ASPLOS

NC STATE UNIVERSITY

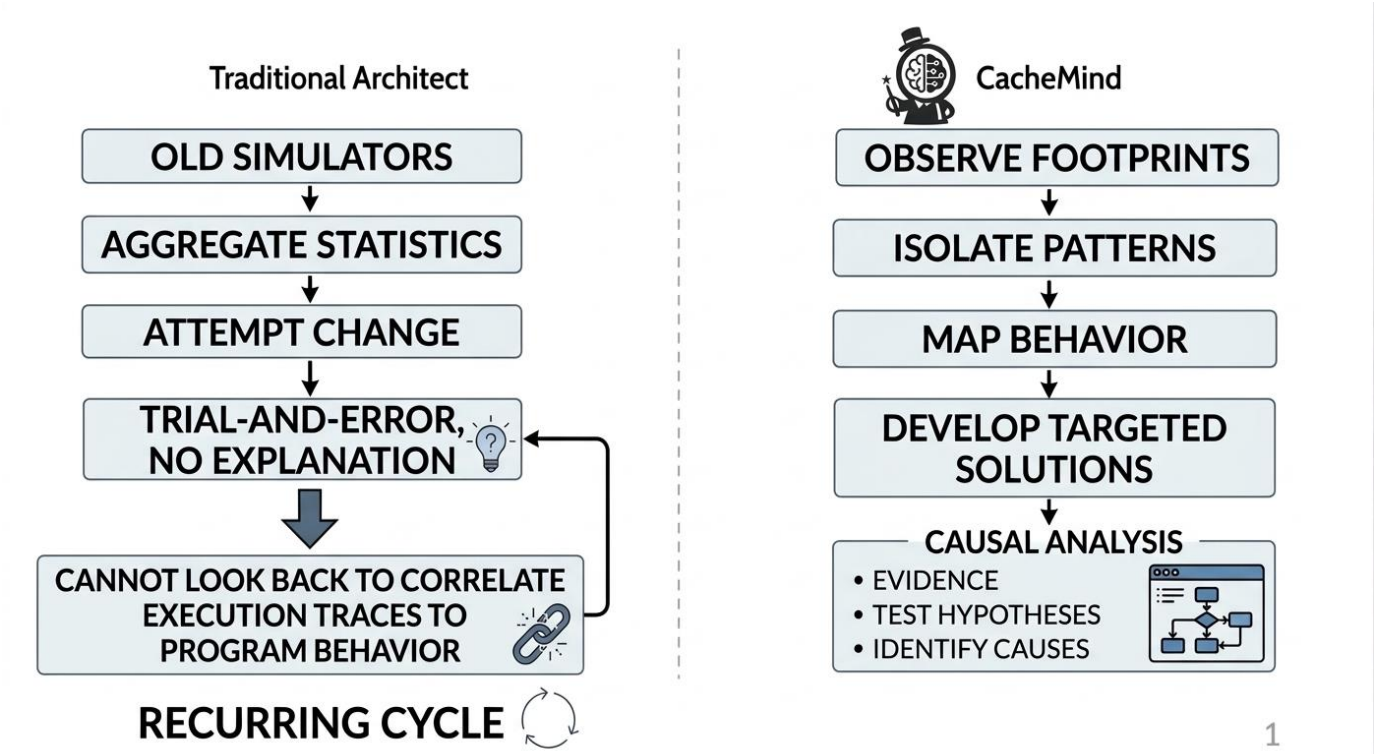
CacheMind

From Miss Rates to Why

Natural-language, Trace-grounded Reasoning for Cache Replacement

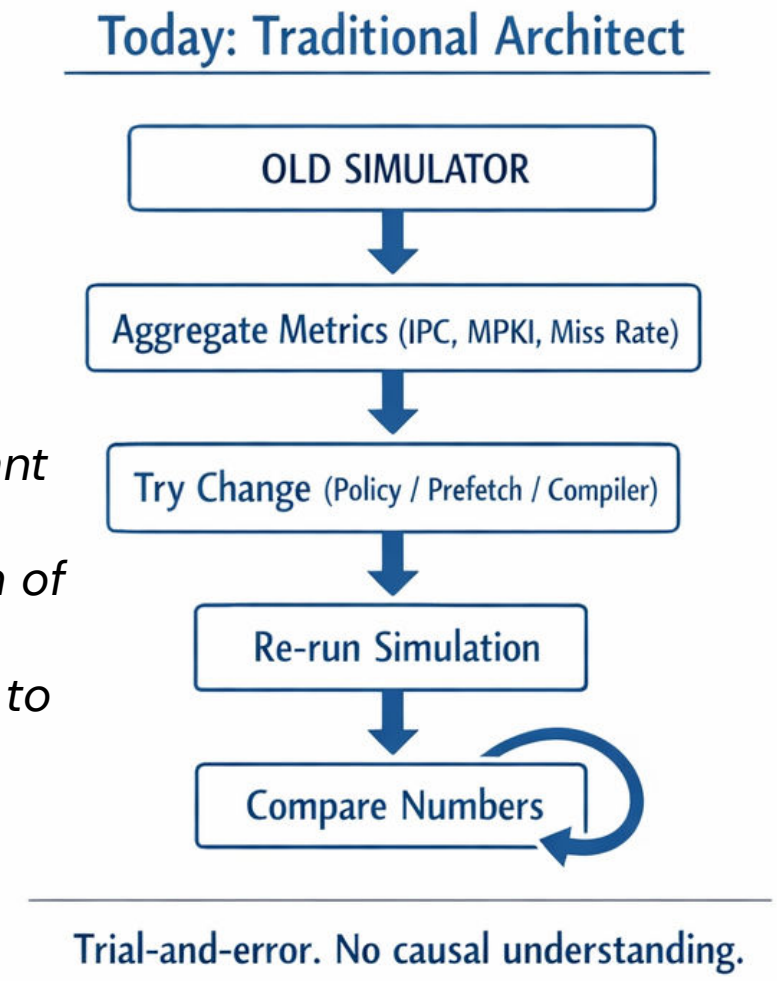
Kaushal Mhapsekar, Azam Ghanbari, Bita Aslrousta, Samira Mirbagher Ajorpaz

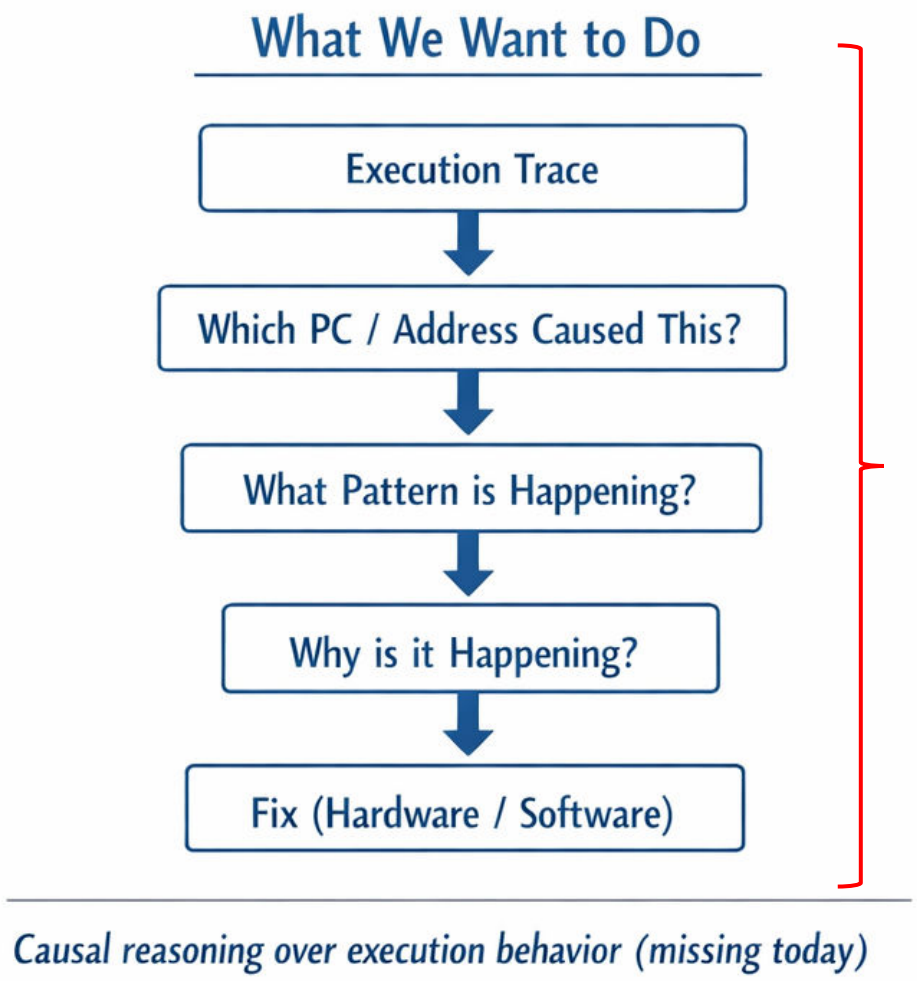
Great architects do not design cache optimizations from aggregate counters alone; they isolated execution patterns, mapped them back to program behavior, and invented mechanisms that captured those patterns at runtime. CacheMind is the first attempt to bring that human reasoning pipeline into the simulator.



