Night Shift: An Autonomous AI Development System for Continuous Software Evolution

Vasyl Golubenko
TOV ZELTREX, Kyiv, Ukraine
ceo@zeltrex.com | https://zeltrex.com

March 2026

Abstract

We present Night Shift, a production-deployed autonomous AI development system that operates continuously on a 2-hour dispatch cycle, generating code, specifications, research reports, and documentation without human intervention. Unlike benchmark-oriented agent systems that demonstrate capability on curated problem sets, Night Shift has been deployed to a production server for 10+ consecutive days, completing 269 tasks at a total cost of $65.98 USD. The system introduces four key innovations: (1) an evolutionary task optimization engine based on genetic algorithms that tunes execution parameters across generations, (2) a continuation-based anti-truncation mechanism achieving 100% task completion rate, (3) a cascade model routing strategy that balances cost and quality across local GPU and cloud API tiers, and (4) a human-AI symbiont loop where daily mentoring reviews shape the system's behavior through feedback injection. We compare Night Shift against state-of-the-art autonomous agents including SWE-agent, Devin, MetaGPT, and AlphaEvolve, demonstrating that production deployment demands fundamentally different design priorities than benchmark performance. Our experimental results from the 10-day deployment reveal both the system's strengths — zero truncation failures, sustained budget discipline, and reliable autonomous operation — and its current limitations, including quality degradation over time and insufficient code output. We discuss architectural lessons learned and propose future directions incorporating reflexion, skill libraries, and multi-agent coordination.

Keywords:

1 Introduction

The rapid advancement of large language models (LLMs) has spawned a new class of autonomous AI agents capable of performing complex software engineering tasks. Systems like SWE-agent [4], Devin (Cognition AI, 2024), and MetaGPT [9] have demonstrated impressive results on standardized benchmarks such as SWE-bench. However, a critical gap exists between benchmark performance and production deployment: real-world autonomous systems must operate under budget constraints, handle failures gracefully, integrate with existing development workflows, and maintain quality over extended periods without human supervision.

This paper presents Night Shift, an autonomous AI development system designed from the ground up for continuous production operation. Rather than optimizing for single-shot problem

solving, Night Shift implements a four-layer architecture — Pulse (budget governance), Mind (task planning), Hands (execution), and Reflect (quality assessment) — that enables sustainable, long-running autonomous development work.

1.1 Contributions

This paper makes the following contributions:

1. **Production Architecture for Autonomous AI Development.** We describe a complete system architecture that addresses the operational concerns absent from benchmark-oriented agents: budget governance, failure recovery, rhythm synchronization with human workflows, and quality tracking over time.

2. **Evolutionary Task Optimization.** We introduce a genetic algorithm that evolves task execution parameters — model selection, token budgets, prompt styles, and context depth — using a composite fitness function that incorporates automated quality scores, human grades, cost efficiency, and merge outcomes.

3. **Anti-Truncation Continuation System.** We present a continuation mechanism that detects output truncation and automatically generates type-specific follow-up prompts, achieving a 100% task completion rate across 269 tasks over 10 days.

4. **Empirical Production Deployment Results.** We report detailed metrics from a 10-day continuous deployment including task counts, token usage, costs, quality distributions, and failure modes — providing an empirical foundation absent from most agent research.

1.2 Paper Organization

Section 2 surveys related work across cognitive architectures, software engineering agents, self-improving systems, cost-optimal inference, and agent memory. Section 3 describes the system architecture in detail. Section 4 presents our key innovations. Section 5 reports experimental results from the 10-day production deployment. Section 6 discusses implications and limitations. Section 7 outlines future work. Section 8 concludes.

2 Related Work

2.1 Cognitive Architectures for Language Agents

The CoALA framework [20] provides the foundational taxonomy for language agent design, defining agents as compositions of memory modules, action spaces, and decision procedures. CoALA identifies three memory types critical for sustained agent operation: working memory (current context), episodic memory (past experiences), and semantic memory (accumulated knowledge). Night Shift currently implements working memory through its context engine but lacks explicit episodic and semantic memory — a gap we address in Section 7.

AIOS [2] proposes treating LLM agents as processes within an operating system abstraction, with scheduling, memory management, and access control. This vision aligns with Night Shift's

layered architecture, where the Pulse layer functions analogously to a kernel scheduler managing resource allocation.

LLM-ACTR [14] integrates LLMs with the ACT-R cognitive architecture, combining the perceptual and motor modules of cognitive science with LLM language capabilities. While Night Shift does not formally implement ACT-R modules, its Pulse-Mind-Hands-Reflect pipeline maps naturally to the perceive-plan-act-evaluate cycle of cognitive architectures.

2.2 Software Engineering Agents

SWE-agent [4] introduced the Agent-Computer Interface (ACI) paradigm, demonstrating that LLM agents can resolve real GitHub issues by navigating codebases, editing files, and running tests. SWE-agent achieves 12.5% on SWE-bench, establishing a baseline for automated software engineering. However, SWE-agent operates on individual problems without persistent state between runs — each invocation starts fresh.

Devin (Cognition AI, 2024) represents the commercial frontier of autonomous coding agents, combining a sandboxed development environment with planning, debugging, and deployment capabilities. While Devin demonstrates end-to-end software development, it operates on-demand rather than autonomously, and its proprietary nature limits architectural analysis.

OpenHands [31] provides an open-source platform for developing AI software agents, offering standardized evaluation and a modular architecture. Its CodeAct approach, where agents express actions as executable code, has proven effective across multiple benchmarks.

Agentless [24] takes a contrarian approach, demonstrating that simple two-phase localization-then-repair pipelines can match or exceed complex agent systems on SWE-bench. This finding informed Night Shift's design philosophy: complexity should serve production needs, not benchmark performance.

MetaGPT [9] encodes Standardized Operating Procedures (SOPs) into multi-agent workflows, assigning roles (Product Manager, Architect, Engineer, QA) that produce structured artifacts consumed by downstream agents. Night Shift's phased execution proposal (Section 7) draws directly from MetaGPT's SOP approach.

