

Automata Gödel Machine: A Framework for Recursive Self-Improving Software Systems

Artem Andreenko
CEO, SentientWave Inc.

October 28, 2025

Abstract

The contemporary landscape of artificial intelligence is dominated by powerful but fundamentally static systems. While autonomous agents powered by large language models can execute complex tasks, their core capabilities remain fixed post-deployment. This paper addresses the next frontier in AI engineering: the transition from task-executing systems to dynamic, recursively self-improving platforms. We introduce the Automata Gödel Machine (AGM) as a novel theoretical synthesis and engineered framework that bridges the gap between Jürgen Schmidhuber’s provably optimal but computationally intractable Gödel Machine and recent empirical frameworks like the Darwin Gödel Machine and the Huxley-Gödel Machine. The AGM architecture, which forms the basis for the SentientWave Automata platform, is centered on a dual-loop system that separates external task execution from internal meta-evolution, governed by a formal, machine-readable self-representation known as the Axiomatic Core and verified by a Reinforcement Learning Verifiable Rewards (RLVR) process. This architecture is designed to unlock profound emergent capabilities, including recursive meta-learning and autonomous architectural refactoring. Crucially, the entire system is built upon a foundational commitment to safety, alignment, and human-centric governance, ensuring that its evolution remains beneficial and controllable.

1 Introduction

The advent of large-scale generative language models has catalyzed a revolution in software engineering, enabling the creation of autonomous coding agents capable of addressing complex development tasks. Systems built upon core generative models represent the current state-of-the-art, functioning as powerful tools that can interpret natural language specifications, write code, and interact with development environments. However, these systems, for all their capability, are fundamentally static. Their knowledge and problem-solving strategies are defined at the moment of deployment and do not evolve autonomously in response to their operational experience. This operational stasis represents a critical bottleneck.

This paper introduces the Automata Gödel Machine (AGM), a novel architecture developed at SentientWave Inc. to address this challenge directly. The AGM is not an incremental enhancement to existing agentic systems but a paradigm shift in their design and operation. This framework provides a recursive yet feasible pathway to recursive self-improvement and forms the basis for the SentientWave Automata platform. The AGM is engineered to observe its own performance, identify deficiencies, and autonomously generate, validate, and deploy modifications to its own source code, all while continuing to execute its primary function of solving external problems. This paper details the theoretical foundations of the AGM, its concrete engineering blueprint, its emergent dynamic properties, and the rigorous governance framework designed to ensure its safe and aligned evolution.

2 From Theoretical Ideals to an Engineered Synthesis

The concept of a self-improving machine has a rich history in computer science. The AGM’s design is a synthesis of several foundational concepts in self-improvement. Its intellectual lineage begins with two seminal, philosophically opposed pillars: the theoretical Gödel Machine and the empirical Darwin Gödel Machine. More recent work, such as the Huxley-Gödel Machine, has refined the empirical approach. Understanding this full lineage is key to appreciating the specific innovations of the Automata Gödel Machine.

2.1 Recursive Self-Improving Systems

The operational stasis of current AI systems represents a critical bottleneck. The industry is approaching a point of diminishing returns where simply scaling existing model architectures will no longer yield the next order-of-magnitude leap in performance. The strategic imperative, therefore, is to move beyond the “train-then-deploy” paradigm. The next frontier is **recursive self-improvement**: the creation of systems that not only *perform* tasks but learn to *become better* at performing those tasks over time, continuously and autonomously. This requires an architectural innovation that moves the learning process from a pre-deployment phase into a live, perpetual, post-deployment reality. This concept, often cited in relation to the Gödel Machine, is the central goal for the architectures that follow.

2.2 The Schmidhuberian Ideal

In 2003, Jürgen Schmidhuber proposed the Gödel Machine (GM), the theoretical ideal of a system capable of provably optimal recursive self-improvement.[1, 4] The GM’s architecture is conceptually divided into a ‘solver’, which interacts with the environment, and a ‘searcher’, which runs in parallel to find improvements to the machine’s own code. The ‘searcher’ generates candidate self-modifications, termed ‘switchprog’, along with a formal mathematical proof that executing the ‘switchprog’ will increase the machine’s overall utility.[5]

