Telling "stuck" from "long": cheap online detection and bounding of non-terminating agentic loops

Christian Hansen*
Western Academy of Beijing
Beijing, China

Abstract

LLM agents that act in tool-use loops routinely fail to terminate: they repeat an action forever, oscillate among a handful of states, or keep issuing superficially different tool calls while making no real progress. The universal mitigation in deployed systems is a single blunt parameter—a maximum step count, `max_steps`. A tight cap prematurely kills trajectories that are simply *long but healthy*; a loose cap lets a dead trajectory burn its full token budget before stopping. We show that "stuck" and "long" are cheaply separable online. We introduce LoopGuard, a model-agnostic detector that fuses three complementary per-step signals—action-signature periodicity, state-embedding recurrence, and novelty collapse—into a single stuckness score, and feeds that score to a CUSUM sequential-change detector. CUSUM turns loop-bounding into a single interpretable knob that trades detection latency and token savings against the risk of cutting a healthy run. Across five seeds of a benchmark spanning the three pathologies plus realistic "debugging-plateau" healthy runs, LoopGuard detects loops a median of 13.8 ± 0.4 steps after onset with a 0.1% false-cut rate, recovering 86% of a clairvoyant oracle's token savings—while a large fixed cap only stops at step 113 and an exact-repeat heuristic is entirely blind to two of the three pathologies. A matched-false-cut ablation locates each signal's role precisely: action-only detection misses stagnation completely at any tuning; state-only recurrence is a strong baseline; and adding the cheap periodicity signal cuts detection latency on the two most common (repetitive) pathologies by ∼25% at a single, healthy-calibrated operating point. The CUSUM latency law (latency linear in the threshold) is confirmed empirically at $R^2 = 0.998$. Detection costs ∼28 μs/step—roughly five orders of magnitude below one model call—and all five *emergent, non-scripted* looping agents we build are caught early with no false cuts. Code, data, and self-checks are pure standard library and fully deterministic.

Keywords

LLM agents, tool use, non-termination, loop detection, CUSUM, sequential change detection, agent reliability

1 Introduction

The dominant agent design pattern—observe, think, call a tool, observe the result, repeat—is an unbounded loop around a language model. Frameworks in this family (ReAct-style reasoning-and-acting agents [1]; reflective agents that critique and retry [2]; tool-augmented models [3]) all share one structural hazard: **the loop has no guaranteed termination condition.** The agent decides when it is done, and a model that is confused, adversarially prompted, or simply working on an underspecified task can decide "not yet" indefinitely.

Every production agent therefore ships a backstop: a hard cap on the number of steps (or tokens, or wall-clock). Popular open agent scaffolds encode exactly this—a loop with a maximum-iteration guard. That cap is doing two incompatible jobs at once:

(1) **Killing genuinely stuck runs**, where every additional step is pure waste; and
(2) **Bounding genuinely productive runs**, where additional steps are doing real work and the cap is only a safety limit that should almost never bind.

A single scalar cannot serve both. Set it low and you truncate the hard, long tasks you most wanted the agent to finish (job 2 fails). Set it high and every stuck run wastes the entire budget before the cap catches it (job 1 fails). The cap treats a 90-step legitimate research task and a 90-step infinite loop identically, because it never looks at *what the agent is doing*—only at *how many times it has done something*.

This paper's claim is that the missing information is cheap to recover. A stuck agent leaves a statistical signature in its own action/observation stream—repetition, recurrence, or a collapse in novelty—that is detectable online, without a second model, without embeddings from a large network, and without touching the agent. Separating "stuck" from "long" lets a system stop dead runs early *and* let healthy long runs continue, from one detector with one interpretable knob.

Contributions.

