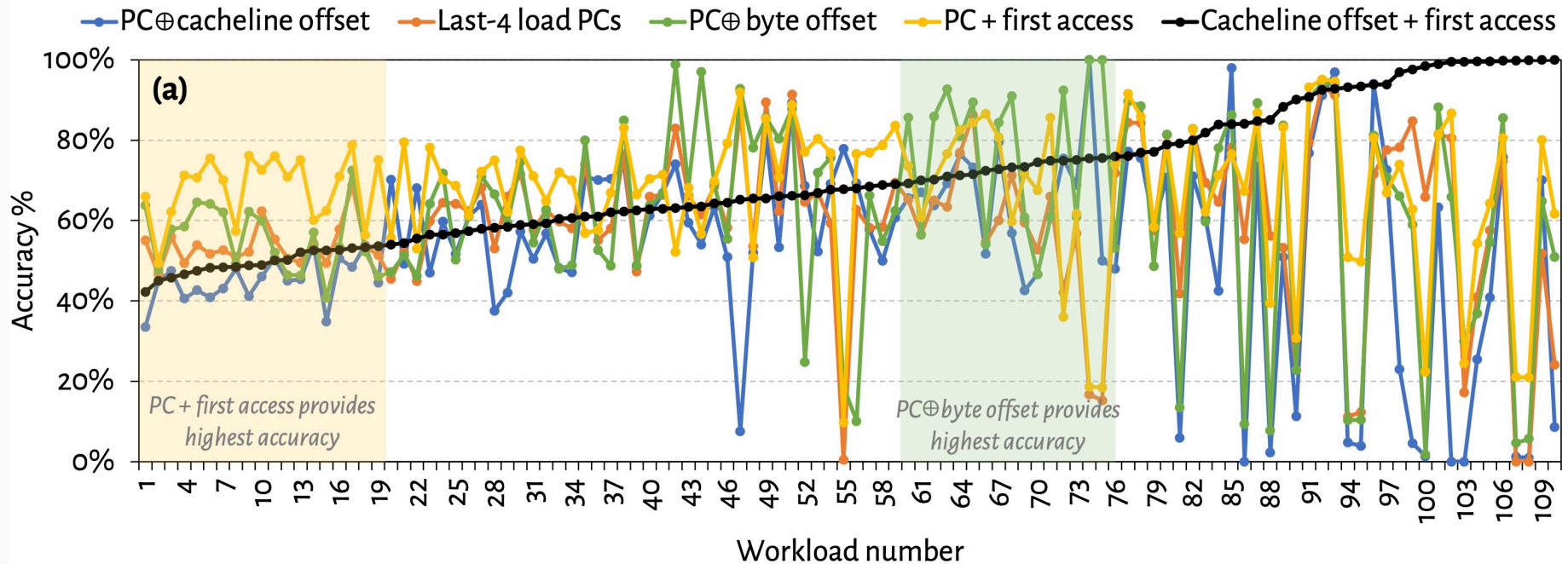


Effect of Different Features



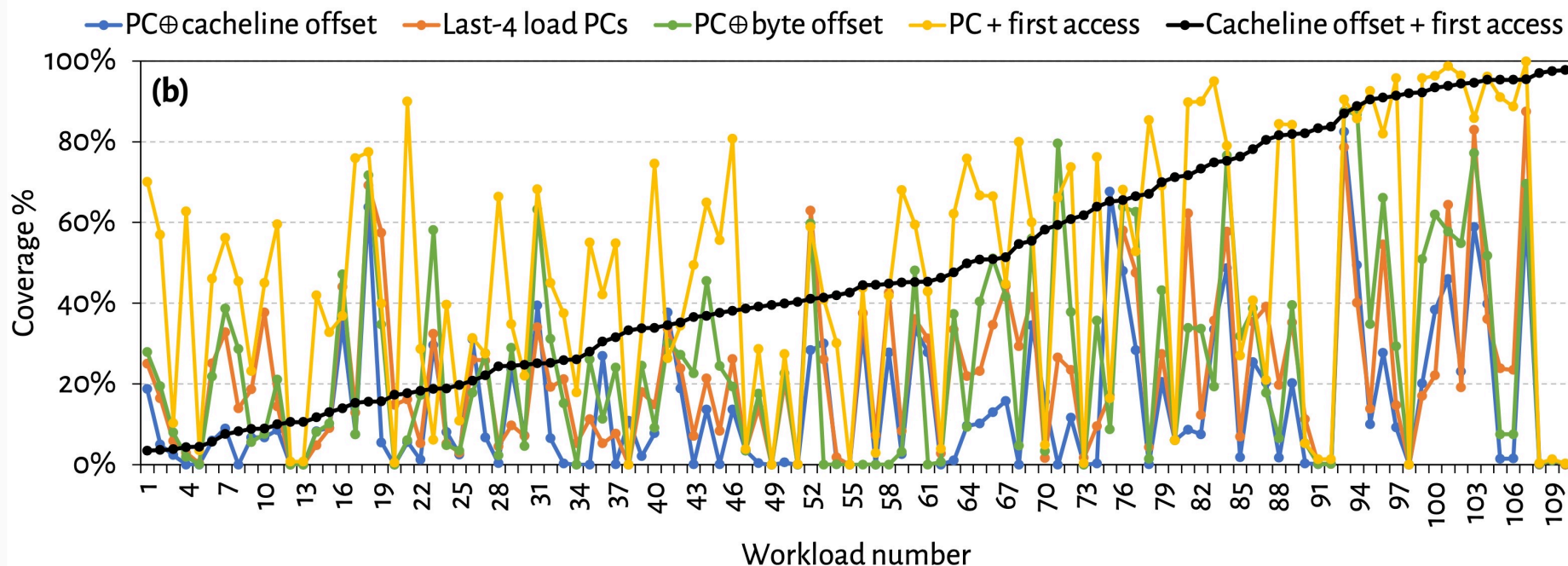
Combination of features provides both **higher accuracy and higher coverage** than any individual feature

