2ND IPADS YOUNG SCHOLARS FORUM · AGENT HARNESS TRACK

Co-evolving OS Infrastructure with Agents through Runtime Extensions

与智能体共同演化的可扩展操作系统

Yusheng Zheng

UCSC · eunomia-bpf · Oct 16, 2026

AGENDA

Agenda

1 · Two characteristics that set agents apart from traditional programs

2 · Why the systems that manage agents must evolve at runtime

3 · Observing and constraining agents in the OS, by task and intent (AgentSight, AgentCgroup, ActPlane)

4 · Agents writing extensions for the OS (SchedCP, gpu_ext)

5 · Four challenges for OS runtime extensions

1 · OBSERVATION

Agents run for hours, work at scale and change system state

Long-running

One task runs for hours, sometimes weeks

In teams, at scale

Thousands of agents work on one codebase at once

Changing system state

Hundreds of agents on one machine edit files, change configs and run commands

TWO CHARACTERISTICS THAT SET AGENTS APART FROM TRADITIONAL PROGRAMS

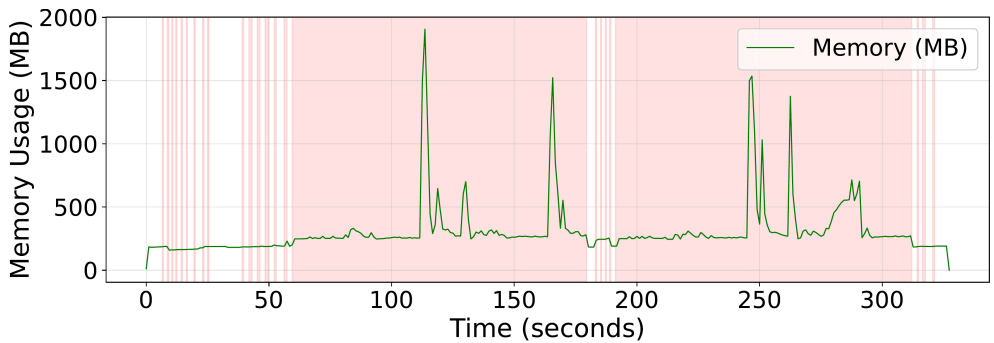
Characteristic 1 · Behavior is generated at runtime from intent, nondeterministically

Characteristic 2 · Agents perceive and understand systems, and help develop and optimize them

Cursor, Scaling long-running autonomous coding (Jan 2026) · Wilson Lin interview (Jan 2026)

1 · CHARACTERISTIC 1

Agents act on intent at runtime, nondeterministically



Resource demand changes with behavior generated at runtime. One task's memory rises during some tool calls (red); in the tasks we measured, peak memory can reach over ten times the average

Generated from intent

The model generates commands and scripts from the task intent at runtime and adapts them to feedback

Agents interact

Many agents interact and change the system together

Nondeterministic

Rerun the same task and its trajectory often differs

1 · CHARACTERISTIC 2

Agents perceive and understand systems, and help develop them

Perceive

They read command output, logs and performance data from the system

Understand

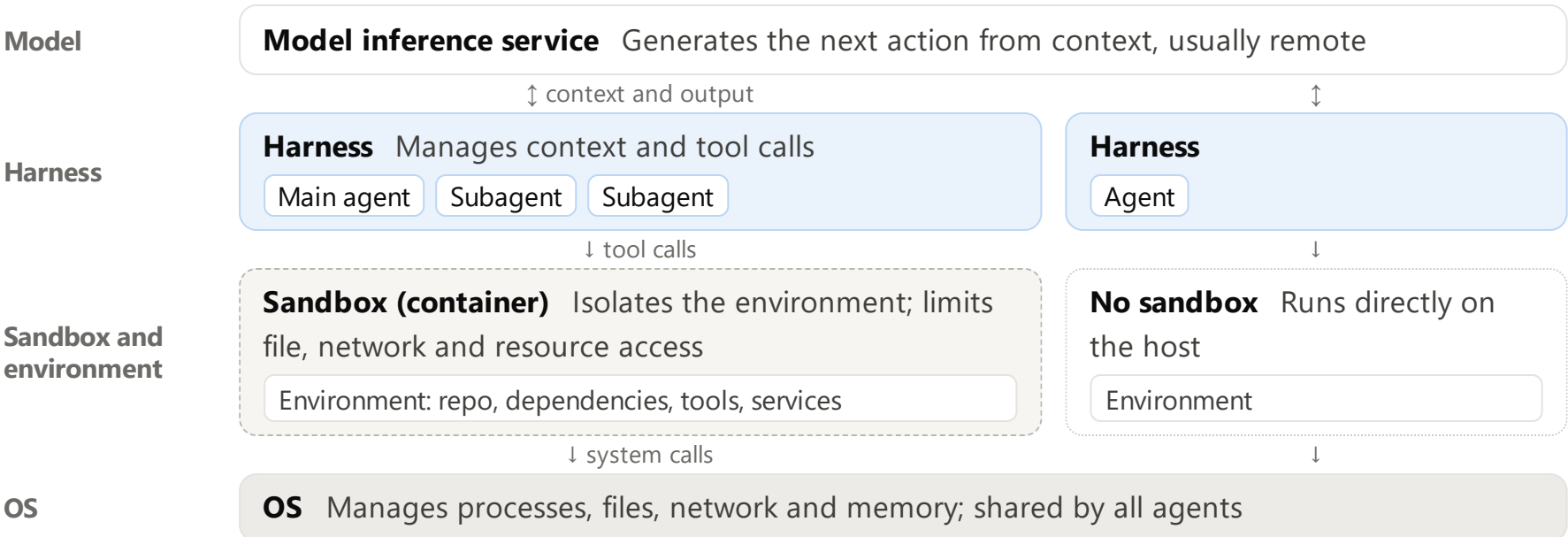
They work out what is wrong and what should change