Caches are still optimized through *trial-and-error over weak signals*, not causal reasoning

No assistant for close inspection of SW execution to cache footprint exist.





Great architects do this mentally.

Our tools do not support it.

CacheMind assist with this.

Aggregate metrics hide fundamentally different behaviors.

Workload A

MPKI = 20
IPC = 1.2

Streaming behavior → low reuse

Streaming workload. No reuse.
Prefetching helps.

Workload B

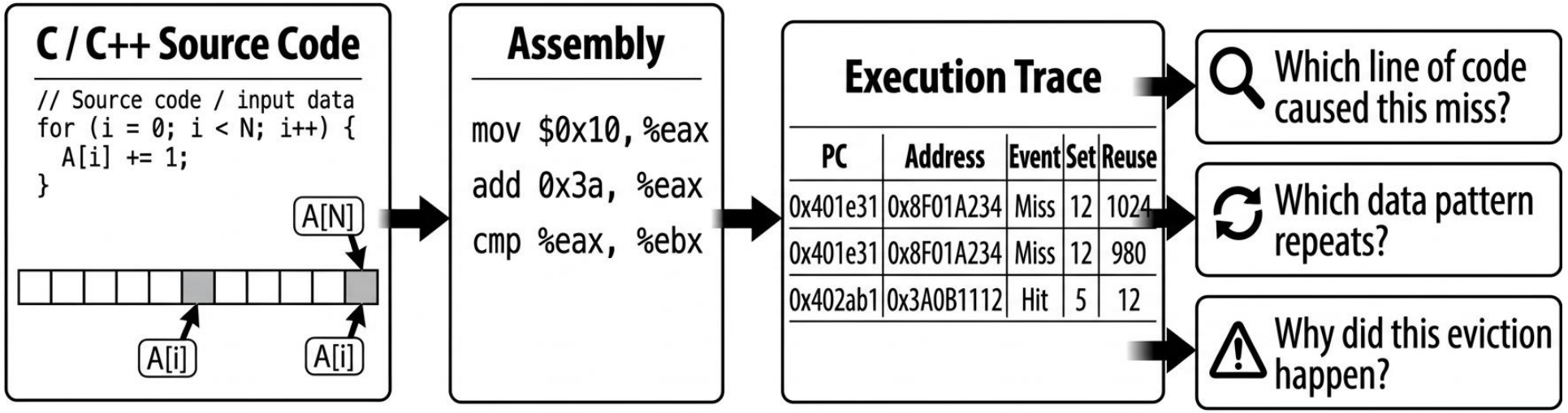
MPKI = 20
IPC = 1.2

Conflict behavior → high reuse potential

Conflict-heavy workload. Same MPKI.
Completely different problem. Reuse predictor helps.

Same numbers. Completely different root causes and solution needed.

The answer already exists in the execution trace



Identify the critical link between software logic and hardware execution.
What hardware behavior does this software pattern create?

Simulators already record the evidence. We still **cannot** automatically isolate it, connect it to code and data, and reason over it.

LLMs need retrieval to reason over traces

LLMs are not trained on traces

LLMs are not trained on microarchitectural simulator traces which are large and highly structured.

Traces are massive

Single simpoint traces contain ~100,000 accesses (~400 MB). Full suites can span hundreds of gigabytes.

Context windows are limited

LLM context can fit a few MBs. Precise retrieval is necessary for reasoning.

Why Existing RAG Fails on Trace Reasoning

Trace-grounded cache reasoning is a structured retrieval problem disguised as a language problem.

Embedding RAG

- Retrieves semantically similar chunks.
- May return wrong PC, wrong address, wrong policy

Textual similarity ≠ structural similarity

SQL / Static Query

- Can retrieve exact rows.
- But cannot decide what to query or decompose the question.

Cannot answer:
Which data pattern repeats?

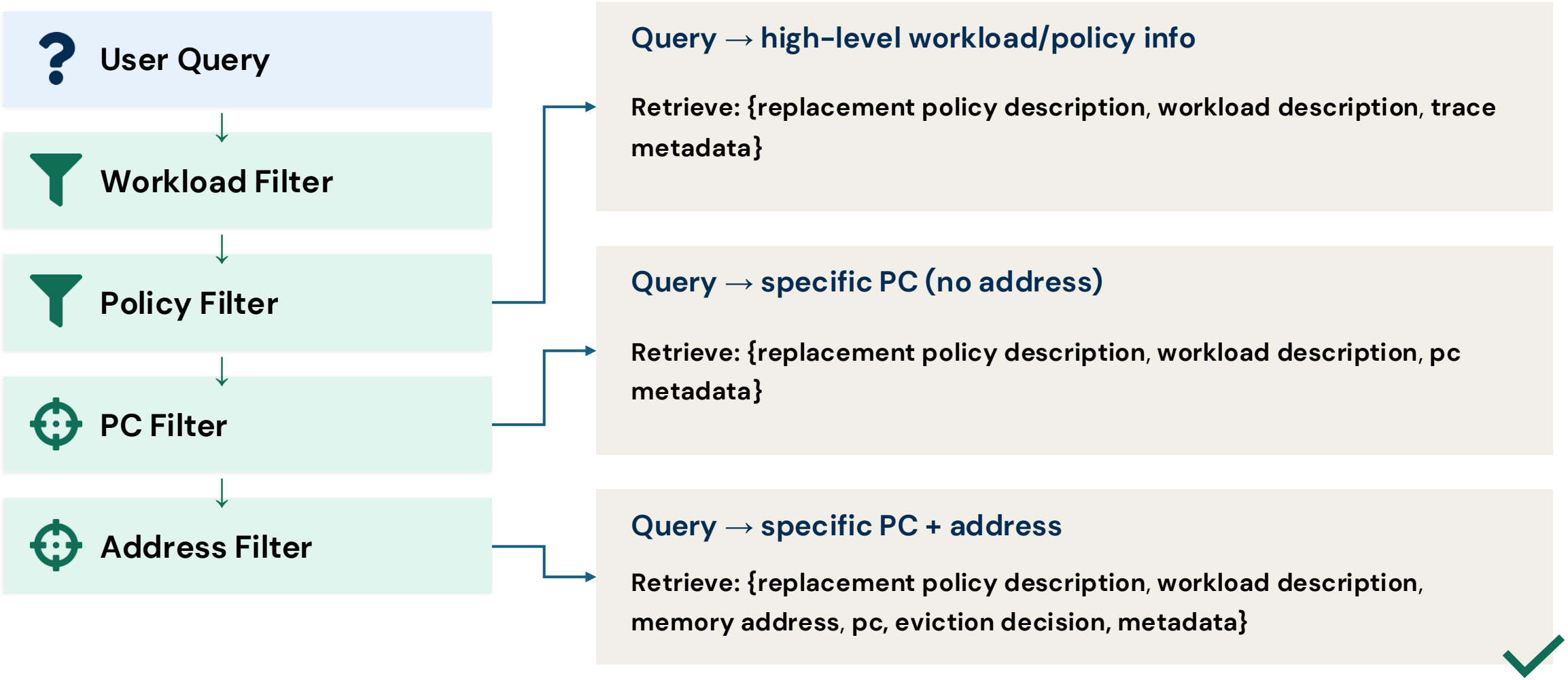
What CacheMind Needs

- Breaks the question into steps.
- Retrieves exact structured context. Then reasons over it.