Are All Features Required? (1)



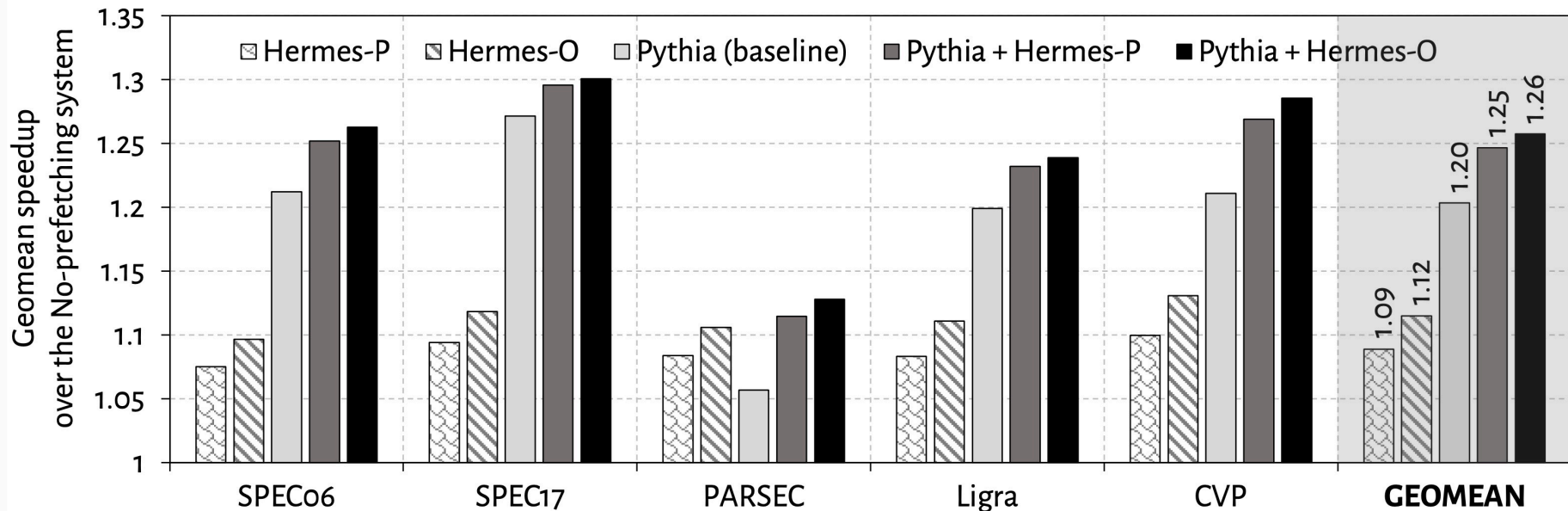
No single feature individually provides **highest prediction accuracy** across **all** workloads

Are All Features Required? (2)



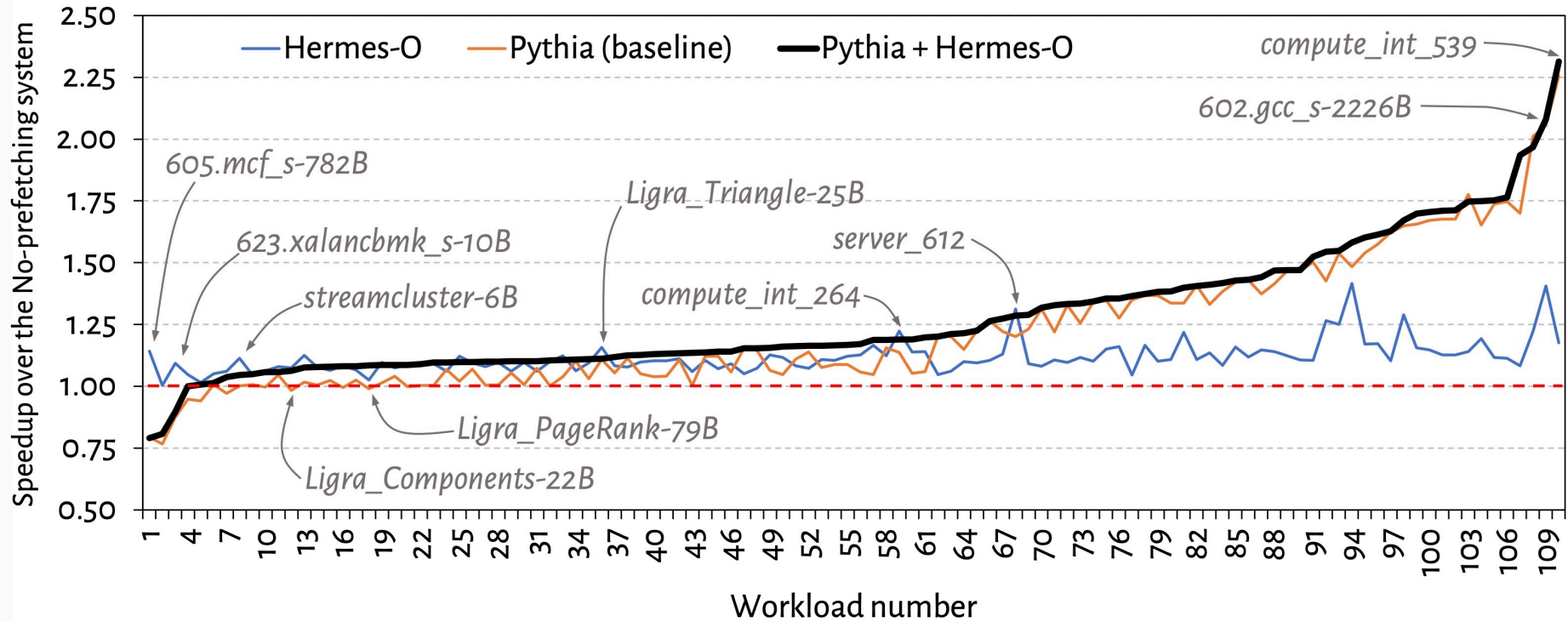
No single feature individually provides **highest prediction coverage** also across **all** workloads

