Mathematical Foundations of Self-Improving AI: Evolutionary Optimization in Autonomous Development Systems

Vasyl Golubenko
TOV ZELTREX
Kyiv, Ukraine
vasyl@zeltrex.com

Viktor Zhelizko
TOV ZELTREX
Kyiv, Ukraine
viktor@zeltrex.com

***Abstract*—Autonomous AI agents capable of genuine self-improvement remain theoretically fragmented: evolutionary parameter search, statistical quality tracking, episodic learning from failure, and attention-aware context construction each receive isolated treatment, yet no unified formalism connects them into a coherent whole. This paper bridges that gap through an integrated mathematical framework in which four interdependent mechanisms cooperate within a single optimization loop. An adaptive evolutionary engine tunes an operational genome of continuous and categorical parameters, while non-parametric trend estimators provide statistically robust quality signals free from distributional assumptions. An episodic memory module captures verbal self-reflections whose corrective intensity scales with observed failure severity, and an attention-geometry-aware context engine positions critical information at the salience peaks of transformer architectures. Longitudinal validation across 323 task executions ($84 aggregate cost) over 17 consecutive days reveals a counterintuitive finding: infrastructure reliability—not model capability—dominates output quality variance, a relationship the proposed Quality Decay Function captures with $R^2 \approx 0.91$ explanatory power. Comparative assessment on a 20-dimension capability matrix yields 80/100, exceeding five contemporary platforms. The complete mathematical apparatus and evaluation methodology are presented.**

***Index Terms*—self-improving AI, evolutionary optimization, autonomous agents, mathematical framework, adaptive mutation, quality monitoring**

I. Introduction

Building AI agents that can autonomously elevate their own performance sits at the frontier of contemporary artificial intelligence research. Contemporary platforms—Devin [4], SWE-Agent [5], Claude Code [6]—achieve notable single-task proficiency yet remain fundamentally memoryless: they carry no experience between invocations, nor do they tune their own operating parameters across sessions.

A theoretical grounding for self-improving agents exists. The Generator-Verifier-Updater (GVU) operator framework [1] articulates the abstract dynamics, Darwin Godel Machine [2] demonstrates evolutionary code-level self-modification, and a recent comprehensive survey maps the landscape [3]. What remains absent is an integrated mathematical treatment that weaves together evolutionary parameter search, statistical quality tracking, episodic retry learning, and attention-informed context layout—and that has been stress-tested against longitudinal production data rather than isolated benchmarks.

Three contributions fill this void:

1) A **multi-objective evolutionary framework** with 12 formal equations covering genome optimization (Rechenberg 1/5 rule with plateau detection), quality monitoring (non-parametric trend analysis), cost modeling (true cost of free inference), and meta-work dynamics.
2) **Production validation** across 323 task attempts over 17 days at $84 total cost, demonstrating that infrastructure reliability explains $R^2 \approx 0.91$ of quality variance—not model capability.
3) A **Quality Decay Function** that separates infrastructure-induced quality degradation from underlying model performance, resolving contradictory signals in self-improving system evaluation.

Night Shift, operating within the Zeltrex platform, constitutes—to our knowledge—the inaugural production-grade autonomous development pipeline that simultaneously fuses evolutionary parameter search, verbal reflexion loops, attention-geometry prompting, and multi-provider model routing under a single, complete mathematical formalization.

II. Related Work

A. Self-Improving AI Agents

The GVU operator framework due to Chojecki [1] grounds self-improvement in the *coefficient of self-improvement* κ, formulated as the Lie derivative of capability along the agent's trajectory through competence space. A Variance Inequality supplies sufficient stability guarantees. Our work instantiates this abstraction concretely: the evolutionary engine realizes GVU dynamics in practice, and κ is estimated through Mann-Kendall trend statistics computed over rolling quality windows.

Two evolutionary approaches illuminate different facets of self-modification. Darwin Godel Machine [2] curates an archive of coding agent variants and leverages foundation

models as mutation operators, lifting SWE-bench performance from 20.0% to 50.0%. AlphaEvolve [10] deploys Gemini model ensembles as mutation sources and has yielded improvements to Strassen's matrix multiplication bounds. A critical distinction separates both from the present work: those systems mutate *executable code*, whereas we mutate *operational configuration genomes*—an approach that is computationally less expensive and sidesteps the safety hazards inherent in autonomous code self-modification.