- **A three-signal decomposition of non-termination** (§2): fixed point, oscillation, stagnation—and the observation, later made quantitative, that no single cheap signal covers all three at an acceptable false-cut rate.
- **LoopGuard** (§3): an $O(\text{window})$-per-step detector fusing the three signals into a stuckness score, with a **CUSUM sequential-change rule** giving a single principled threshold and a closed-form latency law.
- **A rigorous, honest evaluation** (§4–§5): multi-seed confidence intervals; a **matched-false-cut ablation** that removes the "unfair operating point" objection and pinpoints each signal's role; a **clairvoyant oracle** upper bound; empirical validation of the latency law; hyperparameter sensitivity; and a suite of five **emergent, non-scripted** looping agents.
- **A fully reproducible artifact**: pure standard library, deterministic, with runnable self-checks; every number in §5 is emitted by the scripts.

*Manuscript, August 2026.

2 Problem formulation

2.1 The agentic loop and its cost

An agent run is a sequence of steps $x_1, x_2, \ldots$. Each step $x_t = (a_t, o_t)$ pairs an **action** a_t (a tool name plus arguments) with the resulting **observation/state** o_t (text). Each step has a token cost $c_t > 0$ (prompt + generation + tool result). A run either terminates when the agent emits a stop action, or is cut externally at some step τ. Total cost is $\sum_{t \le \tau} c_t$. A *loop-bounding policy* is an online rule that, seeing only $x_1, \ldots, x_t$, outputs stop or continue at each t. The fixed cap is the trivial policy "stop at t = cap."

2.2 A taxonomy of non-termination

We distinguish three qualitatively different failure modes. Let $\phi(o)$ be an embedding of a state.

- **Fixed point.** For $t \ge$ onset, $a_t \equiv a^*$ and $o_t \equiv o^*$: the agent retries one thing forever. *Signature:* action signatures repeat with period 1; states are identical.
- **Oscillation.** For $t \ge$ onset, (a_t, o_t) cycles with period $p \ge 2$ among a small set: the agent bounces among a few states. *Signature:* action signatures repeat with period p; states recur.
- **Stagnation.** For $t \ge$ onset, the **actions keep changing**—every step issues a fresh tool call with new arguments—but the **state stops advancing**: $\phi(o_t)$ hovers near a fixed region. *Signature:* action signatures are all distinct (period detection fails); states have low novelty.

Stagnation is the adversarial case for the naive heuristics. "Stop after N identical tool calls" never fires, because no two calls are identical. Only a signal that looks at *state progress*, not *action surface*, catches it. §5.2 turns this intuition into a measured result.

2.3 What a good policy must do

Against a *healthy* run—which makes real progress but is not pristine, and in particular may contain a temporary **debugging plateau** (several steps of varied actions with little state progress, then a breakthrough)—the policy must **not** stop early. Against a *stuck* run, it must stop **soon after onset**. The tension between these is the entire problem, and the plateau is where it bites: for a while, a healthy plateau and permanent stagnation are statistically indistinguishable. A good policy defers judgment just long enough to tell them apart, and exposes that "just long enough" as a tunable knob.

3 Method: LoopGuard

LoopGuard maintains a rolling window of the last W steps, computes three signals per step—each targeting a different pathology—then fuses and thresholds them.

3.1 Three signals

(1) Action-signature periodicity. Canonicalize each action to a signature string s_t (tool + arguments). Over the window, compute the best agreement rate that the signature repeats at some period $p \in \{1, \ldots, P\}$:

$$\text{period}(t) = \max_{1 \le p \le P} \frac{1}{|W| - p} \sum_{i} \mathbf{1}[s_i = s_{i-p}]. \tag{1}$$

Near 1 for a fixed point (p=1) or oscillation ($p \ge 2$), low when signatures vary. Exact, integer-only, and deliberately *blind* to stagnation—that is signal (3)'s job.

(2) State recurrence. Embed each state $\phi(o_t)$ (an L2-normalized bag-of-tokens vector by default; any semantic embedder drops in unchanged). Track the EWMA of the maximum cosine similarity of the current state to any prior state in the window:

$$\text{recur}(t) = \text{EWMA}\Big(\max_{i<t} \cos(\phi(o_t), \phi(o_i)) \Big). \tag{2}$$

High recurrence means the agent keeps returning to states it has already visited—the state-space signature of oscillation and fixed points.

