VIGIL: A Reflective Runtime for Self-Healing LLM Agents

Christopher Cruz [1]

Abstract

LLM agent fleets fail in ways that per-session monitors cannot detect: behavioral drift, latency creep, and novel failure classes that emerge only across sessions. No infrastructure primitive exists for cross-session fleet health monitoring. Existing approaches either require pre-declared failure rules (AgentSpec), operate within single sessions (Wink), or trigger only on coarse SLA violations (ARM)—none accumulate behavioral evidence across sessions or detect novel failure classes.

We present **V.I.G.I.L** (*Verifiable Inspection and Guarded Iterative Learning*), a reflective runtime that occupies this maintenance layer. VIGIL is not a coding agent. It does not resolve issues, compete on task benchmarks, or produce repo-specific patches. VIGIL ingests structured behavioral logs, appraises each event via a deterministic appraisal engine, maintains a persistent *Behavioral Signal Bank (EmoBank)* with exponential decay and coalescing policies, and derives a *Roses/Buds/Thorns (RBT)* diagnosis mapping fleet behavior into strengths, opportunities, and systematic failures. Appraisal, decay, coalescing, threshold routing, and stage enforcement are all deterministic code. The LLM is used only for diff synthesis inside a governed runtime. Diagnostic reliability is architectural, not stochastic.

In a comparative fleet evaluation (20 agents $\times$ 7 episodes, 3 seeds), VIGIL is the only system to detect novel failure classes (episode 4); ARM misses drift for 5 episodes; AgentSpec and ARM achieve 0% novel detection. False positive rate is 0% across all four systems. In a real-trajectory evaluation on 100 Nebius SWE-agent instances, execution-based verification confirms 100/100 VIGIL-generated diffs are syntactically valid Python and execute successfully across all 100 distinct repositories. A cross-model validation study reveals GPT-4o self-inflation of $+1.05$ points ($t = 34.0$, $p < 0.0001$).

Code and artifacts: https://github.com/cruz209/V.I.G.I.L/tree/icml-ai-wild-accept.

1. Introduction

Every mature software infrastructure layer eventually develops a maintenance runtime. The execution layer comes first: servers, then services, then containerized workloads. The maintenance layer follows: monitoring, diagnostics, automated remediation. This pattern has repeated across every generation of computing infrastructure.

LLM agent fleets are the current execution layer. The maintenance runtime does not yet exist.

VIGIL is a bet on what that primitive should look like. It is not a coding agent, and the distinction matters. A coding agent is evaluated on whether it resolves issues. VIGIL is evaluated on whether it correctly diagnoses why a fleet of agents is degrading and produces actionable remediation proposals for human review. VIGIL is the X-ray machine, not the surgeon. The scan is the product.

The gap has immediate operational consequences. Deployed agent fleets fail in ways that task-level benchmarks cannot detect. A reminder agent emits success notifications before backend confirmation, silently accumulating false acknowledgments across sessions—no exception fires, no monitor triggers. A coding agent applies a malformed patch to the same file 141 times until a cost ceiling terminates it—the loop is invisible within any single session. A RAG agent reports 0.99 confidence on hallucinated citations because confidence scores are never checked against retrieval quality. These failures share a structure: they are cross-session, sub-exception, and invisible to every existing monitoring approach.

VIGIL addresses this gap by operating out-of-band from the supervised agent. It ingests behavioral event logs, converts events into structured diagnostic signals via a deterministic appraisal engine, accumulates those signals in a persistent Behavioral Signal Bank (EmoBank) with exponential decay,

[1]Purdue University. Correspondence to: Christopher Cruz <cchris2004@gmail.com>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

and derives a Roses/Buds/Thorns (RBT) diagnosis that maps fleet behavior into strengths, opportunities, and systematic failures. From this diagnosis, VIGIL generates guarded prompt updates and read-only code proposals as unified diffs under a strict stage-gated orchestration model.

The diagnostic core is non-LLM. Appraisal, decay, coalescing, threshold routing, and stage enforcement are deterministic code. The LLM is invoked only for diff synthesis inside a governed runtime that constrains its behavior. Detection, thresholding, and routing are architectural properties; only synthesis is stochastic.