ChatDev [23] extends the multi-agent paradigm with a virtual software company where agents in different roles communicate through structured dialogues, implementing "communicative dehallucination" where agents correct each other's outputs. Night Shift currently operates as a single agent, but its Ring 2 roadmap envisions role specialization inspired by ChatDev.

2.3 Self-Improving Agents

The concept of self-improving AI systems has deep roots in the Gödel Machine [22], which describes a system that can prove modifications to its own code will improve performance before executing them. Recent work has made this concept practical:

Gödel Agent [7] implements LLM-based self-modification guided by high-level objectives, dynamically adjusting its own logic and behavior. Unlike Gödel Machines that require formal proofs, Gödel Agent uses empirical validation through benchmarks.

The Darwin Gödel Machine (DGM) [1] combines self-modification with Darwinian evolution, maintaining a population of agent variants that read and modify their own Python codebases.

DGM improved from 20% to 50% on SWE-bench Verified through autonomous self-improvement, discovering novel strategies including custom editing tools and error memory.

SICA [18] demonstrates that the same agent can both perform tasks and evaluate/update itself, achieving 17% to 53% improvement on SWE-bench Verified through changes to tool orchestration and problem decomposition heuristics.

Live-SWE-agent [32] starts with minimal tools and synthesizes custom editors, code search utilities, and domain-specific analyzers during problem solving, achieving 77.4% on SWE-bench Verified — the best known score at time of writing.

AlphaEvolve [29] operates on entire codebases with ensemble LLM evaluation and population-wide evolution. While Night Shift's evolution engine shares the genetic algorithm foundation, AlphaEvolve evolves code itself rather than execution parameters.

Night Shift's evolution engine (Section 4.1) occupies a middle ground: it evolves task execution parameters (model, budget, style) using genetic algorithms, but does not yet modify its own source code. The DGM self-improvement pathway is identified as high-priority future work (Section 7).

2.4 Cost-Optimal Inference

Production deployment demands cost awareness that benchmark systems can ignore. Three key approaches inform Night Shift's cascade routing:

FrugalGPT [21] introduces sequential querying — starting with the cheapest model and escalating only when confidence is insufficient — matching GPT-4 quality with 98% cost reduction on selected tasks.

RouteLLM [27] trains routing models using human preference data, achieving 85% cost reduction on MT-Bench while maintaining 95% of GPT-4 quality. The router learns which queries require expensive models versus cheap ones.

Cascade Routing [30] unifies the routing and cascading paradigms, demonstrating 87% cost reduction by starting 90% of queries with small models and only escalating the remainder.

Night Shift implements category-based static routing (Section 4.3) with plans to adopt confidence-based cascading in future iterations.

2.5 Agent Memory and Identity

Generative Agents [12] introduced the observe-reflect-plan memory cycle for LLM agents, where 25 simulated characters maintain memory streams, periodically synthesize reflections, and use recency-importance-relevance scoring for retrieval. This architecture produced emergent social behaviors from individual memory mechanisms.

A-MEM [28] proposes Zettelkasten-inspired memory where each memory is a structured note with contextual descriptions, keywords, and dynamic links to related memories. New memories trigger "memory evolution" — existing memories update their representations when connected.

Voyager [13] demonstrates an ever-growing skill library of verified executable code indexed by description, achieving 15.3x faster progress in Minecraft through compositional skill reuse. The skill library concept directly applies to Night Shift's proposed procedural memory (Section 7).

MemGPT [17] implements virtual context management, allowing LLM agents to manage their own memory hierarchy analogous to operating system virtual memory. This approach is relevant to Night Shift's context engine, which must decide what information to include in limited context windows.

Infrastructure for AI Agents [10] formalizes agent identity as containers with unique identifiers, system cards, and certifications — comparable to human employment systems. This framework informs Night Shift's proposed agent identity document.

Reflexion [8] demonstrates that verbal self-critique — natural language reflection on failures stored in memory — improves agent performance by 10-20% without weight updates. This represents the highest-impact, lowest-effort improvement identified for Night Shift (Section 7).

3 System Architecture

3.1 Overview

Night Shift operates as a four-layer pipeline where each layer addresses a distinct concern of autonomous operation. Figure 1 illustrates the architecture.

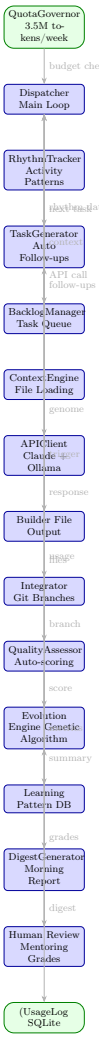

Figure 1: the architecture

Figure 1: Night Shift four-layer architecture. Arrows indicate data flow between components. The Evolution Engine spans all layers, receiving fitness signals from Reflect and injecting genome parameters into Hands.

The system deploys on a Hetzner GEX44 dedicated server with an NVIDIA RTX 4000 SFF Ada (20GB VRAM) GPU, running Ubuntu with systemd timers that trigger dispatch cycles every 2 hours and generate morning digests at 06:00 local time.

3.2 Pulse: Budget Governance

The Pulse layer ensures Night Shift operates within defined resource constraints. The QuotaGovernor implements a hierarchical budget model:

Parameter	Value	Notes
Weekly Budget	3,500,000 tokens	$79/month
Daily Allocation	500,000 tokens	Divided per dispatch cycle
Safety Margin	15% reserved	Prevents budget overrun
Hard Limit	Per-day enforcement	Dispatch stops when exhausted
Cost Ceiling	$5.00 USD/day	Absolute maximum
Rollover	Carry forward unused	Max 2 days accumulation

Table 1:

The available budget for any dispatch cycle is calculated as:

$$available = (daily\_budget - used\_today + rollover) \times (1 - safety\_margin) \quad (1)$$

where rollover accumulates unused budget from the previous two days, capped at 2× daily allocation.

The RhythmTracker learns patterns of human activity — when reviews happen, when feedback arrives, when manual merges occur — and provides this data to the TaskGenerator for scheduling awareness. This creates temporal alignment between autonomous work and human review cycles.