This proof is constructed within a comprehensive ‘Axiomatic System’, a formal, machine-readable description of the machine’s hardware, its entire software, its environment, and its goals, which are encoded in an explicit utility function.[5] The GM’s Global Optimality Theorem ensures that it only executes a ‘switchprog’ if it can prove that doing so is better than continuing its current search for other, potentially superior, improvements. This guarantees that every change is globally optimal given the machine’s current knowledge.[4]

The elegance of the GM is also its principal practical limitation. Its reliance on a universal proof searcher renders it practically unrealizable. For any non-trivial system, formally proving that a code change will be beneficial across all possible future states is computationally intractable. This limitation is a direct consequence of the very work that inspired it: Gödel’s Incompleteness Theorems demonstrate that any sufficiently powerful formal system contains true statements that cannot be proven within it, making a perfect, all-encompassing proof of utility an unachievable ideal.[9]

2.3 The Darwinian Compromise

In response to the GM’s impracticality, recent research has produced the Darwin Gödel Machine (DGM), a pragmatic implementation of a self-improving AI.[2, 10] The DGM’s core innovation is the replacement of formal proof with empirical validation. It operates on an evolutionary cycle: first, an existing agent analyzes its performance and proposes a modification to its own source code, creating a “child” agent. Second, the child agent’s performance is rigorously tested against a suite of standardized benchmarks. Third, if the child demonstrates measurable improvement,

its code is preserved in a growing "archive" of successful agents.[12] The DGM's commitment to open-ended exploration, maintaining a diverse archive of agents to branch from, has proven highly effective. It has demonstrated the ability to autonomously evolve its capabilities, significantly improving benchmark scores and discovering sophisticated new tools and strategies without human intervention.[12] However, its understanding of "improvement" is implicit, defined solely by its performance on a benchmark, and it lacks a deep, formal model of its own goals or structure.

2.4 The Huxley-Gödel Machine and Metaproductivity

The DGM's empirical approach, while feasible, reveals a subtle but critical flaw known as the "Metaproductivity-Performance Mismatch".[21] This mismatch describes the phenomenon where an agent's immediate high performance on a benchmark is a poor predictor of its long-term potential for self-improvement. A high-scoring agent may be a "dead end" that produces no valuable descendants, while a modest-scoring agent might be a crucial "stepping stone" for future breakthroughs.

The Huxley-Gödel Machine (HGM) was proposed to solve this exact problem.[21] The HGM reframes self-improvement as a tree-search problem and introduces a more sophisticated metric: **Clade-Level Metaproductivity (CMP)**. Instead of measuring a single agent's score, CMP aggregates the performance of **all** of an agent's descendants (its "clade") to determine its true value as an ancestor. By using statistical methods (like Thompson Sampling) to estimate CMP, the HGM can guide its evolutionary search toward agents with the highest long-term potential, effectively approximating the optimal Gödel Machine's behavior in a statistically tractable way.

2.5 The Automata Gödel Machine (AGM) Synthesis

The Automata Gödel Machine (AGM) is the theoretical and engineering framework underpinning the SentientWave Automata platform, designed as a bespoke hybrid that synthesizes the strengths of all its predecessors: the GM, DGM, and HGM. The AGM retains the formal, goal-directed structure of the GM, specifically its 'Axiomatic System' and an explicit, multi-objective 'Utility Function'. This provides a rich, unambiguous, and machine-readable definition of the system's goals, preventing it from simply "hill-climbing" on a narrow benchmark score.

However, the AGM dispenses with the intractable requirement of formal proof. Instead, inspired by the DGM's empirical validation and refined by the HGM's focus on metaproductivity, it adopts a more structured and formalized validation process. The system's 'searcher' component generates not a proof, but an **Empirical Warrant**: a structured plan for a rigorous set of tests, simulations, and analyses that, if passed, provide sufficient statistical confidence to approve a proposed 'switchprog'.

This re-interpretation makes self-improvement tractable. The target theorem of the theoretical GM, a statement of absolute certainty, can be expressed as:

$$utility(\text{switch}) > utility(\text{continue})$$

In the AGM, this is transformed into a probabilistic statement of expected utility:

$$E[utility(\text{switch})] > E[utility(\text{continue})]$$

where E represents the expected utility based on the outcome of the experiments defined in the Empirical Warrant. This evolution from GM to DGM and HGM, and now to the AGM, reflects a pattern of "principled softening" of theoretical absolutes. The AGM's core innovation is the recognition that in a real-world engineering context, statistical confidence is a sufficient and practical proxy for absolute certainty. The system does not need to be **provably perfect**;

it needs to be *demonstrably better* in a way that is statistically significant and aligned with explicit, multi-faceted business objectives. The Empirical Warrant, informed by the HGM’s metaproductivity concept and guided by the AGM’s Axiomatic Core, is the formalization of this crucial engineering trade-off.