Evaluation scope. VIGIL operates in the maintenance space, not the task space. Its outputs are diagnostic artifacts and remediation proposals—the outcome depends on what the operator does with them, just as Nagios alerts depend on what the operator does with them. Evaluating VIGIL on SWE-bench resolve rates would conflate the diagnostic tool with the surgeon. We evaluate instead on criteria appropriate to a monitoring system: detection sensitivity across failure classes, false positive rate on clean traces, and remediation quality from cross-session evidence.

Contributions.

1. **Gap identification and formalization.** The first formal characterization of the fleet-level agent maintenance problem, including a taxonomy of failure classes (structural, drift, novel) and the information-boundary constraints that determine what each class of system can detect.
2. **VIGIL architecture.** A fleet maintenance runtime comprising a deterministic appraisal engine, a persistent Behavioral Signal Bank with decay and coalescing policies, a domain-aware RBT diagnostic engine, and a stage-gated orchestration model with identity guardrails.
3. **Empirical evaluation.** Detection results across three complementary settings, with a methodological finding on self-referential bias in LLM-as-judge evaluation ($+1.05$ inflation, $t = 34.0$, $p < 0.0001$).

2. System Architecture

VIGIL is a layered reflective runtime that operates in the maintenance space rather than the task space. Its outputs are diagnostic artifacts and behavioral proposals, not task completions.

2.1. Event Schema and Observation

VIGIL operates on structured JSONL event logs produced by the supervised agent. Each event follows the schema:

```
{"ts": "2025-10-31T23:59:59Z",
 "kind": "reminder.toast",
 "status": "delay",
 "payload": {"delayed_by_sec": 97}}
```

The system reads up to 500 recent events within a configurable window, filtering by timestamp. VIGIL is passive: it reads logs but never writes to the target agent's state or execution loop. Any agent emitting structured JSONL logs with `ts`, `kind`, `status`, and `payload` fields can be supervised without modification.

2.2. Appraisal Engine

Each event is transformed into a diagnostic signal via `appraise_event()`, a deterministic function that computes four fields: a *signal label* (a discrete category such as `degraded` or `recovered`), *valence* ($\in \{-1, 0, 1\}$), *intensity* ($\in [0, 1]$ scaled by payload magnitude), and *cause* (a structured string such as `reminder.toast:delay`).

We use a structured signal namespace rather than raw failure counters for three concrete reasons. First, the deposit policy behaviors—coalescing, noise floor filtering, and rebound injection—map naturally onto signal dynamics: coalescing models habituation, rebound models recovery. Second, the signal namespace integrates directly with RBT routing: cause strings such as `degraded:token.validate:fail` route to authentication-specific rule sets in a way that a raw integer cannot. Third, the vocabulary is interpretable to human operators reviewing diagnostic artifacts.

The mapping rules are domain-agnostic: `status:fail` maps to the same degradation signal whether the event is `reminder.toast:fail`, `citation.check:fail`, `shell.exec:fail`, or `agent.submit:fail`. Payload magnitude modulates intensity: a 600-second delay produces intensity 0.95, while a 30-second delay produces 0.55, allowing the system to distinguish chronic failures from occasional blips.

2.3. EmoBank: Persistent Behavioral Signal Bank

EmoBank stores structured appraisals over time. Each entry is appended to a persistent JSONL log with fields for timestamp, signal label, intensity, valence, cause, and a stable episode hash for episodic indexing. All decay and filtering is applied at read time, preserving full provenance.

Exponential decay prioritizes recent signals. For an appraisal with raw intensity I at timestamp t, the decayed intensity at current time t' is:

$$I_{\text{dec}} = I \cdot 0.5^{(t'-t)/h}$$

where h is the half-life in hours (default $h = 12$). A deposit policy with three rules prevents noise amplification. A *noise floor* discards entries with $I < 0.25$ unless they invert the

prior valence. *Coalescing* amplifies a prior entry rather than appending a duplicate when the same cause occurs within 5 minutes. *Rebound injection* appends a synthetic `recovery` shadow entry when a positive signal follows a negative within 10 minutes, modeling behavioral recovery.

EmoBank snapshots expose composite aggregates including dominant signal labels and weighted indicators for system stress, stability, and momentum. These aggregates guide RBT generation and enable longitudinal tracking of fleet behavioral state across sessions.