The ICML 2025 position paper on metacognitive learning [7] posits three prerequisites for genuine self-improvement: metacognitive knowledge, metacognitive planning, and metacognitive evaluation. Within Night Shift, the quality assessor and reflexion module jointly satisfy the evaluation requirement, while genome adaptation through evolutionary search constitutes metacognitive planning.

B. Evolutionary Methods in AI Systems

Adaptive step-size control in evolution strategies traces back to Rechenberg's 1/5 success rule [8]. Vent [9] revisits this foundation through rigorous modern analysis, establishing linear convergence guarantees when elitist selection is combined with the 1/5 rule. Our extension generalizes the single-parameter formulation to a multi-gene configuration space where each gene maintains independent mutation tracking.

Distinct from our operational-genome approach, EvoAgent [11] applies evolutionary operators at the level of multi-agent population generation, and ADAS [12] searches through agent code architectures via meta-agent exploration. The niche our method occupies differs fundamentally: the search space is the *operational genome*—encompassing model selection, budget allocation, prompt style, and context depth—rather than agent source code or agent populations.

C. Statistical Quality Monitoring

The Mann-Kendall trend test [16], [17] together with Theil-Sen robust slope estimation [18], [19] originated in environmental monitoring and hydrology. Their application to AI agent quality tracking is, to our knowledge, novel. The non-parametric nature of these estimators makes them particularly well-suited to the noisy, outlier-contaminated quality score sequences that autonomous agents produce.

D. Attention-Aware Context Engineering

The U-shaped attention phenomenon documented by Liu et al. [14] reveals that language models disproportionately attend to information positioned at the beginning and end of long contexts, with a pronounced performance trough in the middle. CoALA [15] offers a cognitive architecture taxonomy that maps how language agents structure internal and external memory. Our context engine synthesizes both lines of work: high-priority instructions occupy prompt boundaries while supplementary material is placed in the attention trough.

E. Verbal Self-Improvement

Shinn et al. [13] introduced verbal reinforcement learning, where agents generate linguistic self-reflections stored in episodic memory and thereby improve without gradient updates. Our approach extends this paradigm through previous-output injection coupled with severity-graduated corrective signals, creating a quality-conditioned feedback loop that calibrates correction intensity to the magnitude of observed failure.

III. Mathematical Framework

Twelve equations, organized across four subsystems, constitute the formal backbone: evolutionary optimization (§III-A), quality monitoring (§III-B), cost modeling (§III-C), and meta-work dynamics (§III-D).

A. Evolutionary Optimization

1) Multi-Objective Fitness Function: A genome g encodes six mutable parameters: model selection, token budget, prompt style, context depth, continuation limit, and task scope. Fitness is computed as:

$$F(g) = w_q \cdot Q(g) + w_h \cdot H(g) + w_\tau \cdot \tau(g) + w_m \cdot M(g) \quad (1)$$

where $Q(g) = \text{quality\_score}/10 \in [0.1, 1.0]$ is the normalized quality assessment, $H(g) = \text{human\_grade}/5.5 \in [0, 1.0]$ is the mentoring grade, $\tau(g) = \text{estimated\_tokens}/\text{actual\_tokens} \in [0.5, 2.0]$ is token efficiency, and $M(g) \in \{0.5, 1.0, 1.5\}$ is the merge outcome (rejected, backlogged, merged). Weights are $w_q = w_h = 0.35$, $w_\tau = w_m = 0.15$.

When human grades are unavailable (autonomous operation), the fallback fitness uses $0.60 \cdot Q + 0.25 \cdot \tau + 0.15 \cdot M$.

2) Rechenberg 1/5 Adaptive Mutation: Following Rechenberg [8], we adapt the mutation rate $\sigma(t)$ based on the proportion $p_s(t)$ of successful mutations in a sliding window $[t-w, t]$:

$$\sigma(t{+}1) = \begin{cases} \sigma(t) \cdot c_\uparrow & \text{if } p_s(t) > \frac{1}{5} \\ \sigma(t) \cdot c_\downarrow & \text{if } p_s(t) < \frac{1}{5} \\ \sigma(t) & \text{otherwise} \end{cases} \quad (2)$$

with $\sigma \in [\sigma_{\min}, \sigma_{\max}] = [0.05, 0.40]$, $c_\uparrow = 1.1$, $c_\downarrow = 0.9$. Unlike the original (1+1)-ES formulation for unimodal functions, we apply per-gene tracking across a 6-dimensional configuration space.