Single-Core Performance

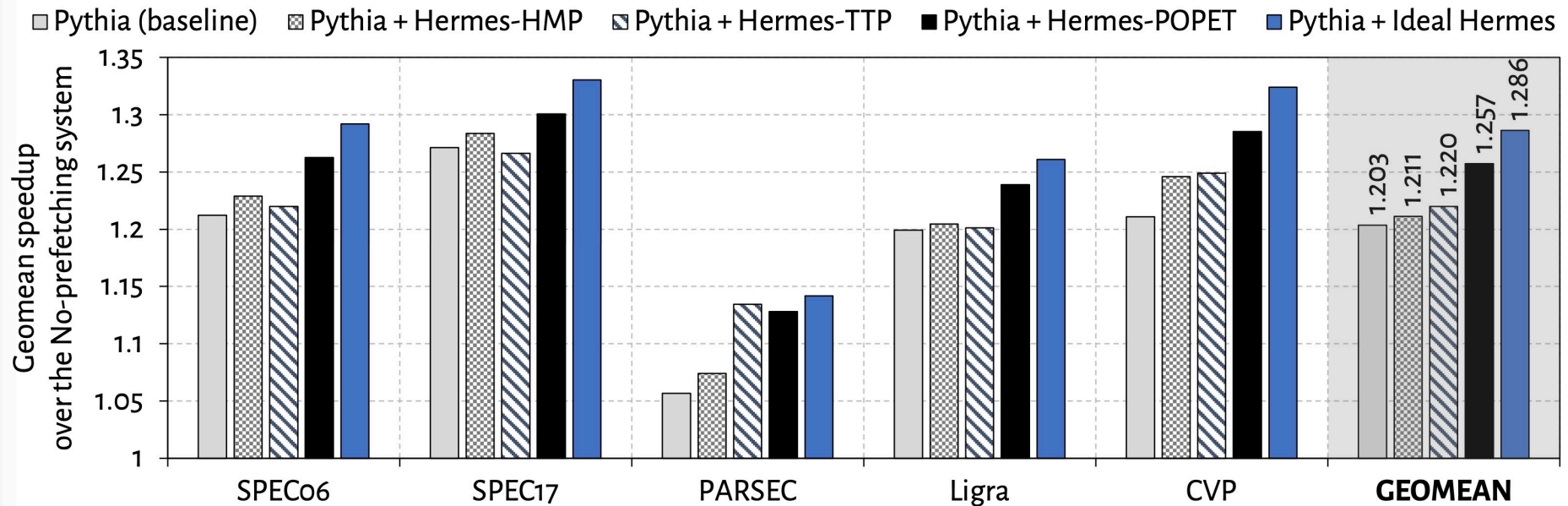


Hermes in combination with Pythia
outperforms **Pythia alone** in every workload category