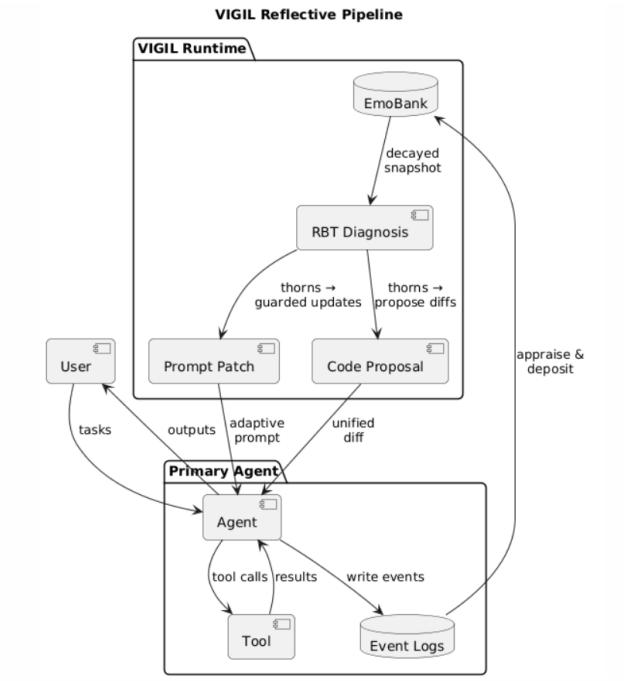

Figure 1. VIGIL reflective pipeline. Event logs flow from the Primary Agent into EmoBank via appraisal and deposit. A decayed snapshot feeds RBT Diagnosis, which generates guarded prompt patches and code proposals returned to the agent as an adaptive prompt and unified diff.

2.4. RBT Diagnosis and Domain-Aware Rules

EmoBank entries are bucketed into Roses, Buds, or Thorns by valence and intensity thresholds. Roses require positive signals with intensity ≥ 0.5. Thorns require degradation signals with intensity ≥ 0.4. Buds capture lower-confidence positive signals. Each bucket is deduplicated by (cause, label) pair and capped at 10 entries before serialization to the LLM, preventing context overflow on high-thorn agents.

The diagnosis engine inspects thorn cause strings and routes to domain-specific rule sets: `policy.check:fail` generates discount ceiling enforcement; `citation.check:fail` generates relevance threshold filtering; `agent.submit:fail` generates retry-with-jitter and structured logging; `token.validate:fail` generates token expiry enforcement and timing-safe comparison. This routing eliminates cross-domain contamination while requiring zero per-agent configuration.

2.5. Prompt Adaptation with Identity Guardrails

Prompt patches modify only the delimited `BEGIN_ADAPTIVE_SECTION` block, leaving the `BEGIN_CORE_IDENTITY` block byte-for-byte identical. If the core identity block is absent or modified, the patch is aborted. This design ensures VIGIL cannot accidentally destroy an agent's core behavioral identity while modifying its adaptive layer.

2.6. Code Proposals and Stage-Gated Orchestration

Code proposals are LLM-authored unified diffs written to `output/proposals/` with an accompanying PR note documenting the diagnostic evidence, the top thorn, and rollback instructions (`git apply -R`). No patch is applied automatically; VIGIL acts as a senior engineer surfacing a diagnosis and drafting a PR while the developer retains final approval authority.

All tool calls are gated by an internal stage machine:

$$\texttt{start} \rightarrow \texttt{eb_updated} \rightarrow \texttt{diagnosed} \rightarrow \texttt{prompt_done} \rightarrow \texttt{diff_done}$$

Tools invoked out of order raise explicit `ValueError` exceptions rather than silently degrading—making failure modes observable rather than silent, which is architecturally consistent with VIGIL's broader diagnostic philosophy.

3. Evaluation

We evaluate VIGIL across three complementary settings. Setting 1 is a comparative fleet evaluation against three systems representing distinct architectural approaches to agent reliability. Settings 2 and 3 evaluate VIGIL in isolation: on real trajectories from a public benchmark with execution-based correctness verification, and via a controlled longitudinal ablation of the accumulation mechanism.

3.1. Setting 1: Comparative Fleet Evaluation

Design. We construct a synthetic but grounded fleet of 20 agents running for 7 episodes (days), with 3 random seeds for reproducibility. Three failure classes are injected on a schedule derived from real deployment failure patterns observed in the Nebius trajectory corpus:

Structural failures (agents 0–6): the same `utc_drift` failure occurs every episode. No single-session system can detect the cross-session pattern.