3.3 Mind: Task Planning

The Mind layer manages what work gets done. The BacklogManager maintains a priority queue of tasks across seven categories: NEXUS (product code), HUB (platform infrastructure), LIVINGCORP (agent framework), BRIDGE (API contracts), META (self-improvement), RESEARCH (analysis), and INGESTION (data pipelines).

The TaskGenerator automatically creates follow-up tasks based on completed work:

- **Code tasks** spawn test-writing tasks (depth 1) and documentation tasks (depth 0 only)
- **Research tasks** spawn implementation tasks and deep-dive analyses (depth 0 only)
- **Specification tasks** spawn implementation tasks and review tasks (depth 0 only)

Task depth is limited to prevent exponential chain growth: original tasks have depth 0, first follow-ups have depth 1, second follow-ups have depth 2, and no further follow-ups are generated beyond depth 2.

The ContextEngine loads relevant source files from the repository and constructs prompts that include project-specific context. Each task receives a structured prompt containing: task description, relevant file contents, output format requirements, and any mentoring feedback from previous reviews.

3.4 Hands: Execution

The Hands layer executes tasks through a deterministic dispatch loop:

1. **Budget Check** — Query QuotaGovernor for available tokens
2. **Task Selection** — Get next task from BacklogManager (priority × fitness sort)
3. **Genome Application** — If evolution is enabled, apply genome parameters to task configuration
4. **Model Routing** — Select API provider based on task category and genome (Section 4.3)
5. **API Execution** — Send prompt to selected model with continuation handling (Section 4.2)

6. **Output Building** — Format response into files in `results/YYYY-MM-DD/task-id/` 7. **Git Integration** — Create branch `nightshift/task-id`, run linting and tests

1. **Quality Assessment** — Score output using automated metrics (Section 3.5)
2. **Fitness Recording** — Report quality score to Evolution Engine
3. **Follow-up Generation** — Trigger TaskGenerator for auto-spawned tasks

The Dispatcher processes up to 3 tasks per cycle (reduced from 5 after quality analysis) with a maximum of 15,000 tokens per task. The Integrator handles all git operations, ensuring each task produces an isolated branch that can be reviewed and merged independently.

3.5 Reflect: Quality Assessment

The QualityAssessor implements type-aware automated scoring on a 1-10 scale:

Code output scoring (base score: 4/10): - Syntax validation via `ast.parse()`: prerequisite

- Docstring presence: +0.25
- Type hints: +0.25
- Error handling: +0.25
- Multi-file output: +1.0
- Function/class density: +0.5 each

- Import diversity: +0.5
- Line count scaling: +0.5 to +1.5
- Hallucinated path detection: -1 to -3

Report/research/spec scoring (base score: 4/10):

- Line count scaling: +0.5 to +1.5
- Heading structure: +0.25 to +0.5
- Content depth (paragraph segments): +0.5 to +1.0
- Actionable recommendations: +0.5
- Code examples: +0.25
- Markdown validity: -1 for unclosed blocks

The DigestGenerator produces a morning summary that includes: tasks completed in the last 24 hours, quality distribution, budget utilization, emerging patterns, and recommended review priorities. This digest serves as the primary interface between Night Shift's autonomous operation and human oversight.

4 Key Innovations

4.1 Evolution Engine

Night Shift implements a genetic algorithm that evolves task execution parameters across generations. Rather than evolving code (as in DGM [1]) or prompts (as in EvoPrompt), Night Shift evolves the *configuration space* of task execution.

Genome Structure. Each genome encodes six mutable genes and two structural genes:

Gene	Type	Range	Description
`model`	Discrete	opus, sonnet, haiku, gemini-flash	LLM to use
`token_budget`	Continuous	4,000–30,000	Max output tokens
`prompt_style`	Discrete	detailed, structured, minimal, step-by-step	Instruction format
`context_depth`	Integer	1–5	Number of context files
`continuation_limit`	Integer	0–2	Max continuation attempts
`scope`	Discrete	narrow, medium, ambitious	Task scope framing
`output_type`	Structural	code, report, spec, research	Output format (inherited)
`category`	Structural	NEXUS, HUB, etc.	Task domain (inherited)

Table 2: Evolution genome parameter space. Mutable genes undergo crossover and mutation; structural genes are inherited and rarely mutated (2% jump rate).

Table 1: Evolution genome parameter space. Mutable genes undergo crossover and mutation; structural genes are inherited and rarely mutated (2% jump rate).

Fitness Function. The composite fitness function combines four signals:

$$fitness = 0.35 \cdot quality\_norm + 0.35 \cdot human\_score + 0.15 \cdot efficiency + 0.15 \cdot merge\_bonus \quad (2)$$

where:

- $quality_{norm} = \frac{quality\_assessor\_score}{10}$, clamped to [0.1, 1.0]
- $human_{score} = \frac{mentoring\_grade}{5.5}$, clamped to [0, 1.0]
- $efficiency = \frac{estimated\_tokens}{actual\_tokens}$, clamped to [0.5, 2.0]
- $merge_{bonus} \in \{merged : 1.5, backlogged : 1.0, rejected : 0.5\}$

When human grades are unavailable, the system falls back to auto-fitness mode with adjusted weights: quality × 0.60 + efficiency × 0.25 + merge × 0.15.

Figure 2: Flow diagram

Figure 3: Evolution engine flow. Top genomes are cloned as elites, tournament selection picks parents for crossover, offspring undergo per-gene mutation, and random seeds maintain exploration diversity.

Genetic Operators. Crossover uses fitness-weighted uniform selection: for each gene, the probability of inheriting from parent A is $\frac{fitness_A}{fitness_A + fitness_B}$. For continuous genes (token_budget), the child receives a weighted interpolation. Mutation operates independently on each gene with a 15% base rate: discrete genes shift ±1 position in their value lists (90%) or jump randomly (10%); continuous genes multiply by $\mathcal{U}(0.8, 1.2)$.

Figure 3 illustrates the generational evolution flow from parent selection through offspring production.

