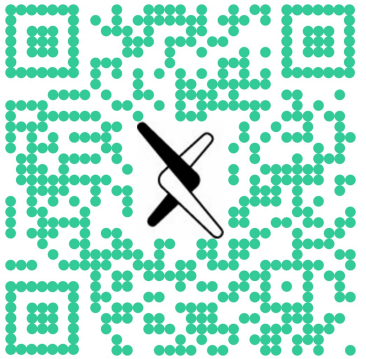
ArchAgent

AGENTIC AI-DRIVEN COMPUTER ARCHITECTURE DISCOVERY

Raghav Gupta[1†], Akanksha Jain[2], Abraham Gonzalez[2],
Alexander Novikov[3], Po-Sen Huang[3], Matej Balog[3], Marvin Eisenberger[3],
Sergey Shirobokov[3], Ngân Vũ[3], Martin Dixon[2],
Borivoje Nikolić[1], Parthasarathy Ranganathan[2], Sagar Karandikar[1]

[1]University of California, Berkeley
[2]Google [3]Google DeepMind



Motivation

Slowdown of classical compute scaling

Exploding compute demand

Specialization across the stack

What's next?

Goal

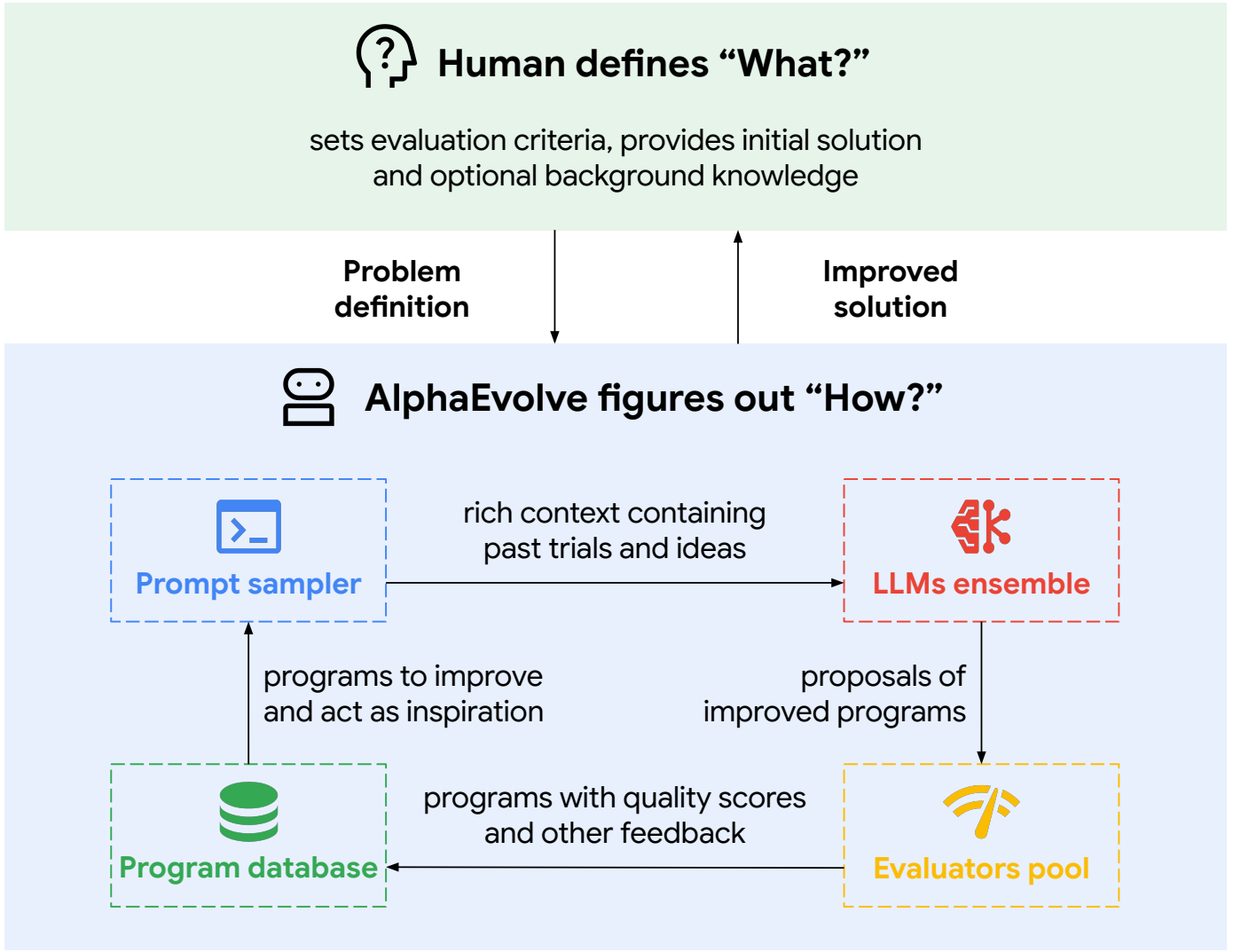
Can we use agentic AI techniques
to discover new
(micro)architectural mechanisms?

AlphaEvolve

An evolutionary coding agent for scientific and algorithmic discovery

Evolutionary Search

Driven by LLMs



Computer Architecture Competition Setup

Target

Cache Replacement
Branch Prediction
Data Prefetching
Value Prediction

Evaluator

Software μarch simulator
- Trace-driven
- O(10 KIPS)

ChampSim, gem5

Benchmarks

SPEC, GAP, hyperscale traces (Google, Qualcomm), etc.

