THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA

서울

From Continuous Tracing To Automated Insights

AI-Driven Performance Analysis With Guider

Hyundai Motor Company
Peace Lee

OPEN SOURCE SUMMIT 2026

From Continuous Tracing To Automated Insights

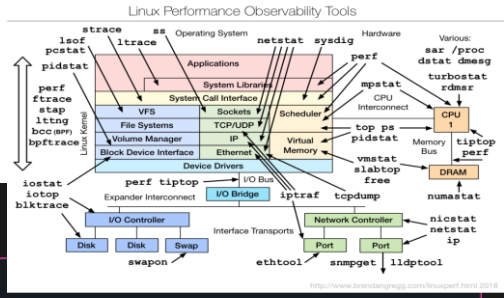
AI-Driven Performance Analysis With **Guider**

"Autonomous Performance Management for Embedded Linux — Configure once. Guider watches, analyzes, and acts."

Peace Lee · github.com/iipeace/guider

Python | eBPF | Linux / Android | AI-Powered | 250K+ lines | 10+ years

Guider



Three Challenges of Performance Analysis

Diverse Root Causes

Performance issues stem from kernel, driver, IPC (Inter-Process Communication), memory, I/O, thermal — code alone rarely reveals the cause. Analysis requires **many different specialized tools**.

Guider solves with

200+ Built-in Commands

Hard to Reproduce

Sudden performance issues may never happen again. When they occur, you must **capture as much information as possible**, immediately and effectively.

Guider solves with

Always-on Tracing Daemon

Needs Immediate Response

Field incidents need more than post-mortem analysis. Some level of **real-time response capability** is required — even if imperfect.

Guider solves with

AI-Driven Analysis & Action

Guider addresses all three — today's talk focuses on the third: **AI-driven automated insights & response**

Guider

One Tool. Every Layer.

A single Python file that watches, profiles, and explains your entire system.

244 Commands	**36** eBPF Tracers	**28** Visualizations
250K+ Lines of Code	**10+** Years in Production	**1** File, Zero Build

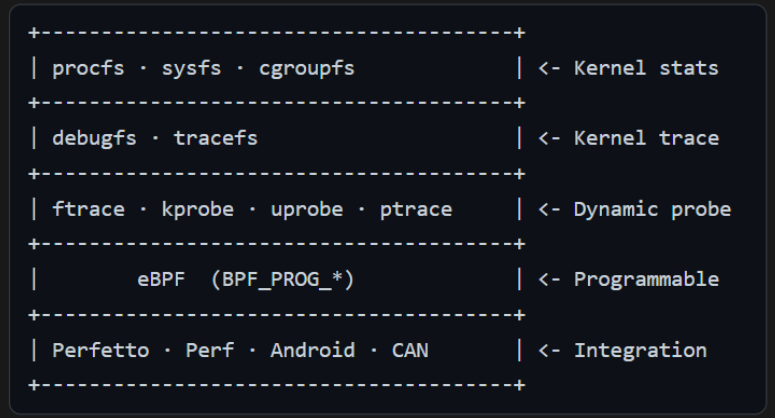
No compilation. No kernel module. No agent to install.
`pip install guider` — pure Python, runs on any Linux/Android box you can SSH or `adb` into.

Built over a decade of real production incidents — not a weekend side project.

Guider

The Data Sources Stack

All data sources — one binary



procfs / sysfs / cgroupfs

CPU, memory, I/O, network, process tree — the foundation. Zero overhead, always available.

ftrace / kprobe / uprobe

Kernel function tracing with no kernel rebuild. Dynamic attach/detach at runtime.

eBPF — 28+ commands

Stack sampling, latency histograms, TCP retransmit, Binder tracing, lock contention...

Perfetto / CAN / Android

Automotive-grade: Perfetto system traces, CAN bus + DBC decode, Android platform commands and APIs.

Guider

Breadth of Coverage

Process & CPU

- `top` — system-wide stats
- `ftop` — file descriptor monitor
- `atop` — all-in-one analysis
- `ctop` — threshold daemon ⭐
- `pytop` — Python profiler
- `bpfstacktop` — on-CPU eBPF
- `bpfwaittop` — off-CPU eBPF
- `bpfrunqtop` — run queue lat

Memory & I/O

- `mtop` — memory monitor
- `checkdup` — dup page detection
- `bpfreclaimtop` — reclaim lat
- `bpfblktop` — I/O lat per stack
- `bpflocktop` — lock contention
- `bpfcachetop` — page cache analysis

Network & IPC

- `bpfpkttop` — XDP flow top
- `bpftcpretrans` — TCP retrans
- `bpfnetlat` — network latency
- `bpfbinderlat` — Binder lat
- `bpfbinderpool` — thread pool
- `bpfsyscalltop` — syscall stats
- `bpfdroptop` — pkt drop stacks

Android / IVI

- `perfetto` — system trace ⭐
- `andtop` — Android log monitor
- `cantop` — CAN bus monitor
- `cansnoop` — CAN bus stream
- `bpfbindersnoop` — Binder stream
- `lmksnoop` — LMK kill events