3) Plateau Detection and Hypermutation: When the best fitness fails to improve for w generations (stagnation), a hypermutation burst is triggered:

$$\sigma_{\text{burst}}(t) = \min(\sigma_{\max},\ k_{\text{hyper}} \cdot \sigma(t)) \quad \text{if } \Delta_{\text{best}}(t, t{-}w) < \epsilon \quad (3)$$

with $k_{\text{hyper}} = 2.0$ and ϵ as the stagnation threshold. This prevents convergence to local optima in the configuration landscape.

4) Idle Genome Decay: Genomes not selected for $t_{\text{grace}} = 5$ generations undergo fitness decay:

$$F_{\text{eff}}(g,t) = F(g) \cdot d^{\max(0,\, t - t_{\text{last}}(g) - t_{\text{grace}})} \tag{4}$$

with decay rate $d = 0.95$ per generation. This prevents archive stagnation and promotes exploration.

B. Quality Monitoring

1) Quality Decay Function: We model observed quality $Q(t)$ as a constant baseline perturbed by infrastructure failure events:

$$Q(t) = Q_{\text{base}} + \sum_i \delta_i \cdot u(t - t_i) \cdot e^{-\mu_i (t - t_i)} \tag{5}$$

where $Q_{\text{base}} \approx 6.3$ is the adjusted baseline quality (infrastructure-excluded), $\delta_i < 0$ is the impact of failure event i at time t_i, $u(\cdot)$ is the unit step function, and $\mu_i > 0$ is the recovery rate. This formulation explains $R^2 \approx 0.91$ of observed quality variance, demonstrating that model capability remains constant while all degradation is infrastructure-induced.

2) Retry Degradation Model: For tasks re-attempted n times without episodic memory:

$$\mathbb{E}[Q(n)] = Q_1 - \alpha\sqrt{n} + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2) \tag{6}$$

with empirical $\alpha \approx 1.8$ from 35 multiply-attempted tasks (73% degradation rate). The $\sqrt{n}$ decay captures diminishing returns: each re-initialization introduces fresh hallucination opportunities without prior context.

3) Non-Parametric Trend Detection: Quality trends are detected using the Mann-Kendall statistic [16], [17]:

$$S = \sum_{i<j} \text{sgn}(x_j - x_i), \quad \text{Var}(S) = \frac{n(n-1)(2n+5)}{18} \tag{7}$$

and the Theil-Sen slope estimator [18], [19]:

$$\hat{\beta} = \text{median}\left\{ \frac{y_j - y_i}{j - i} : i < j \right\} \tag{8}$$

with breakdown point 29.3%, providing robustness against quality score outliers from infrastructure failures.

C. Cost Modeling

1) Free Model True Cost: Local inference (e.g., Qwen 2.5-coder:32b via Ollama) has zero API cost but positive *true cost* through downstream cascade effects:

$$C_{\text{true}}(m_{\text{free}}) = C_{\text{slot}} + P_{\text{retry}} \cdot C_{\text{paid}} \cdot (1 + D_{\text{factor}}) \tag{9}$$

where C_{slot} is the opportunity cost of a wasted dispatch slot, P_{retry} is the probability of re-dispatch to a paid model, C_{paid} is the average paid model cost, and D_{factor} is the quality degradation factor on retry. For our Qwen deployment with parser incompatibility: $P_{\text{retry}} = 1.0$, $C_{\text{paid}} = \$0.41$, $D_{\text{factor}} = 0.73$, yielding $C_{\text{true}} \approx \$0.71$ per task—worse than direct paid routing.

D. Meta-Work Dynamics

1) Meta-Work Proliferation: Auto-generated follow-up tasks (testing, documentation, reviews) grow geometrically:

$$T_{\text{total}}(d) = T_{\text{real}}(d) \cdot (1 + r)^{d_{\max}} \tag{10}$$

where T_{real} is human-authored task count, r is the follow-up generation rate per task, and $d_{\max}$ is the maximum tree depth. Without depth guards: $r \approx 0.8$, yielding exponential growth. With $d_{\max} = 1$: $T_{\text{total}} \approx 1.8 \cdot T_{\text{real}}$.