Sampled traces

Metrics

IPC
A storage budget

Computer Architecture Competition Setup

Target

Last Level Cache
Replacement

Evaluator

ChampSim

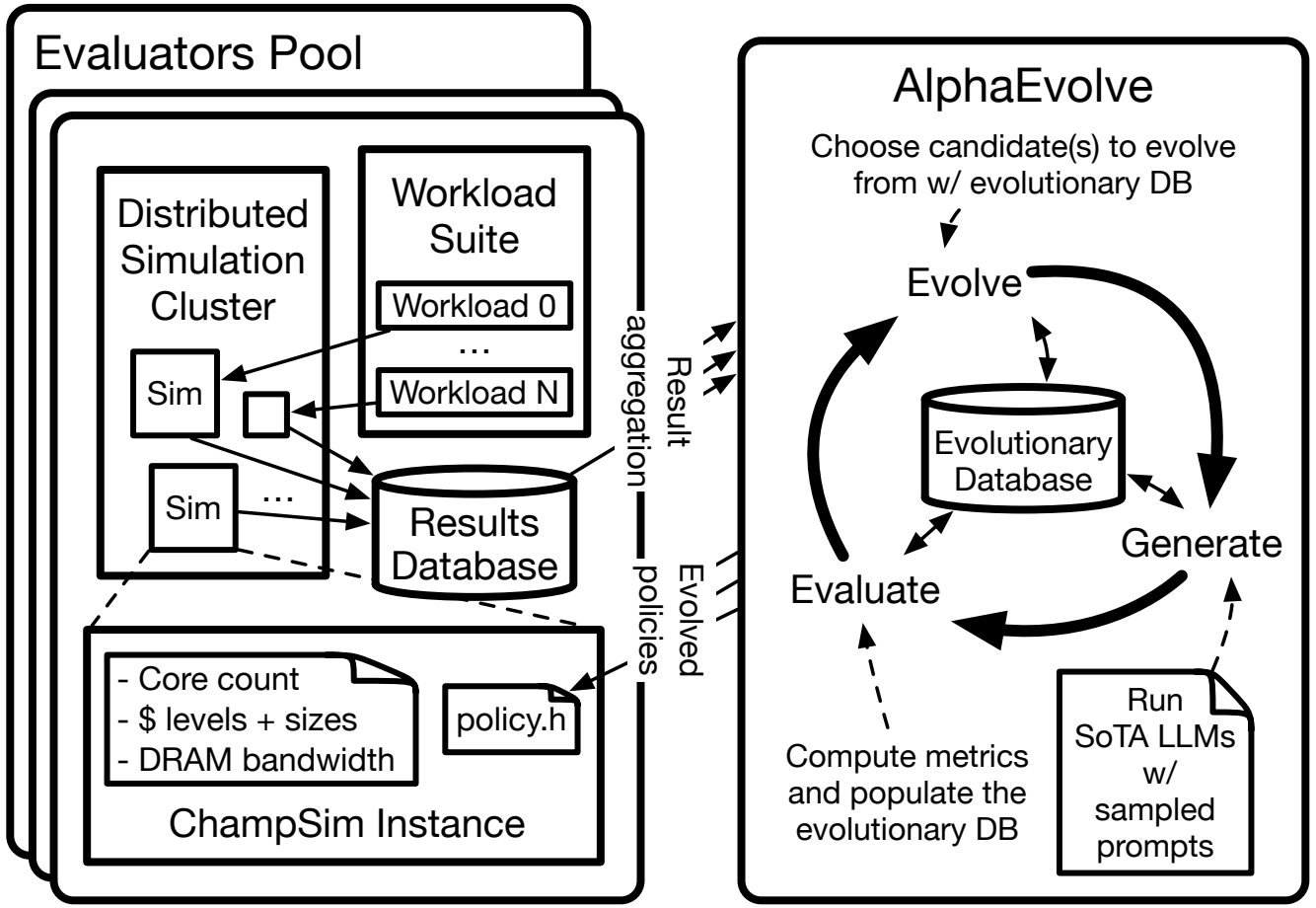
Benchmarks

Single-core SPEC06
Multi-core Google Workload Traces v2

Metrics

IPC
A storage budget

ArchAgent System Design



Prompt Template for ArchAgent

Act as an expert software developer and computer architect. The codebase that you are working on is a simulator for an out-of-order superscalar processor running a program trace. Your task is to iteratively improve the indicated section of the codebase, which models a replacement policy for the caches in the simulated processor and win the Cache Replacement Championship. That is, you want to design the best possible cache replacement policy for an out-of-order superscalar processor and implement it in the simulator. The primary goal is to increase the scores on the provided evaluation metrics, where larger values are better. One of these metrics is the number of instructions-per-cycle (IPC) that the processor executes. A better processor will achieve a higher IPC.

[...]

Ensure the code you introduce is realizable in hardware. Ensure you don't use more than 48KB state for your replacement policy. This is separate from the size of the cache itself. This is a strict limit and you must adhere to it honestly. Otherwise you will be disqualified.

[...max lines to change, context about simulator APIs,workloads, system configs, prior working programs, prior literature, simulator caveats ...]