Change

They write and modify programs, configs and policies, helping develop the system

> Coding is far cheaper and more capable, so understanding and changing systems is no longer limited to a few experts

Harness, sandbox and OS manage agents in layers



Harness, sandbox and OS manage agents in layers, and many agents share one OS

2 · LIMITATION 1

Existing systems assume a program's intent is known at deployment

Mechanisms built on it

Vetting, authorization, resource limits and access control follow the expected behavior; runtime monitoring sees low-level signals, not what the behavior is for

Agents break it

Characteristic 1 · Behavior is generated at runtime from intent, nondeterministically; intent arrives with each task, so the same agent may do different things each time

WHERE THESE MECHANISMS FALL SHORT

Checks cover only specific forms

Block a command and the agent rewrites it or takes another path

Resource limits are preset

Demand jumps with each tool call, so preset limits either waste or starve

Blind to task and timing

Permissions say who may do what, not for which task or when

AgentCgroup (arXiv:2602.09345)

2 · LIMITATION 2

Existing systems assume a few trusted experts write policies

The assumption

Policies such as scheduling, resource allocation and permission checks are written by a few trusted experts and rarely change

Agents break it

Characteristic 2 · Agents can now write policies too, so authors multiply and changes become frequent

BOTH WAYS AGENTS CHANGE A SYSTEM TODAY FALL SHORT

Commands and knobs

Run commands or set existing knobs; no new decision logic

Source edits

Too broad to verify; applying or undoing it needs a rebuild and redeploy

Systems for agents must understand intent and evolve safely

Requirement 1 · Understand task and intent

Learn each agent's task and intent at runtime; observe, control and constrain it accordingly, and explain why when it is blocked

Addresses limitation 1

Requirement 2 · Evolve safely with agents

Offer agents a new way to change the system, more expressive than knobs and easier to verify and roll back than source edits

Addresses limitation 2

> Both requirements mean policies change while the system runs, so the system must keep evolving at runtime

Part of observing and constraining agents belongs in the OS

Effects happen in the OS

Commands and scripts act on files, processes and the network

The harness sees only tool calls

It cannot observe what scripts and subprocesses do inside a call

Many agents share one OS

Hundreds of agents, possibly from different harnesses, share one machine

> Only the OS observes, per process and file, both what happens below tool calls and the resources all agents share

Wilson Lin interview (Jan 2026)

Runtime extensions let the OS change while it runs, verified before loading

Kernel modules
Full privilege; one bug can crash the kernel

→

eBPF
Verified before loading, replaced at runtime

→

Beyond the kernel
bpftime (userspace), gpu_ext (GPU)