Single-Core Performance Line Graph



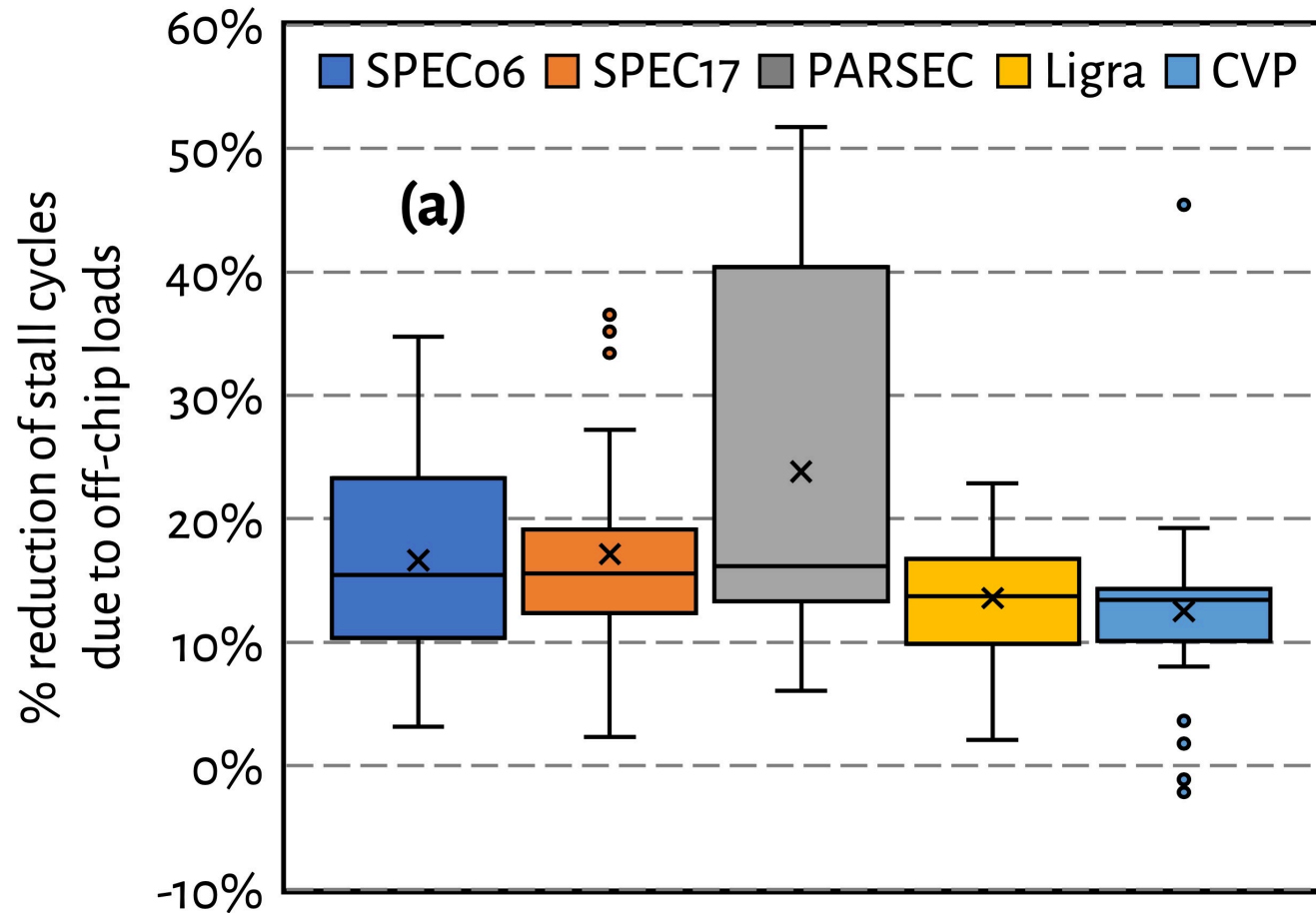
Single-Core Performance Against Prior Predictors



POPET provides higher performance benefit
than prior predictors

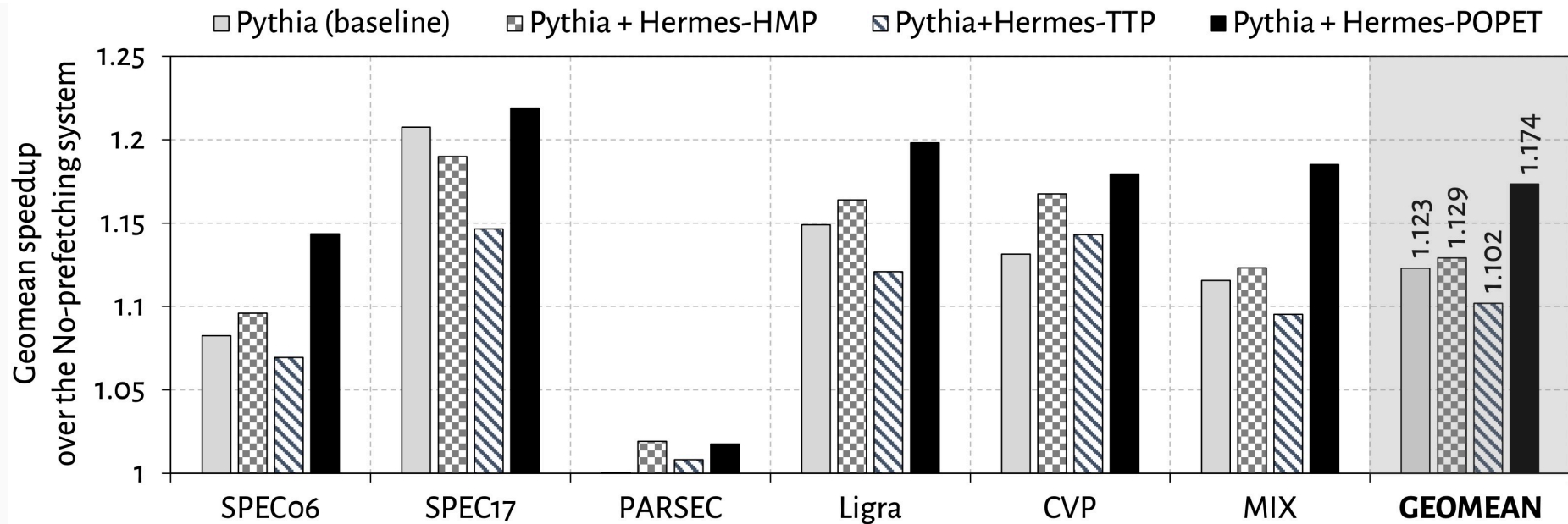
Hermes with POPET achieves nearly **90%** performance improvement of the Ideal Hermes