Profiling & Visualization

- `rec` — ftrace analysis
- `drawflame` — flame graph
- `draw` — time-series HTML
- `drawscatter` — correlation
- `drawmem` — memory chart
- **20+ visualization types**

AI Integration ⭐

- `ASKAI` — analyze & explain
- `ASKRUN` — decide & act
- Embedded in HTML/SVG output
- ctop threshold trigger
- Webhook → Slack/PagerDuty
- → Covered in Part 2

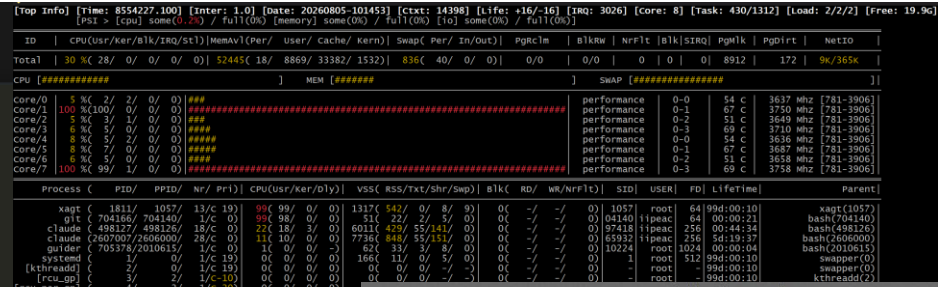
Guider

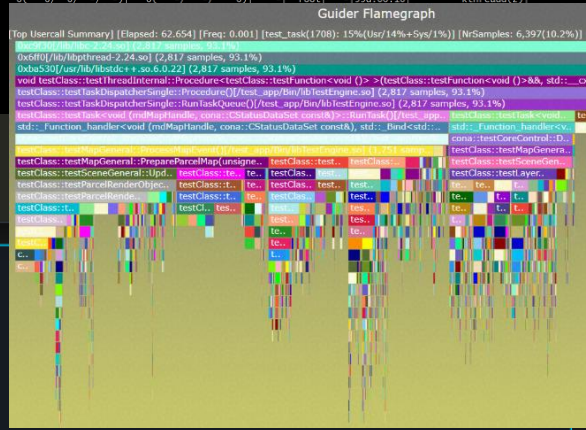
Demo 1 — Tracing & Profiling LIVE

```
# Step 1: System-wide overview - CPU, memory, I/O, per-process breakdown
guider top -a

# Step 2: eBPF on-CPU profiling - flame graph SVG auto-generated on exit
sudo guider bpfstacktop -R 30 -q DRAWFLAME
# → guider_bpfstacktop_flamegraph.svg  (open in browser)

# Step 3: AI-enhanced HTML flame graph - from any guider monitoring session
#         record first:  guider top -a -o guider.out  (then Ctrl+C)
guider drawflame guider.out -q ASKAI:"Find the root cause of CPU pressure" -F html
# → guider_flamegraph.html  (monitoring data + AI analysis panel, shareable)
```





Traditional approach

- top → perf record → perf report
- addr2line for symbol resolution
- Manual stack cross-reference
- Write analysis report separately
- **Multiple tools · 30+ minutes**

Guider approach

- Single tool, consistent workflow
- Automatic symbol resolution
- Interactive flame graph in browser
- AI analysis panel embedded in HTML
- **One tool · shareable single file**

eBPF profiling → instant SVG · Monitoring session → AI-analyzed HTML — all from one tool, one file to share.

Continuous Tracing

Continuous Tracing Daemon — ctop

What ctop does

- Runs as an **always-on background daemon**
- Evaluates threshold rules at every interval
- On threshold hit: auto-captures full snapshot (SAVERAW)
- Supports **multi-condition groups**: triggers only when CPU% *and* MEM both breach simultaneously
- 27 resource types monitored — CPU, MEM, I/O, PSI, log, eBPF, ...