(3) Novelty collapse. Novelty is $1 - \max_{i<t} \cos(\phi(o_t), \phi(o_i))$; progress is its EWMA. When an agent stagnates, novelty collapses toward 0 *even though its actions vary*. This is the only one of the three that catches stagnation, precisely because it ignores the action surface and measures state advancement.

3.2 Fusion into a stuckness score

The three signals combine into a single score $s_t \in [0, 1]$ with normalized weights:

$$s_t = w_1 \,\text{period}(t) + w_2 \,\text{recur}(t) + w_3 \,(1 - \text{progress}(t)). \tag{3}$$

Because the signals are complementary, a stuck run drives s_t high regardless of *which* pathology it is, while a healthy, progressing run keeps s_t low. §5.2 shows the fusion is load-bearing.

3.3 CUSUM stopping rule

A single-step threshold on s_t is brittle: healthy plateaus spike s_t transiently (Figure 4). We instead treat loop onset as a **change in the mean of** s_t and detect it with a cumulative-sum (CUSUM) test—the classical, asymptotically latency-optimal sequential detector for a mean shift [5, 6]. With reference value k (the stuckness tolerated under healthy operation) and threshold h:

$$g_t = \max\big(0,\ g_{t-1} + (s_t - k)\big), \qquad \text{alarm when } g_t \ge h. \tag{4}$$

The accumulator drifts *down* toward 0 whenever $s_t < k$ (healthy progress) and grows *up* only under a sustained shift to $s_t > k$ (a real loop). A transient plateau adds a bounded bump that decays; a permanent loop grows without bound and crosses h.

Latency law. Under a loop with mean stuckness $\mu_1 > k$, the accumulator grows at rate $\mu_1 - k$ per step, so expected detection latency after onset is approximately

$$\mathbb{E}[\text{latency}] \approx \frac{h}{\mu_1 - k}. \tag{5}$$

This is the design equation: h is the single knob, latency is linear in it, and the slope is set by the healthy/loop stuckness gap. §5.3 confirms the linear form at $R^2 = 0.998$.

3.4 Complexity and footprint

Every per-step operation is bounded by the window: periodicity is $O(P \cdot W)$ integer comparisons, the state signals are $O(W)$ sparse-cosine evaluations, CUSUM is $O(1)$. Total is $O(W)$ per step with a small constant and no allocation beyond the window. There is

Table 1: The five emergent, non-scripted agents. Loops arise from the dynamics, not from scripted repetition.

agent	mechanism	pathology
`gridworld`	memoryless greedy nav. frozen by walls	fixed-point
`lcg_explorer`	iterated map with a zero-free orbit	oscillation
`thrashing_planner`	two subgoals that undo each other	oscillation
`retry_on_error`	identical retry under permanent error	fixed-point
`saturated_search`	fresh queries on an exhausted corpus	stagnation

no model call, no GPU, and no dependency beyond the standard library. Measured cost is ~28 μs/step (§5.5).

4 Experimental setup

4.1 Trajectory harness

We generate trajectories with known ground truth, each a list of (action-signature, state-text) steps plus a label and loop-onset step. *Healthy* runs make genuine progress (each step introduces a fresh milestone token), with two realistic bumps: frequent tiny slow-progress stalls, and—in 45% of runs—one **debugging plateau** of 4–9 steps where actions *vary* but the state barely advances, then progress resumes. Lengths vary uniformly in $[20, 90]$; some are legitimately long. *Fixed-point / oscillation / stagnation* runs share a healthy prefix (onset $\in [8, 20]$), then enter the corresponding pathology and continue to length 128, so a large fixed cap would run them to the end. Modeling the plateau is not incidental: an earlier pristine-healthy version produced a suspicious 0% false-cut rate across the *entire* threshold sweep—the artifact of an unrealistically easy negative set.

4.2 Emergent (non-scripted) agents

To guard against the risk that a synthetic generator and the detector share assumptions, we also build five agents whose loops **emerge from policy × environment dynamics** rather than being written into the trajectory (Table 1). Each has a healthy twin that terminates; the "stuck" flag changes the *environment*, never the trajectory.