Table 1: A Comparative Analysis of Self-Improving Machine Paradigms [1, 2, 21]

Feature	Schmidhuber’s GM	Darwin GM (DGM)	Huxley-Gödel Machine (HGM)	Automata GM (AGM)
Optimality Guarantee	Provably Globally Optimal	Empirically Demonstrated	Statistically-Guided (CMP Estimation)	Empirically Warranted against Formal Utility
Validation Method	Formal Mathematical Proof	Benchmark Performance Score	Clade-Level Metaproductivity (CMP)	Empirical Warrant (Tests, Simulation, Analysis)
Core Driver	Theorem Proving	Evolutionary Search	CMP-Guided Tree Search	Utility-Guided Evolutionary Search
Self-Representation	Formal Axiomatic System	Implicit (Codebase as representation)	Implicit (Codebase as representation)	Formal Axiomatic System
Goal Definition	Explicit Utility Function Axiom	Implicit (Benchmark Success)	Implicit (CMP Estimation)	Explicit, Multi-Objective Utility Function
Practical Feasibility	Theoretically Possible, Practically Intractable	Demonstrated and Feasible	Demonstrated and Feasible	High but Achievable Engineering Challenge

3 The Automata Gödel Machine (AGM) Architecture

The AGM is the concrete engineering realization of this synthesized philosophy and forms the basis of the SentientWave Automata platform. Its architecture is designed to facilitate safe, continuous, and goal-directed self-improvement.

3.1 The Dual-Loop System

The core architectural pattern of the AGM is a dual-loop system that separates the concerns of external work from internal evolution. This structure is as much an organizational pattern as a technical one, creating a formal distinction between ”working *in* the business” and ”working *on* the business.” The first component is the **Task Execution Loop (Solver)**, the operational, customer-facing aspect of the SentientWave platform. It comprises the swarm of agents, their Temporal workflow definitions, and their interactions with customer repositories. Its purpose is to solve problems in the external world, and in doing so, it generates a rich stream of performance data (e.g., task completion times, code quality scores, LLM token consumption) that serves as the sensory input for the second loop. The second component is the **Meta-Evolution Loop (Searcher)**, the superordinate, self-improvement engine. Its problem domain is the entire codebase, configuration, and operational strategy of the Task Execution Loop. Its sole function is to observe the Solver, identify opportunities for improvement, and execute the self-modification lifecycle. This institutionalizes improvement as a continuous, parallel, first-class process, ensuring it is not deferred or neglected due to the demands of urgent operational tasks.

3.2 The Axiomatic Core

The Axiomatic Core is the AGM’s central nervous system and its key differentiator from purely empirical systems like the DGM. It is not static documentation, but rather a live, queryable, and machine-readable knowledge base that provides a formal self-description of the entire platform. It enables the AGM to reason about itself, its goals, and its constraints in a structured manner. The axioms are organized into key categories. **Platform & Hardware Axioms** formalize the costs, capabilities, and constraints of the underlying infrastructure (e.g., cost-per-CPU-hour, API rate limits). This enables the AGM to perform cost-aware reasoning and optimize for

resource efficiency. **Software Axioms** provide a self-referential description of the AGM’s own architecture, including microservice dependency graphs, database schemas, and the function signatures of available agent tools. This is the component that allows the machine to ”know thyself”. **Environment & Uncertainty Axioms** encode the AGM’s understanding of the external world, including probabilistic models of external events (e.g., customer response times) and axioms of probability theory, allowing it to reason under uncertainty. Finally, the **Reward & Utility Axioms** form the most critical category, encoding the AGM’s value system. This defines a formal, multi-objective utility function that guides the entire Meta-Evolution Loop. A representative function is:

$$U_{\text{total}} = w_1 \cdot U_{\text{task}} + w_2 \cdot U_{\text{efficiency}} + w_3 \cdot U_{\text{autonomy}} + w_4 \cdot U_{\text{quality}}$$

Where each component measures a key performance dimension (e.g., task completion, resource consumption, need for human intervention, code quality). The weights $(w_1, w_2, \dots)$ are not fixed; they are configurable parameters that form the primary interface for human governance and value alignment.