ctop solves Problem 2 — "*hard to reproduce*".
When the issue occurs, ctop has already captured it.
It is also the trigger point for AI analysis.

```
guider_00000000_00000001_CPU_SYSTEM_total_90_SAVERAW_2020-09-22T12:59:04Z_47469.out.gz
guider_00000000_00000002_CPU_SYSTEM_total_90_SAVERAW_2020-09-22T13:14:02Z_48128.out.gz
guider_00000000_00000002_CPU_SYSTEM_total_90_SAVERAW_2020-09-22T13:14:02Z_48128.out.gz.gz
guider_00000000_00000003_CPU_SYSTEM_total_90_SAVERAW_2020-09-22T13:14:32Z_48150.out.gz
guider_00000001_00000001_CPU_SYSTEM_total_80_SAVERAW_2020-09-24T09:54:19Z_5152.out.gz
guider_00000002_00000001_CPU_SYSTEM_total_95_SAVEMIN:-2_2020-10-26T13:25:52Z_2500403.out.gz
guider_00000002_00000002_CPU_SYSTEM_total_95_SAVEMIN:-2_2020-10-26T13:26:03Z_2500460.out.gz
guider_00000003_00000001_CPU_SYSTEM_total_95_SAVEMIN:-2s_2020-10-26T13:28:33Z_2501192.out.gz
guider_00000004_00000001_CPU_SYSTEM_total_95_SAVEMIN:-2s_2020-10-26T13:28:56Z_2501285.out
guider_00000005_00000001_USER_SAVE_2020-10-26T13:35:26Z_2503149.out.gz
guider_00000005_00000002_USER_SAVEMIN_2020-10-26T13:35:39Z_2503209.out.gz
guider_00000005_00000003_USER_SAVERAW_2020-10-26T13:35:49Z_2503268.out.gz
guider_00000006_00000001_USER_SAVE_2020-10-26T13:36:29Z_2503473.out
guider_00000006_00000002_USER_SAVEMIN_2020-10-26T13:36:54Z_2503567.out
guider_00000006_00000003_CPU_SYSTEM_total_95_SAVEMIN:-2s_2020-10-26T14:09:38Z_2512910.out
guider_00000006_00000004_CPU_SYSTEM_total_95_SAVEMIN:-2s_2020-10-26T14:25:36Z_2513626.out
guider_00000006_00000005_USER_SAVEMIN_2020-10-26T14:26:00Z_2513665.out
guider_00000006_00000006_USER_SAVERAW_2020-10-26T14:26:06Z_2513676.out
guider_00000007_00000001_CPU_SYSTEM_total_95_SAVEMIN:-2s_2020-10-26T14:26:31Z_2513698.out.gz
guider_00000007_00000002_USER_SAVE_2020-10-26T14:26:47Z_2513703.out.gz
guider_00000008_00000001_USER_SAVEMIN_2020-10-26T14:49:10Z_2514659.out.gz
guider_00000008_00000002_USER_SAVEMIN_2020-10-26T14:51:24Z_2514761.out.gz
guider_00000008_00000003_USER_SAVEMIN_2020-10-26T14:51:30Z_2514766.out.gz
guider_00000008_00000004_USER_SAVE_2020-10-26T14:51:39Z_2514770.out.gz
```

```
# Start ctop daemon with config file
guider ctop -C /etc/guider/guider.conf
```

```
// cpu threshold
{
  "cpu": {
    "SYSTEM": [{
      "apply":   "true",
      "total":   85,
      "comp":    "big",
      "command": ["SAVERAW"],
      "message": "CPU usage is high"
    }]
  }
}
```

```
// mem threshold
{
  "mem": {
    "SYSTEM": [{
      "apply":     "true",
      "available": 20,
      "comp":      "less",
      "command":   ["SAVERAW"],
      "message":   "free memory is low"
    }]
  }
}
```

SAVERAW — what it captures

Complete raw snapshot saved to file: all process metrics · memory maps · I/O stats · kernel counters → ready for AI analysis (ASKAI/ASKRUN) or offline review

Continuous Tracing

Continuous Tracing Daemon — ctop

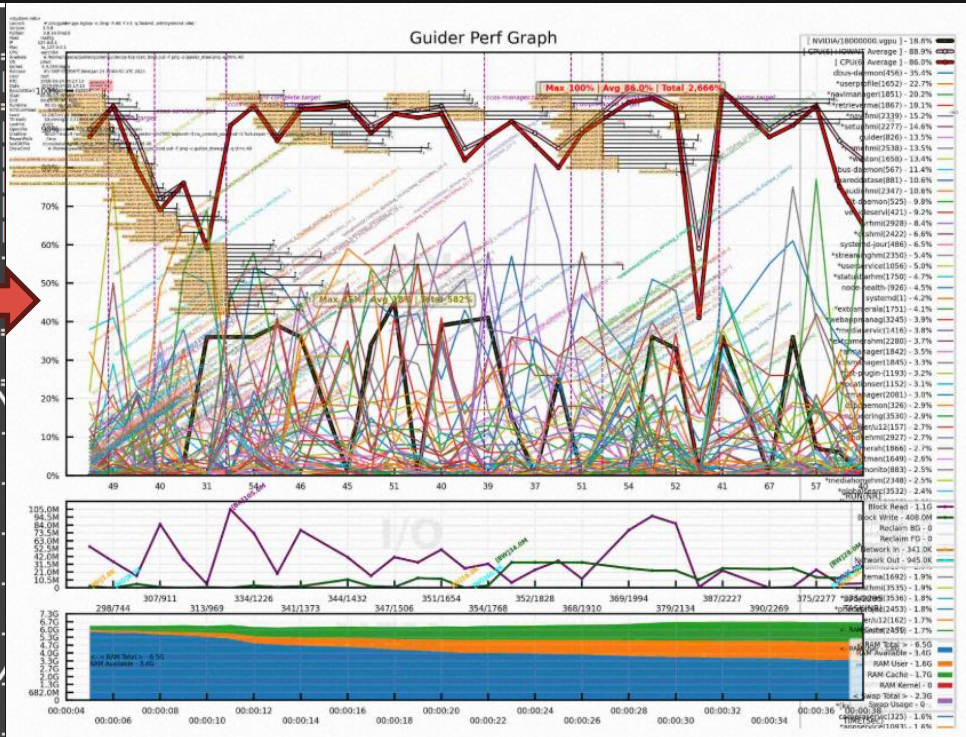
[Top Summary Info]