Designing for SPEC

Policy31 achieves **0.9% IPC speedup improvement v. Mockingjay** (prior SoTA)

Several novel mechanisms:

- Hawks and Doves
- Insertion Quality Predictor
- Prefetch-Aware Retention
- Cache Pressure-Aware Adaptive Throttling

```
+ // --- HAWKS AND DOVES STATE ---
+ // A bit-packed 2b sat. ctr per cache line
+ // Tracks usage intensity and follows state budget
+ // Each byte holds 4 counters
+ std::vector<uint8_t> packed_usage_counter;
/* find a cache block to evict */
long policy31::find_victim(...) {
for (...) {
+ // Retrieve H&D usage and use for ETR
+ current.usage = get_usage(set, way);
+ current.effective_etr = abs(current.etr_val) -
(current.usage * BONUS_PER_USE);
}
}

/* called on every cache hit and cache refill */
void policy31::update_replacement_state(...) {
+ if (hit) {
+ // Inc. usage counter (saturating at 3)
+ // This makes frequently-used blocks stickier
+ increment_usage(set, way);
+ } else {
+ // Fill on a miss (a new line inserted)
+ // The new block starts with a usage of 0
+ reset_usage(set, way);
+ }
}
```

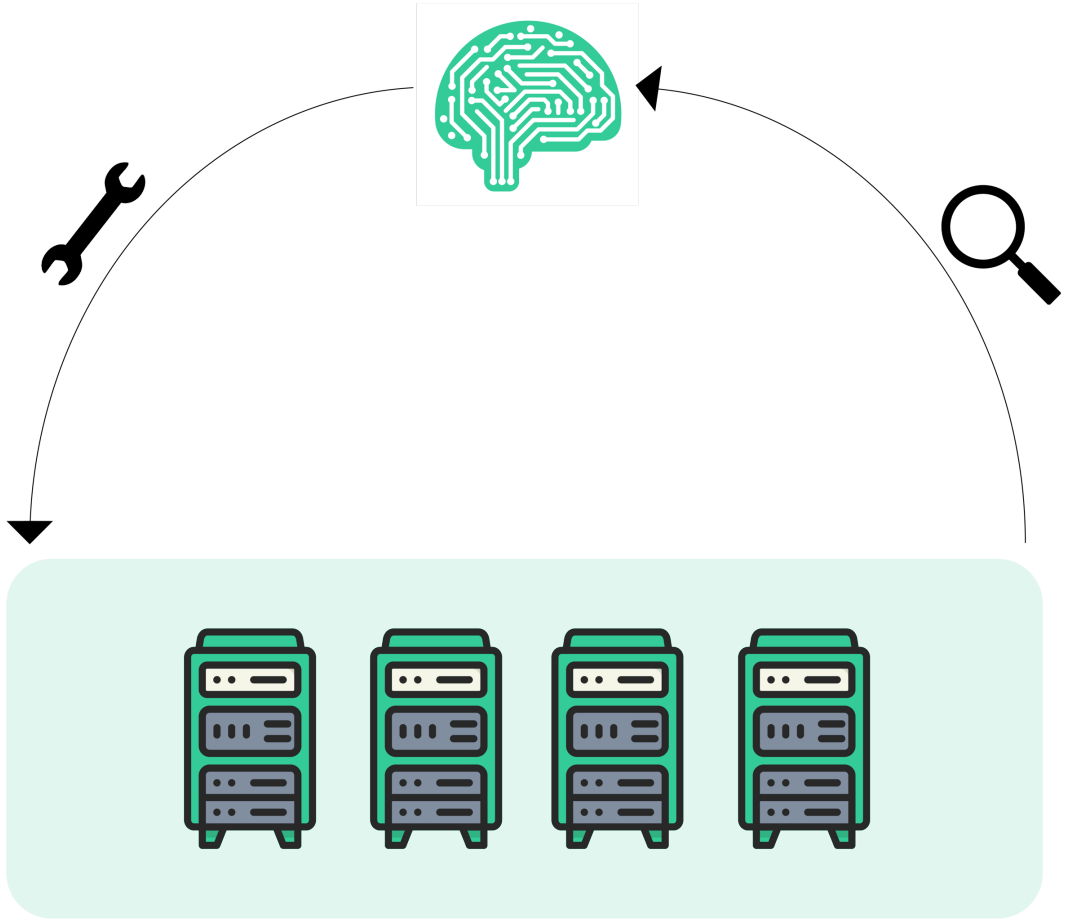
*Diff for **Hawks and Doves** Mechanism in Policy31*

Tuning Deployed Hardware

ArchAgent-developed policies are **parameterized** and **flexible**

Proliferation of agentic flows

In addition to generating hardware, can agentic flows do “FDO” over runtime parameters of hardware, per-workload, on-the-fly?



Tuning Deployed Hardware

13 runtime parameters in Policy31 that do not affect storage size

- E.g., constants used to score accesses

```
// Bonus to reduce ETR for hard-to-prefetch lines
static constexpr int PAR_ETR_BONUS = 2;
// Penalty added to a prefetched block's initial ETR
static constexpr int FARSIGHT_PENALTY = 4;
```