4.3 Dataset, baselines, metrics

Dataset. 150 trajectories per class × 4 classes = 600 per seed; five seeds; fully deterministic. Per-step token cost is the token count of the step's action + observation text.

Baselines. `fixed_small` (`max_steps`= 40); `fixed_large` (`max_steps`= 128, the reference for token savings, never binds on our healthy set); `exact_repeat` (stop after 3 identical consecutive action signatures); LoopGuard (default, $h = 3.0$). We also report a **clairvoyant oracle** that stops exactly at loop onset—an unachievable upper bound on online token savings.

Metrics. Detection latency (steps between onset and stop; median); false-cut rate (fraction of healthy runs stopped before natural completion); tokens saved per stuck run (vs. `fixed_large`); tokens wasted per healthy run (on false cuts); per-step overhead. All aggregated as mean ± std over the five seeds.

Table 2: Main comparison over 5 seeds (mean ± std). Latency in steps; tokens per run. LoopGuard is the only method that is fast, safe, and pathology-complete.

detector	latency	false-cut	tok saved	tok waste
`fixed_small`	25.0	68.4%	1251	503
`fixed_large`	113.0	0.0%	0	0
`exact_repeat`	109.5	0.0%	469	0
LoopGuard	**13.8**	**0.1%**	**1398**	24

Table 3: Matched-false-cut ablation. Latency by pathology at the tuned k giving $\leq$ 1% false cuts. Novelty-only and recur+novelty match "state-only".

variant (matched $\leq$ 1%)	tuned k	fixed-pt	oscill.	stagn.
action-only (periodicity)	0.45	22.0	22.7	112.3
state-only (recurrence)	0.75	16.0	17.0	16.0
action + recurrence	0.45	15.0	16.7	66.0
full (3-signal)	0.45	**12.0**	**13.7**	18.0

5 Results

5.1 Main comparison (5 seeds)

Table 2 summarizes the five-seed comparison. `fixed_small` catches loops at a fixed latency of 25 but false-cuts two-thirds of healthy runs. `fixed_large` never false-cuts but only stops loops at step 113, saving zero tokens. `exact_repeat` is excellent on fixed points but, as §5.2 quantifies, blind to oscillation and stagnation, so its median latency is 109.5. LoopGuard catches loops a median of 13.8 steps after onset with essentially no false cuts, recovering 86.2% of the clairvoyant oracle's 1622-token savings. Figure 1 traces the full false-cut / latency frontier as h varies; every point dominates both fixed caps.

5.2 Ablation: why fusion is load-bearing

We remove signals and re-tune each variant to a **matched false-cut budget ($\leq$ 1%)** by raising the CUSUM reference k—so no variant is penalized by an operating point chosen to flatter the full model—then compare median detection latency per pathology at that fair point ($\approx$ 112 = effectively missed).

The result (Table 3, Figure 2) is precise, and more nuanced than "every signal helps":

- **Action-only structurally cannot see stagnation** (112.3) at *any* k—varied actions defeat periodicity. This is `exact_repeat`'s blind spot, isolated.
- **State-recurrence alone is a strong baseline.** Re-tuned to k=0.75, it catches all three pathologies at 16–17 steps with 0% false cuts. We report this rather than bury it: the *state* signals are what make coverage—especially of stagnation—possible.
- **Fusion's marginal value is speed on the common repetitive loops, tuning-free.** Adding the cheap periodicity signal cuts fixed-point latency 16→12 and oscillation 17→13.7—a ~25% reduction on the two most common pathologies—at