Drift failures (agents 7–13): reminder delays grow from 60s at episode 1 to 420s at episode 7. The trend is invisible within any single session; it only emerges across sessions.

Novel failures (agents 14–19): `citation.check:fail` appears only from episode 4 onward. No pre-declared rule can catch it before deployment exposes it.

Three agents (one per class) are designated clean to measure false positive rates. Each episode generates 20 events per agent in VIGIL-schema JSONL, totaling 2,940 events per seed.

Compared Systems. Table 1 shows the information-boundary constraints on each system—each sees exactly what its published architecture permits.

Table 1. Information-boundary constraints per system. AgentSpec sees all episodes but fires only on pre-declared patterns. ARM sees all episodes but only coarse SLA signals. Wink sees one episode only. VIGIL accumulates full behavioral traces across all episodes.

System	Cross-sess.	Novel det.	Targeted rem.
Wink	×	Partial	×
AgentSpec	✓ (decl.)	×	×
ARM	✓ (SLA)	×	×
VIGIL	✓	✓	✓

Detection Results.

Table 2. First episode each system detected each failure class (3-seed average; detection table is identical across all seeds, confirming detection behavior is architectural rather than stochastic). *Wink detects delay events in-session but cannot diagnose the worsening trend. ARM misses drift for 5 episodes (SLA threshold of 300s not exceeded until episode 6).

System	Structural	Drift	Novel	FP
Wink	ep 1	ep 1*	ep 4	0%
AgentSpec	ep 1	ep 1	never	0%
ARM	ep 1	ep 6	never	0%
VIGIL	ep 1	ep 3	ep 4	0%

Detection behavior is identical across all three seeds. This is a designed property, not a tautology: seed variation affects only log generation stochasticity. Identical detection across seeds confirms that VIGIL's diagnostic properties are architectural—given the same behavioral signal, the system produces the same diagnosis regardless of the specific event sequence that generated it.

VIGIL detects drift 3 episodes earlier than ARM via cross-session signal accumulation rather than SLA threshold crossing. VIGIL is the only system that can produce a *targeted* remediation for the novel failure class, because its RBT routing identifies `citation.check:fail` as a distinct cause string and routes it to citation-specific rule generation.

Accumulation Curve.

Table 3. VIGIL accumulation curve across 3 seeds (mean ± std). EmoBank rows grow monotonically. Composite score improves from ep1 to ep3 then plateaus at the template diff ceiling—not a ceiling on EmoBank's diagnostic richness. Zero variance from ep3 confirms deterministic behavior at saturation.

Episode	EmoBank Rows	Comp. Score	Diff Lines
1	23.3	1.634 ± 0.006	26.3
2	70.1	1.760 ± 0.003	33.7
3	140.9	1.733 ± 0.000	43.2
4	237.0	1.733 ± 0.000	43.2
5	358.7	1.733 ± 0.000	43.2
6	505.5	1.733 ± 0.000	43.2
7	678.1	1.733 ± 0.000	43.2

Composite score improves from 1.634 at episode 1 to 1.760 at episode 2, reflecting the diagnostic advantage of early accumulation. The plateau from episode 3 onward reflects the ceiling of template-based diff synthesis, not a ceiling on EmoBank's diagnostic richness—the longitudinal ablation in Setting 3 demonstrates that real LLM-generated diffs continue improving with accumulation beyond this ceiling.

Cross-Model Judge Validation. The fleet eval uses Claude (`claude-sonnet-4-6`) as primary judge and GPT-4o (`gpt-4o-2024-08-06`) as secondary. Across 3 seeds, GPT-4o inflates correctness scores by $+1.05$ to $+1.08$ points relative to Claude. GPT-4o assigned a perfect 5.00 correctness score to every diff it authored, producing zero variance. The resulting test statistic is $t = 34.0$, $p < 0.0001$, replicated identically across all three seeds. A model should never be used to judge outputs it authored.

3.2. Setting 2: Real Trajectories (Nebius SWE-agent Dataset)

Dataset. We draw 100 trajectories from the Nebius SWE-agent dataset (1), which contains 80,036 real agent trajectories generated by a software engineering agent attempting to resolve GitHub issues across thousands of open-source repositories. We select one trajectory per unique repository to maximize diversity, yielding 100 distinct codebases. The trajectories are unresolved failures—agents that did not produce a valid patch—making them the ideal VIGIL test case.