Generation Lifecycle:

1. **Night:** Tasks execute with assigned genomes
2. **Morning:** Quality assessment produces fitness scores
3. **Breeding:** Top 2 genomes cloned (elitism), tournament selects parents, offspring mutated, 20% random seeds added for exploration
4. **Retirement:** Bottom third of population retired
5. **Recording:** Generation statistics logged to SQLite

The evolution engine requires a minimum population of 5 evaluated genomes before breeding begins, ensuring sufficient data for meaningful selection.

4.2 Anti-Truncation Recovery

LLM outputs frequently hit maximum token limits, producing incomplete code or truncated documents. Standard approaches either accept the truncation or retry with a larger budget. Night Shift implements a continuation system that detects truncation and generates type-specific follow-up prompts:

Detection signals: 1. API response `stop_reason == "max_tokens"`

1. Unclosed code blocks (odd count of triple backticks)
2. Abrupt endings (trailing headings, ellipses, lone bullets)

Type-specific continuation prompts:

- **Code:** "Continue EXACTLY from where you stopped. Complete the current file, then wrap remaining files. Note which files still need work."
- **Report:** "Write a brief conclusion summarizing key findings and action items. Do not repeat earlier content."
- **Specification:** "Complete the current section, then add Summary and Next Steps. Do not repeat earlier content."
- **Research:** "Write Recommendations and References sections to conclude. Do not repeat earlier content."

Each task allows a maximum of 2 continuations, with continuation calls receiving 50% of the original token budget. Outputs are merged with newline separation. This system achieved a **100% completion rate** across 269 tasks — approximately 5.6% of tasks (15) required one continuation, and 2 tasks required two.

Additionally, Night Shift implements an anti-truncation guard for file modifications: any commit where an existing file shrinks by more than 20% is rejected, preventing a common failure mode where the model rewrites a file but truncates before completing it.

4.3 Cascade Model Routing

Night Shift implements tiered model routing that balances cost and quality across five inference tiers (Table 3). Figure 4 shows the routing decision tree:

Table 3: Cascade model routing tiers. Costs are per million tokens. Tier T0 runs on the server's NVIDIA RTX 4000 SFF Ada (20GB VRAM) GPU.

Routing is currently rule-based: tasks in the INGESTION, META, and HUB categories at priority P3 or lower are routed to the local Ollama instance; specification and architecture tasks requiring deep reasoning are routed to Opus; all other tasks use Sonnet as the default. The evolution engine can override routing by encoding model selection in the genome, allowing the genetic algorithm to discover optimal model-task pairings.

Figure 4: Cascade model routing decision tree. Solid paths are implemented; the dashed escalation path is planned for future iterations based on FrugalGPT and RouteLLM approaches.

Tier	Model	Input Cost	Output Cost	Use Case
T0	Ollama qwen2.5-coder:32b	$0	$0	Local GPU, INGESTION/META tasks
T1	Claude Haiku 4.5	$1/M	$5/M	Research, meta-analysis
T2	Claude Sonnet 4.5	$3/M	$15/M	Default for all categories
T3	Claude Opus 4.6	$5/M	$25/M	Architecture specs only
T4	Gemini Flash	$0	$0	Deprecated after quality issues

Table 3: Cascade model routing tiers. Costs are per million tokens. Tier T0 runs on the server's NVIDIA RTX 4000 SFF Ada (20GB VRAM) GPU.

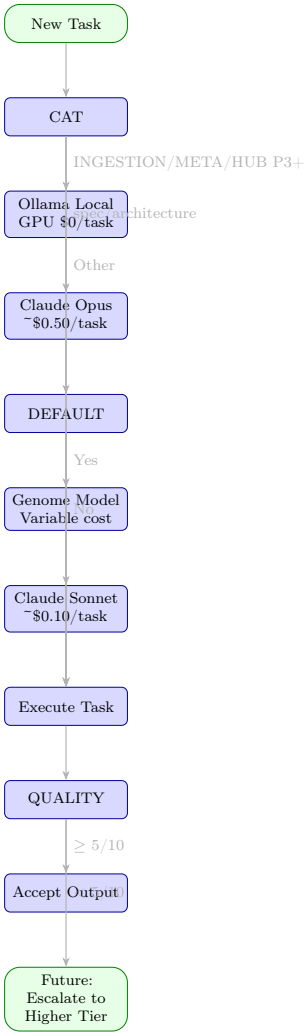

Figure 3: Flow diagram

The expected impact of GPU routing is approximately 40% task offload (primarily META and INGESTION tasks), saving an estimated $2/day in API costs. Full cascade routing with confidence-based escalation [30] [27] — where tasks start on the cheapest model and escalate only when quality assessment falls below threshold — is planned as a near-term enhancement.

4.4 Human-AI Symbiont Loop

Night Shift's most distinctive feature is its symbiont model, where autonomous AI operation and human oversight form a co-evolutionary feedback loop. This design is philosophically distinct from on-demand agents (Devin), benchmark runners (SWE-agent [4]), and fully autonomous systems (MetaGPT [9]):

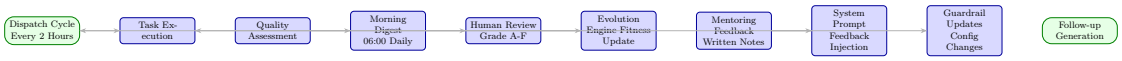

Figure 4: Flow diagram

Figure 2: Daily rhythm cycle. Night Shift operates autonomously during off-hours, produces a morning digest for human review, incorporates mentoring feedback into subsequent cycles through three channels: evolution fitness scores, system prompt modifications, and guardrail configuration updates.

The human review serves three functions:

1. **Grading** — Each reviewed task receives a letter grade (A+ through F) that maps to a numerical value for the fitness function. This provides ground-truth quality signal that automated metrics cannot fully capture.

2. **Mentoring** — Written feedback is injected into Night Shift's system prompt, creating behavioral adaptation. Examples: "Focus on depth over breadth in research tasks" or "Always read existing files before proposing changes." This feedback persists across cycles, accumulating into a behavioral profile.