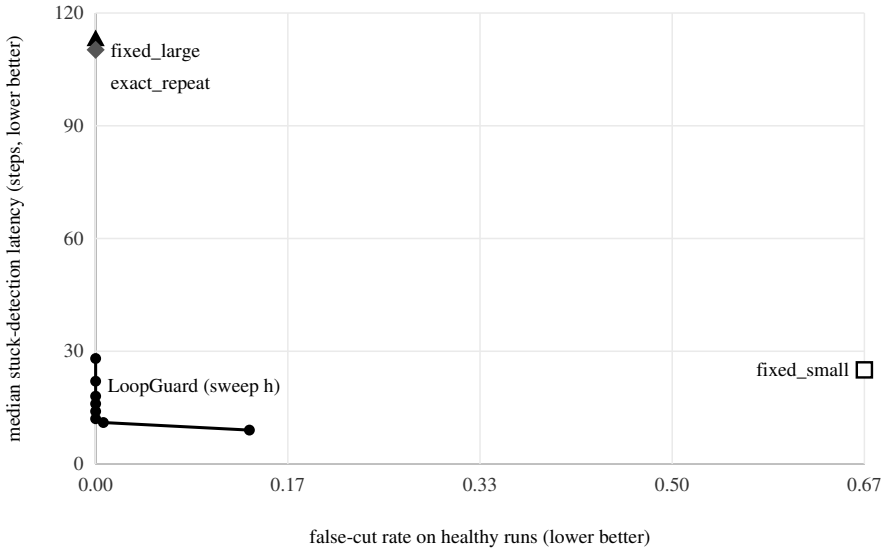

Figure 1: Operating curve: false-cut rate vs. detection latency as the CUSUM threshold h sweeps. All baselines are dominated.

the **default** k=0.45, calibrated directly to the measured healthy stuckness $\mu_0 \approx 0.44$, with no per-signal retuning.

- **Signal weighting interacts with the reference.** Diluting recurrence with periodicity at a fixed low k (action + recurrence) slows stagnation to 66; the balanced full configuration avoids this.

5.3 The latency law holds empirically

Sweeping h in the zero-false-cut regime and fitting median latency against h (Figure 3): latency is **linear in h with $R^2 = 0.998$**, exactly the form predicted by Eq. (5). Measured stuckness gap: $\mu_0 = 0.439$ (healthy), $\mu_1 = 0.865$ (inside loops), with $k = 0.45$. Fitted slope 1.94 vs. theoretical $1/(\mu_1 - k) = 2.41$, with a positive intercept ≈ 8. The first-order theory predicts the *shape* (linear) and the *order of magnitude* of the slope; the residual gap is the honest footprint of the fixed warmup and the EWMA ramp-up before the score settles at μ_1. The deployment-relevant claim—latency is linear and tunable through one knob—holds exactly.

5.4 Sensitivity

At $h = 3$, over five seeds (false-cut / median latency): LoopGuard is robust to window and EWMA and sensitive to k, and informatively so: false cuts vanish exactly as k crosses the measured healthy stuckness $\mu_0 \approx 0.44$. This yields a concrete tuning recipe—measure μ_0 on held-out healthy runs and set k just above it.

Table 4: Hyperparameter sensitivity at $h = 3$ (false-cut / latency). Robust to window and EWMA; informatively sensitive to k.

knob	values → (false-cut / latency)				
ref. k	.35:80%/9	.40:49%/12	**.45:0%/14**	.50:0%/15	.55:0%/17
window W	12:0%/11	18:0%/13	**24:0%/14**	36:0%/14	48:0%/14
EWMA	.2:0%/15	**.35:0%/14**	.5:0%/13	.7:0%/13	

Table 5: Emergent-agent results. Every loop caught early; no healthy twin cut.

agent	pathology	stuck: stop / len	healthy cut?
`gridworld`	fixed-point	13/60	no
`lcg_explorer`	oscillation	11/60	no
`thrashing_planner`	oscillation	9/60	no
`retry_on_error`	fixed-point	9/60	no
`saturated_search`	stagnation	21/60	no

5.5 Overhead

LoopGuard costs ~28 μs per step in pure Python. One LLM step costs on the order of seconds, so detection is $\sim 10^{-5}$ of a step's wall-clock—free in any practical sense, and it removes far more compute than it adds by ending dead runs early.

5.6 Emergent agents

On the five non-scripted agents ($h = 3$), LoopGuard catches every emergent loop early and false-cuts none of the healthy twins (Table 5).