Effect on Stall Cycles



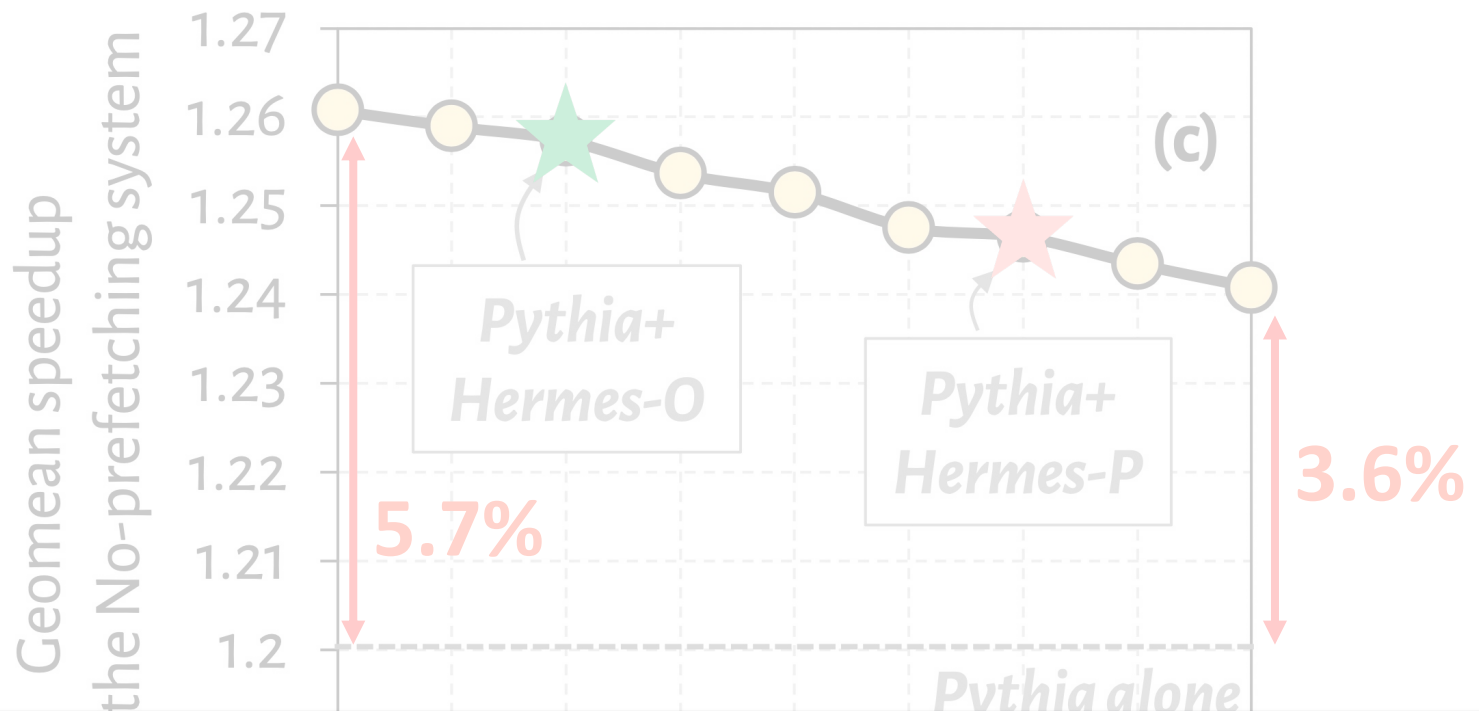
Hermes reduces off-chip load induced stall cycles on average by **16.2%** (up-to **51.8%**)