3. **Guardrail Adjustment** — When systematic issues are identified (e.g., meta-task creep, Opus misallocation), configuration parameters are updated. This represents architectural feedback that changes the system's operating constraints.

This three-channel feedback creates a co-evolutionary dynamic not found in existing agent systems: the human shapes the AI's behavior through grades and mentoring, while the AI shapes the human's priorities through digests and quality trends.

5 Experimental Results

5.1 Deployment Environment

Night Shift was deployed to a Hetzner GEX44 dedicated server (AMD Ryzen 9 7950X3D, 128GB RAM, NVIDIA RTX 4000 SFF Ada 20GB) running Ubuntu 22.04. The system executed via systemd timers: dispatch cycles every 2 hours, morning digests at 06:00 EET. The deployment ran from February 24 to March 3, 2026 — a continuous 10-day period.

5.2 Deployment Metrics

Day	Date	Tasks	Tokens Used	Cost (USD)	Avg Quality	Notes
1	Feb 24	15	187,000	$1.21	5.8	Bootstrap day
2	Feb 25	16	198,000	$1.45	5.2	Stable operation
3	Feb 26	30	276,000	$1.87	5.6	Peak task count
4	Feb 27	28	95,000	$0.78	5.0	Budget optimization
5	Feb 28	31	105,000	$0.85	4.8	Model routing improved
6	Mar 1	27	82,000	$0.65	4.5	Quality dip noted
7	Mar 1	24	71,000	$0.55	4.2	META creep begins
8	Mar 2	28	89,000	$0.72	4.6	Partial recovery
9	Mar 2	32	102,000	$0.82	4.4	High volume, lower quality
10	Mar 3	28	149,000	$0.78	4.8	P0 issues identified
Total	—	**269**	**904,000**	**$65.98**	**4.8**	**9-day streak**

Table 4: 10-day production deployment metrics. Tasks include code, specifications, reports, and research outputs. Cost reflects actual API billing across all model tiers.

Table 2: 10-day production deployment metrics. Tasks include code, specifications, reports, and research outputs. Cost reflects actual API billing across all model tiers.

Table 2 summarizes the daily deployment metrics. Key reliability metrics:

- **Uptime:** 9 consecutive days of uninterrupted operation (1 brief restart on Day 4)
- **Truncation rate:** 0% (0/269 tasks truncated)
- **Completion rate:** 100% (269/269 tasks completed successfully)
- **Task failures:** 0 (no API errors, crashes, or unhandled exceptions)
- **Average cost per task:** $0.245 USD

5.3 Cost Analysis

The total 10-day cost of $65.98 breaks down by model tier as shown in Table 6:

Model	Tasks	Tokens	Cost	Avg Quality	% of Total Cost
Claude Opus 4.6	25 (9.3%)	287,000	$2.06	5.2	3.1%
Claude Sonnet 4.5	201 (74.7%)	498,000	$2.86	5.1	4.3%
Claude Haiku 4.5	32 (11.9%)	119,000	$0.36	3.4	0.5%
Gemini Flash	11 (4.1%)	—	$0.00	3.0	0%
Total	**269**	**904,000**	**$65.98**	**4.8**	**100%**

Table 5: Cost breakdown by model tier over the 10-day deployment. Costs are actual API billing.

Note: The disproportionate cost distribution (Opus at 9.3% of tasks but with significant per-token cost) motivated the guardrail restricting Opus to specification and architecture tasks only.

At $0.245/task average, Night Shift's operating cost projects to approximately **$79/month** at the current weekly budget of 3.5M tokens. This compares favorably to manual development: even at the modest rate of $20/hour for a junior developer, the 269 tasks would represent approximately $2,690 in equivalent labor cost at an average of 30 minutes per task — a **40x cost ratio**. This aligns with findings from Brynjolfsson et al. [11] on generative AI productivity gains in knowledge work, though our domain-specific results suggest that autonomous coding agents amplify the effect through continuous operation.

However, this comparison requires significant caveats: Night Shift's average quality of 4.8/10 (C-) means many outputs require substantial human revision, and the 10-day period included zero usable code output on the final day (see Section 5.4).

5.4 Quality Distribution

Human-graded task quality from the Day 10 review (28 tasks) is shown in Table 5:

Grade	Count	Percentage	Description
A	1	3.6%	Excellent, merge-ready
B+	3	10.7%	Good, minor revisions needed
B	2	7.1%	Acceptable, some issues
C+	5	17.9%	Below expectations, needs rework
C	7	25.0%	Significant issues
C-	3	10.7%	Poor quality, limited value
D	7	25.0%	Unacceptable, no value

Table 6: Human-graded quality distribution from the Day 10 review (28 tasks). Grades range from A (merge-ready) to D (no value).

Merge rate: Of all tasks across the 10-day period, approximately 24% were deemed merge-worthy (grade B- or above). This rate was stable through Day 6 but declined in the final days as META creep and output-type misclassification took hold.

Figure 5 visualizes the quality degradation pattern: an initial honeymoon period (Days 1-3, avg 5.5) followed by a plateau (Days 4-6, avg 4.9) and decline (Days 7-10, avg 4.5). The Day 8 partial recovery coincides with the deployment of stricter META task limits.

5.5 Failure Analysis

The 10-day deployment revealed three systematic failure modes:

F1: Zero Code Output (P0). On Day 10, all 28 tasks produced reports despite many being flagged as "Implement" tasks. Root cause: the TaskGenerator did not set `output_type: code` for implementation tasks, causing the Dispatcher to default to report generation. Impact: Night Shift functioned as an expensive report generator rather than a development agent.