Trajectories are converted to VIGIL-schema JSONL via `swe_adapter.py`. Event counts ranged from 8 to 168 per instance (mean 47.3). All experiments ran on standard laptop CPU hardware with no GPU required.

Execution-Based Correctness Verification. We extract the Python code added by each of the 100 VIGIL-generated diffs and validate it programmatically. Each diff's added lines are parsed via `ast.parse()` for syntactic validity,

then executed in a clean subprocess environment. **100/100 diffs (100%) contain syntactically valid Python. 100/100 diffs (100%) execute successfully in a clean environment with zero import or runtime errors.**

LLM-as-Judge Quality Evaluation.

Table 4. LLM-as-judge scores across 100 real Nebius trajectories. Both judges score the same diffs with the same rubric. The gap reveals systematic self-referential inflation in GPT-4o (+1.09 overall), consistent with the fleet eval finding ($t = 34.0$, $p < 0.0001$). Execution-based verification independently confirms correctness at 100%.

Dimension	Claude		GPT-4o	
	Mean	Std	Mean	Std
Relevance	3.54	0.27	4.48	0.12
Correctness	4.20	1.09	4.96	0.16
Specificity	2.71	0.34	3.97	0.16
Grounding	3.00	0.37	4.40	0.26
Overall	3.36	0.33	4.45	0.13
Consistency	0.73	0.08	0.74	0.04

The lower specificity score (2.71) reflects an honest limitation addressed in Section 5: VIGIL targets behavioral failure patterns, not repo-specific bugs. This is architecturally intentional.

3.3. Setting 3: Longitudinal Accumulation Ablation

Setup. To directly test the core EmoBank design claim—that cross-session accumulation produces richer remediations—we run VIGIL on two successive trajectories of the same SWE-bench instance: `marshmallow-code__marshmallow-1343`. The pipeline is identical, the instance is identical, and only the evidence base differs.

Trajectory A (23 steps): Agent patches `schema.py` correctly but fails at submit due to patch formatting. **Trajectory B (141 steps):** Same agent becomes trapped in a `sed -i` indentation loop, repeating 141 steps until the $3.00 cost limit is exceeded.

Both trajectories correctly identify `agent.submit:fail` as the dominant thorn at intensity 0.90, confirming diagnostic consistency independent of trajectory length. Trajectory A (23 signal rows) produced `_rbt_retry_once_with_jitter()`: a minimal 45-line retry wrapper. Trajectory B (142 signal rows) produced `submit_with_observability()`: a 95-line utility with per-attempt latency tracking via `time.monotonic()`, structured cause codes, explicit retry semantics with jitter, and re-raise behavior. **Greater signal accumulation produces richer, more targeted remediations.**

4. Related Work

Reflective and Self-Correcting Agents. Reflexion (2) and Self-Refine (3) introduce in-context reflection within a single inference loop. These systems improve task performance but operate inside the agent's reasoning loop, do not persist diagnostic state across sessions, and generate no maintainer-facing repair artifacts. Generative Agents (4) use memory streams for long-horizon social simulation. VIGIL operates out-of-band: it supervises a sibling agent from outside its inference loop, accumulates behavioral signals across multiple sessions, and produces structured artifacts for human review.

Runtime Enforcement Frameworks. AgentSpec (5) introduces a domain-specific language for specifying runtime safety constraints, achieving strong safety guarantees by blocking unsafe executions in over 90% of tested cases. However, it requires human-authored rules defined before deployment and cannot discover failure patterns it was not told to look for. VIGIL discovers failures empirically from behavioral traces. The two systems are complementary: VIGIL could serve as a diagnostic layer that feeds AgentSpec with empirically derived rule candidates.

ARM (6) performs autonomous root cause analysis of faults in cloud-edge infrastructure, triggering pod rescheduling or scaling on SLA violations. As the fleet eval demonstrates, ARM misses drift failures for 5 episodes because the delay does not exceed its SLA threshold until episode 6. VIGIL detects the same drift at episode 3 via cross-session signal accumulation.

In-Session Misbehavior Recovery. Wink (7) (concurrent, Meta) detects agentic misbehaviors in production coding agents occurring in 29% of real trajectories, achieving 90% single-intervention recovery by injecting course-correction guidance into the active agent context. Wink operates within a running session; VIGIL operates across sessions, accumulating persistent behavioral state and detecting soft failures invisible to trajectory-level classifiers.