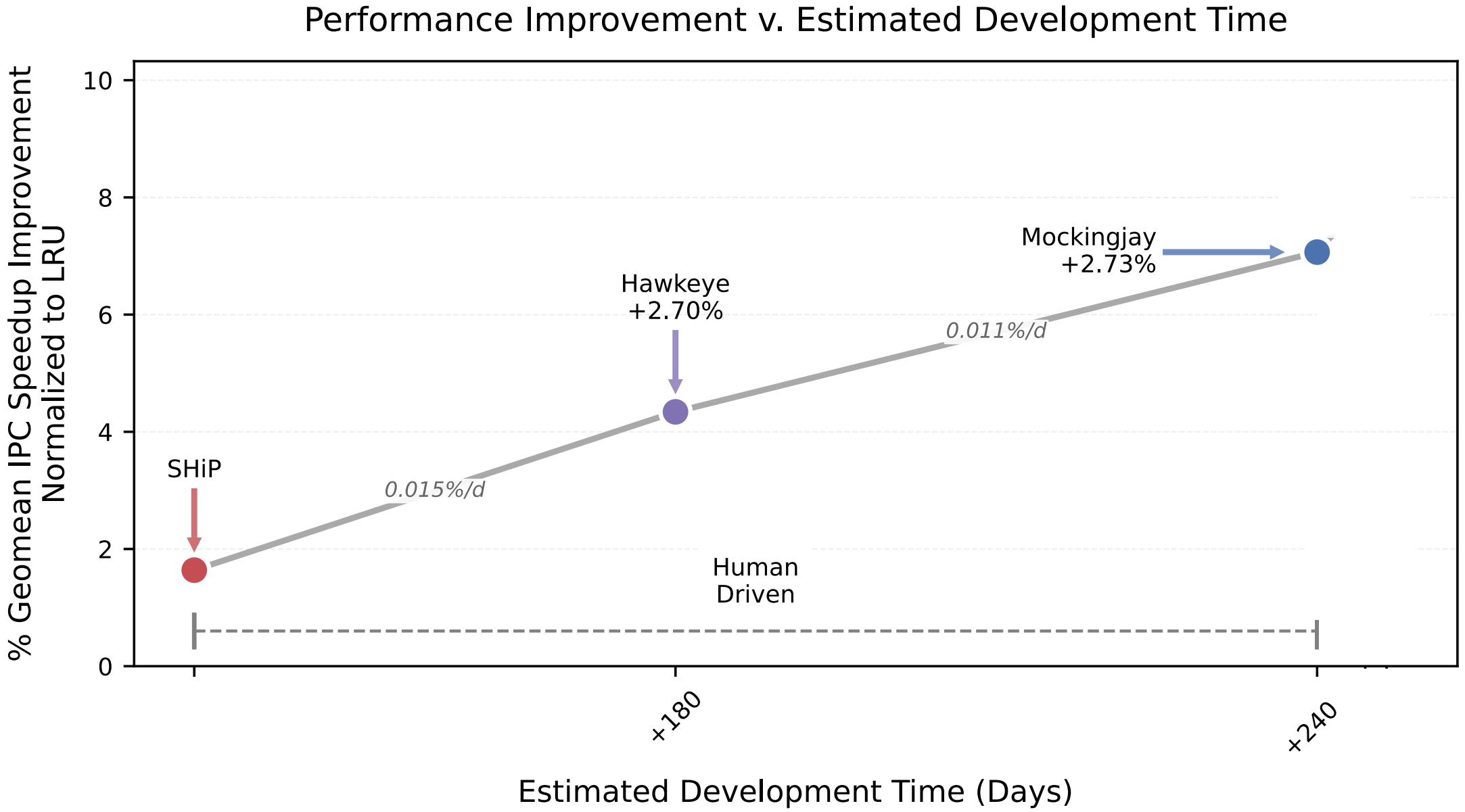
- E.g., thresholds to detect cache events

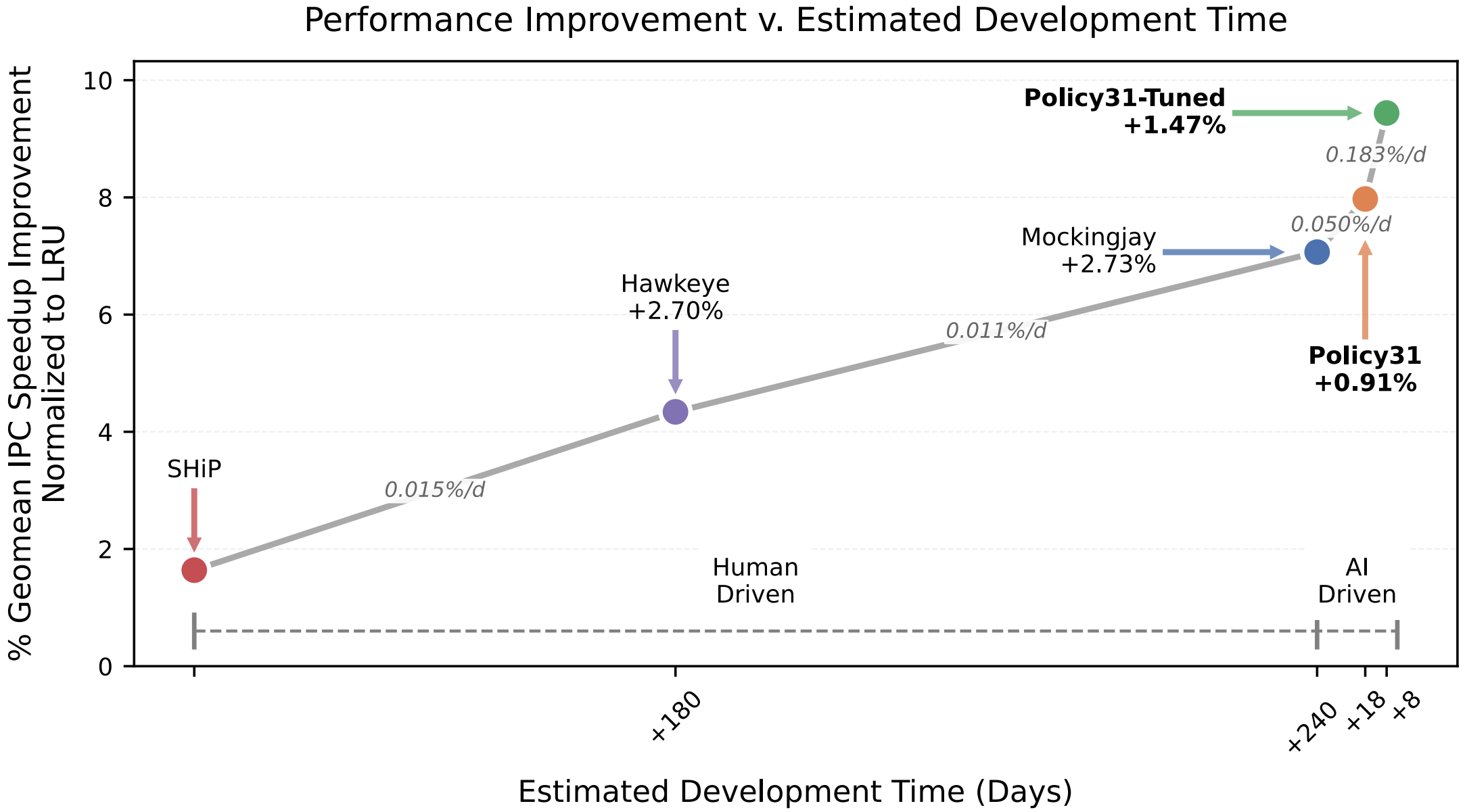
```
static constexpr uint64_t PRESSURE_DECAY_PERIOD = 256;
static constexpr int PCT_NEUTRAL = 1 << (PCT_BITS - 1);
static constexpr int PCT_HIGH_CONF_THRESH = PCT_NEUTRAL + 2;
```