Eight-Core Performance



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

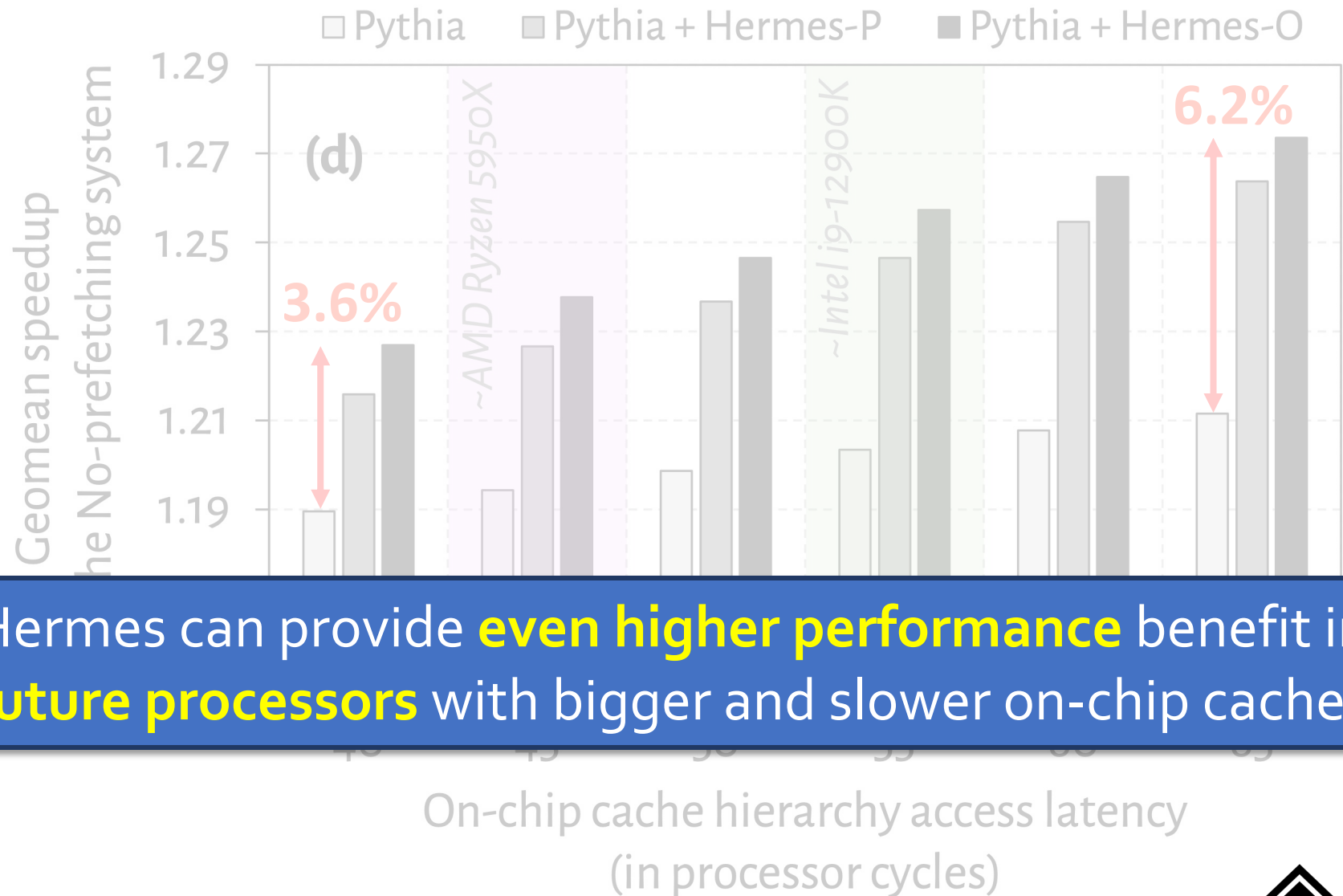
Effect of Hermes Request Issue Latency



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

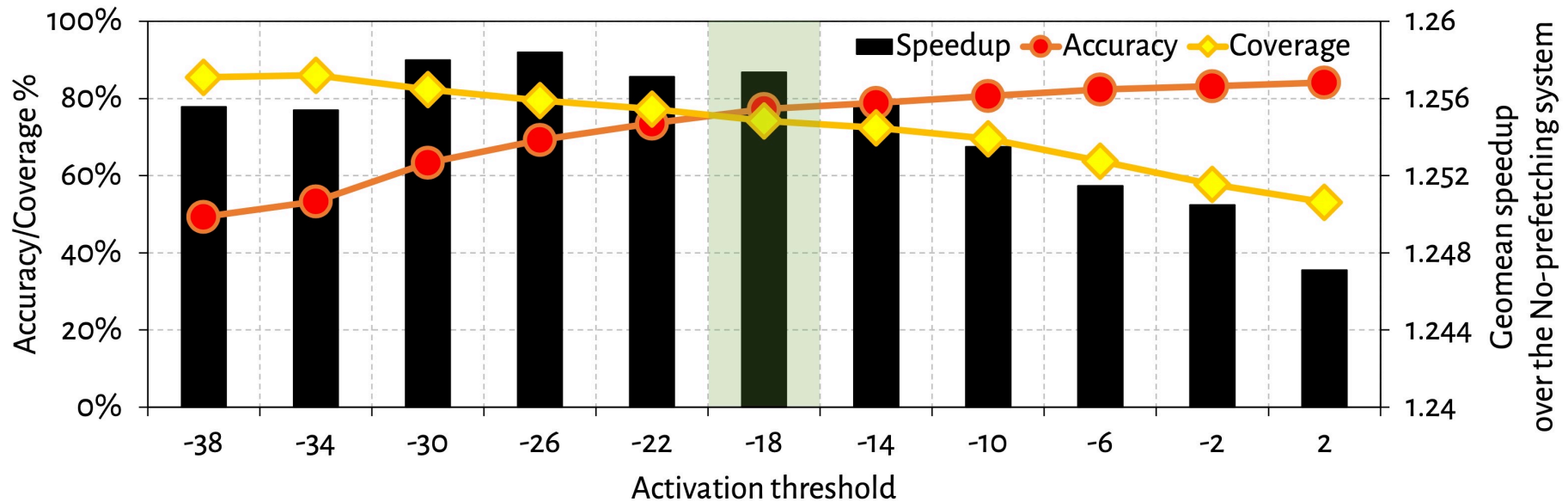
Hermes request issue latency
(in processor cycles)