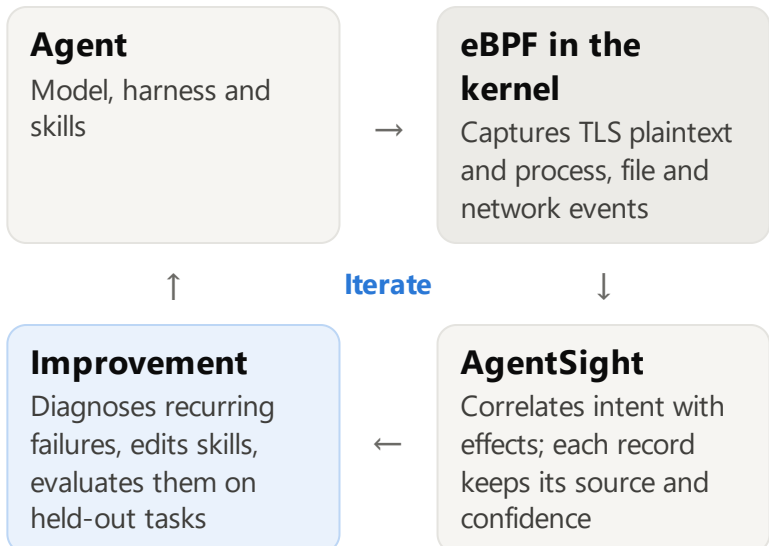
Since Linux 6.12, schedulers can be written in eBPF

> Changing at runtime lets the OS load policies by task and intent; verifying before loading lets it accept agent-written ones

sched_ext (Linux 6.12)

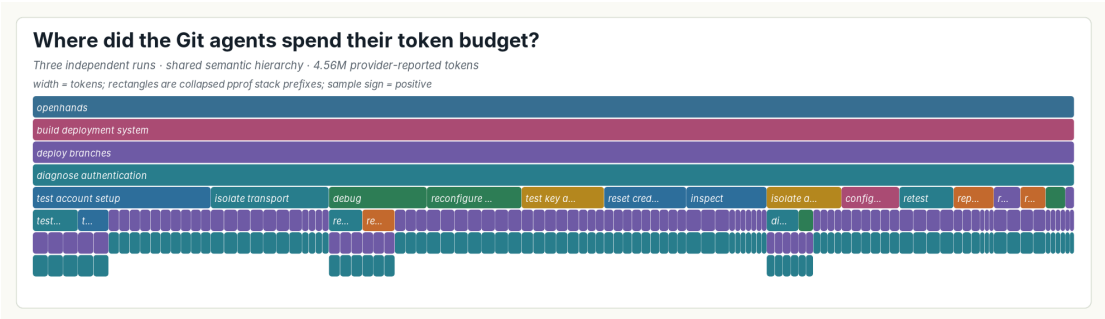
AgentSight: observing system behavior and iteratively improving agents

- **Problem**: agents' own records are getting richer, but effects in child processes, files and the network fall outside them
- **Method**: eBPF captures TLS plaintext and system events without modifying the agent; evidence keeps four sources apart, with confidence
- **Result**: finds prompt injection, idle reasoning loops and multi-agent coordination bottlenecks; typical overhead < 3%
- **Improvement**: recurring failures become skill edits, evaluated on held-out tasks and returned to the agent



Paper: AgentSight, arXiv:2508.02736 · Code: github.com/eunomia-bpf/agentsight

Profiling long-running agents



- **Problem**: profilers attribute resources to code paths, but an agent's cost belongs to tasks such as "diagnose" or "compare branches"
- **Method**: a semantic operation stack splits agent trajectories recursively, replacing the call stack, and exports standard pprof profiles that render as flame graphs
- **Result**: aggregates tokens, time and operations across runs to locate failures and cost hot spots; fault localization up to **56%** better

Paper: arXiv:2609.20301 · Code: github.com/eunomia-bpf/agentsight

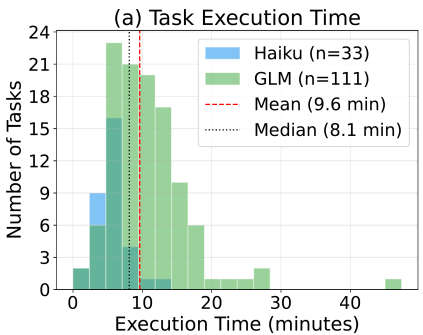
3 · REPLAY · Agent Nebula

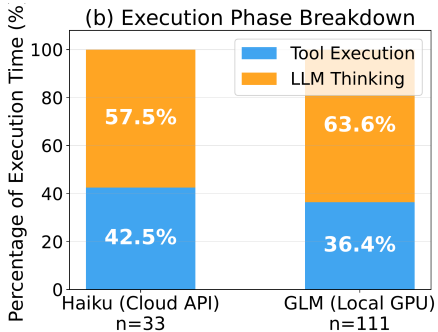
Replaying agents' system behavior trajectories