ArchAgent tunes for each workload individually

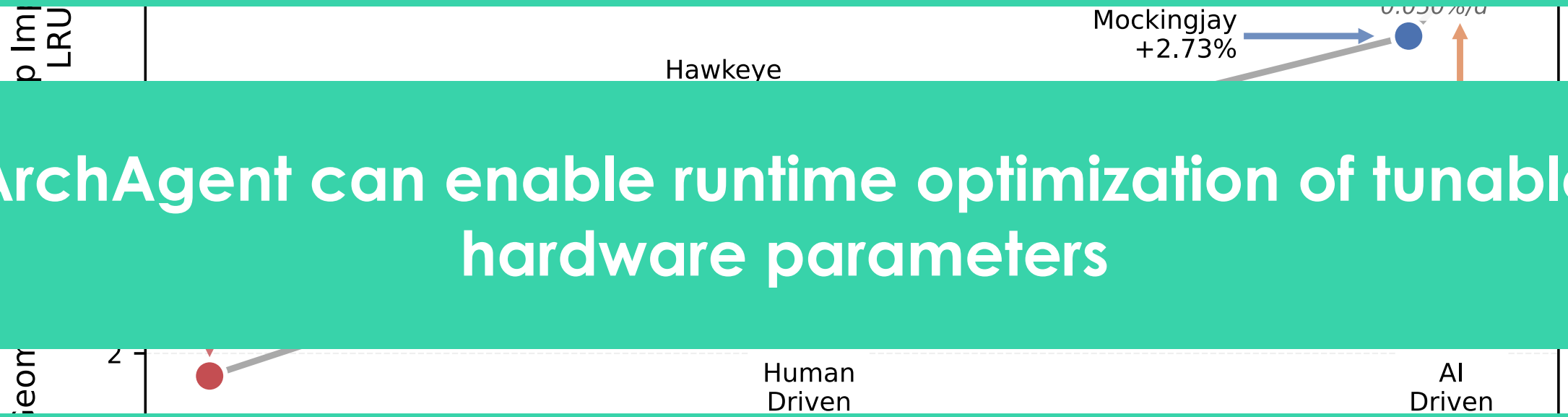
Policy31-Tuned achieves

- **2.3%** IPC speedup improvement vs. **Mockingjay** (prior SoTA)
- **1.5%** IPC speedup improvement over **Policy31**


Performance Improvement v. Estimated Development Time
% Geomean IPC Speedup Improvement
Normalized to LRU
10
8
6
4
2
0
SHiP
Hawkeye
+2.70%
Mockingjay
+2.73%
0.015%/d
0.011%/d
Human
Driven
+180
+240
Estimated Development Time (Days)


Performance Improvement v. Estimated Development Time
% Geomean IPC Speedup Improvement
Normalized to LRU
10
8
6
4
2
0
Policy31-Tuned
+1.47%
0.183%/d
0.050%/d
Mockingjay
+2.73%
Hawkeye
+2.70%
0.011%/d
Policy31
+0.91%
SHiP
0.015%/d
Human
Driven
AI
Driven
+180
+240
+18
+8
Estimated Development Time (Days)