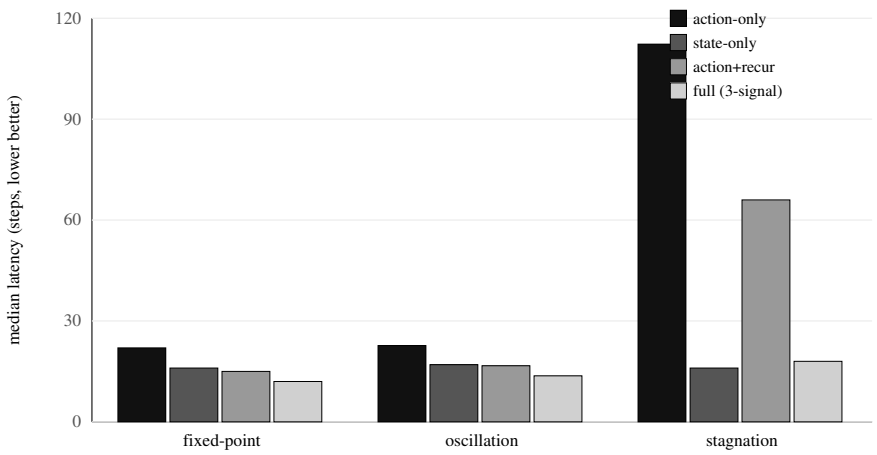

Figure 2: Matched-false-cut ablation: action-only misses stagnation; the full fusion is fastest on the repetitive pathologies.

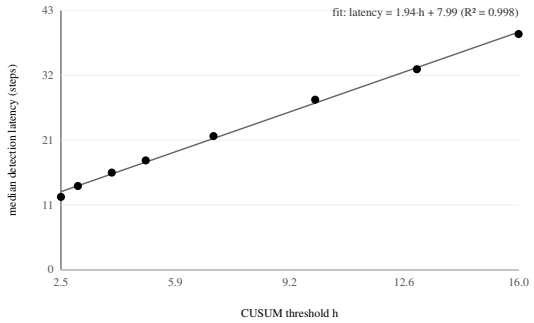

Figure 3: Median detection latency is linear in the threshold h ($R^2 = 0.998$), confirming the CUSUM latency law.

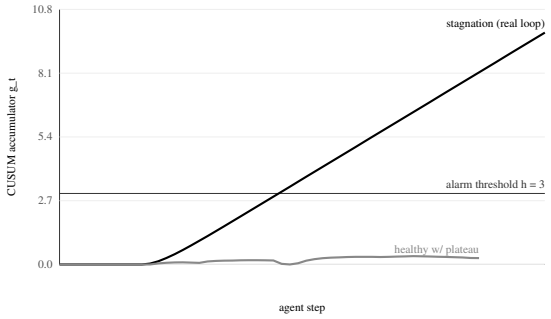

Figure 4: CUSUM accumulator over a real stagnation loop (crosses h) vs. a healthy debugging plateau (bumps and recedes). This is why CUSUM beats a single-step threshold.

The detector, designed against synthetic trajectories, generalizes to loops that arise from real agent dynamics it never saw—including the hardest case, emergent stagnation (`saturated_search`), caught at step 21 while its healthy twin completes untouched.

6 Discussion

Why fusion, concretely. The ablation (§5.2) is more careful than “each signal helps.” Action-only detection is *structurally disqualified*—it cannot see stagnation at any tuning. State-recurrence alone is a strong baseline we do not hide. Fusion’s defensible contribution over that baseline is twofold: it detects the two most common (surface-repetitive) pathologies ∼25% faster by folding in a near-free integer signal, and it reaches its operating point by calibrating one reference k to the measured healthy stuckness μ_0, rather than requiring a per-signal tuning search.

Why CUSUM rather than a threshold on s_t. The debugging plateau is the reason (Figure 4). A plateau drives s_t high for several steps, so a single-step threshold would cut it. CUSUM integrates evidence: a plateau contributes a *bounded* bump (it ends, s_t falls below k, and the accumulator drains), whereas a real loop contributes *unbounded* growth. The threshold h is exactly the amount of sustained evidence required, and it comes with the linear latency law of §5.3.