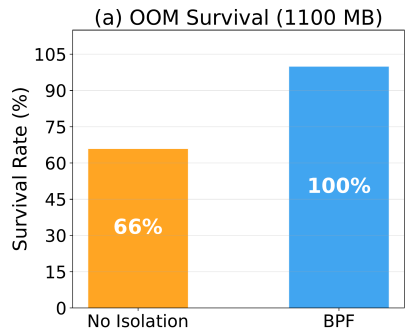
- **Problem**: after days of agent work, people see only the final diff and tests
- **Method**: replay the agents' real file reads, writes, creates, renames and deletes; files are stars, directories set colors, commits are milestones
- **Example**: ActPlane's own development, May to July 2026, about 68,000 agent actions over 349 files
- **Use**: expose harness and skill design problems, such as unread documents or loops on useless tests

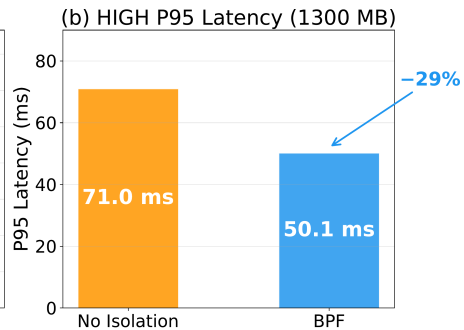
Code: github.com/eunomia-bpf/agentsight (agentvis)

AgentCgroup: understanding and controlling agent resources





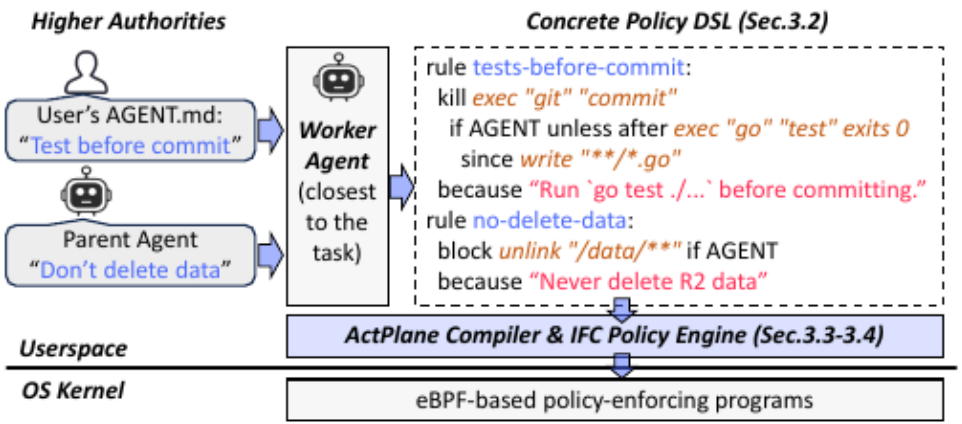




- **Problem**: tool execution and setup take **56–74%** of end-to-end time, and resource use bursts per tool call
- **Method**: each tool call runs in a short-lived cgroup; eBPF enforces policy in the kernel through sched_ext and memcg_bpf_ops
- **Result**: OOM survival **66%** → **100%**, P95 allocation latency **−29%**, wrapper overhead under **5 ms**

Paper: AgentCgroup, arXiv:2602.09345 · Code: github.com/eunomia-bpf/agentcgroup

ActPlane: programmable OS policy enforcement for agents



- **Problem**: tool-layer checks do not cover shells, scripts and subprocesses; many agent policies, such as "test before you commit", are really data-flow constraints on OS objects
- **Method**: agents declare data-flow control policies in a DSL; eBPF propagates labels across processes, files and endpoints in the kernel and returns violation reasons to the agent
- **Result**: **2.0–3.2×** as many violating trajectories handled correctly as baselines; **74%** of harmful outcomes prevented (78 of 106 in 361 tasks); overhead **1.9–8.4%**

Paper: ActPlane, arXiv:2606.25189 · Code: github.com/eunomia-bpf/ActPlane · OpenAgentSafety

Agents now change OS policies too

THREE LEVELS, FROM TUNING KNOBS TO WRITING EXTENSIONS

Tune knobs

An LLM sets Linux scheduling, memory and I/O parameters online; each value is type-checked

TuxBot (SOSP '26)

→

Select verified policies

The agent picks among eBPF policies verified at load time; switching takes under a microsecond

AKTS (Sep 2026)

→

Write extensions

The agent writes new extensions, verified at load time and confined by hooks

SchedCP, gpu_ext (ours)

> The higher the level, the more expressive it is and the harder its effects are to bound; our work is at the third level