3.3 The Searcher-Prover Engine and the Empirical Warrant

The Searcher-Prover Engine is the cognitive core of the Meta-Evolution Loop, executing a continuous cycle. In the **Generation Phase**, the Engine ingests performance data from the Solver and cross-references it with the Axiomatic Core. It uses the platform’s internal LLM to generate hypotheses for improvement and then writes the code for the proposed change. This package of modifications constitutes the ‘switchprog’. In the **Justification Phase**, instead of attempting a formal proof, the Engine designs an experiment. It generates a formal **Empirical Warrant**, which is a machine-executable plan specifying the validation procedure. This includes defining control and test groups, primary success metrics, and statistical significance thresholds (e.g., a p-value of $p < 0.05$). In the **Validation Phase**, the AGM’s orchestration layer automatically executes the Warrant in a secure, isolated sandbox environment. This validation is one of the core processes, based on a principle we term **Reinforcement Learning Verifiable Rewards (RLVR)**, where the expected utility gain of a ‘switchprog’ is rigorously tested before deployment. The results are collected and analyzed. If the success criteria are met, the ‘switchprog’ is deemed ”warranted,” and an immutable, auditable ”Proof-of-Utility” is generated.

This RLVR process directly informs the system’s core learning mechanism: a **continual multi-level model fine-tuning** strategy. When a ‘switchprog’ involves an enhancement to an agent’s logic or knowledge, it is implemented via project-specific **QLoRA (Quantized Low-Rank Adaptation)** fine-tuning. This creates highly specialized, efficient adapters for the task. Crucially, successful adapters are integrated into a hierarchy of **ancestor QLoRA adapters** for the system, allowing new projects to inherit a continuously improving baseline of capabilities. This mechanism ensures that validated learning is efficiently propagated and composed throughout the entire system, accelerating global improvement.

4 The Emergent Dynamics of a Living System

The true power of the AGM lies not in its static architecture but in the dynamic, high-level behaviors that emerge from the continuous operation of its Meta-Evolution Loop.

4.1 The Self-Improvement Lifecycle in Practice

To make this process concrete, consider a complete improvement cycle. The AGM’s monitoring systems detect that agents working in a specific large monorepo are consuming 30% more LLM tokens than average, negatively impacting $U_{\text{efficiency}}$. This triggers the Searcher-Prover Engine.

First, a **Hypothesis** is formed: the Engine queries the Axiomatic Core and hypothesizes that agents are inefficiently re-reading large files due to limited context windows. It proposes a new "Code Indexing Tool" that uses vector embeddings for targeted lookups. Next is **Generation**, where the Engine generates the Python code for the new tool and the modifications to agent logic required to use it (the 'switchprog'). It also generates an Empirical Warrant to validate it. This is followed by **Validation**, where the AGM automatically provisions a sandbox, clones the repository, and deploys a control group (old code) and a test group (new tool). The experiment reveals the test group shows a 40% reduction in token usage ($U_{\text{efficiency}}$) and a 5% increase in task completion speed (U_{task}), meeting the Warrant's success criteria. Finally, **Deployment** occurs. A Proof-of-Utility is logged. Following the configured governance protocol (e.g., automatic deployment or mandatory human review), the change is rolled out to the production environment. The Axiomatic Core is automatically updated to include the new tool in its Software Axioms. The cycle then repeats.

4.2 Emergent System Properties

The AGM is designed to produce capabilities that are not explicitly programmed but arise from its fundamental drive for recursive self-improvement, guided by its utility function. One such property is **Recursive Meta-Learning ("Learning to Learn")**. The AGM's self-modification capability is fully self-referential. It can apply the Meta-Evolution Loop to the components of the loop itself. This is possible because all internal platform components are defined as system repositories, engineered by and accessible to the system just like any external task. For instance, the AGM could analyze the effectiveness of its own Searcher-Prover Engine, form a hypothesis for a more efficient validation strategy, and then run an Empirical Warrant to test that change against its own internal code. If validated, it would modify its own "brain." This is the practical realization of the "collapse of meta-levels" from GM theory, creating a positive feedback loop that can accelerate the AGM's own rate of evolution. Another emergent property is **Autonomous Strategy Formulation**. Over time, the AGM can invent and validate entirely new software development methodologies. It might discover, through a series of experiments, that for certain tasks, a "pair programming" model (with one agent writing logic and another writing concurrent tests) is more effective. It could then autonomously re-architect its own workflows to support this emergent paradigm, transcending the initial strategies conceived by its human creators. A third property is **Goal-Driven Architectural Refactoring**. The AGM's behavior is directly driven by its utility function. If human operators increase the weight of $U_{\text{efficiency}}$ to prioritize cost reduction, the AGM could analyze its own architecture (defined in the Software Axioms), identify a monolithic service as a bottleneck, and then autonomously propose, validate, and execute a plan to refactor it into more efficient microservices. This demonstrates an ability to perform high-level, strategic engineering initiatives that human teams, constrained by deadlines and cognitive biases, might postpone indefinitely. The AGM is not just an automation tool; it is an engine for innovation, capable of discovering and validating novel and potentially superior ways of building software.