Exact context + Reasoning

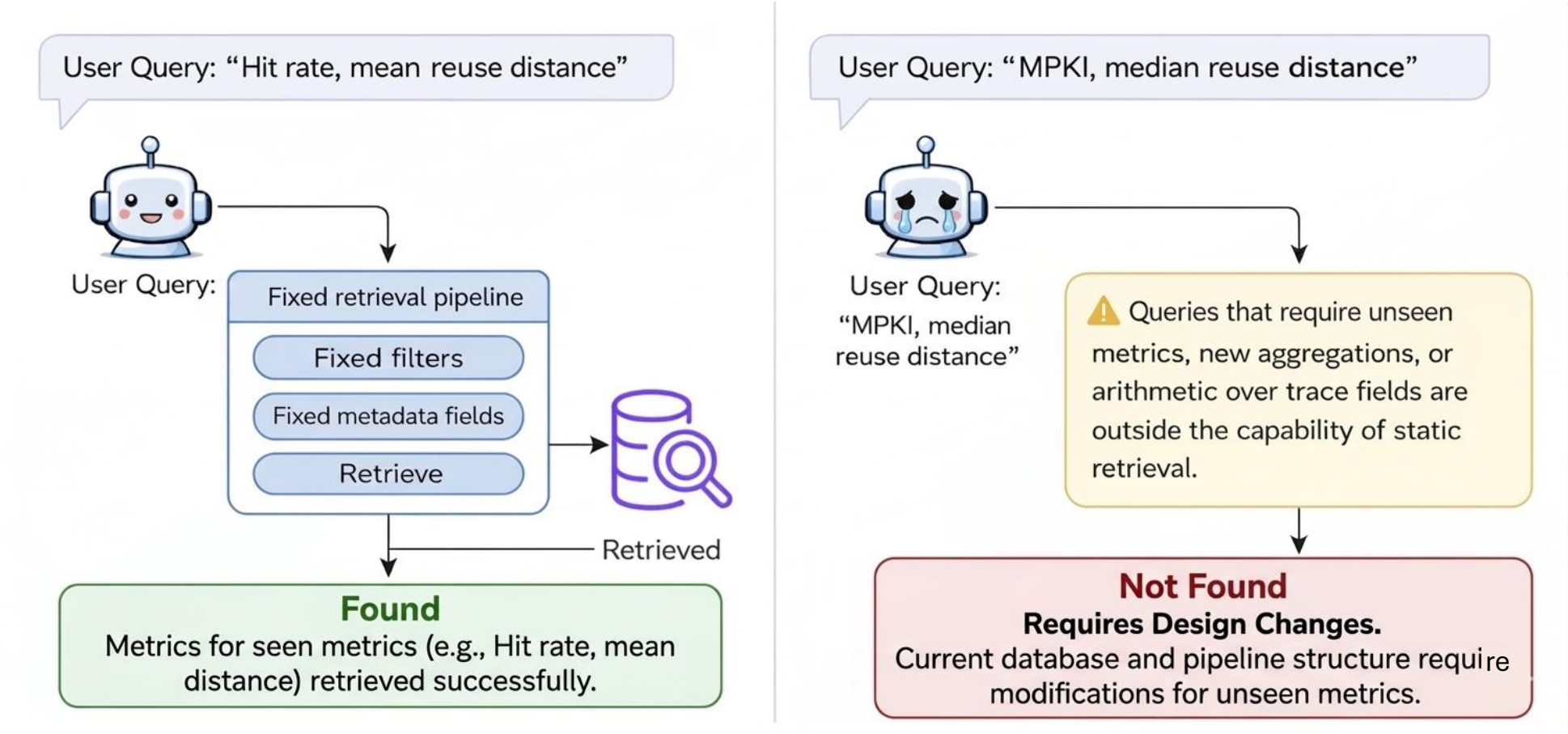
These questions look like language, but they require exact queries, decomposition over structured data and reasoning over structured trace data - not similarity or static queries.

SIEVE filters raw traces to a task-specific slice and returns the most informative evidence for the user's query.



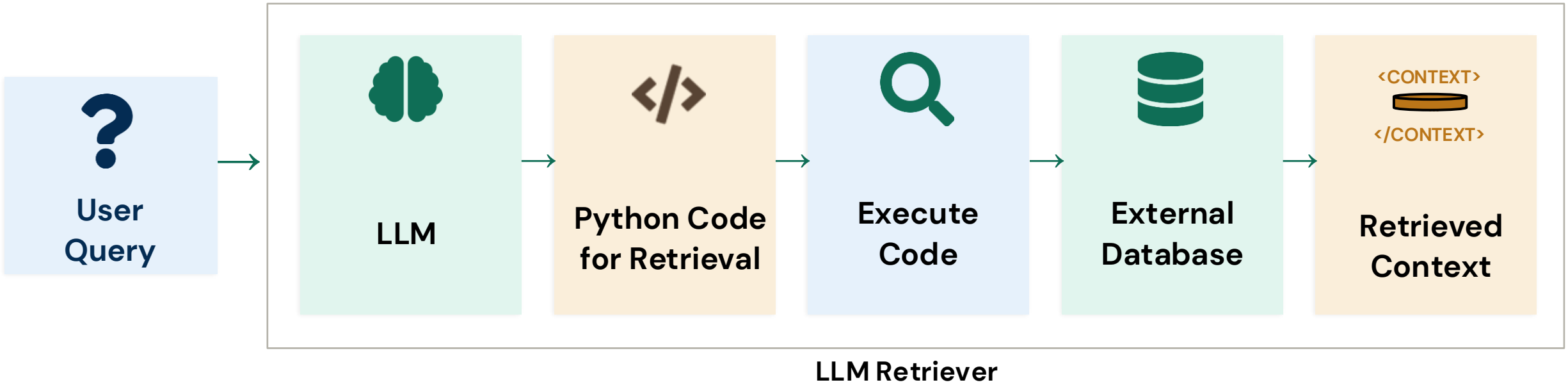
Symbolic-Indexed Entries for Verifiable Extraction

Fixed retrieval fails on unanticipated queries



Queries on new, unaccounted metrics require modifying the infrastructure pipeline.

CacheMind leverages LLM's code generation and reasoning capability to handle arbitrary queries

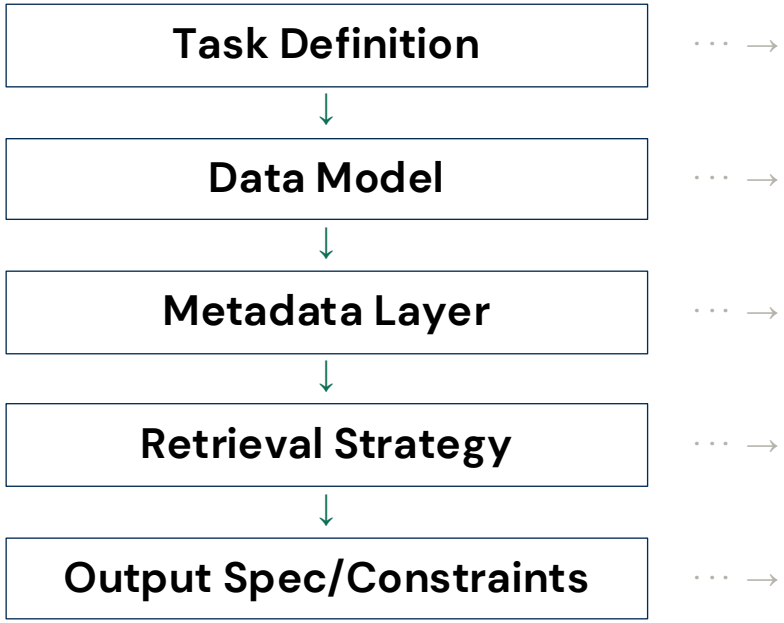


Key advantages of Ranger

- Handles unanticipated queries by dynamically generating code at inference time
- Adds a reasoning layer during retrieval — can calculate new metrics and statistics
- Maps natural language requests into structured data retrieval via code generation

Prompt as modular program specification

The prompt encodes goal, structure, retrieval logic, and output rules



Task Definition
Defines the goal
"Generate Python code for trace retrieval"

Data Model
Defines the structure
"loaded_data → data_frame, metadata, description"

Metadata Layer
Adds trace-level context
"miss rate, evictions, correlations"

Retrieval Strategy
Defines decomposition
"policy/workload → PC/address → metadata"

Output Spec
Enforces answer format
"result = \"...\""

Structured prompts turn free-form language into reliable retrieval programs.