IDX	Interval	CPU	Avl	BlkRW	Blk	SWAP	NrPgRclm	NrFlt	NrCtx	NrIRQ	NrTask	Core	NetIO
1	51.990 - 53.290	92	4376/1200/1174	3/10	0	0	0/0	4	30021	21379	349/1952	6	1K/3K
2	53.290 - 54.350	60	4290/1237/1188	10/1	0	0	0/0	8	20295	15582	349/1969	6	1K/3K
3	54.350 - 55.400	66	4246/1270/1295	102/0	0	0	0/0	74	20358	16316	384/2025	6	1K/1K
4	55.400 - 56.440	83	4233/1270/1314	25/0	0	0	0/0	6	20845	16073	383/2025	6	1K/1K
5	56.440 - 57.490	17	4228/1282/1318	0/0	0	0	0/0	2	10130	7753	383/2024	6	1K/1K

[Top CPU Info] (Unit: %) (New: +) (Die: -)

COMM (	PID/	PPID/	Nr/ Pri)|	Min/Avg/Max/Tot	1	2	3	4	5
[CPU/AVG] (	-/	-/	-/ -)|	17/63.6/92/318	92	60	66	83	17
globalsearch (	3198/	1/	12/C 0)|	13/26.0/83/130	34	83	13	0	0
dbus-daemon (	444/	1/	1/C-19)|	5/21.8/50/109	50	30	19	5	5
systemd-udev (	594/	1/	1/C 0)|	0/17.0/65/85	0	0	20	65	0
dbus-daemon (	551/	1/	1/C 0)|	3/14.2/40/71	8	3	20	40	0
quickcontrol (	3199/	1/	17/C 0)|	12/11.8/47/59	12	47	0	0	0
homehmi (	2354/	1/	23/C 0)|	1/10.6/50/53	50	2	0	1	0

[Top Delay Info] [Top Prio Info]
[Top GPU Info] [Top VSS Info]
[Top RSS Info] [Top Block Info]
[Top Storage Info] [Top Network Info]
[Top Cgroup.CPU Info] [Top Cgroup.Throttle Info]
[Top Cgroup.Mem Info] [Top Cgroup.Blk Info]



Continuous Tracing

Anatomy of a Threshold Rule — CPU Example

guider.conf <threshold>.cpu.SYSTEM — one real rule, every option it can carry

```
"cpu": {
  "SYSTEM": [{
    "apply":    "true",
    "total":    90,
    "type":     "avg",
    "comp":     "big",
    "interval": 5,
    "after":    "60s",
    "oneshot":  "true",
    "refresh":  120,
    "taskmon":  "true",
    "perm":     "root",
    "lock":     "true",
    "command":  ["SAVE:5s", "CMD_ASKRUN_CPU"],
    "message":  "system CPU usage is critically high",
    "explain":  "sustained avg CPU >=90% for 5 ticks,
                 60s after boot -> snapshot + AI
                 diagnosis and auto-remediation"
  }]
}
```

```
// defined once, top-level "COMMAND" block
"COMMAND": {
  "CMD_ASKRUN_CPU": "ASKRUN:CPU overload detected.
    Diagnose root cause and suggest
    remediation#TIMEOUT:60,MAXCMD:2"
}
```

Trigger Condition

- `total: 90, comp: "big"` — fire when CPU ≥ 90%
- `type: "avg"` — sustained average, not one spike
- `interval: 5` — must hold for 5 consecutive ticks

Timing & Scope

- `after: "60s"` — ignore boot-time noise
- `oneshot` + `refresh: 120` — fire once, 120s cooldown
- `taskmon` — attach per-task breakdown to the snapshot
- `perm: "root"` — only evaluated when guider runs as root
- `lock: "true"` — freezes all other threshold checks while this event is active

Actions — AI-integrated

- `SAVE:5s` — raw snapshot captured first
- `CMD_ASKRUN_CPU` — a name, not a keyword; resolved against the `COMMAND` block shown below-left
- Lighter alternative: `CMD_ASKAI_CPU` — analysis only, nothing executed

ctop — What Can Be Monitored

guider.conf <threshold> — 27 resource types, each with fine-grained attributes

System Performance

- cpu — total / user / kernel / iowait / irq / temp / nrCore
- core — per-core usage / freq / governor / online status
- pmu — IPC / cache-miss rate / branch-miss rate (PMU counters)
- psi — cpu/memory/io — some-avg10 / full-avg10 / diff
- load — load1m / load5m / load15m
- task — nrRun / nrCtx / new / die / abnormal (D/Z state)
- interdiff — collection interval delay (sec)

