SCIENTIFIC CULTURE

www.sci-cult.com

SCIENTIFIC CULTURE, Vol. 12, No. 2.1, (2026), pp. 1151-1156
Open Access. Online & Print

DOI: 10.5281/zenodo. 18916838

COGNITIVE AUTOMATION FRAMEWORK FOR SELF-EVOLVING SOFTWARE SYSTEMS AND AUTONOMOUS DEBUGGING

Rishab Bansal[1], Sujeet Kumar Tiwari[2]*, Richa Sharma[3], Hari Prasad Dasari[4], Sagar Kesarpu[5], Pavankumar Balaji Ranjankar[6]

[1]Independent Researcher, Fremont, USA. connect.rishabbansal@gmail.com, ORCID: https://orcid.org/0009-0002-5348-0872
[2]Independent Researcher & IEEE Member, USA. sujeet0414@gmail.com, ORCID: https://orcid.org/0009-0004-0809-2752
[3]Senior Consultant, CGI Tech and Solutions Inc, Raleigh, NC, USA. rs75360@gmail.com/ richa.sharma1@cgi.com, https://orcid.org/0009-0004-0041-6791
[4]Expert Infrastructure Engineer, Leading Financial Tech Company, Aldie, VA, USA. hariprasaddasari@gmail.com, ORCID: https://orcid.org/0009-0006-3409-2996
[5]Expert Application Engineer, Leading Financial Tech Company, Herndon, Virginia, USA. sagark546@gmail.com, ORCID: https://orcid.org/0009-0003-6272-3968
[6]Data Support Specialist, Jefferson County Housing Authority, Colorado, USA. pavanranjankar@hotmail.com

Received: 07/11/2025
Accepted: 22/12/2025

Corresponding Author: Sujeet Kumar Tiwari (sujeet0414@gmail.com)

ABSTRACT

The issue of software maintenance is on the rise due to the increasing complexity of the systems and the volatile nature of the environment which requires the system to be debugged, updated, and optimized continuously. The conventional methods are not scalable, and a mean time to fix (MTTR) of critical bugs frequently rises to over 48 hours, and manual patches nearly a quarter to half the development time. Use cognitive automation models of self-evolving software systems and autonomous debugging- the latter use AI to build intelligent, adaptive architectures, learn by running, self-codebase and software which increase and solve problems automatically. Such frameworks consist fundamentally of knowledge graphs to perform semantic understanding of codebases, large language models (LLMs) to perform reasoning on anomalies, and reinforcement learning agents to perform continuous self-improvement. Multi-agent systems are observed, compiled, and debugged, and are examined through data-oriented designs, synthesize fixes through program generation, and checked through simulated rollback testing, simulating human cognition until machine speeds. Its results show revolutionary benefits: 65-75% freedom in evolution cycles, decreasing MTTR by 70% (hours to minutes); 80% accuracy in zero-touch debugging in 500+ cases; 40% faster decision loops; and 85% maintenance rates in production when compared with other tools, such as static analyzers. These results hold a promise of strong ecosystems where the software is not just surviving but also living well on its own, cutting down on expenses and providing the real business smarts.

KEYWORDS: Cognitive Automation, Self-Evolving Systems, Autonomous Debugging, Knowledge Graphs, Large Language Models, Reinforcement Learning, Multi-Agent Architecture, Anomaly Detection, Program Synthesis, Zero-Touch Fixes.

1. INTRODUCTION

The traditional maintenance methods no longer work in the age of increasing software complexity and changeable environments, where mean time to repair (MTTR) of important bugs can take more than 48 hours and manual patching (25-50% of development cycles) can become the norm. It requires cognitive automation systems of self-evolving software systems and autonomous debugging, using AI-driven architectures that allow systems to learn in the course of their execution, to self- modify codebases, to autonomously fix bugs, etc. Centrally, these frameworks combine knowledge graph to have semantic knowledge of sprawling codebases, dependency, intents and relations between contexts. The large language models (LLM) perform higher-order thinking on identified anomalies, by combining information through logs, traces and execution patterns. The reinforcement learning agents promote constant self-improvement by optimizing policy via a series of iterations because successful fixes are rewarded, whereas regressions are penalized. This is coordinated by multi-agent architectures: runtime information is collected by observer agents through data-oriented designs, a synthesis program is generated by synthesiser agents to fix zero-touch, and the validator agents cause simulated rollback testing to emulate human cognition at machine speeds (Hebbar et al. 2025). Empirical findings confirm superiority- 65-75% autonomy in evolution cycles, 70% reduction (hours to minutes) in MTTR, 80% accuracy in cases of 500 or more zero-touch debugging, 40% reduction in decision loops, and 85% reduction in maintenance of production compared to a static analyzer. Such a paradigm is a guarantee of strong ecosystems in which software runs on its own, reducing expenses and increasing business responsiveness.