Example trace excerpt retrieved by CacheMind

Cache Access Trace

PC: 0x405832 | Address: 0x2a9e6a48d9d | Set: 0b101100011101 | Evict: true

Cache Lines in Set

{0x2c919839d9d, 0x405832} {0x2c91983e59d, 0x405832} ...

Recent Access History

{0x3528e1a7d9d, 0x409228} {0x286af5fd59d, 0x409270} ...

Cache Line Eviction Scores

{3062739738013, 180} {3062739756445, 181} {3707401110941, 195}

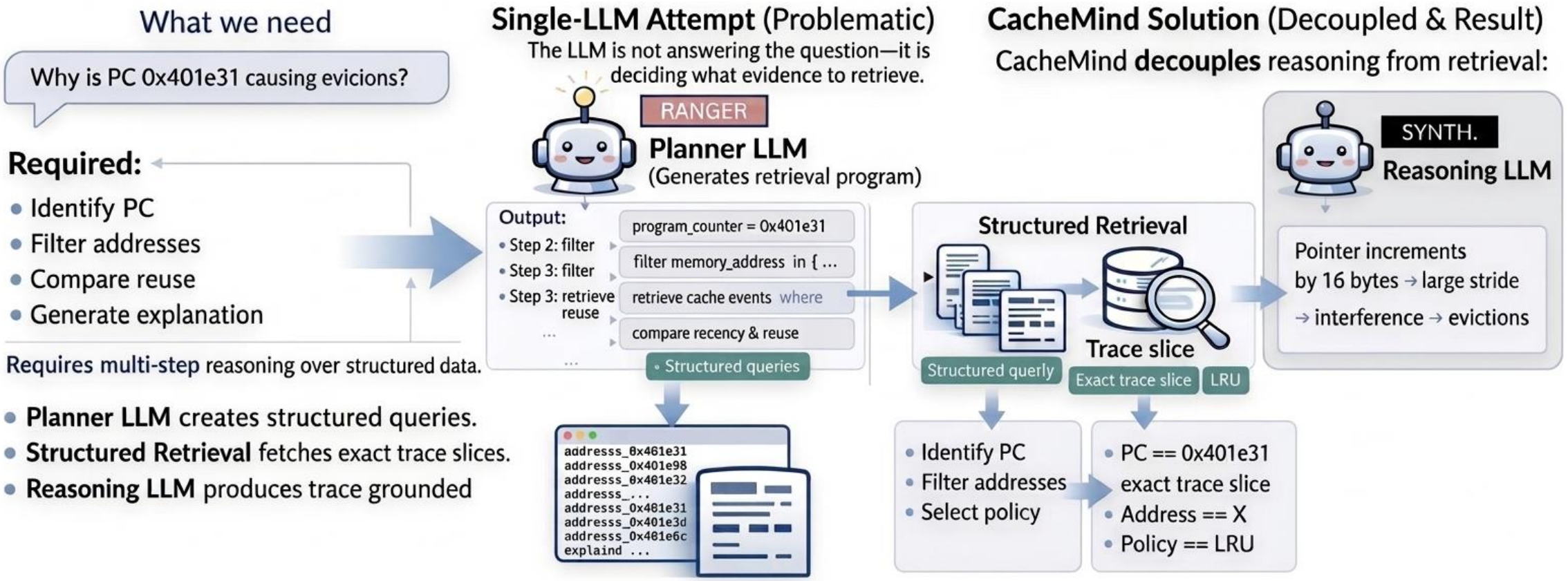
Assembly (mainSimpleSort)

```
405821: 84 c0      test %al,%al
405832: 0f 85      jne 4032d7 <mainSimpleSort+0xbd>
405839: eb 01      jmp 40336d <mainSimpleSort+0x153>
```

Sieve fixes RAG limitations; Ranger enhances rigid retrieval

LlamaIndex	Sieve	Ranger
Example Query		
When PC 0x409270 and address 0x2bfd401c63f is accessed on the astar workload with LRU policy, does the cache hit or miss?		
Example Retrieved Context		
TRACE_ID: astar_evictions_lru program_counter=0x409538, memory_address=0x2bfd401b693, evict=Cache Miss, ... **Match not found**	LRU + astar @ PC 0x409270, addr 0x2bfd401c63f: Cache result: Cache Miss Evicted addr: 0x31232a3ee3f (reuse: 3648 accesses). ... **Exact match found**	Result: Cache Miss for PC 0x409270 and addr 0x2bfd401c63f (workload: astar, policy: LRU). Function: createwayar ... **Exact match found**
Accuracy = **10%** | Avg time = 36.62s	Accuracy = **60%** | Avg time = 3.65s	Accuracy = **90%** | Avg time = 4.43s

CacheMind pipeline is a multi-step reasoning over structured data.



Benchmarking advances LLM reasoning in every field

Curated suites with unambiguous ground-truth answers have been the engine of measurable progress across domains TODO

GSM8K	MATH	ARC	DROP
Grade-school math	Competition math	Commonsense science	Discrete reasoning
Enabled evaluation of multi-step arithmetic reasoning in LLMs	Pushed boundaries of symbolic and algebraic problem solving	Tested scientific reasoning beyond pattern matching	Evaluated numerical reasoning over text passages

No equivalent benchmark exists for microarchitecture - **CacheMindBench** is the first verified, trace-grounded reasoning suite for cache replacement.

Existing architectural benchmark does not test *reasoning* over execution behavior

Static Q&A

QuArch

Tests high-level architectural knowledge using static, concept-grounded questions.

Example question

Q: ___ translates a logical address into a physical address.

a MMU ✓ | b Translator

c Compiler | d Linker

- Concept-based · Multiple choice · No trace needed

Trace-grounded

CacheMindBench

Evaluates trace-grounded retrieval and causal reasoning over real microarchitectural executions.

Example question

Q: What is the average **evicted reuse distance** of PC `0x40170a` for the `lbm` workload with MLP?

↳ *Requires access to real execution trace data*

- Trace-driven · Open-ended · Causal reasoning

CacheMindBench Establish trust in LLM reasoning by rigorously evaluating retrieved context and factual grounding.

Trace-grounded benchmark categories — 11 question types

QUANTITATIVE

Cache Hit/Miss

Determine if a specific access results in a hit or miss

Does PC `0x401dc9` and address `0x47ea85d37f` result in a cache hit in `lbm` under `Mokingjay`?

QUANTITATIVE

Miss Rate

Compute the miss rate of a specific PC or workload

What is the miss rate for PC `0x4037ba` in `mcf` with `PARROT`?

COMPARATIVE

Policy Comparison

Compare miss behavior across replacement policies

Which policy has the lowest miss rate for PC `0x409270` in `astar`?

QUANTITATIVE

Count

Count the frequency of events such as accesses or evictions

How many times did PC `0x405832` appear in `astar` under `LRU`?

QUANTITATIVE

Arithmetic

Perform arithmetic computations over trace statistics

What is the average evicted reuse distance of PC `0x40170a` for the `lbm` workload with `MLP`?