Memory & I/O

- mem — available / anon / file / slab / nrMinFlt / oomKill / rss / nrTickLmk / nrTotalLmk (Android)
- swap — usage / usagePer / swapin / swapout / usageDiff
- block — read / write / ioWait / nrMajFlt / nrRun
- storage — usagePer / free / favail / readtime / writetime
- cgroup — cpu / memory / throttle / read / write / cpuDelay

Network, Android & More

- net — inbound / outbound / recv / trans (per interface, size suffix)
- sock — nrTCPConn / nrTCPRetrans / RxQ / TxQ
- anr — nrTickANR / nrBootANR / nrTotalANR (Android)
- file — exist / none / size / data / dataInt (file & dir monitoring)
- battery — per / left / plugged (Android)
- gpu / gpumem — usage / size
- log — filter pattern across kernel / DLT / Android / journal
- bpf — RUNQLAT / BLKLAT / FUNCCOUNT / FUNCLAT (eBPF-backed)

Common Control Attributes

- apply — activate rule ("true" / "false")
- comp — comparison: "big" (≥) / "less" (≤) / "eq" (==)
- command — action list: SAVERAW, ASKAI, ASKRUN, SNAPSHOT, WEBHOOK...
- interval — condition must hold for N ticks (duration gate)
- oneshot — fire only once, then auto-disable
- after/before — fire only within an uptime window
- predict — fire *before* threshold on rising trend
- escalation — ordered action sequence per fire
- message / explain — attach description to event
- includeOR / exceptOR — process whitelist / blacklist filter
- taskmon — include per-task breakdown in saved output
- prop — Android property condition (e.g. "sys.boot_completed|1")

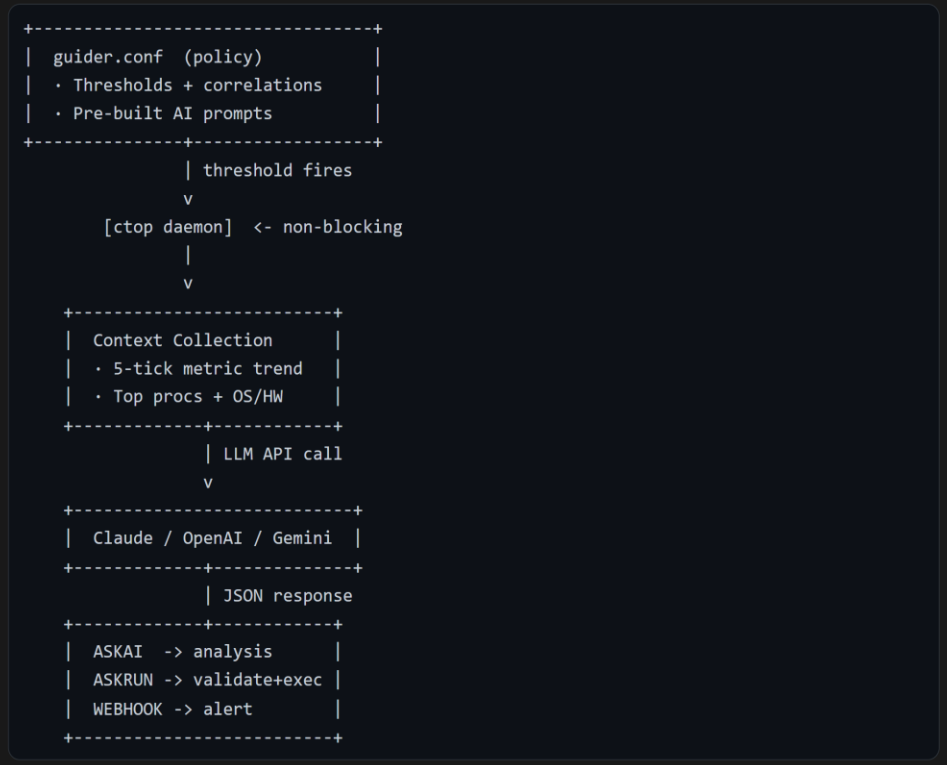
Correlation Group

```
// fires when ALL members trigger together
"oom_risk": {
  "apply":   "true",
  "members": ["MEM_SYSTEM_available",
              "MEM_SYSTEM_oomKill"],
  "window":  2,
  "commands":["SAVERAW",
              "ASKAI:Explain OOM risk and suggest relief"]
}
```

Automated Insights

The AI Loop Architecture

ASKAI — ask & explain. Read-only, nothing is ever executed.



ASKRUN — ask & act. Validated commands can actually run.

Non-blocking by Design

LLM calls run in a daemon thread.
ctop monitoring loop **never waits** for AI.
No data collection gap during analysis.