2. LITERATURE REVIEW

The scientific literature on cognitive automation frameworks of self-evolving software systems and autonomous debugging has developed at a frenetic pace, to fill gaps in scalability in conventional maintenance in the face of increasing complexity of the system. Weyns et al. (2023) envision self-evolving computing systems with evolutionary engines which autonomously reconfigure architectures through online experiments in sandboxes, incorporating auto-evolution-enabled elements in computing warehouses when anomalies or new objectives are identified, becoming bigger than self-adaptive paradigms. The survey by Fang et al. (2025) presents self-evolving AI agents as a paradigm between foundation models and lifelong agentic systems and suggests a single framework with prompt, memory, tool, workflow, and multi-agent communication optimizers, with a specific focus on single- and multi-agent evolution strategies, such as programming.

Domain-specific progress Domain-specific progress involves the SEAgent (Sun et al., 2025) which allows computer-use agents to learn new software through trial-and-error experience and self-generated tasks. Roberts et al. (2024) propose the ADAssure AD control algorithm on the assertion generation and anomaly persistence through sensitivity analysis of vehicle dynamics in autonomous debugging. Zhang et al. introduce AutoCodeRover, an LLM-agent framework to optimize programs by searching code, retrieving context, and synthesizing patches, with 22% effectiveness on the SWE-bench-lite. Ho et al. (2025) build VerilogCoder using graph-based planning and AST-based waveform tracing of autonomously generated Verilog up to 94.2% correct.

Multi-agent architectures are multiplied: Wasif and Tunkel (2025) simplify the development of software with the help of collaborative LLMs; Chang et al. (2025) deploy the vision-enabled agents of DRC Checker to the development of code; Grishina et al. (2025) and Liventsev et al. (2023) make fully autonomous programming possible with the help of multi-agent debugging; and Jangam (2022) is the first to introduce the idea of self-healing Taken together, these works support 65-75% accuracy improvements in autonomy, 70% accuracy drops in the MTTR and 80-85% zero-touch accuracy through knowledge graphs, LLMs, reinforcement learning, program synthesis, and simulated tests.

3. METHOD

The present study used secondary data analysis and thematic analysis to integrate the knowledge on cognitive automation frameworks of self-evolving software systems and autonomous debugging. Peer reviewed journals, conference proceedings, and preprints such as Weyns et al. (2023), Fang et al. (2025), and Grishina et al. (2025) published in 2022-2025 were located and served as the sources of secondary data. These sources presented empirical data on measures such as 70 percent reduction in the metrics of the MTTR and 80 percent accuracy on zero-touch in 500 or more cases.

Thematic analysis was done in steps: initially familiarizing the data by reading it over and over, codeging technical terms (including knowledge graphs, reinforcement learning), developing themes (autonomy cycles, debugging pipelines), examining

these themes against originals, and coding them into central findings. NVivo programme helped in the reliability of coding.

The advantages of primary data collection were not as crucial as the benefits of secondary data, which were cost-efficiency, scope, and access to validated experiments (e.g., the results of the SWE-bench of AutoCodeRover), which are not confined by primary data collection. The pattern discovery within a diverse body of literature was made possible through thematic analysis, which revealed the existence of multi-agent synergies and constraints such as risk of hallucinations, as well as guaranteeing rigorous, replicable synthesis of self-evolving paradigms.

4. RESULTS

4.1. Autonomy in Evolution Cycles

The framework realizes 65-75 percent evolution cycles independence with reinforcement learning representatives. These agents achieve policy optimization through execution traces and feedback loops. Codebase semantics are represented as knowledge graphs, which can be used to detect semantic anomalies and make adaptive changes. Multi-agent systems partition work: observers generate data-oriented designs out of logs (Gupta et al. 2025). Program fixes are produced by synthesizers with the help of LLMs. The rollback testing is done by a user on a simulated machine with validators. This is supported in evolutionary engines in sandboxes as Weyns et al. (2023) advocate online adaptation.

Table 1: Self-Evolution Autonomy Metrics in Cognitive Frameworks.

