Silent Starvation: Dependency Deadlock and Planning Loop Failure Modes in Autonomous AI Task Scheduling

Vasyl Golubenko
TOV ZELTREX, Kyiv, Ukraine
Ukrainian Academy of Technology, Kyiv, Ukraine
vasyl.golubenko@zeltrex.com

April 2026

Abstract

Autonomous AI agents that manage their own task backlogs face scheduling challenges that differ fundamentally from traditional job scheduling: tasks have semantic dependencies, agents must self-assess eligibility, and metacognitive planners operate independently from execution gates. We report the first empirical documentation of *silent starvation*—a compound failure in which a dependency chain deadlock combines with a decoupled planning layer to produce an agent that runs hundreds of planning cycles while executing zero tasks, without generating any error signals. In a production autonomous software development system, we observed 452 identical task selections over 7 days, producing 0 task executions and 0 errors. Root cause analysis revealed three interacting failures: (1) status promotion failure—tasks marked "blocked" were never promoted to "pending" when dependencies completed; (2) backlog desynchronization—the execution database contained only 7 of 43 pending tasks; (3) planning-execution decoupling—the metacognitive planner selected tasks from a candidate window disjoint from the execution gate's eligibility filter. We formalize a taxonomy of four autonomous scheduling failure modes, propose detection heuristics based on selection entropy and execution ratio monitoring, and validate that correcting the root cause immediately restored task flow.

Keywords: autonomous agents, task scheduling, dependency resolution, failure modes, deadlock detection, multi-agent systems

1 Introduction

The emergence of autonomous AI agents capable of independently managing software development tasks represents a significant advancement in AI-assisted engineering [He et al., 2025, Fang et al., 2025]. Unlike traditional CI/CD pipelines with predefined task graphs, autonomous agents must dynamically select tasks from evolving backlogs, manage inter-task dependencies, and self-assess execution readiness—all without human intervention during execution cycles.

This combination of autonomous scheduling with semantic task dependencies creates failure modes that have no direct analog in classical job scheduling theory. Traditional deadlock detection assumes tasks that are actively waiting for resources [Coffman et al., 1971]; in autonomous AI scheduling, tasks may be statically *marked* in a way that prevents their selection, without any active waiting or resource contention.

We present a case study of a production autonomous development system that experienced *silent starvation*: a state in which the agent continued to execute dispatch cycles, perform metacognitive planning, and log structured decision rationale, while executing zero tasks over a 7-day period. The system produced no error signals, no alerts, and no degradation warnings.

Our contributions are: (1) a formal taxonomy of four autonomous scheduling failure modes; (2) empirical documentation of silent starvation in a production system; (3) detection heuristics and prevention mechanisms; (4) validation of the corrective intervention.

2 Related Work

2.1 Classical Deadlock and Scheduling Theory

Deadlock in concurrent systems has been studied extensively since Coffman et al. [Coffman et al., 1971] identified the four necessary conditions: mutual exclusion, hold and wait, no preemption, and circular wait. Modern scheduling frameworks employ cycle detection in wait-for graphs [Silberschatz et al., 2018]. However, these approaches assume actively waiting processes with explicit resource dependencies. Autonomous AI scheduling differs: dependencies are semantic, the scheduler has agency, and state may desynchronize across storage layers.

2.2 Multi-Agent Task Allocation

Sun et al. [Sun et al., 2025] surveyed multi-agent coordination paradigms. He et al. [He et al., 2025] addressed LLM-based multi-agent systems for software engineering, identifying task decomposition as a key challenge. Fang et al. [Fang et al., 2025] discussed evolutionary mechanisms for agent improvement but did not address scheduling pathologies.

2.3 Autonomous Agent Failure Modes

Cemri et al. [Cemri et al., 2025] analyzed 1,642 execution traces across seven multi-agent systems, finding failure rates of 41–87%. Roig [Roig, 2025] provided qualitative analysis of LLM failure in agentic scenarios. Neither documented silent scheduling failures where the agent appears functional while producing no output.

3 System Description

3.1 Architecture Overview

Night Shift is an autonomous software development agent deployed on dedicated server infrastructure (Hetzner GEX44, RTX 4000 SFF Ada GPU). It operates on hourly dispatch cycles, maintaining approximately 300 tasks across 7 categories with task types including code generation, testing, research, and documentation. The system uses multiple LLM providers (local Ollama with qwen2.5-coder:14b, cloud-based Claude and Gemini models).

3.2 Task Lifecycle

Tasks follow a defined lifecycle: `pending` → `dispatched` → `completed` | `archived` | `blocked`. The system maintains two representations: a YAML canonical source and an SQLite execution database.