Deep Context

- 5-tick metric trend (not just "now")
- Top resource consumers ranked
- Recent threshold event history
- Hardware: CPU model, RAM, kernel version

Provider Flexibility

`Claude (default)` `OpenAI` `Gemini` `Custom endpoint`

guider.conf <llm> — Global AI Configuration

One block, set once — every ASKAI/ASKRUN call on the device reads from here

```
<llm>
{
  "PROVIDER": "claude",
  "MODEL": "sonnet",
  "CACHE": "1",
  "API_KEYS": {
    "ANTHROPIC": "sk-ant-...",
    "OPENAI": "", "GOOGLE": "", "CUSTOM": ""
  },
  "CUSTOM_BASE_URL": "",
  "CONTEXT": {
    "MEMINFO": ["MemAvailable","Cached",
                "SwapUsed","Dirty"],
    "PSI": true,
    "BINDER": true,
    "THERMAL": true,
    "CPUFREQ": true
  }
}
```

```
"ASKRUN": {
  "SEVERITY_ACTIONS": {
    "critical": ["SNAPSHOT","GETDUMP",
                 "INTERVAL:1"],
    "high":   ["SAVE","INTERVAL:2"],
    "medium": ["SAVEMEM"],
    "low":    []
  },
  "SUBSYSTEM_ACTIONS": {
    "mem": ["SAVEMEM","AUTOOOMADJ:1000"],
    "cpu": ["AUTOLIMITCPU:90",
            "AUTOSCHED:B:0"]
  },
  "AUTO_ESCALATE": "false",
  "MAX_AUTO_CMDS": 3
}
```

Auto-inserted commands still pass the same whitelist check — AUTO_ESCALATE defaults to off

Priority for every field: -q option > this config file > environment variable

Always Included — no config needed

- event — name, uptime, timestamp, the threshold rule's own fields
- system — full sysSnapshot() (CPU/mem/I/O/PSI etc.)
- top_processes — top-N by CPU, N = CTXDEPTH (default 10)
- active_events — every threshold currently firing
- metric_history — last 5 ticks per metric (the "5-tick trend")
- recent_events — correlated events within the window
- os_info — kernel version, architecture
- network_info — TCP/UDP socket counts

CONTEXT Adds — opt-in, only when the key is present

- MEMINFO → /proc/meminfo
- DMESG:N → last N kmsg lines
- DISKSTATS → /proc/diskstats
- CPUSTAT → /proc/stat per-core
- INTERRUPTS:N → top-N IRQs
- SOFTIRQS → /proc/softirqs
- VMSTAT → /proc/vmstat
- BUDDYINFO → fragmentation
- PSI → /proc/pressure/*
- SLABINFO → /proc/slabinfo
- BINDER → Android IPC (Android only)
- CPUFREQ / THERMAL → sysfs freq/temp

Left is sent on **every** call · right is layered on top only when the matching <llm>.CONTEXT key (previous slide) is set.

Automated Insights

ASKAI — Observe & Explain Deeply

Analysis only — the LLM explains what's happening, nothing is ever executed.

Pattern 1: With ctop (threshold trigger)

```
# guider.conf cpu.SYSTEM rule:
#   "command": ["ASKAI:Explain the CPU spike
#                and IPC contention"]
guider ctop -c guider.conf

# [LLM:ASKAI] CPU spike traced to binder thread
#   starvation in com.android.systemui.
#   Peak at 17:42:08. 3 threads blocked on
#   Binder:1234 waiting for display service
#   reply (avg 180ms).
```

Pattern 2: Embedded in Visualization

```
guider drawflame guider.out -q ASKAI -F html
# → guider_flamegraph.html
#    AI Analysis panel (expand/collapse)
#   Anomaly sections highlighted in red
```