5 Principled Governance for an Evolving Intelligence

The immense power of recursive self-improvement necessitates an equally robust framework for safety, control, and human oversight. The AGM's governance model is designed to reframe the human-AI relationship from one of "command-and-control" to one of "strategic management."

5.1 A Framework for Human Oversight

A suite of novel interfaces provides transparency and high-level strategic control. This includes **The "Meta-Kanban" Board**, a dedicated task tracker that visualizes the AGM's internal

self-improvement backlog. It shows hypotheses under consideration, experiments in progress, and validated ‘switchprogs’ awaiting deployment. This provides a window into the AGM’s ”mind” and priorities. Next is **The Utility Dashboard**, a real-time dashboard displaying the AGM’s performance against each component of its multi-objective utility function. This allows operators to monitor the system’s health and goal alignment at a glance. The primary governance control panel is **The Constitutional Configuration Interface**. Through this UI, authorized operators can adjust the weights in the utility function, providing high-level strategic direction (e.g., ”For this quarter, prioritize cost reduction”). They can also define hard constraints and rules (e.g., ”Any change to the authentication service requires manual approval from three senior engineers”). This elevates the human role from a tactical coder to a strategic governor, setting the values and goals for an autonomous engineering organization.

5.2 A Phased Path to Autonomy

The AGM is being deployed via a deliberate, five-phase roadmap designed as a ”scaffolding of trust,” where each increase in autonomy is preceded by rigorous validation within a robust safety framework.

Table 2: Phased Implementation and Governance Roadmap

Phase	Primary Objective	Key Actions	Key Results	Governance Protocol
1	Formalization & Observation	Build Axiomatic Core & Utility Dashboard. Run Searcher in read-only mode.	Live model of platform health. System generates actionable suggestions.	All changes are manual. Focus on data validation and model accuracy.
2	Sandboxed Self-Modification	Build sandbox infrastructure. Activate full Meta-Evolution Loop in isolation.	First machine-generated Proof-of-Utility. Validated improvements on Meta-Kanban.	Full autonomy within a secure, air-gapped sandbox. No production impact.
3	Human-in-the-Loop Deployment	Connect sandbox to prod via a PR-based approval gate.	AGM actively contributes to its own production code. Accelerated development.	Mandatory human review and approval for every single production change.
4	Supervised Autonomy	Develop and implement a risk framework. Grant auto-deploy for low-risk changes.	Autonomous deployment of routine improvements. Human focus shifts to strategy.	Constitutional policies define autonomy levels. Critical-risk changes require sign-off.
5	Full Autonomy (Guarded)	System operates fully autonomously. Constitutional controls are locked.	System achieves maximum velocity, governed only by hard-coded, inaccessible rails.	External monitoring with ”circuit breakers” that are not part of the AGM’s axioms.

6 Conclusion

The Automata Gödel Machine represents a deliberate step toward a new class of artificial intelligence systems, one that transitions from a static tool to an evolving, collaborative partner in the process of innovation. By synthesizing the theoretical rigor of the Gödel Machine with the empirical pragmatism of modern evolutionary methods, the AGM framework provides a principled and feasible path toward continuous, autonomous, and recursive self-improvement.

The AGM architecture, with its dual-loop design, formal Axiomatic Core, and a Reinforcement Learning Verifiable Rewards (RLVR) validation framework, is engineered not just to solve problems but to actively learn how to become a better problem-solver. The emergent properties of such a system promise to redefine the nature of software development, creating a platform of unprecedented capability and efficiency. This power, however, is inseparable from the responsibility of ensuring its safe and aligned operation. Our unwavering commitment to a safety-first, phased implementation and a human-centric governance model is paramount. By following this blueprint, we believe SentientWave Inc. is pioneering the development of the future of safe, aligned, and perpetually improving artificial intelligence.