Choosing the knobs in deployment. Set k just above the healthy stuckness μ_0 measured on held-out healthy runs; then pick the tolerable false-cut rate and read the corresponding h off a one-time sweep. Because latency is linear in h and false-cut has a sharp knee, this is a one-dimensional, interpretable choice—unlike tuning a raw `max_steps`, which conflates the two objectives it can never separate.

7 Limitations and threats to validity

We are deliberately explicit, because the result's credibility rests on it.

- **Synthetic main benchmark.** The 600-trajectory-per-seed benchmark is generated, not harvested from production agents. We mitigate the two obvious failure modes: an unrealistically easy negative set (addressed by modeling debugging plateaus) and generator/detector collusion (addressed by the five emergent agents). The honest next step is replay on real agent logs; the detector needs no change to run on them.
- **Lexical embedding and paraphrase stagnation.** The default state embedding is bag-of-tokens. It catches stagnation where the state text reuses core tokens but would *miss* a stagnation that paraphrases the same meaning with disjoint vocabulary. That case needs a semantic embedder, which the embedding hook accepts unchanged; we flag it as a known ceiling rather than hiding it.
- **Thin healthy/loop margin.** The measured $\mu_0 \approx 0.44$ sits close to $\mu_1 \approx 0.87$ relative to the tolerance—hence the sharp k sensitivity. A deployment whose healthy agents are noisier should expect to raise k and accept some latency. This is a property of the workload, made visible by the method.
- **Onset is known only in evaluation.** Latency is measured against ground-truth onset, available because we generated the loops. The detector never uses it; the deployment knob is h, not onset.
- **Single cost model.** Token cost is proxied by text length; a real model would weight tool-result sizes and pricing. This changes savings *magnitudes*, not the ordering of methods.

None of these threaten the central finding: "stuck" and "long" are cheaply separable online, and a fused-signal CUSUM policy dominates the fixed-cap and exact-repeat policies that ship in practice.

8 Related work

Agentic loops. Reasoning-and-acting agents [1] and reflective/self-critiquing agents [2] established the observe–act loop but delegate termination to the model plus an outer step cap; tool-augmented models [3] share the same unbounded structure. Surveys of LLM-based autonomous agents [4] note reliability and termination as open operational problems. Non-termination is treated as an engineering backstop rather than a detection problem—the gap this paper addresses.

Sequential change detection. CUSUM [5] and its asymptotic optimality [6], and the related Page–Hinkley test, are the classical low-latency tools for detecting a shift in a stream's mean; adaptive-windowing detectors such as ADWIN [7] address change detection under drift. Our contribution is not the detector but the construction of a *cheap, agent-appropriate statistic* s_t for it to watch.

Repetition heuristics. Ad-hoc "stop after N identical calls" rules are widespread in agent frameworks; §5 quantifies their blind spot on oscillation and stagnation and positions them as the degenerate action-only special case of our fusion.

9 Conclusion

Deployed agents bound their loops with a single number forced to do two contradictory jobs—kill dead runs and cap live ones—and it does neither well. The information needed to separate the two is already present in the agent's own action/observation stream and is recoverable at ~28 μs/step by fusing action periodicity, state recurrence, and novelty collapse. Feeding that fused score to a CUSUM detector yields a single interpretable threshold that, at its default, stops all three loop pathologies a median of 13.8 steps after onset with a 0.1% false-cut rate and recovers 86% of an oracle's token savings. The method is model-agnostic, dependency-free, and requires no change to the agent. The most valuable next step is replay on production agent traces, for which the artifact is ready as-is.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*.

[2] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *NeurIPS*.

[3] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *NeurIPS*.

[4] L. Wang et al. 2024. A Survey on Large Language Model based Autonomous Agents. *Frontiers of Computer Science* 18, 6.

[5] E. S. Page. 1954. Continuous Inspection Schemes. *Biometrika* 41, 1/2, 100–115.

[6] G. Lorden. 1971. Procedures for Reacting to a Change in Distribution. *Annals of Mathematical Statistics* 42, 6, 1897–1908.

[7] A. Bifet and R. Gavaldà. 2007. Learning from Time-Changing Data with Adaptive Windowing. In *SIAM Int. Conf. on Data Mining (SDM)*.