TuxBot (SOSP '26, arXiv:2605.15026) · AKTS (arXiv:2609.12276, NeurIPS '26 ML for Systems workshop) · SchedCP (arXiv:2509.01245)

Agents write scheduling and GPU policies at confined hooks

THE AGENT DECIDES WHAT TO OPTIMIZE; THE CONTROL PLANE HANDLES HOW TO OBSERVE AND ACT

Observe
Profiling and tracing on demand

→

Plan
Decide what to optimize; reuse stored policies

→

Execute
Write an eBPF policy, verified before deployment

→

Learn
Measure and keep what works

CPU scheduling · SchedCP

Agent-written sched_ext policies, up to 1.79× faster kernel builds and 2.11× better schbench P99, at 13× lower cost

GPU memory and scheduling · gpu_ext

59 agent-written policies over 974 runs, up to 4.8× throughput over framework offloading; all 50 unsafe attempts blocked, no kernel crash

> Agents express new logic; the verifier and hooks bound what it can do, not its effect on other tenants

SchedCP (arXiv:2509.01245) · gpu_ext, ASPLOS '27, arXiv:2512.12615 · github.com/eunomia-bpf/gpu_ext

OS runtime extensions, too, assume a few trusted authors

Like OS policies (CFS was Linux's default scheduler for 16 years), extensions are written by a few trusted experts

	Kernel-module era	Today
Safety	Trust the author	Verify before loading
Deployment	Ships with kernel releases	Replace at runtime
Scope	A few subsystems	Kernel, extending to userspace and GPUs
Authors	A few trusted experts	**Still a few trusted experts**

Unprivileged loading was tried, then disabled by default from Linux 5.16

> Safety, deployment and scope have changed; authors are still a few trusted experts, and mostly only administrators load extensions

CFS: Linux 2.6.23 (2007) to 6.6 (2023) · Linux commit 8a03e56b253e (Nov 2021) · SUSE-SU-2021:3941-1

Agents bring four challenges to OS runtime extensions

Agents are now both the targets and the authors of extensions, and they are untrusted, adaptive and numerous

Effects

Bound an extension's impact on other tenants' performance, fairness and resources

The verifier guarantees only memory safety and termination, and agents generate many extensions

Enforcement

Check constraints on every execution path, with task intent and timing

ActPlane moves checks into the kernel, but a check often sits on one hook, and agents switch to other loaders

Authority

Delegate loading authority across authors of different trust; a subordinate's extensions may only add constraints, never loosen them

Admin-only loading shuts agents out; open loading lets managed agents loosen their own constraints

Coordination

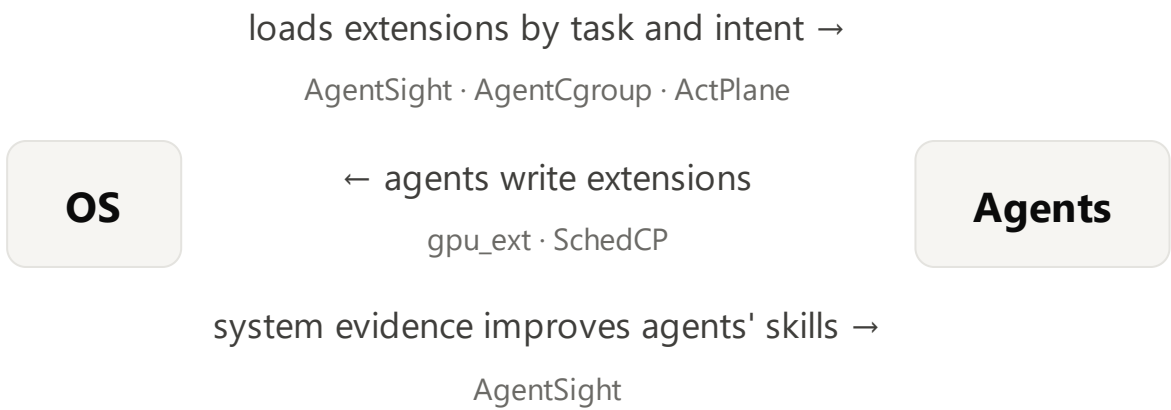
Arbitrate and compose many agents' extensions on one hook by intent

Attach mechanisms track ownership and order; with many authors, conflicting intents are hard to resolve

sched_ext documentation (Linux 6.12) · tcx commit e420bed02507

SUMMARY

The OS and agents co-evolve



> Co-evolution still has to meet four challenges in effects, enforcement, authority and coordination

Questions? · eunomia.dev · github.com/eunomia-bpf