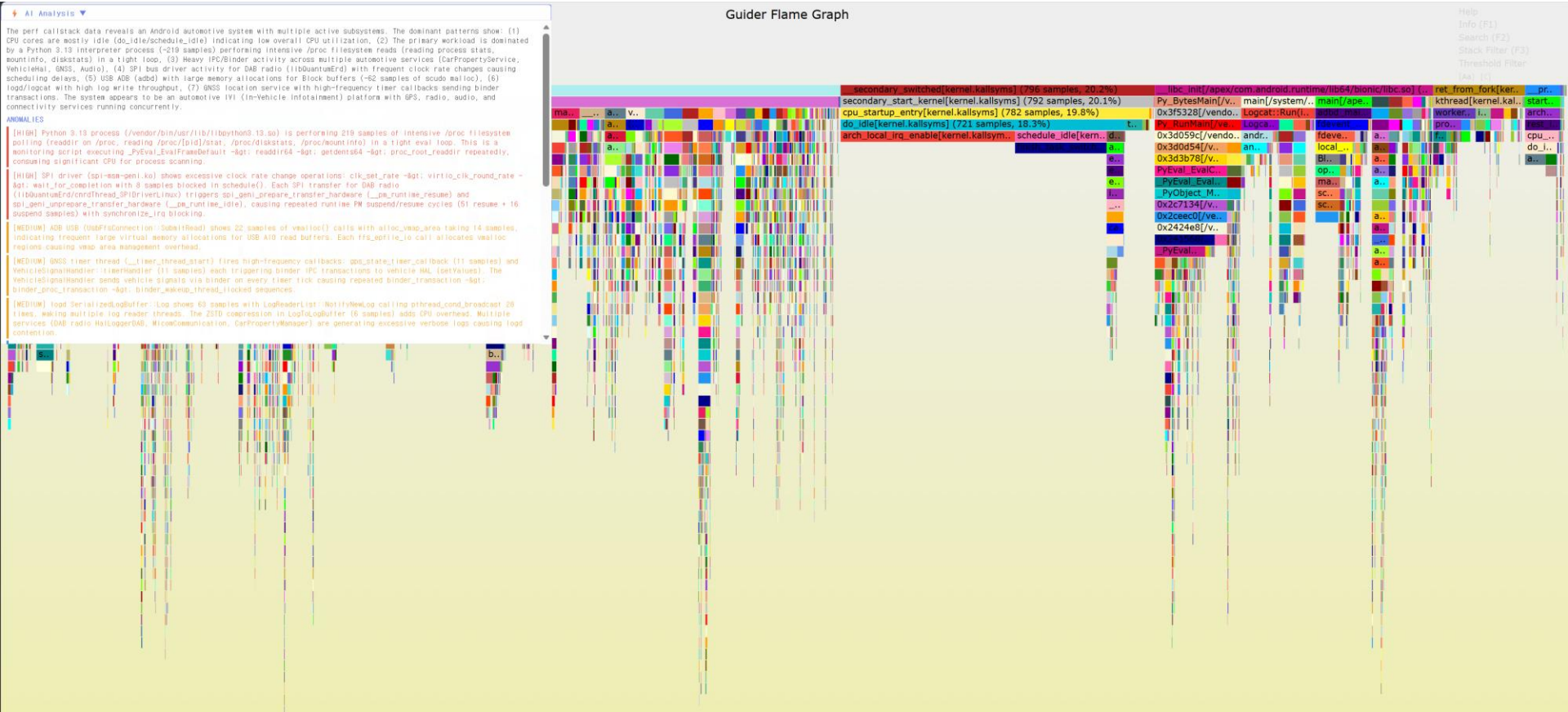
What ASKAI Sees

- 5-tick metric trend (not just a snapshot)
- Top processes by CPU/mem/I/O
- Stack frames (when drawflame mode)
- Previous threshold events
- Hardware: CPU model, total RAM, kernel
- Android: Binder threads, LMK state

One HTML file = data + AI analysis + context

Share it — they get everything.

Automated Insights



Automated Insights

Anatomy of a Threshold Rule — ASKRUN Example

guider.conf <threshold>.mem.SYSTEM — the real rule behind the next slide's JSON response

```
"mem": {
  "SYSTEM": [{
    "apply":     "true",
    "available": 200,
    "comp":      "less",
    "type":      "avg",
    "interval":  3,
    "after":     "30s",
    "oneshot":   "true",
    "refresh":   300,
    "taskmon":   "true",
    "perm":      "root",
    "autoTarget":"topRss",
    "command":   ["SAVE", "CMD_ASKRUN_OOM"],
    "message":   "system available memory is
                  critically low",
    "explain":   "sustained avg available memory
                  <200MB for 3 ticks -> snapshot +
                  AI diagnosis, DRYRUN-safe by default"
  }]
}
```

```
// defined once, top-level "COMMAND" block
"COMMAND": {
  "CMD_ASKRUN_OOM": "ASKRUN:OOM risk detected.
    Memory pressure is critical. Diagnose and
    suggest safe preemptive relief
    #TIMEOUT:60,MAXCMD:3,DRYRUN:true"
}
```

Trigger Condition & Scope

- `available: 200, comp: "less", type: "avg", interval: 3` — sustained <200MB, not one dip
- `after: "30s"`, `oneshot` + `refresh: 300` — ignore boot noise, fire once, 5-min cooldown
- `taskmon, perm: "root"` — per-task breakdown, root-only
- `autoTarget: "topRss"` — governor tracks the top memory consumer, so AUTOOMADJ/AUTOSIGNAL below act on the right process

Actions — ASKRUN wired in

- `SAVE` → `CMD_ASKRUN_OOM`, resolved against the `COMMAND` block below-left
- `DRYRUN:true` baked into the template — always previews, regardless of severity

Inline Options — `#OPT1:v1,OPT2:v2`

- `TIMEOUT:<sec>` — LLM request timeout (60)
- `DRYRUN:<bool>` — log, don't execute
- `MAXCMD:<n>` — max commands to run (3)
- `CTXDEPTH:<n>` — top-N procs in context (10)
- `PROVIDER:<name>` — override LLM provider
- `MODEL:<name>` — override LLM model
- `FEEDBACK:<sec>` — re-check effect after exec