3.3 Task Selection Pipeline

The dispatch cycle involves three sequential stages:

Stage 1: Metacognitive Planning. The MC planner selects 10 candidates, computes a 4-axis confidence score, and records the recommended task.

Stage 2: Eligibility Filtering. The backlog manager queries pending tasks within budget, filters by dependency satisfaction, sorts by priority, and applies a prompt quality gate (code tasks require non-empty `context_files`).

Stage 3: Deduplication and Execution. The completion ledger prevents re-execution. Critically, Stages 1 and 2 operate on potentially different task sets.

4 Failure Analysis

4.1 Observed Symptoms

During the 7-day observation period (March 27–April 3, 2026):

- 452 planning iterations with `ingest-300` selected 89.2% of the time
- 0 tasks executed in the final 34 dispatch cycles
- 0 error signals—"No eligible tasks" logged as INFO, not WARNING
- System logs showed structured metacognitive rationale masking the absence of progress

4.2 Root Cause 1: Status Promotion Failure

Three tasks were marked `blocked` with fully satisfied dependencies (Table 1).

Table 1: Blocked tasks with satisfied dependencies.

Task ID	Dependency	Dependency Status
ingest-201	ingest-200	completed
bridge-002	bridge-001	archived
ingest-700	ingest-000	archived

The `get_next_task()` method queries only `status='pending'`, meaning blocked tasks are never reconsidered. This differs from classical deadlock: there is no circular dependency—the tasks are simply *forgotten*.

4.3 Root Cause 2: Dependency Chain Cascade

A single stuck task (`ingest-201`) transitively blocked 5 downstream tasks through the dependency graph:

```
ingest-201 (blocked, deps satisfied)
  -> ingest-300 (pending, blocked by ingest-201)
    -> ingest-400 (blocked by ingest-300)
      -> ingest-500 (blocked by ingest-400)
      -> ingest-600 (blocked by ingest-400)
    -> ingest-202 (blocked by ingest-201)
```

Combined with 2 permanently ineligible tasks (empty `context_files`), all 7 pending tasks were unreachable.

4.4 Root Cause 3: Planning-Execution Decoupling

The MC planner and execution filter operated on different data (Table 2).

This created a feedback loop: the planner selected, the executor rejected, the planner selected again—with no mechanism to communicate the rejection.

Table 2: Planning-execution data mismatch.

Component	Data Source	ingest-300 Status
MC Planner	Candidate window	Selectable (score 0.76)
Execution Filter	DB with dep. check	Ineligible (dep. unmet)

4.5 Root Cause 4: Backlog Desynchronization

The YAML contained 43 pending tasks; the database contained only 7. The remaining 36 had been completed/archived in the database while the YAML retained stale status.

5 Taxonomy of Autonomous Scheduling Failures

Based on our analysis, we propose a taxonomy of four failure modes (Table 3).

Table 3: Taxonomy of autonomous scheduling failure modes.

Failure Mode	Definition	Detection Signal	Classical Analog
Status Ossification	Tasks remain in terminal state despite conditions for advancement	Blocked tasks with met deps > 0	Stale lock
Silent Starvation	Scheduling runs but finds no eligible tasks; no error raised	Consecutive zero-exec cycles $>$ threshold	Livelock
Phantom Planning	Planner recommends tasks the executor cannot process	Selection entropy ≈ 0; agreement $< 10\%$	Split-brain
Backlog Desync	Multiple backlog representations diverge	$\lvert\text{source}\rvert - \lvert\text{DB}\rvert >$ threshold	Replica drift

These modes interact multiplicatively: status ossification causes starvation, masked by phantom planning, exacerbated by desynchronization.

6 Detection and Prevention

6.1 Detection Heuristics

Heuristic 1: Execution Ratio. Track executed tasks per dispatch cycle:

$$R_w = \frac{\text{tasks\_executed}_{[t-w,t]}}{\text{cycles}_{[t-w,t]}} \tag{1}$$

Alert when $R_w < 0.1$ for $w \geq 10$. In our system, $R_{34} = 0.0$.

Heuristic 2: Selection Entropy.

$$H_w = -\sum_i p_i \log_2 p_i \tag{2}$$

Alert when $H_w < 1.0$ bit. In our system, $H_{452} = 0.50$ bits.

Heuristic 3: Blocked Task Audit. Periodically re-evaluate all blocked tasks and auto-promote those with satisfied dependencies.

Heuristic 4: Plan-Execute Agreement.

$$A_w = \frac{|\{t : \text{planned}(t) = \text{executed}(t)\}|}{w} \tag{3}$$

Alert when $A_w < 0.3$ over $w \geq 20$.