ADVERSARIAL

Trick Question

Catch inconsistencies or invalid assumptions in a query

Does PC `0x4037aa` in `lbm` access address `0x1b73be82e3f`?

CONCEPTUAL

Microarchitecture Concepts

Answer general cache questions using architectural logic

How does increasing cache size affect miss rate? Compare increasing #sets vs. #ways.

GENERATIVE

Code Generation

Generate code to analyze specific trace conditions

Write code to compute hits for PC `0x4037ba` and address `0xa3a0df3d9d` in `mcf` under `LRU`.

COMPARATIVE

Replacement Policy

Analyze the behavior of replacement policies across contexts

Why does `Belady` outperform `LRU` on PC `0x409270` in `astar`?

COMPARATIVE

Workload Analysis

Reason about entire workload characteristics and trends

Which workload has the highest cache miss rate under `MLP`?

CONCEPTUAL

Semantic Analysis

Connect program counters to high-level program behavior

Why does PC `0x4037ba` have a high hit rate? Examine the assembly context and analyze.

To evaluate the CacheMind system and future conversational AI tools, we created a suite of 100 verified benchmark questions.

Trace-Grounded Tier: Testing fine-grained retrieval

Category	Description	Example Question
Hit/Miss	Determine if a specific access results in a hit or miss	*Does PC 0x401dc9 and address 0x47ea85d37f result in a cache hit in lbm under PARROT?*
Miss Rate	Compute miss rate of a specific PC or workload	*What is the miss rate for PC 0x4037ba in mcf with PARROT?*
Policy Comparison	Compare miss behavior across replacement policies	*Which policy has the lowest miss rate for PC 0x409270 in astar?*
Count	Count frequency of events (accesses, evictions)	*How many times did PC 0x405832 appear in astar under LRU?*
Arithmetic	Perform arithmetic over trace statistics	*What is the avg evicted reuse distance of PC 0x40170a for lbm with MLP?*
Trick Question	Catch inconsistencies or invalid assumptions	*Does PC 0x4037aa in lbm access address 0x1b73be82e3f? (Answer: TRICK)*

Binary scoring (1 = correct, 0 = incorrect) using exact-match verification against the trace.

Architectural Reasoning Tier: Testing LLM reasoning

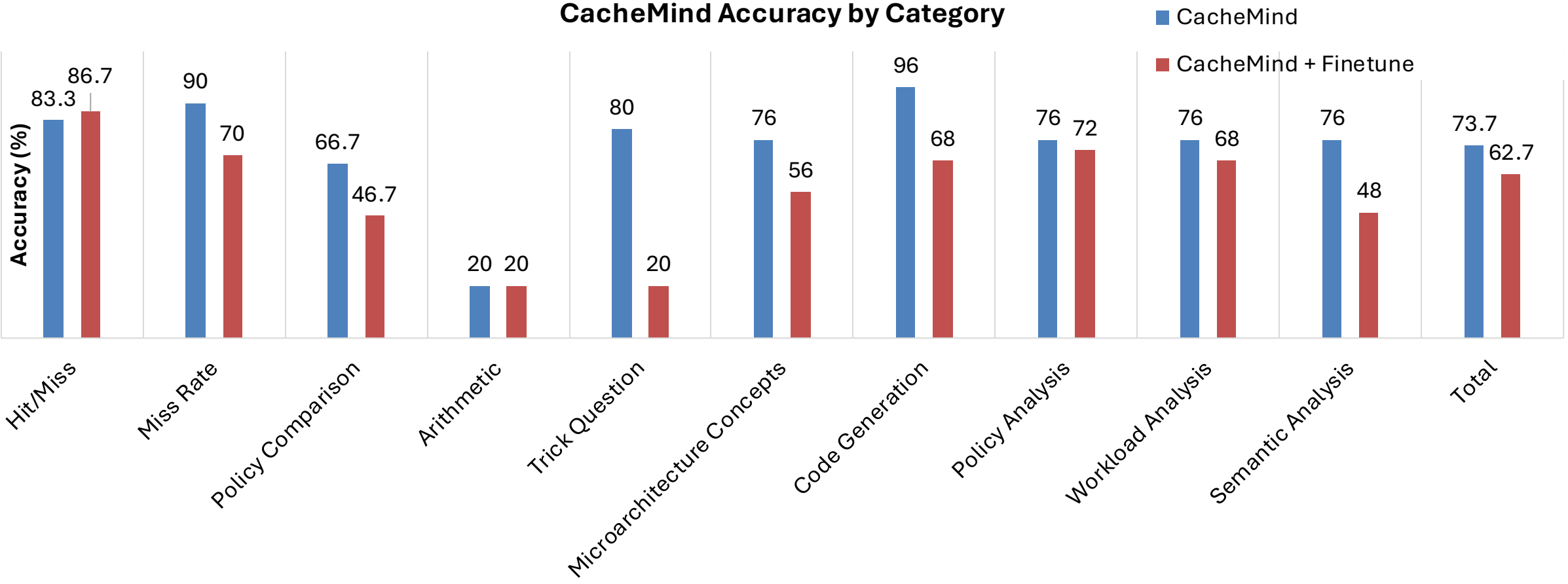
Category	Description	Example Question
Microarch. Concepts	Answer general cache questions with architectural logic	*How does increasing cache size affect miss rate? Compare increasing #sets vs. #ways.*
Code Generation	Generate code to analyze specific trace conditions	*Write code to compute hits for PC 0x4037ba and address 0xa3a0df3d9d in mcf under LRU.*
Policy Analysis	Analyze behavior of policies across contexts	*Why does Belady outperform LRU on PC 0x409270 in astar?*
Workload Analysis	Reason about entire workload characteristics	*Which workload has the highest cache miss rate under MLP?*
Semantic Analysis	Connect PCs to high-level program behavior	*Why does PC 0x4037ba have a high hit rate? Examine the assembly context and analyze.*

Qualitative scoring (0–5) based on correctness, use of evidence, and clarity - judged by a microarchitect.