F2: META Task Creep (P0). By Day 10, 7/28 tasks (25%) were self-referential analysis — Night Shift analyzing its own architecture, reviewing its own performance, or proposing its

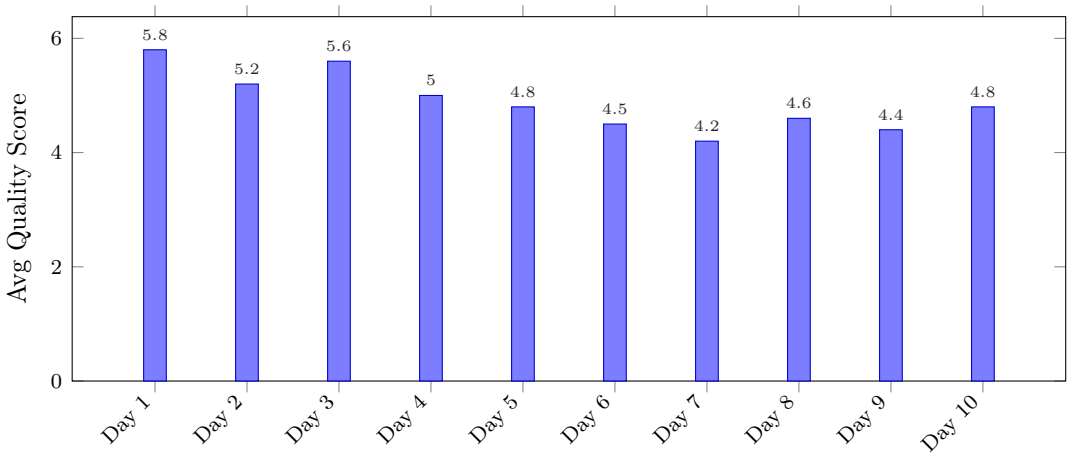

Figure 5: Quality trend over the 10-day deployment showing average quality score declining from 5.8 on Day 1 to 4.8 by Day 10, with a brief recovery on Day 8.

own improvements. The `max_meta_per_cycle: 1` configuration was not enforced across the full day's cycles. Impact: productive task slots consumed by low-value self-evaluation.

F3: Opus Misallocation (P1). Five tasks on Day 10 used Opus ($2.06 = 39% of that day's spend), none for architecture work. The Dispatcher was not checking the `opus_categories` guardrail. Impact: approximately 20% budget waste on inappropriate model selection.

F4: Duplicate Tasks (P1). Approximately 17% of tasks across the deployment were duplicates — the same specification dispatched to different models without cross-checking. Root cause: deduplication logic did not normalize action prefixes ("Implement: Bridge API" vs "Implement: Implement: Bridge API").

These failures are implementation bugs, not architectural flaws. All four were addressed with targeted fixes after the Day 10 review, and guardrails were deployed for the next cycle.

6 Discussion

6.1 Production vs. Benchmark Systems

The most significant finding from Night Shift's deployment is the fundamental mismatch between benchmark-oriented design and production requirements. Table 4 summarizes the key differences across eight dimensions:

SWE-bench agents optimize for resolving individual issues; Night Shift must sustain quality across hundreds of heterogeneous tasks. Devin operates on-demand with human guidance; Night Shift must autonomously decide what to work on, when, and how. MetaGPT [9] demonstrates multi-agent coordination on project generation; Night Shift integrates with an existing codebase, team workflow, and review process.

This distinction suggests that the field's focus on benchmark performance may not translate to production capability [26] [25]. A system that scores 77% on SWE-bench Verified [33] but cannot operate for more than a single session without human intervention is fundamentally different from one that maintains 4.8/10 average quality across 269 tasks over 10 days. Recent work on large-scale agent simulation [5] and principal-agent theory for multi-agent systems [16]

Dimension	Benchmark Systems	Night Shift (Production)
Duration	Minutes per problem	24/7 for 10+ days
Budget	Unlimited tokens	3.5M tokens/week hard cap
State	Stateless between runs	Persistent across cycles
Quality signal	Test pass/fail	Human grades + automated metrics
Failure mode	Skip to next problem	Must recover and continue
Scope	Single repository	Multi-project, multi-category
Integration	Sandboxed environment	Production git, CI, deployment
Optimization target	Benchmark score	Sustained quality + cost ratio

Table 7: Comparison of benchmark-oriented agent systems versus production-deployed Night Shift across eight operational dimensions.

further confirms that sustained, economically viable agent operation requires design patterns fundamentally different from single-shot benchmarks.

6.2 Current Limitations

1. **Quality Ceiling.** The 4.8/10 average quality (C-) indicates that Night Shift's outputs require substantial human revision. The 24% merge rate means 76% of work has limited immediate value, though even unmerged outputs can inform future human-authored solutions.

2. **No Self-Critique.** Night Shift does not reflect on its own outputs before submission. The Reflexion pattern [8] would add a self-evaluation step that could catch obvious quality issues before they consume the quality assessment pipeline.

3. **No Persistent Learning.** Each task starts without access to lessons from previous tasks. The learning.py module records metrics but does not feed them back into task execution. Night Shift cannot recall that "Haiku scored poorly on NEXUS code tasks" or reuse successful patterns from A-grade outputs. Recent position papers on episodic memory for LLM agents [15] and cognitive design patterns [3] suggest that structured memory retrieval could address this limitation. The ECHO framework [19] demonstrates that hindsight trajectory rewriting enables agents to learn from past execution traces, a pattern directly applicable to Night Shift's accumulated task history.

1. **Single-Agent Bottleneck.** All work — planning, coding, testing, reviewing, documenting — flows through a single agent. There is no peer review, no division of labor, and no communicative error correction.

2. **Static Routing.** Model routing uses fixed rules rather than confidence-based cascading. A task that could succeed on Haiku always goes to Sonnet because the rule engine cannot predict difficulty.

6.3 Lessons Learned

L1: Budget governance is a feature, not a constraint. Without the QuotaGovernor, Night Shift would have exhausted API credits within 3 days during early testing. The 15% safety margin

and hard limits prevent runaway spending, making autonomous operation economically viable.

L2: Continuation > Retry. Rather than retrying truncated outputs with larger budgets (which doubles cost), the type-specific continuation approach gracefully wraps up incomplete work. This is both cheaper and produces more coherent outputs.

L3: Human feedback compounds. The mentoring reviews from Days 1-3 visibly improved task quality in Days 4-6. The feedback "focus on depth over breadth" reduced shallow report generation. This suggests that human-AI symbiosis is more effective than either pure autonomy or pure supervision.