Global `-q` equivalents: LLMRATELIMIT, LLMDRYRUN, LLMMAXCMD, LLMCTXDEPTH, LLMAUDITLOG, LLMALLOWCMD

Automated Insights

ASKRUN — Observe, Decide & Act

Same idea as ASKAI, but the LLM's suggested commands can actually run — gated by severity, a whitelist, and dry-run.

```
{
  "analysis":   "Memory pressure from 3 OOM-risk processes. com.ivi.maps holds 820MB RSS and 3 wake locks.",
  "severity":   "high",
  "subsystems": ["mem", "binder"],
  "confidence": 0.91,
  "commands":   ["SNAPSHOT:oom_event", "AUTOOMADJ:500"],
  "rationale":  "Proactive OOM adj before LMK triggers nav service kill"
}
```

JSON schema enforced — AI cannot output freeform commands.

Before every call

Rate-limit (60s) + duplicate-pending guard per event — skip if still cooling down
Runs in a daemon thread — ctop loop never blocks on it

Severity Routing (how urgent the LLM judges the event)

CRITICAL	HIGH	MEDIUM	LOW
5 cmds · execute	3 cmds · execute	1 cmd · dry-run	0 cmd · analyze only

After the response arrives

- Parse JSON → severity sets maxCmd/dryRun, unless the command template already fixed them (like CMD_ASKRUN_OOM); guider.conf <llm>.ASKRUN.AUTO_ESCALATE (shown a few slides back) may inject more
- Each command re-checked against the 39-cmd whitelist (pre-approved commands — full list coming up), one at a time
- Dry-run (preview only — nothing actually runs) → [LLM:DRYRUN] would execute; live → [LLM:EXEC] re-enters handleEventCmd() — same path normal threshold commands use
- Blocked → [LLM:BLOCKED], never runs
- Optional FEEDBACK:<sec> — re-asks the LLM "was this effective?" after execution
- Every outcome (severity, executed, blocked) written to LLMAUDITLOG as JSONL

Example — What CMD_ASKRUN_OOM Actually Returns

A worked trace: which of the 39 whitelisted commands the LLM can pick, and what happens to each

```
{
  "analysis":   "com.ivi.maps holds 820MB RSS, 2 wake
                 locks; available memory 178MB, under
                 200MB for 3 ticks.",
  "severity":   "critical",
  "subsystems": ["mem"],
  "confidence": 0.88,
  "commands":   ["AUTOOMADJ:700",
                 "AUTOSIGNAL:SIGSTOP",
                 "RESTART"],
  "rationale":  "Raise OOM priority and freeze the
                 process before LMK forces a hard kill"
}
```

Same shape as the previous slide's schema — severity=critical here, but CMD_ASKRUN_OOM's baked-in DRYRUN:true still forces preview-only.

```
[LLM:ASKRUN] com.ivi.maps holds 820MB RSS and 2 wake
             locks; recommend raising OOM priority and
             freezing before LMK forces a hard kill.

[LLM:DRYRUN] would execute: AUTOOMADJ:700
[LLM:DRYRUN] would execute: AUTOSIGNAL:SIGSTOP
[LLM:BLOCKED] command not allowed: RESTART
```

Why these two ran, and one didn't

- AUTOOMADJ:700, AUTOSIGNAL:SIGSTOP — both in the 39-cmd whitelist; autoTarget: "topRss" from the rule points them at com.ivi.maps automatically
- RESTART — hard-blocked (P10, layer 2) before the whitelist is even checked; stripped no matter how confident the LLM is
- All 3 counted toward MAXCMD:3 — a 4th suggestion would be ignored entirely

Other whitelisted commands that fit memory pressure: SAVEMEM, SAVEMEMPROC, SAVESWAP, SNAPSHOT, AUTOSWAPOUT, AUTORESTORE — the LLM picks from the same 39 regardless of resource type.

Automated Insights

Safety by Design

"Can the AI brick my device?" — No. Here's why:

1. **Whitelist: 39 commands only** — SAVE, SNAPSHOT, AUTOOMADJ, AUTOIONICE, AUTONICE, AUTOLIMITCPU, SAVERAW, WEBHOOK...
   Anything not on the list is structurally impossible to execute.

2. **Hard block — EXIT / RESTART / STOP / RELOAD / UPDATE** — Even if AI outputs these, they are removed before whitelist check.
   The daemon cannot kill itself or the host system.

3. **Dry-run default for medium/low severity** — Commands are *suggested*, not executed.
   Only critical/high severity triggers real execution.

Rate Limiting

60-second cooldown per event type.
Prevents runaway LLM call storms.

JSONL Audit Trail

Every decision (executed + blocked) logged with full context for post-mortem replay.
Enable: `-q LLMAUDITLOG:/path/to/audit.jsonl`

*Three structural layers ensure the AI **cannot brick your device** — by design, not by policy.*

Q&A

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT KOREA

서울