Effect of Cache Hierarchy Access Latency



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

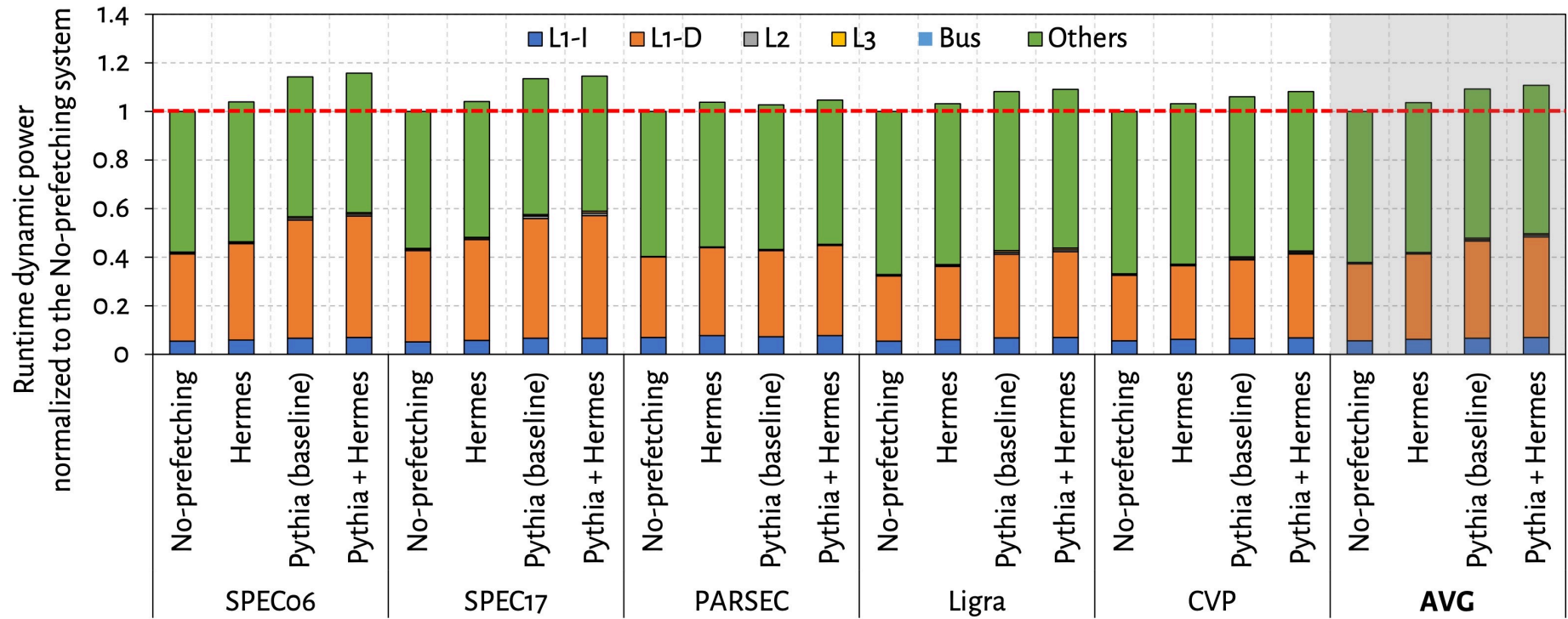
Effect of Activation Threshold



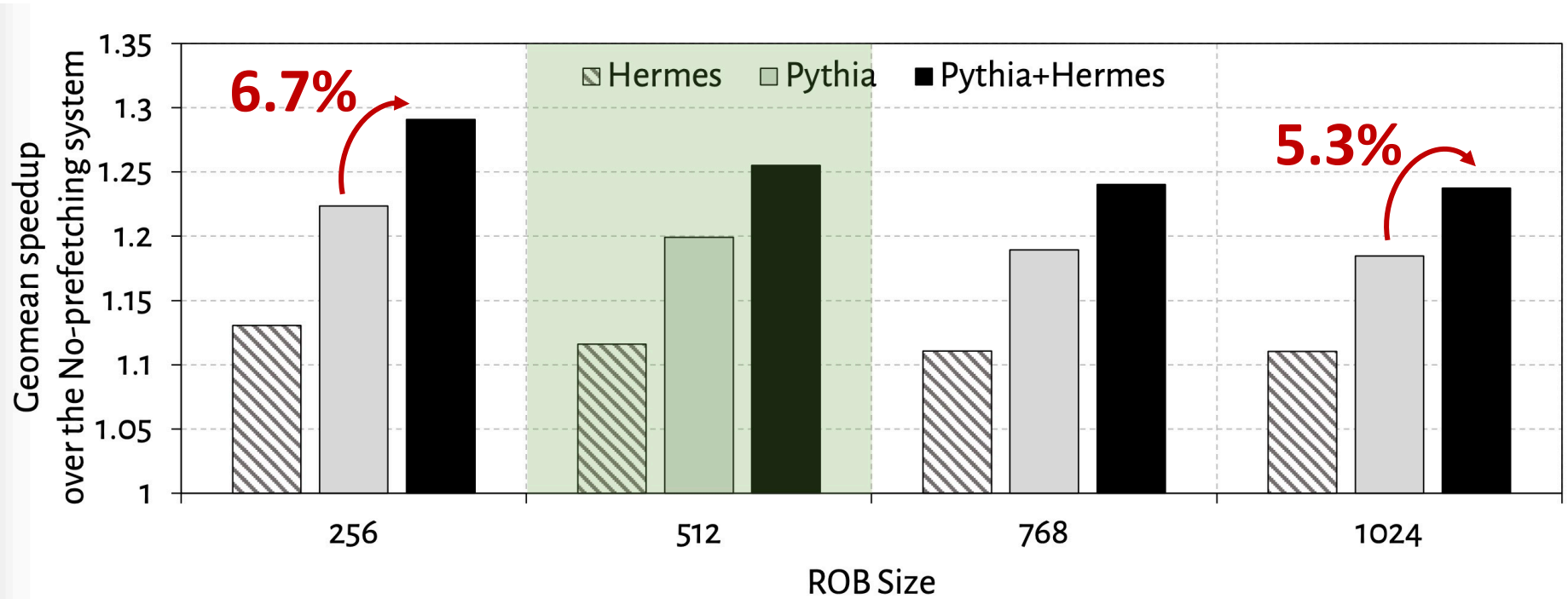
With increase in activation threshold