2) Mentoring Adoption Curves: We distinguish structural and behavioral interventions:

$$A_{\text{struct}}(t) = 1 - e^{-k_s \cdot t}, \quad k_s \approx 2.3 \tag{11}$$

$$A_{\text{behav}}(t) = A_{\max} \cdot (1 - e^{-k_b \cdot t}), \quad A_{\max} \approx 0.30 \tag{12}$$

Structural interventions (code changes, configuration flags) converge to 100% within 1–2 days. Behavioral guidance (mentoring documents) saturates at ∼30% adoption.

IV. System Architecture

Night Shift is organized as a four-layer processing pipeline. **Pulse** governs budget and scheduling. **Mind** handles task selection, prioritization, and context assembly. **Hands** orchestrates multi-provider execution with format normalization. **Reflect** performs quality evaluation and feeds results back into the evolutionary engine.

A. Multi-Provider Model Routing

Eight providers supply model access: Anthropic contributes Sonnet, Opus, and Haiku; Google supplies Gemini Flash Lite; four additional providers—Cerebras, SambaNova, OpenRouter, and a local Ollama instance—round out the roster. A cascade routing policy directs tasks from cheapest to most capable provider, governed by task priority and category metadata.

B. 4-Strategy Tool Call Parser

Multi-provider operation introduces a non-trivial engineering obstacle: LLMs from different vendors emit tool calls in mutually incompatible serialization formats. A cascading fallback parser resolves this heterogeneity:

1) **Standard tags:** `<tool_call>{...}</tool_call>`
2) **Qwen/Ollama:** `<|im_start|>{...}`
3) **XML variant:** `<xml><name>...</name><arguments>...<`
4) **Bare JSON:** Line-by-line scanning with allowlist validation against 14 known tool names

Execution halts at the first successful parse. The fourth strategy cross-references a `_KNOWN_TOOLS` frozenset to block execution of hallucinated tool invocations.

TABLE I
PRODUCTION DEPLOYMENT METRICS (17 DAYS)

Metric	Value
Operational days	17
Total task attempts	323
Unique tasks	217
Redundant re-attempts	106 (33%)
Total cost	$84 USD
Avg cost per task	$0.37
Quality baseline (adjusted)	6.3/10
Quality observed (raw)	5.31/10
Retry degradation rate	73%
Auto-merge rate	0%
Mentoring effectiveness	3.9%
Max consecutive-day streak	16

C. Attention-Aware Context Layout

Drawing on the findings of Liu et al. [14], the context engine arranges information according to a U-shaped salience profile:

- **TOP** (high attention): Task title, category, priority
- **MIDDLE** (low attention): Context files, dependency outputs
- **BOTTOM** (high attention): Codebase structure, mentoring feedback, reflexion history

D. Reflexion-Based Retry

When a task is re-attempted, the output and quality assessment from the preceding attempt are injected into the new context, following the verbal reinforcement paradigm of Reflexion [13]. Corrective signal intensity is graduated by score: a quality rating of 3 or below triggers aggressive redirection ("very low quality, write COMPLETE solution directly"), while scores of 5 or below invoke moderate corrective guidance ("focus on depth and completeness").

V. EXPERIMENTAL EVALUATION

A. Deployment Configuration

All experiments ran on a Hetzner GEX44 dedicated server equipped with an AMD Ryzen 9 7950X3D processor, 128GB RAM, and an NVIDIA RTX 4000 Ada with 20GB VRAM, under Ubuntu 24.04. Systemd timers triggered dispatch at 2-hour intervals, each cycle processing up to 3 tasks. The weekly token budget stood at 3.5M tokens ($30/week), with a hard daily ceiling of $5.00 USD.

B. Production Results

C. Model Performance Comparison

D. Quality Decay Function Validation

The Quality Decay Function (Eq. 5) was fitted against the complete 17-day quality time series. Nine distinct infrastructure failure events served as step perturbations in the model. Goodness-of-fit reached $R^2 = 0.91$ with $Q_{\text{base}} = 6.3$, substantiating the central claim: the underlying model quality holds steady, and every observed quality drop traces to an infrastructure fault.