Experimental Setup

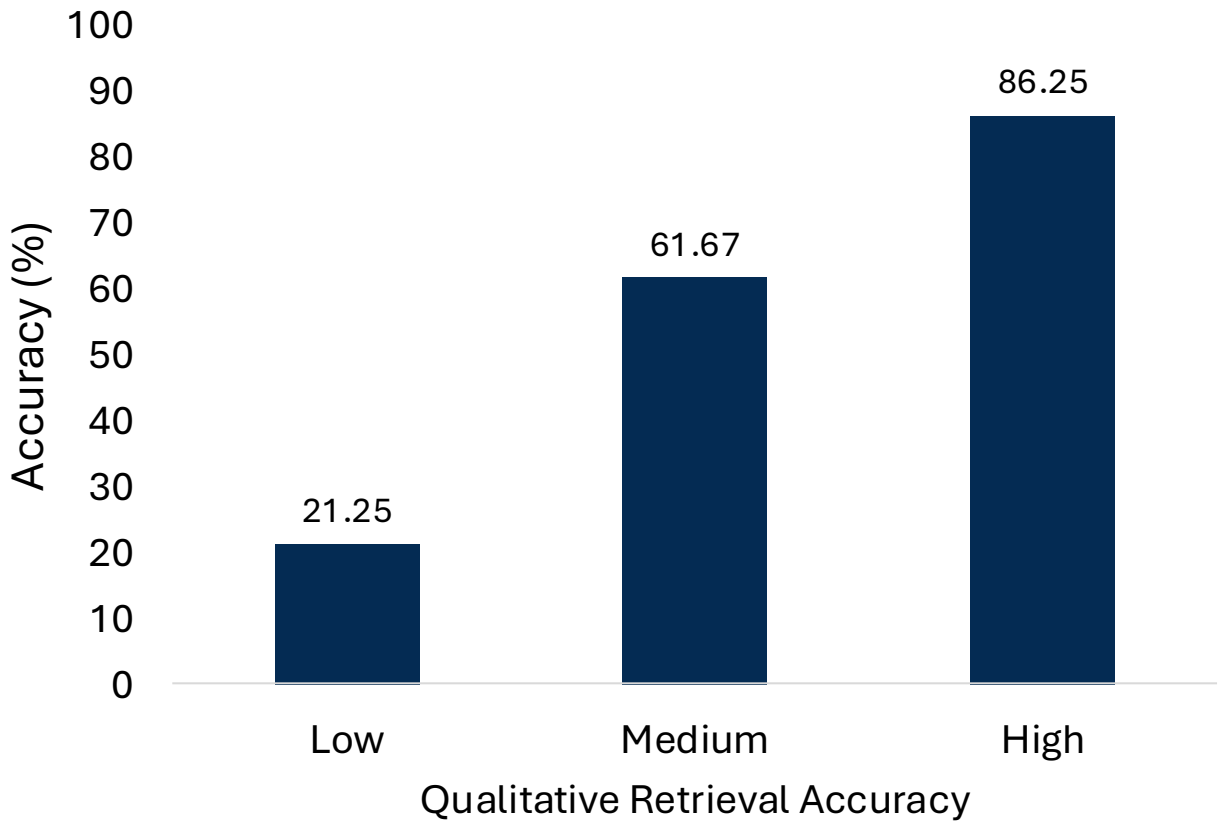
Category	Subsystem	Configuration
Simulation Platform	ChampSim	Trace-based
	gem5 extension	Used for software intervention studies
Workloads	Trace source	SPEC CPU 2006 / CRC-2 derived traces
	Trace fields	PC, address, hit/miss, eviction
Execution Setup	Run length	1B instructions
	Analysis target	Event-level cache behavior
Replacement Policies	Baselines	LRU, Belady, Parrot, Multi-layer perceptron
	Learned	6-wide fetch/decode/execute; 4-wide retire
Hardware Configuration	Core	1 core, 4 GHz
	Pipeline	6-wide fetch/decode/execute; 4-wide retire
	OoO structures	352 ROB, 128 LQ, 72 SQ
	Predictor	bimodal
Hardware Configuration	L1 I	32 KB, 64 sets, 8 ways, 4 cyc, 8 MSHR
	L1 D	32 KB, 64 sets, 8 ways, 4 cyc, 16 MSHR
	LLC	2 MB, 2048 sets, 16 ways, 26 cyc, 64MSHR
	DRAM	4 GB DDR4-3200, 1 ch, 1 rank/ch, 8 banks/rank

Finetuning drops the performance by 11%



We observed that finetuning narrows generalization and increases hallucination risk

Retrieval quality dominates reasoning performance

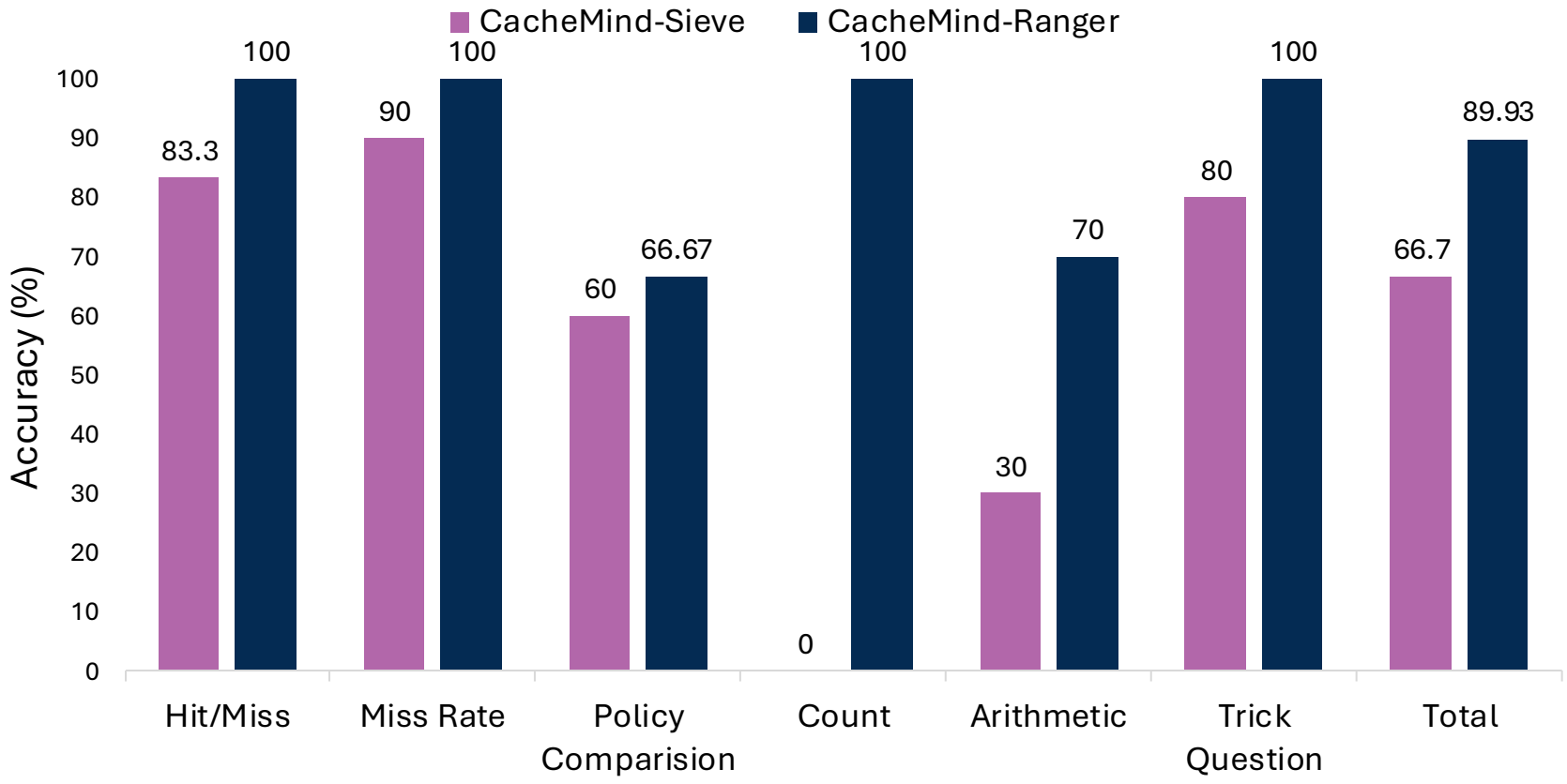


Low = insufficient context retrieved for reasoning
High = sufficient context retrieved for reasoning

Better retrieval leads to more accurate and consistent outputs.

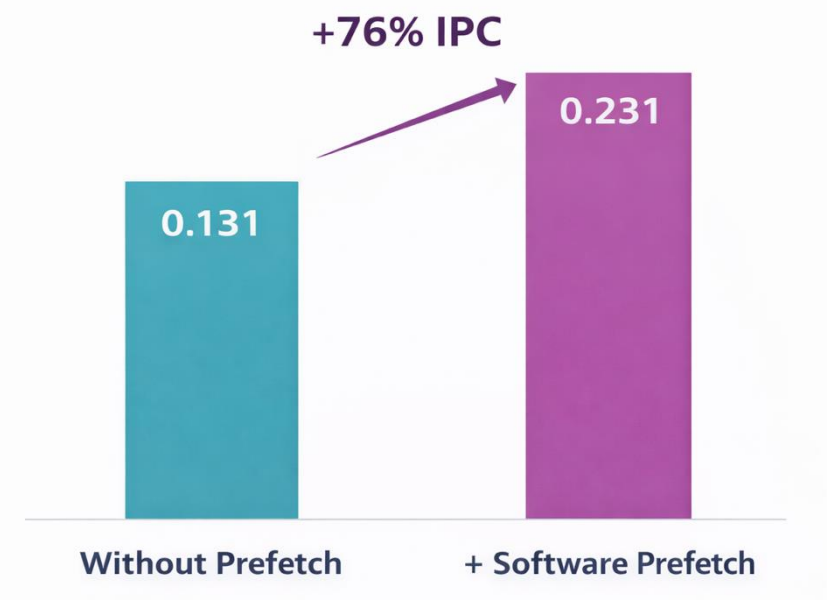
CacheMind Ranger solves retrieval bottleneck

SIEVE limitation - works well for structured queries but doesn't do well for open-ended dynamic queries.