1. Accuracy increases
2. Coverage decreases

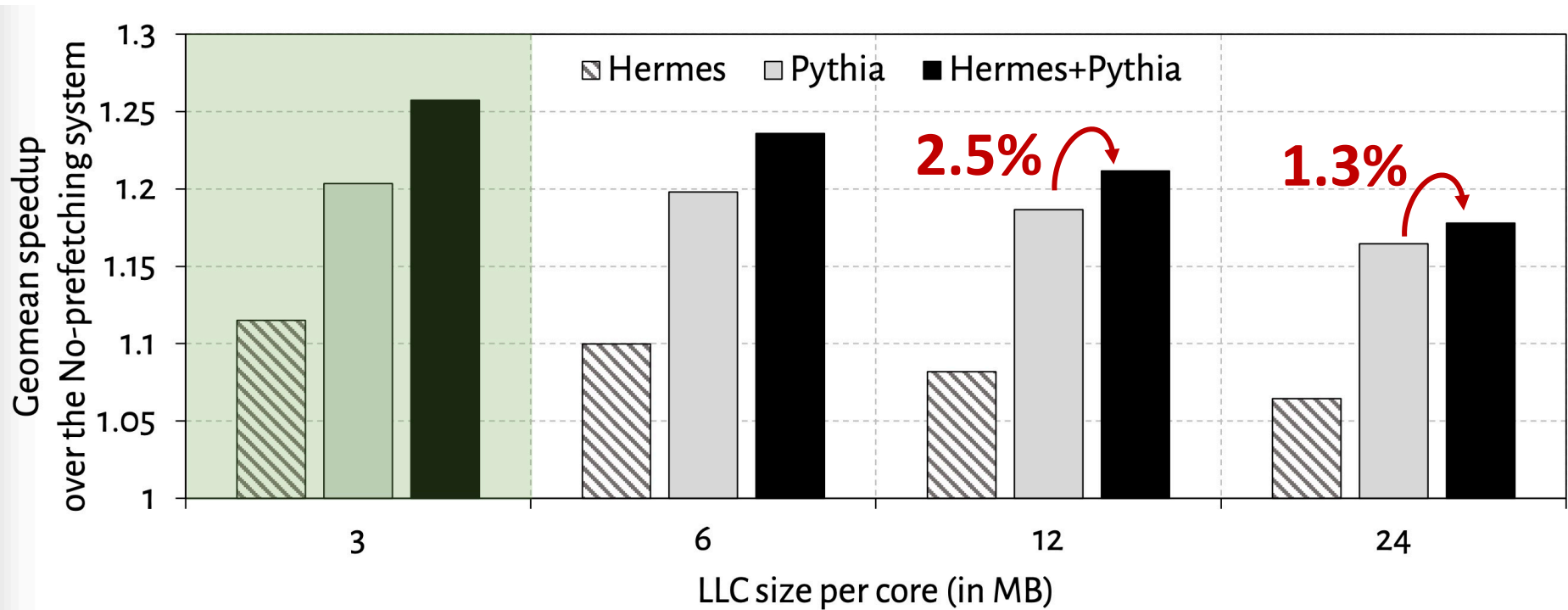
Power Overhead



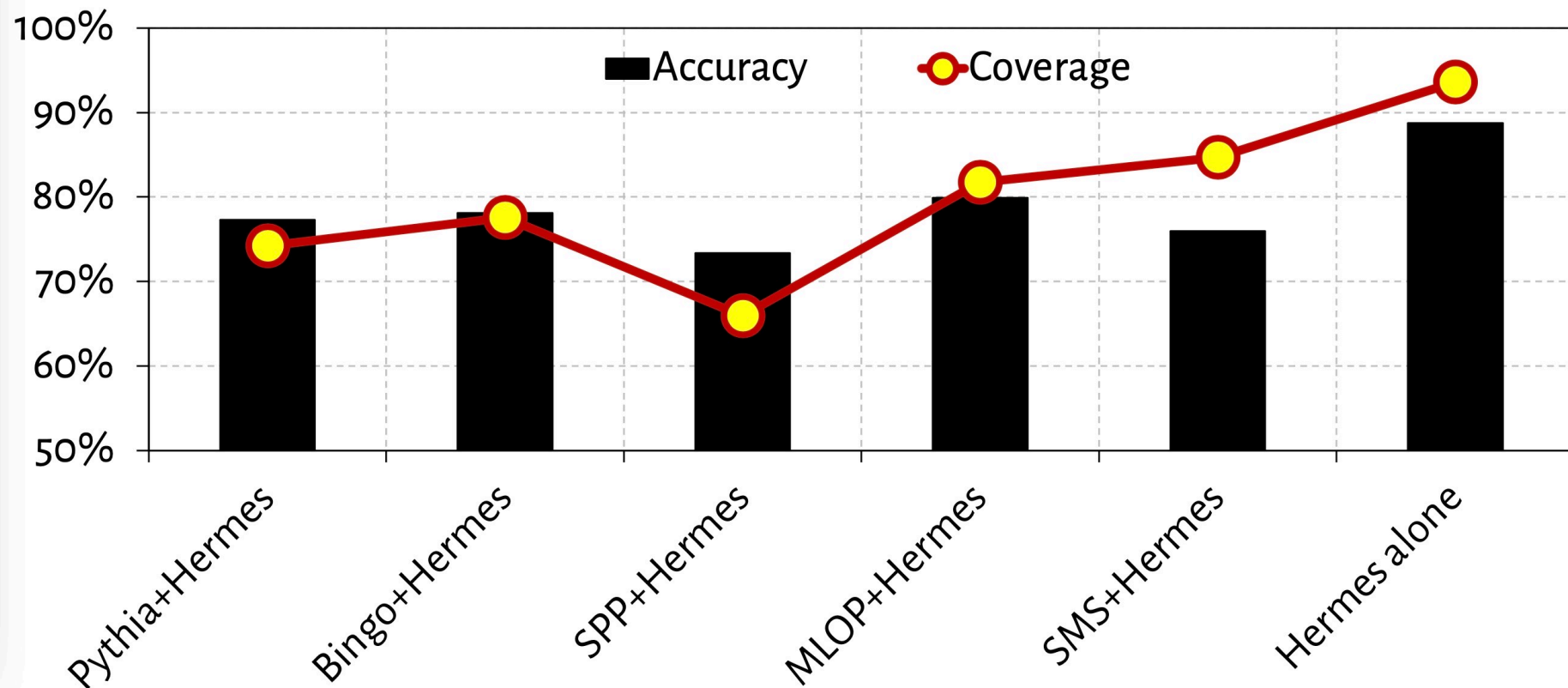
Effect of ROB Size



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