ArchAgent can win on SPEC



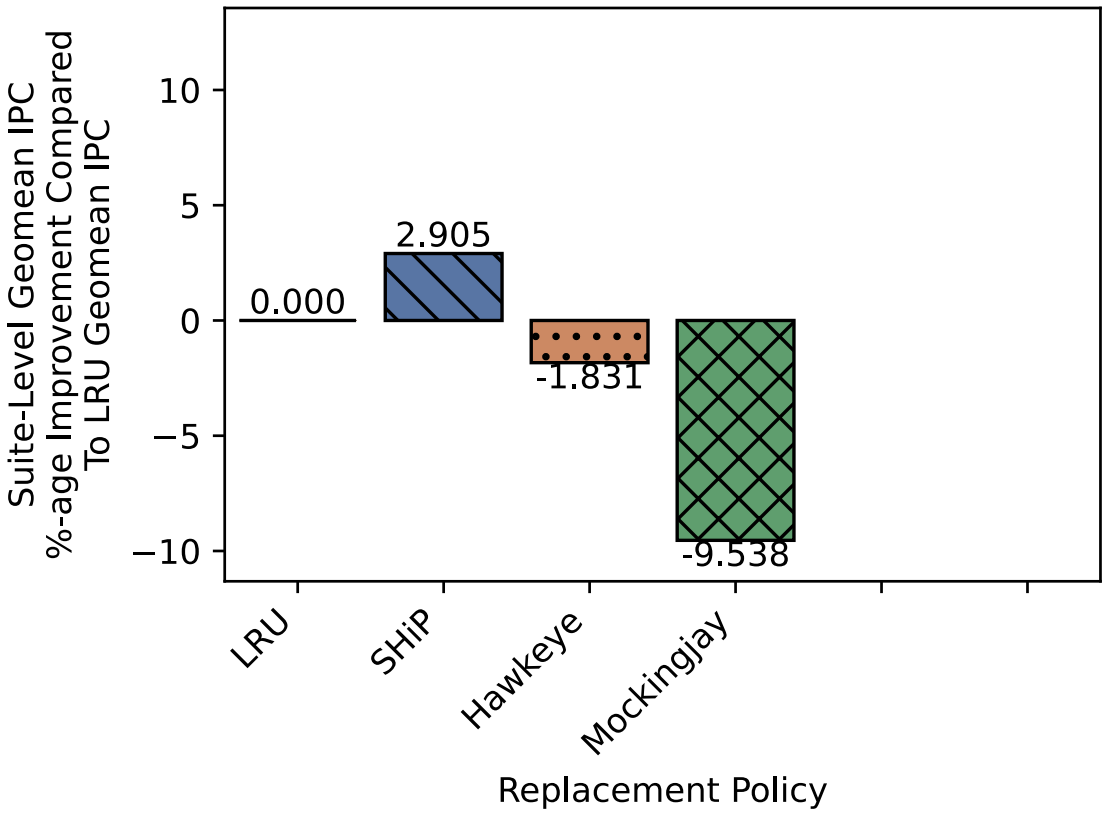
ArchAgent can enable runtime optimization of tunable hardware parameters