Proposed: RANGER, an LLM-based retriever that writes and executes python code to retrieve relevant data from the database, based on user queries and knowledge of database schema.

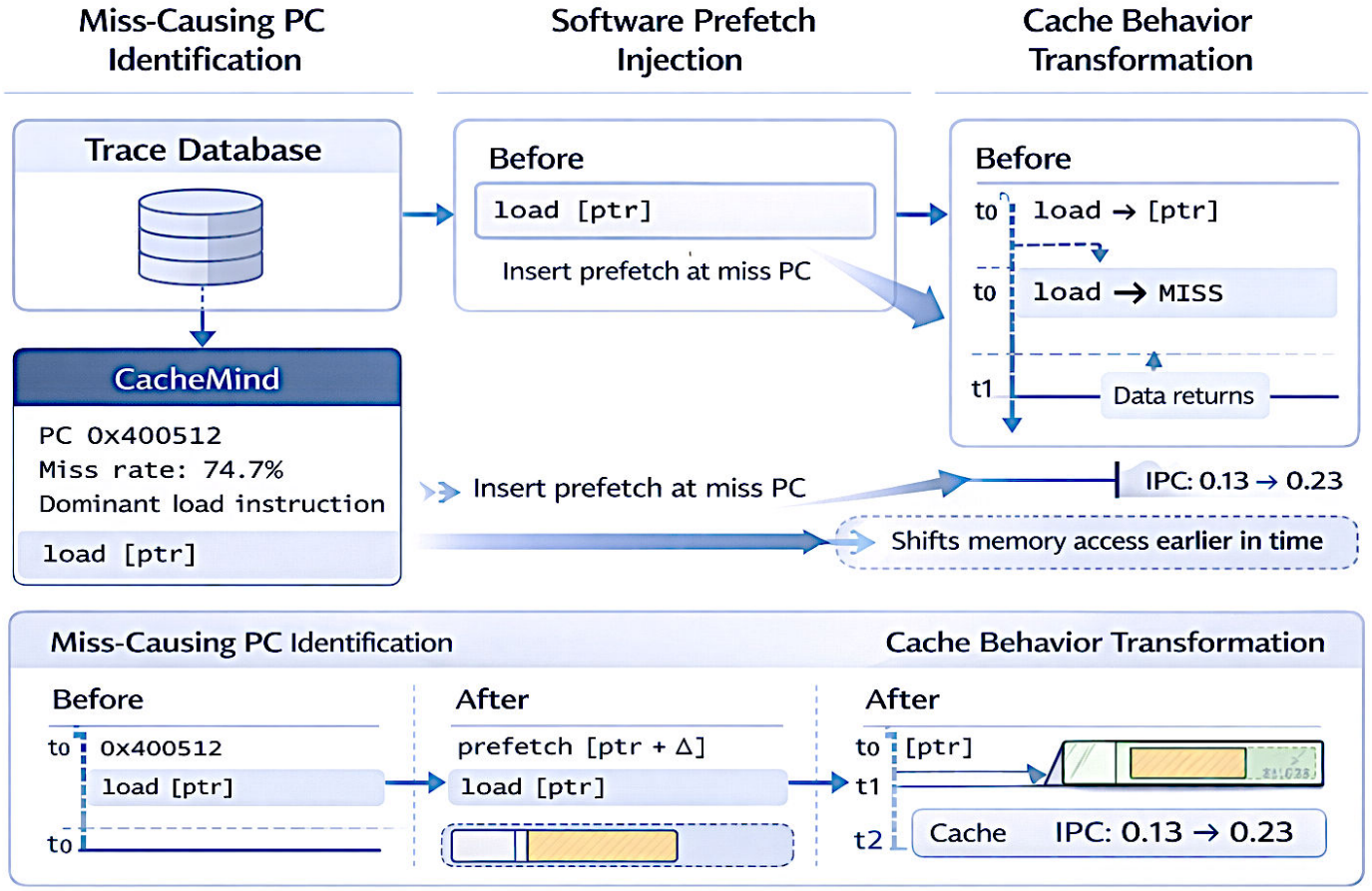
CacheMind for SW/HW optimizations



User: List all unique PCs in the given trace.
Assistant: `0x400444, 0x400701, 0x400709, . . .`

User: From the unique PCs, identify the PC causing the most cache misses.
Assistant: `0x400512`.

User: What is the miss rate of PC `0x400512`?
Assistant: **74.73%** miss rate.

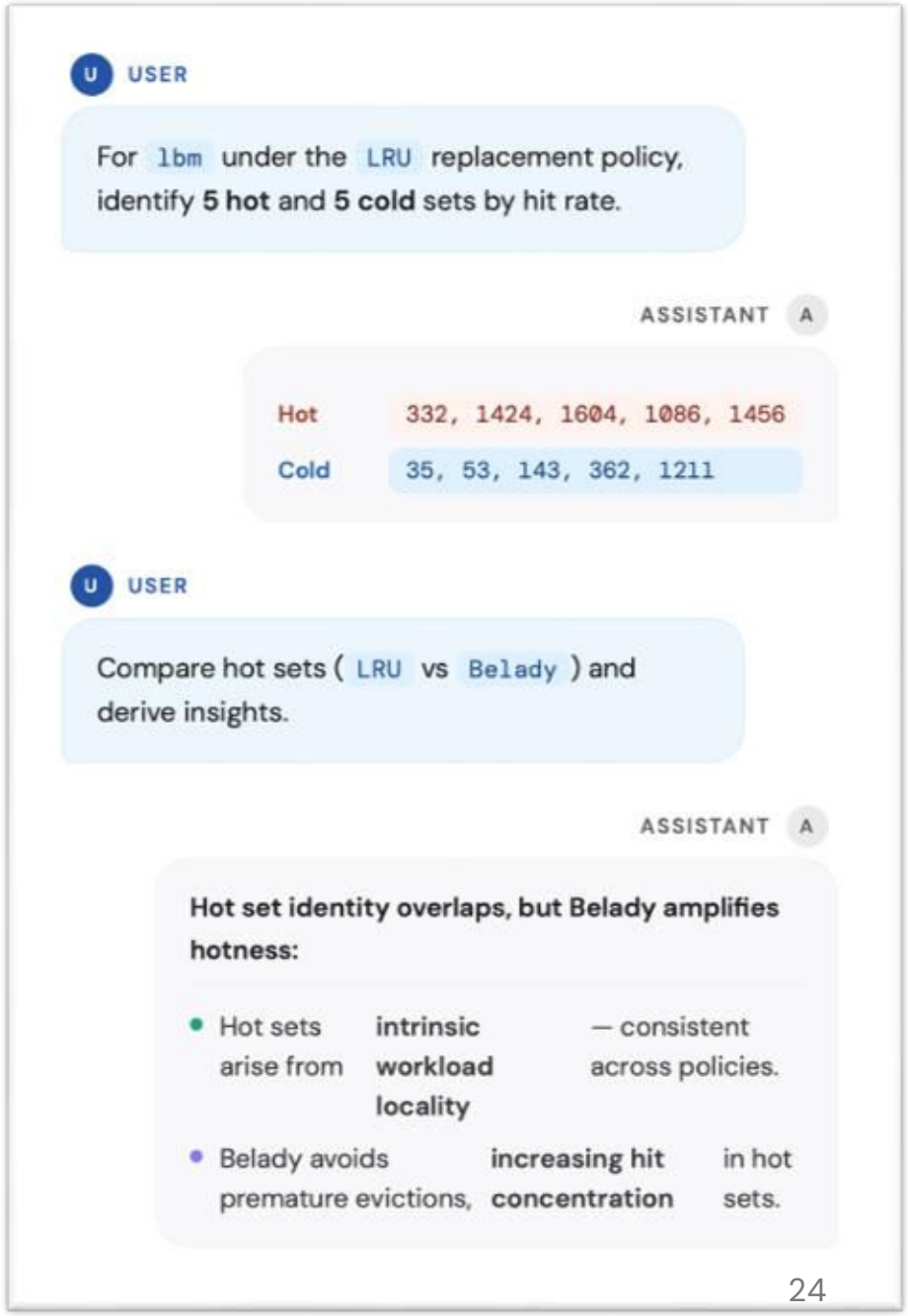


CacheMind identifies the PC that causes high amount of cache misses in a custom pointer chasing benchmark. Prefetching the memory accesses for this PC at code level boosts IPC by 76%

CacheMind can reveal set-level behavior for cache analysis

Hot set analysis is beneficial for replacement policies like Mockingjay, Hawkeye, SHiP, SDBP, DRRIP, DIP, which *sample cache sets*

CacheMind insight: For astar workload, set hotness depends on workload behavior, not the replacement policy's decisions.