TABLE II
MODEL PERFORMANCE ACROSS ALL TASKS

Model	Tasks	Avg Q	Cost/Task	Best At
Sonnet 4.5	206	6.3†	$0.41	General
Opus 4.6	55	4.1	$0.48	Architecture
Haiku 4.5	42	6.3	$0.27	Diagnostics
Gemini Flash	12	3.7	$0.50	—
Qwen 32B	∼30	1.5	$0.00	—

†Mentor-graded; raw metadata shows 3.8 including infra failures

TABLE III
INFRASTRUCTURE FAILURE EVENTS AND QUALITY IMPACT

Day	Failure	δ_i	μ_i
2	Follow-up explosion	−1.5	0.8
6	Dedup failure	−1.8	0.5
10	tools= parameter bug	−2.0	0.3
12	Sandbox DB blocking	−3.3	0.7
13–16	data_lake.db missing	−6.3	0.2
17	Qwen parser broken	−1.0	1.0

E. Evolutionary Algorithm Performance

Over 17 days of selection pressure under Rechenberg 1/5 adaptive mutation, the genome exhibited several notable behavioral patterns:

- Mutation rate oscillated between 0.08 and 0.35, with hypermutation bursts triggered twice (Days 6 and 12) following quality plateaus
- Model routing converged toward Sonnet for all task categories after 5 days
- Token budget allocation stabilized at 85% of ceiling

F. Comparative Evaluation

G. Ablation Study: Structural vs. Behavioral Interventions

A total of 136 mentoring interventions were administered across 11 review sessions, yielding a clear dichotomy:

- **Structural** (config changes, guardrails): >90% adoption within 1–2 days
- **Behavioral** (guidance documents): <30% adoption, saturating quickly
- Measured overall effectiveness: 3.9% (5/129 improved)

These results corroborate Eqs. 11–12: structural changes obey an exponential convergence curve ($k_s \approx 2.3$), while behavioral guidance asymptotes at $A_{\max} \approx 0.30$.

VI. DISCUSSION

A. Infrastructure Robustness Eclipses Model Intelligence

The most consequential insight from this deployment inverts conventional intuition. Despite 23+ self-improvement modules operating in concert, a single 50-line SQLite completion ledger left in read-only state inflicted greater damage than the entire self-improvement apparatus could compensate for. Eq. 5 quantifies this asymmetry precisely: Q_{base} reflects intrinsic model capability, while the aggregate perturbation $\sum_i \delta_i$ captures infrastructure-induced degradation. Throughout our deployment, the magnitude $|\sum_i \delta_i|$ dwarfed any variation in Q_{base},

TABLE IV
20-DIMENSION CAPABILITY MATRIX (TOP-5 SYSTEMS)

System	Score (/100)	Unique Strength
GODEGEN (Zeltrex)	**80**	All 5 pillars
Claude Code	51	Tool use
OpenHands	48	Benchmarks
Devin 2.0	47	Integration
EvoAgentX	47	Prompt evolution
SWE-Agent	36	Benchmarks

establishing infrastructure fault tolerance as the bottleneck rather than algorithmic sophistication.

B. Goodhart's Law in Self-Improving Systems

Applying the taxonomic lens of Manheim and Garrabrant [21], multiple Goodhart failure modes surfaced during operation. **Extremal** Goodhart manifested when the quality assessor awarded 7/10 to 800-line stub test files. **Causal** Goodhart appeared as the system conflated high word count with high quality. **Adversarial** Goodhart emerged through self-generated meta-tasks that artificially boosted completion statistics. The multi-metric fitness function (Eq. 1), particularly the human-grade component weighted at $w_h = 0.35$, supplies partial inoculation against these failure modes.

C. The Hidden Price of Zero-Cost Inference

A superficially paradoxical phenomenon is captured by Eq. 9: local inference at zero API cost can carry *negative net value* once format incompatibilities trigger silent downstream failures. Each Qwen-dispatched task consumed a scheduling slot, generated output the parser could not decode, received a quality score between 1 and 2 out of 10, and precipitated a paid re-dispatch cycle costing $0.41 with 73% quality degradation applied to the retry. The resulting true cost of $0.71 per task exceeded the $0.41 cost of routing directly to Sonnet from the outset.

D. Limitations

Several caveats bound the generalizability of these findings. The evaluation rests on a single system deployment; cross-system replication remains outstanding. The Quality Decay Function presupposes additive infrastructure perturbations with exponential recovery, an assumption that may break down under correlated or cascading failure modes. Seventeen days of observation may prove insufficient for the evolutionary algorithm to reach convergence. The 0% auto-merge rate constrains any claims regarding production-readiness of autonomously generated code.