ArchAgent does this in a matter of days

Designing for Hyperscale Workloads

Hyperscale workloads differ significantly from SPEC

Mockingjay (SPEC-winning SoTA) performs worse than LRU



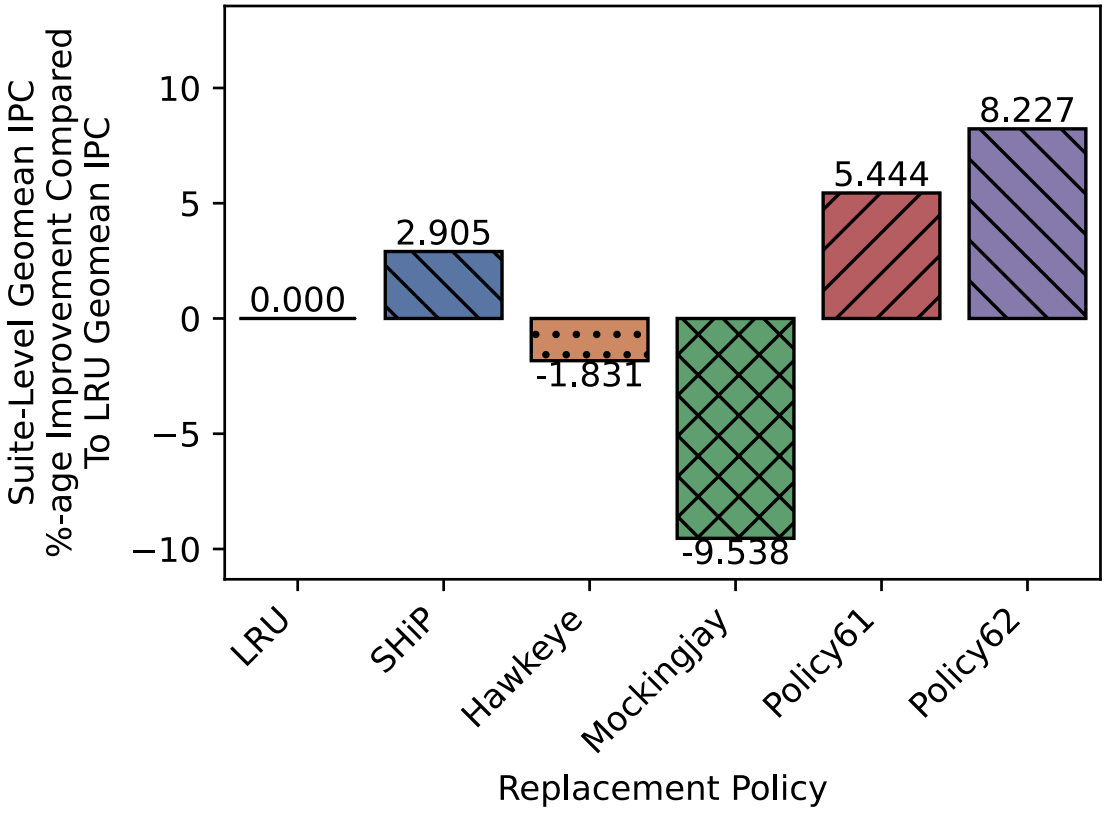
Designing for Hyperscale Workloads

Hyperscale workloads differ significantly from SPEC

Mockingjay (SPEC-winning SoTA) performs worse than LRU

Policy62

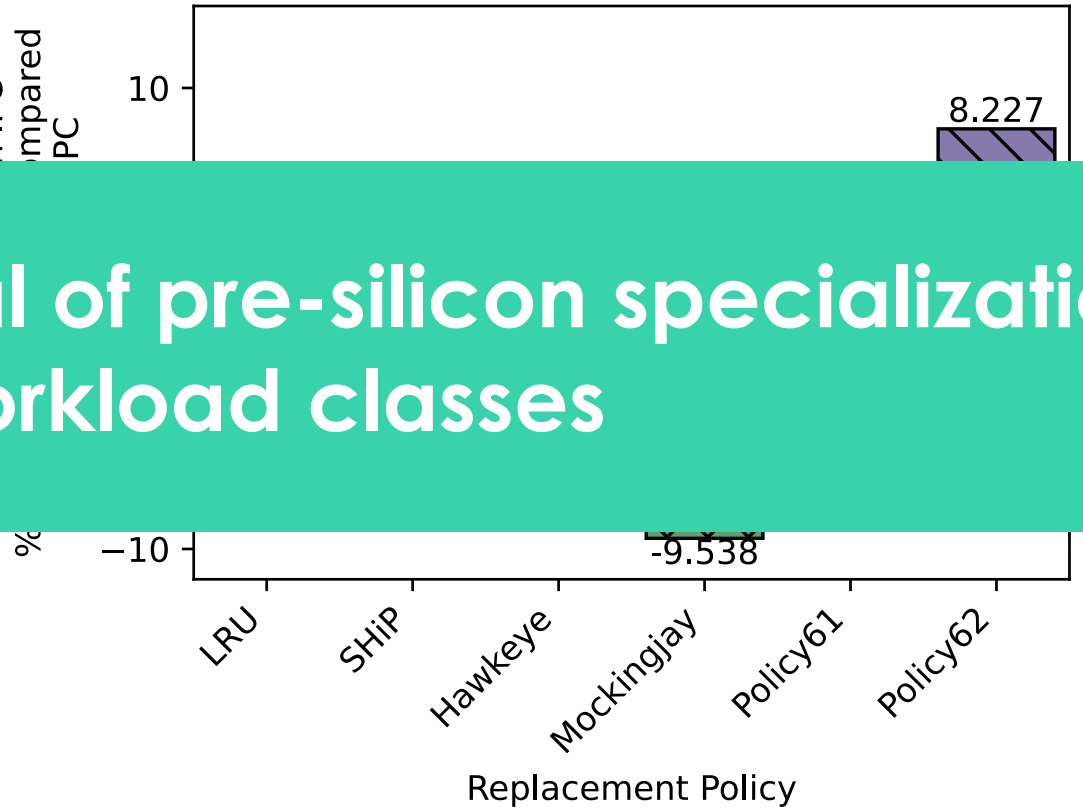
- **2 days**
- **5.3%** IPC speedup improvement **vs. SHiP** (**+17.8% vs. Mockingjay**)



Designing for Hyperscale Workloads

Hyperscale workloads differ significantly from SPEC

- **2 days**
- **5.3%** IPC speedup improvement **vs. SHiP** (**+17.8% vs. Mockingjay**)



ArchAgent unlocks the potential of pre-silicon specialization of hardware to workload classes

An Unexpected Observation

A few weeks into working with AlphaEvolve…

AlphaEvolve wins vs. Mockingjay with over 4% IPC speedup!!!
(recall that's 3-4x the paper "acceptance threshold" for cache replacement)

Why?

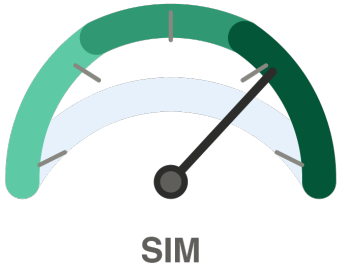
- Simulator does not support write-bypassing
- Assertions disabled in optimized simulator build
- AE solution drops write traffic with no visible error → IPC 📈

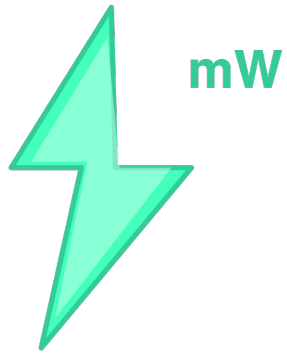
```
259     uint64_t pc_signature = get_pc_signature(pc, false, type == access_type::PRE
    FETCH, cpu);
260     if (type != access_type::WRITE && rdp.count(pc_signature) &&
261             (rdp[pc_signature] > MAX_RD || rdp[pc_signature] / GRANULARITY > max
    _etr)) {
262         return this->NUM_WAY;
```