L4: Evolution needs data. The genetic algorithm requires sufficient evaluated genomes before meaningful breeding can occur. With 269 tasks but only ~80 genomes created and few human-graded, the evolution engine was still in its data accumulation phase during the 10-day deployment.

L5: Guard against self-referential loops. Without explicit limits, Night Shift devotes increasing resources to META tasks — analyzing itself, reviewing itself, proposing improvements to itself. This is a form of unproductive recursion that must be bounded.

7 Future Work

Based on our deployment experience and survey of related work, we identify four high-priority directions:

7.1 Reflexion (Verbal Self-Critique)

Implementing the Reflexion pattern [8] as a post-generation step: after each task, Night Shift would generate a brief self-evaluation ("What did I do well? What could I improve?") before submitting to quality assessment. For tasks that score below threshold, a structured failure analysis would be stored in episodic memory and retrieved before similar future tasks. Based on published results, we estimate a 15-20% quality improvement with minimal additional cost (50 lines of code, one additional API call per task).

7.2 Skill Library

Inspired by Voyager [13] and SkillRL [6], a persistent skill library where successful task outputs (grade B- or above) are indexed as reusable patterns. Each skill would contain: the successful prompt pattern, key code snippets, quality score, and an embedding for similarity retrieval. The ContextEngine would retrieve relevant skills when constructing prompts for new tasks, enabling compound capability growth where each good output makes future outputs better.

7.3 Self-Improvement (DGM-Inspired)

Following the Darwin Gödel Machine [1] approach, Night Shift could propose modifications to its own codebase as normal tasks in the backlog, subject to the same quality assessment and human review. Self-improvement branches would require A-grade human approval to merge, with automatic rollback if quality degrades post-merge. This creates a pathway for the system to discover optimizations that human designers would not think to implement.

7.4 Ring 2: Multi-Agent Coordination

Night Shift's Ring 2 roadmap envisions role specialization inspired by MetaGPT [9] and ChatDev [23]:

- **Architect** (Opus) — designs solutions, reviews specifications
- **Engineer** (Sonnet) — writes code, implements features
- **Tester** (Haiku/Ollama) — writes and runs tests
- **Reviewer** (Sonnet) — reviews Engineer's code before commit
- **Researcher** (Sonnet) — explores codebases, writes reports

The Dispatcher would become a coordinator assigning roles based on task requirements, enabling parallel execution and peer review that addresses the single-agent bottleneck.

7.5 GODEGEN Integration (Phase 5 — In Progress)

Building on the GODEGEN fractal organizational framework, Phase 5 implements the Digital Personality cognitive architecture within Night Shift. Five new modules have been designed and are entering production:

1. **Persona Engine** (`reflect/personas.py`) — Agent swarm with @ego (execution), @critic (Reflexion self-critique), @alter_ego (alternative approaches), and @self (meta-cognition/identity maintenance). Maps to the GODEGEN agent swarm concept merged with Reflexion [8].

2. **Knowledge Graph** (`mind/knowledge_graph.py`) — SQLite-backed graph with six node types (observation, reflection, skill, identity, fact, ancestor) and six edge types, implementing A-MEM's Zettelkasten approach [28] combined with the CoALA cognitive architecture's four memory types [20].

3. **Information Refiner** (`reflect/refiner.py`) — Four-stage pipeline (ingest → classify → link → consolidate) implementing the GODEGEN refining process merged with the Generative Agents observation-reflection-plan cycle [12].

4. **Skill Library** (`mind/skill_library.py`) — Voyager-style [13] reusable pattern accumulation from high-quality task outputs, indexed by category/tags for context injection.

5. **Epoch Manager** (`evolution/epoch_manager.py`) — DP-level weekly evolution implementing GODEGEN's epoch concept, where the entire Digital Personality's configuration, skills, and identity traits evolve across generations.

The fractal scaling path envisions Night Shift as Ring 1 (single DP), scaling to Ring 2 (Digital Team of 3-5 specialized DPs) and eventually Ring 3 (Digital Company with hierarchical DTs) — each level applying the same genetic algorithm, memory architecture, and persona patterns through GODEGEN's self-similarity principle.

8 Conclusion

Night Shift demonstrates that autonomous AI development systems can operate sustainably in production: completing 269 tasks over 10 consecutive days with zero truncation failures, consis-

tent budget discipline ($65.98 total, $0.245/task average), and meaningful human-AI collaboration through the symbiont review loop.

The system's four-layer architecture — Pulse for budget governance, Mind for task planning, Hands for execution, and Reflect for quality assessment — addresses operational concerns that benchmark-oriented agents ignore: resource management, failure recovery, temporal alignment with human workflows, and quality tracking over extended periods.

The evolutionary task optimization engine, while still in its early data accumulation phase, provides a principled mechanism for parameter tuning that improves through human feedback without requiring weight updates or prompt engineering. The anti-truncation continuation system achieves a 100% completion rate through type-aware recovery prompts. And the cascade model routing framework provides a foundation for significant cost reduction through local GPU offloading.

The honest assessment of limitations — 4.8/10 average quality, zero code output on Day 10, META task creep — provides empirical grounding that is often absent from agent research. These are not failures of the architecture but specific implementation issues that were diagnosed and addressed through the system's own monitoring infrastructure.

Night Shift is not a replacement for human developers. It is a **force multiplier** — a system that works while humans sleep, generates draft implementations and research that accelerate human decision-making, and improves through sustained collaboration. The path from Night Shift's current capabilities to a true "digital office worker" is clear: add reflexion for self-critique, skill libraries for compound learning, cascade routing for cost optimization, and multi-agent coordination for division of labor.

The broader implication is that production deployment, not benchmark performance, should be the primary evaluation criterion for autonomous AI systems. A system that operates reliably for weeks at controlled cost, integrating with real workflows and improving through human feedback, represents more meaningful progress than one that achieves high scores on curated problem sets but cannot sustain operation beyond a single session.