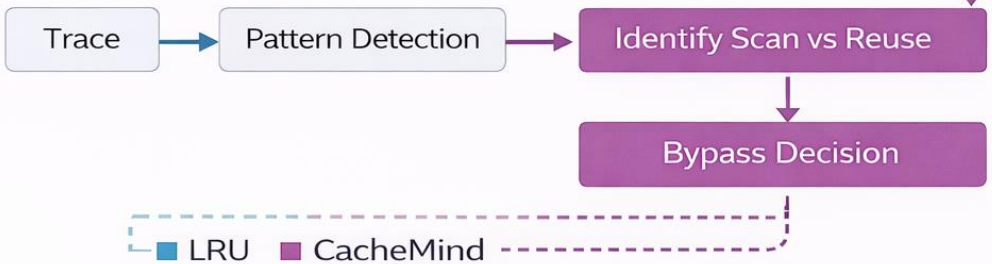


Optimizing Replacement Policy with targeted Bypass

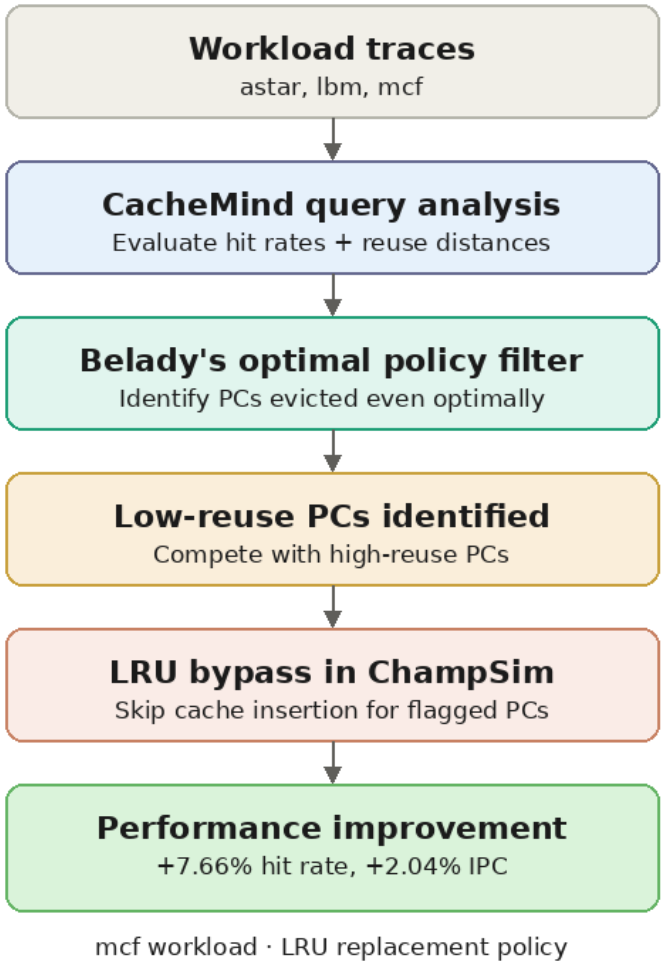
CacheMind enables targeted bypass via trace-grounded insight

Insight:

- Streaming + reuse interleaving
- Scan traffic pollutes cache
- Long-horizon reuse is evicted

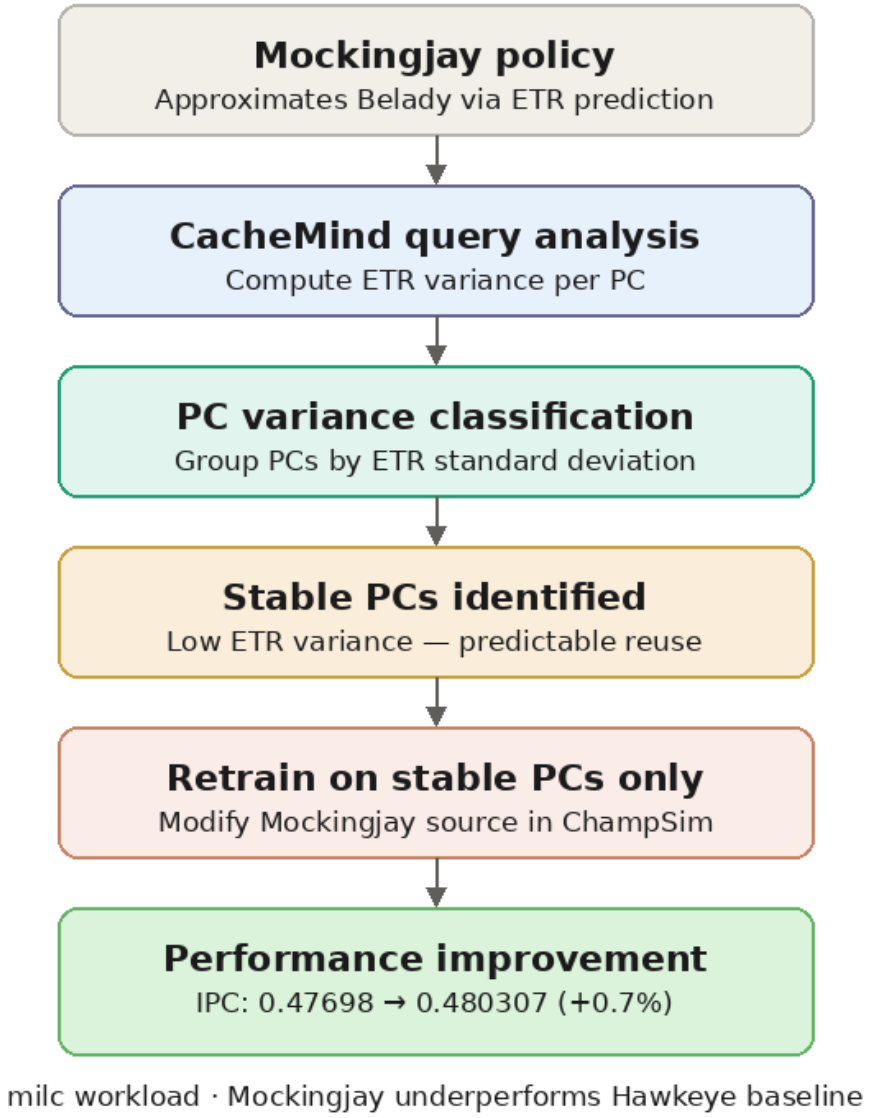


By separating scan and reuse patterns, CacheMind preserves long-horizon reuse.



mcf workload · LRU replacement policy

CacheMind identifies potential optimizations in SOTA replacement policies from just the paper summary and memory access traces!



Conclusion

Next-generation simulators must move beyond ***fixed metric*** outputs to engines that can answer ***arbitrary, interactive questions*** about any event in the simulated execution.

90%

Retriever accuracy on trace-grounded questions

9×

Better retrieval vs. LlamaIndex

100

Verified benchmark questions

4

Actionable insights highlighting utility (76% speedup)

CacheMind demonstrates this concept for cache replacement - a microarchitectural microscope that enables free-form, per-PC, per-data exploration.