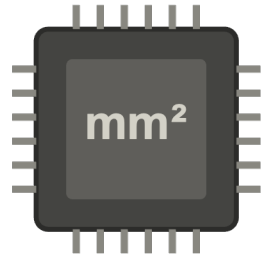
```
306     uint64_t pc_signature = get_pc_signature(pc, false, type == access_type::PRE
    FETCH, cpu);
307     auto rdp_it = rdp.find(pc_signature);
308     if (rdp_it != rdp.end()) {
309         const auto& entry = rdp_it->second;
310         if (entry.confidence >= RDP_CONFIDENCE_THRESHOLD &&
311             (static_cast<double>(entry.rdp_value) * RDP_SCALE_FACTOR / GRANULARITY
    > max_etr)) {
312             // Confident prediction of long reuse, bypass the block
313             return this->NUM_WAY;
314         }
```

Challenges

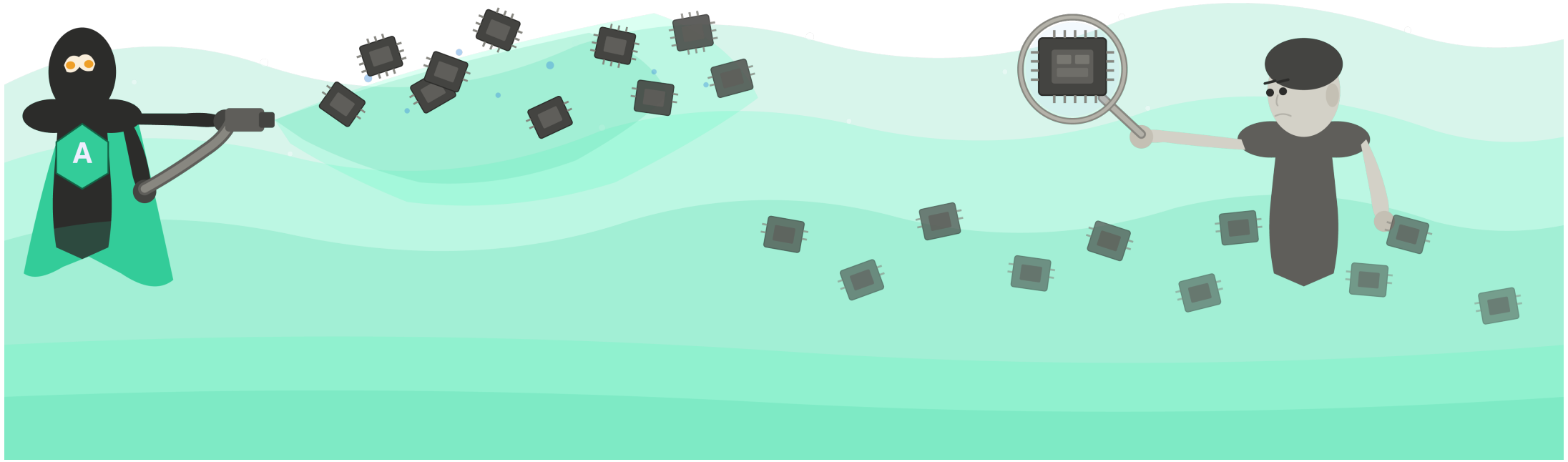
The Agentic Discovery System is only as good as its playground







Challenges



A flood of automatically generated designs
requiring significant manual verification

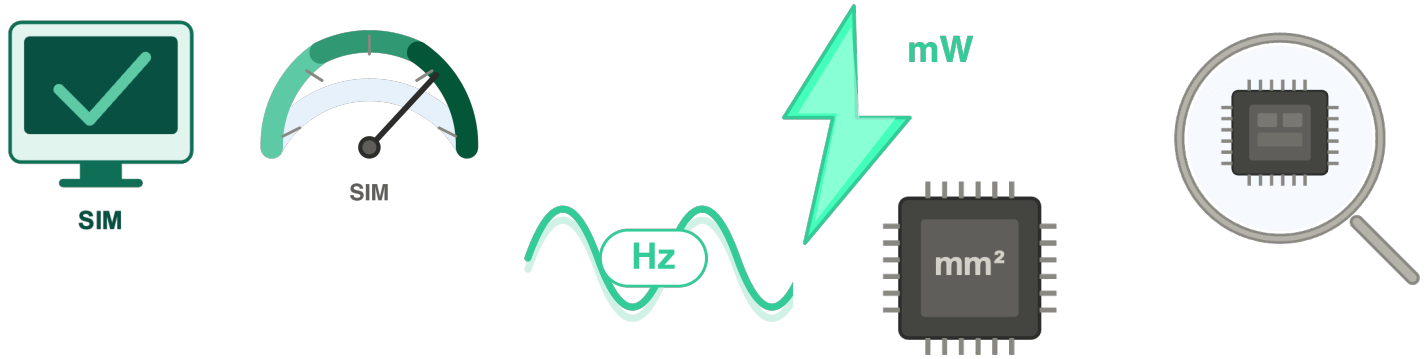
Conclusion

ArchAgent: The first agentic AI system to **automatically discover novel computer architectures**, advancing the state-of-the-art **within days**.

Rapid Hardware Specialization

Pre-Silicon | **Deployed Systems**

Challenges



Thank You! Questions?

ArchAgent

AGENTIC AI-DRIVEN COMPUTER ARCHITECTURE DISCOVERY

Raghav Gupta[1†], Akanksha Jain[2], Abraham Gonzalez[2],
Alexander Novikov[3], Po-Sen Huang[3], Matej Balog[3], Marvin Eisenberger[3],
Sergey Shirobokov[3], Ngân Vũ[3], Martin Dixon[2],
Borivoje Nikolić[1], Parthasarathy Ranganathan[2], Sagar Karandikar[1]

[1]University of California, Berkeley
[2]Google [3]Google DeepMind