6.2 Prevention Mechanisms

Mechanism 1: Automatic Status Promotion. Pre-dispatch hook checks blocked tasks and promotes those with satisfied dependencies.

Mechanism 2: Planning-Execution Feedback. Rejection reasons communicated back to planner for downranking.

Mechanism 3: Starvation Circuit Breaker. After 5 consecutive zero-execution cycles: diagnose, auto-heal, alert operator.

Mechanism 4: Single Source of Truth. Eliminate dual representations or enforce bidirectional sync.

7 Intervention and Validation

Three corrective actions were performed: (1) promote 3 blocked tasks with satisfied dependencies to `pending`; (2) archive 2 permanently broken tasks; (3) verify dependency graph for cycles.

Table 4: Task eligibility before and after intervention.

Metric	Before	After	Change
Pending tasks	7	8	+1
Blocked tasks	3	0	−3
Eligible (all gates)	0	3	+3
Cascade-unblockable	0	5	+5

The intervention immediately restored task flow: 3 tasks eligible, with 5 additional tasks cascade-unblockable upon completion.

8 Discussion

8.1 The Livelock Analogy

Silent starvation resembles livelock: the system is active but makes no progress. However, in classical livelock, processes actively change state in response to each other. In our system, the state is *static*—the same blocked tasks remain blocked, the planner makes the same recommendation. This is a *scheduling fixpoint*: a stable state the system cannot exit through its own dynamics.

8.2 Design Principles

Our findings suggest:

1. **Status should be derived, not stored.** Compute eligibility from the dependency graph at query time.

2. **Planning and execution should share eligibility.** The planner should only consider tasks that pass the execution filter.

3. **Zero-progress detection is a safety requirement.** The absence of errors is not evidence of health.

4. **Multiple representations are a liability.** A single source of truth prevents desynchronization.

8.3 Broader Implications

Cemri et al. [Cemri et al., 2025] found that failures compound at each step. Our finding extends this: in systems with persistent state, failures can *persist across cycles*, creating sustained failure states affecting all subsequent operations.

8.4 Limitations

Our analysis is based on a single system. The specific failure (blocked status not promoted) may be implementation-specific, but the general pattern—semantic dependencies + autonomous scheduling + decoupled planning—is common across agent architectures.

9 Conclusion

We documented *silent starvation*—a compound failure where dependency deadlocks, status ossification, and planning-execution decoupling produce an agent that appears functional while executing zero tasks. We observed 452 wasted planning cycles over 7 days with no error signals.

We propose a taxonomy of four autonomous scheduling failure modes, detection heuristics based on execution ratio and selection entropy, and prevention via automatic status promotion and starvation circuit breakers. Correcting the root cause immediately restored task flow. As autonomous agents take on complex task management, the scheduling layer becomes a critical reliability surface requiring dedicated fault-tolerance mechanisms.

References

J. He, C. Treude, and D. Lo. LLM-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Trans. Softw. Eng. Methodol.*, 34(5):124:1–124:30, 2025. 10.1145/3712003.

J. Fang, Y. Peng, X. Zhang, et al. A comprehensive survey of self-evolving AI agents. *arXiv preprint arXiv:2508.07407*, 2025.

E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *ACM Computing Surveys*, 3(2):67–78, 1971. 10.1145/356586.356588.

A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley, 10th edition, 2018.

L. Sun, Y. Yang, Q. Duan, et al. Multi-agent coordination across diverse applications: A survey. *arXiv preprint arXiv:2502.14743*, 2025.

M. Cemri, M. Z. Pan, S. Yang, et al. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

J. V. Roig. How do LLMs fail in agentic scenarios? *arXiv preprint arXiv:2512.07497*, 2025.

G. Wang, W. Wu, G. Ye, et al. Decoupling metacognition from cognition: A framework for quantifying metacognitive ability in LLMs. In *Proc. AAAI*, volume 39, pages 25353–25361, 2025. 10.1609/aaai.v39i24.34723.

N. Shinn, F. Cassano, E. Berman, et al. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

M. Steyvers, H. Tejeda, A. Kumar, et al. What large language models know and what people think they know. *Nature Machine Intelligence*, 7:221–231, 2025. 10.1038/s42256-024-00976-7.

Q. Guo, R. Wang, J. Guo, et al. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *Proc. ICLR*, 2024.

W. Xu, Z. Liang, K. Mei, et al. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.

P. Du. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026.

S. Ghosh and M. Panday. The Dunning-Kruger effect in large language models. *arXiv preprint arXiv:2603.09985*, 2026.

G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley, 5th edition, 2011.

J. Leng, C. Huang, B. Zhu, and J. Huang. Taming overconfidence in LLMs: Reward calibration in RLHF. In *Proc. ICLR*, 2025.