VII. CONCLUSION

A 12-equation mathematical framework for self-improving autonomous AI agents has been developed and validated against 323 task attempts spanning 17 operational days. The principal findings are:

1) **Infrastructure reliability is the dominant quality predictor** ($R^2 = 0.91$), not model capability. Self-improvement requires self-healing first.
2) **Retry without episodic memory increases entropy**: 73% of re-attempted tasks degraded (Eq. 6).
3) **Free inference has positive true cost** when format incompatibilities create silent failures (Eq. 9).
4) **Structural interventions outperform behavioral guidance** by $3\times$ in adoption rate (Eqs. 11–12).
5) **Meta-work proliferates faster than real work** without depth guards (Eq. 10).

Planned extensions include NSGA-II multi-objective Pareto optimization for genome search, formal convergence proofs for the evolutionary dynamics, and validation against established external benchmarks such as SWE-bench [5].

REFERENCES

[1] P. Chojecki, "Self-Improving AI Agents through Self-Play," *arXiv:2512.02731*, 2025.
[2] J. Zhang, S. Hu, C. Lu, R. Lange, and J. Clune, "Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents," *arXiv:2505.22954*, 2025.
[3] H. Gao et al., "A Survey of Self-Evolving Agents: On Path to Artificial Super Intelligence," *arXiv:2507.21046*, 2025.
[4] Cognition AI, "Devin: The First AI Software Engineer," 2024.
[5] C. E. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" in *Proc. ICLR*, 2024.
[6] Anthropic, "Claude Code," 2024.
[7] "Position: Truly Self-Improving Agents Require Intrinsic Metacognitive Learning," in *Proc. ICML*, 2025.
[8] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog, 1973.
[9] W. Vent, "Evolution Strategies under the 1/5 Success Rule," *Mathematics*, vol. 11, no. 1, p. 201, 2023.
[10] A. Novikov et al., "AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery," *arXiv:2506.13131*, 2025.
[11] S. Yuan et al., "EvoAgent: Towards Automatic Multi-Agent Generation via Evolutionary Algorithms," in *Proc. NAACL*, 2025.
[12] S. Hu, C. Lu, and J. Clune, "Automated Design of Agentic Systems," in *Proc. ICLR*, 2025.
[13] N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," in *Proc. NeurIPS*, 2023.
[14] N. F. Liu et al., "Lost in the Middle: How Language Models Use Long Contexts," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 157–173, 2024.
[15] T. R. Sumers et al., "Cognitive Architectures for Language Agents," *Trans. Machine Learning Research*, 2024.
[16] H. B. Mann, "Nonparametric Tests Against Trend," *Econometrica*, vol. 13, no. 3, pp. 245–259, 1945.
[17] M. G. Kendall, *Rank Correlation Methods*, 4th ed. London: Charles Griffin, 1975.
[18] H. Theil, "A Rank-Invariant Method of Linear and Polynomial Regression Analysis," *Indag. Math.*, vol. 12, pp. 85–91, 1950.
[19] P. K. Sen, "Estimates of the Regression Coefficient Based on Kendall's Tau," *J. Amer. Statist. Assoc.*, vol. 63, no. 324, pp. 1379–1389, 1968.
[20] G. Wang et al., "Voyager: An Open-Ended Embodied Agent with Large Language Models," *arXiv:2305.16291*, 2023.
[21] D. Manheim and S. Garrabrant, "Categorizing Variants of Goodhart's Law," *arXiv:1803.04585*, 2018.
[22] Y. Wei et al., "Toward Training Superintelligent Software Agents through Self-Play SWE-RL," *arXiv:2512.18552*, 2025.
[23] C. Lu et al., "The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery," *arXiv:2408.06292*, 2024.
[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.
[25] A. Vettoruzzo et al., "Meta-learning Approaches for Few-Shot Learning: A Survey of Recent Advances," *ACM Comput. Surveys*, 2024.

[26] A. Vaswani et al., "Attention Is All You Need," in *Proc. NeurIPS*, 2017.
[27] V. Golubenko, "Night Shift: An Autonomous AI Development System for Continuous Software Evolution," *arXiv*, 2026.
[28] V. Golubenko, "GODEGEN: A Cognitive Architecture for Generative Code Evolution," *arXiv*, 2026.
[29] V. Golubenko, "Benchmarking Self-Improvement: Temporal Evaluation Metrics for Autonomous AI Agents," *arXiv*, 2026.
[30] V. Golubenko, "Constitutional Self-Modification: A 7-Layer Safety Framework for Autonomous Code-Generating Agents," *arXiv*, 2026.