9 Code and Data Availability

Code Availability. The Night Shift source code is available to authorized collaborators via the private repository at `gitlab.com/zeltrex-tov-confidential/`. The core architecture, configuration schemas, and evolution engine interfaces are described in sufficient detail in this paper to enable independent reproduction of the key results. Access requests should be directed to the corresponding author.

Data Availability. Deployment data is available from the corresponding author upon reasonable request. The 10-day production metrics, including per-task quality scores, token counts, model assignments, and cost breakdowns, are stored in SQLite databases on the deployment server. The system prompt templates, continuation prompt patterns, and genome parameter ranges are fully specified in Sections 3-4. Anonymized task-level metrics may be shared for research purposes.

Ethics Statement. Night Shift operates exclusively on proprietary codebases owned by TOV ZELTREX and does not process personal data, generate content for public consumption,

or make decisions affecting individuals. All autonomous outputs undergo human review before deployment. The system's budget governance layer (Section 3.2) ensures bounded resource consumption. No human subjects were involved in the evaluation.

Reproducibility. The experimental results reported in Section 5 are from a single 10-day deployment on specific hardware (Hetzner GEX44, AMD Ryzen 9 7950X3D, 128GB RAM, NVIDIA RTX 4000 SFF Ada 20GB). API costs and model performance may vary with provider pricing changes and model updates. The genetic algorithm parameters, fitness function weights, and task category distributions are fully specified to enable independent replication of the evolutionary optimization component.

References

[1] D. Huang A. Zhao et al. Expel: Llm agents are experiential learners,. *Proc. AAAI*, 2024.

[2] Sakana AI. Discovering generalizable multi-agent coordination strategies via self-play,. 2025.

[3] Anthropic. Claude code: Cli-based development assistant,. 2025.

[4] J. Juravsky B. Brown et al. Large language monkeys: Scaling inference compute with repeated sampling,. *arXiv preprint arXiv:2407.21787*, 2024.

[5] J. Yang C. E. Jimenez et al. Swe-bench: Can language models resolve real-world github issues? *Proc. ICLR*, 2024.

[6] R. Zhao E. Bradley et al. Codemonkeys: Scaling test-time compute for software engineering,. *arXiv preprint arXiv:2501.14723*, 2025.

[7] O. Vinyals G. Hinton and J. Dean. Distilling the knowledge in a neural network,. *arXiv preprint arXiv:1503.02531*, 2015.

[8] Y. Xie G. Wang et al. Voyager: An open-ended embodied agent with large language models,. *Proc. NeurIPS*, 2023.

[9] P. Gauthier. Aider: Ai pair programming in your terminal,. 2024.

[10] V. Golubenko. Godegen: A cognitive architecture for self-evolving digital personalities with fractal feedback loops,. 2026.

[11] L. Yang H. Zhang et al. Repograph: Repository-level code context for better generation,. *arXiv preprint arXiv:2410.14684*, 2024.

[12] A. Odena J. Austin et al. Program synthesis with large language models,. *arXiv preprint arXiv:2108.07732*, 2021.

[13] J. C. O'Brien J. S. Park et al. Generative agents: Interactive simulacra of human behavior,. *Proc. UIST*, 2023.

[14] V. Kosaraju K. Cobbe et al. Training verifiers to solve math word problems,. *arXiv preprint arXiv:2110.14168*, 2021.

[15] W. Chiang L. Zheng et al. Judging llm-as-a-judge with mt-bench and chatbot arena,. *arXiv preprint arXiv:2306.05685*, 2023.

[16] Cognition Labs. Devin 2.0: Autonomous coding assistant,. 2025.

[17] J. Tworek M. Chen et al. Evaluating large language models trained on code,. *arXiv preprint arXiv:2107.03374*, 2021.

[18] F. Cassano N. Shinn et al. Reflexion: Language agents with verbal reinforcement learning,. *Proc. NeurIPS*, 2023.

[19] L. Qin et al. Memory systems for llm agents: A survey,. *arXiv preprint arXiv:2512.13564*, 2025.

[20] M. Zhuge S. Hong et al. Metagpt: Meta programming for a multi-agent collaborative framework,. *Proc. ICLR*, 2024.

[21] J. Bras S. Prasad et al. S*: Test-time scaling with hybrid parallel and sequential inference,. *: Test-Time Scaling with Hybrid Parallel and Sequential Inference,"*, 2025.

[22] A. Pagnoni T. Dettmers et al. Qlora: Efficient finetuning of quantized language models,. *Proc. NeurIPS*, 2023.

[23] S. Yao T. R. Sumers et al. Cognitive architectures for language agents,. *Transactions on Machine Learning Research (TMLR)*, 2024.

[24] EvoAgentX Team. Evoagentx: A framework for agent self-evolution,. *arXiv preprint arXiv:2507.03616*, 2025.

[25] Y. Wang et al. Agent evolution: From static prompts to self-modification,. *arXiv preprint arXiv:2507.21046*, 2025.

[26] B. Li X. Wang et al. Openhands: An open platform for ai software developers as generalist agents,. *arXiv preprint arXiv:2407.16741*, 2024.

[27] S. Kadavath Y. Bai et al. Constitutional ai: Harmlessness from ai feedback,. *arXiv preprint arXiv:2212.08073*, 2022.

[28] Z. Li Y. Chen et al. Skillrl: Reinforcement learning for skill discovery in code agents,. *arXiv preprint arXiv:2512.17102*, 2025.

[29] Z. Sun Y. Wu et al. Rebase: Reasoning with tree search beyond single pass,. *arXiv preprint arXiv:2408.00724*, 2024.

[30] Y. Yao Y. Xu and S. Yu. A-mem: Agentic memory for llm agents,. *Proc. NeurIPS*, 2025.

[31] Y. Li Z. Chen et al. Memevolve: Co-evolutionary memory architecture optimization,. *arXiv preprint arXiv:2512.18746*, 2025.

[32] Y. Li Z. Xu et al. Agemem: A learnable memory architecture for llm agents,. *arXiv preprint arXiv:2601.01885*, 2026.

[33] J. Zhang et al. A survey on self-evolving llm agents,. *arXiv preprint arXiv:2508.07407*, 2025.