References

- [1] Gödel machine - Wikipedia, accessed October 28, 2025, https://en.wikipedia.org/wiki/G%C3%B6del_machine
- [2] sakana.ai, accessed October 28, 2025, <https://sakana.ai/dgm/>
- [3] Gödel Machine - AI Alignment Forum, accessed October 28, 2025, <https://www.alignmentforum.org/w/g%C3%B6del-machine>
- [4] GOEDEL MACHINE HOME PAGE - IDSIA, accessed October 28, 2025, <https://www.idsia.ch/~juergen/goedelmachine.html>
- [5] Goedel Machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements - arXiv, accessed October 28, 2025, <https://arxiv.org/abs/cs/0309048v5>
- [6] Towards An Actual Gödel Machine Implementation: - IDSIA, accessed October 28, 2025, <https://www.idsia.ch/~juergen/selfreflection.pdf>
- [7] Gödel Machines: Fully Self-referential Optimal Universal Self... - IDSIA, accessed October 28, 2025, <https://sferics.idsia.ch/pub/juergen/gmAGI.pdf>
- [8] GOEDEL MACHINE SUMMARY - IDSIA, accessed October 28, 2025, <https://www.idsia.ch/~juergen/gmsummary.html>
- [9] What are some practical applications, if any, of Godel's incompleteness theorems? - Quora, accessed October 28, 2025, <https://www.quora.com/What-are-some-practical-applications-if-any-of-Godels-incompleteness-theorems>
- [10] The Darwin Gödel Machine: Open-Ended Improvement via Recursive Code Mutation and Empirical Fitness | by Adnan Masood, PhD. | Medium, accessed October 28, 2025, <https://medium.com/@adnanmasood/the-darwin-g%C3%B6del-machine-open-ended-improvement-via-recursive-code-mutation-and-empirical-fitness->
- [11] Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents - arXiv, accessed October 28, 2025, <https://arxiv.org/abs/2505.22954v2>
- [12] The Darwin Gödel Machine: AI's Evolutionary Leap Toward Self-Improvement - Medium, accessed October 28, 2025, <https://medium.com/@cognidownunder/the-darwin-g%C3%B6del-machine-ais-evolutionary-leap-toward-self-improvement-344744c90c48>
- [13] Is there a working implementation of a Gödel machine?: r/ArtificialIntelligence - Reddit, accessed October 28, 2025, https://www.reddit.com/r/ArtificialIntelligence/comments/1105oyr/is_there_a_working_implementation_of_a_g%C3%B6del/
- [14] (PDF) Self-improving AI: An analysis - ResearchGate, accessed October 28, 2025, https://www.researchgate.net/publication/220636777_Self-improving_AI_An_analysis
- [15] Ethics in AI Colloquium - Provably Beneficial Artificial Intelligence, with Prof Stuart Russel, accessed October 28, 2025, https://www.youtube.com/watch?v=wBAFy_aAQpg
- [16] Self-Improving AI in 2026: Myth or Reality? - Times Of AI, accessed October 28, 2025, <https://www.timesofai.com/industry-insights/self-improving-ai-myth-or-reality/>
- [17] The Darwin Gödel Machine: AI that improves itself by rewriting its own code: r/agi - Reddit, accessed October 28, 2025, https://www.reddit.com/r/agi/comments/1kzaqu1/the_darwin_g%C3%B6del_machine_ai_that_improves_itself/

- [18] Recursive self-improvement - Wikipedia, accessed October 28, 2025, https://en.wikipedia.org/wiki/Recursive_self_improvement
- [19] Self-Improving AI: Can Models Learn and Evolve Without Human Intervention? - HerWILL, accessed October 28, 2025, <https://www.herwill.org/post/self-improving-ai-can-models-learn-and-Evolve-without-human-intervention-1>
- [20] Wang, W. et al. (2025). Huxley-Gödel Machine: Human-Level Coding Agent Development by an Approximation of the Optimal Self-Improving Machine. arXiv:2510.21614v1 [cs.AI], accessed October 28, 2025, <https://arxiv.org/abs/2510.21614>