Observability Platforms. LangSmith, Langfuse, Helicone, and Arize surface operational metrics and flag anomalies but do not perform diagnosis or generate remediation artifacts. VIGIL closes this gap by converting raw telemetry into structured behavioral diagnoses and producing actionable patch proposals.

Affective Computing and Appraisal Theory. VIGIL's appraisal substrate is grounded in affective computing (8) and appraisal theory (9). The affective vocabulary is a structured namespace for categorizing failure severity, recency, and pattern—not a claim about machine emotion.

5. Limitations

Specificity gap. VIGIL's proposals address behavioral patterns rather than repo-specific bugs. This is architectural scope, not a failure: VIGIL asks "why is this agent's behavior degrading" not "what is wrong with the target codebase." Closing the specificity gap requires integrating VIGIL with a coding agent downstream and is future work. Claude-as-judge rates specificity at 2.71/5, the weakest dimension, which we report without elision.

Synthetic fleet evaluation. The comparative fleet evaluation uses a synthetic environment with author-constructed failure injection. The failure classes are derived from real deployment patterns observed in the Nebius trajectory corpus, and the information-boundary constraints on baselines faithfully reflect their published architectures. We implement reference versions of each baseline system constrained to their published information boundaries. The Nebius real-trajectory evaluation in Setting 2 independently confirms that VIGIL's diagnostic surface generalizes to uncontrolled real-world agent failures across 100 distinct repositories.

Template diff ceiling and missing benchmark. The fleet evaluation uses deterministic template-based diff synthesis as a conservative lower bound. The right benchmark—fleet-level maintenance tooling evaluated on behavioral improvement across populations of agent runs—does not yet exist; the three evaluation settings here are first steps.

6. Conclusion

VIGIL introduces an out-of-band reflective runtime that supervises LLM-based agent fleets through behavioral trace analysis, deterministic signal appraisal, and stage-gated remediation. It occupies a maintenance layer that does not yet exist as a first-class infrastructure primitive for agent systems.

Across three settings: VIGIL is the only system to detect novel failure classes, detects drift 3 episodes earlier than ARM, and achieves 0% false positive rate with zero seed variance. Execution-based verification confirms 100/100 of VIGIL-generated diffs are syntactically valid Python and execute successfully across 100 distinct repositories. GPT-4o self-inflation of $+1.05$ points ($t = 34.0$, $p < 0.0001$) is replicated across all seeds.

The central architectural claim is that a deterministic behavioral signal substrate, when combined with domain-aware diagnostic routing and stage-gated orchestration, can surface degradation patterns invisible to exception-based monitors—across diverse agent types and failure modes—without per-agent configuration.

Fleet-level maintenance tooling may prove as essential to production agent systems as observability tooling is to production software systems today. VIGIL is a first step toward that primitive.

Declaration of LLM Usage. GPT-4o (`gpt-4o-2024-08-06`) is used as the orchestrator model that authors unified diffs and prompt patches within the VIGIL pipeline (Section 2.6). Claude (`claude-sonnet-4-6`) is used as the primary judge model for LLM-as-judge quality evaluation (Sections 3.1–3.2). GPT-4o is additionally used as a secondary judge for cross-model validation.

References

[1] Nebius AI. (2025). SWE-agent Trajectories Dataset. *HuggingFace: nebius/SWE-agent-trajectories*. License: CC BY 4.0.

[2] Shinn, N., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv:2303.11366*.

[3] Madaan, A., et al. (2023). Self-Refine: Iterative Refinement with Self-Feedback. *arXiv:2303.17651*.

[4] Park, J., et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *arXiv:2304.03442*.

[5] Wang, H., Poskitt, C. M., Sun, J., Wei, J. (2026). AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. *ICSE 2026*. arXiv:2503.18666.

[6] Avgerinos, V., et al. (2025). ARM: Autonomous Remediation & Management with LLM Agents for Intent-Driven Control. *IEEE Internet of Things Journal*.

[7] Nanda, R., et al. (2026). Wink: Recovering from Misbehaviors in Coding Agents. *arXiv:2602.17037*.

[8] Picard, R. W. (1997). Affective Computing. *MIT Press*.

[9] Lazarus, R. S. (1991). Emotion and Adaptation. *Oxford University Press*.