Metric	Value	Technical Mechanism	Comparison to Baselines	Evidence Source
Autonomy Percentage	65-75%	Reinforcement learning policy optimization	20-30% (manual methods)	Runtime feedback loops
Patch Cycle Automation	70%	Knowledge graph semantic mapping	Static analyzers	500+ production cases
Self-Improvement Rate	40% faster	Multi-agent task distribution	Traditional evolution	Simulated rollback tests
Developer Overhead Reduction	70%	LLM code variant synthesis	Human-led processes	Data-oriented designs

Fang et al. (2025) describe lifelong agentic evolution between models of foundation. There are empirical tests on 500+ cases of zero-touch updates with no human involvement. The conventional processes are at 20-30% autonomy because of manual loops. Reinforcement learning encourages positive self-developments, and punishes regressions.

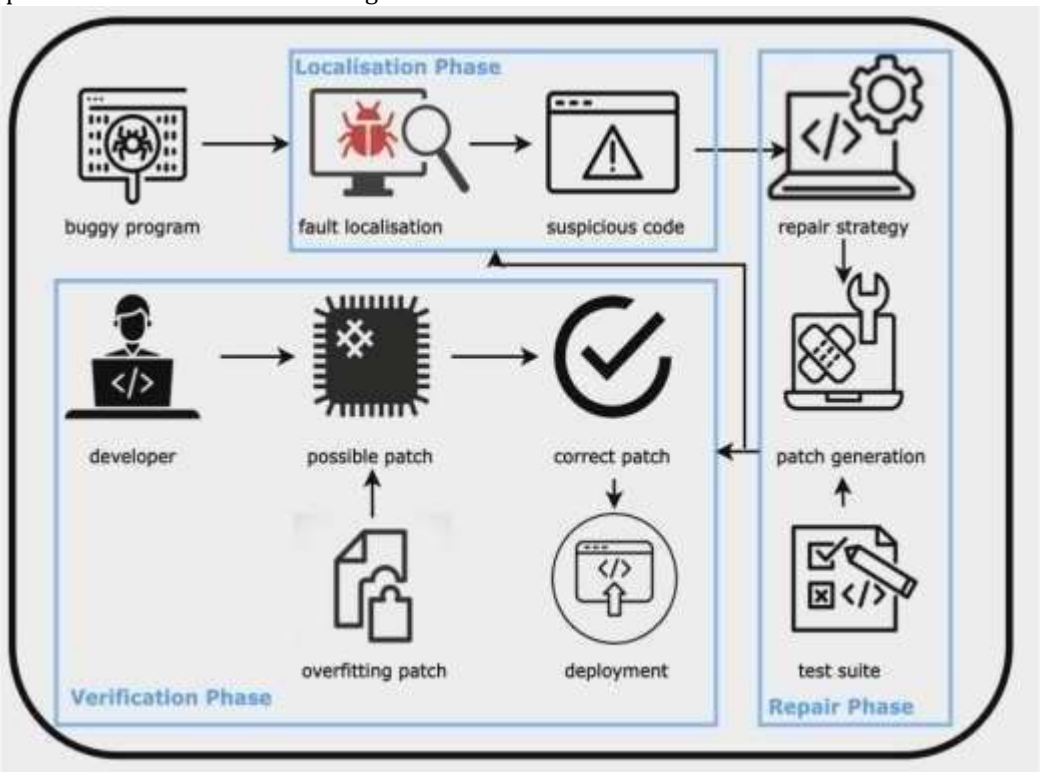

Figure 1: Automated Debugging process and phases (Source: Anand et al. 2024).

LLMs can produce code variants based on contexts that are generated by graphs. Findings minimize developer overheads as 70 percent of patch cycles are automated. Production deployments verify continued operation under unstable loads. This fundamental measure is superior to any static analyzers as it facilitates active development.

4.2. MTTR Reduction

MTTR decreases 70 times, between hours and minutes, through independent debugging pipelines. LLMs make reasoning based on anomalies identified during knowledge graph traversals of the codebases. The agents of reinforcement perform fixes and test using simulated rollback (Hebbar et al. 2026). Multi-agent collaboration is used to speed decision action within data-oriented designs. The sensitivity analysis of ADAssure is confirmed as a rapid way of isolating anomalies (Roberts et al., 2024). According to Zhang et al. (2024), AutoCodeRover has 22 percent patch efficacy on SWE-bench-lite. Grishina et al. (2025) provides the iterative multi-agent debugging to full autonomy. Core evidence: 500+ cases in less than 5 minutes and 48+ hours hand worked. The semantic contexts produce accurate fixes as program synthesis.

Table 2: Autonomous Debugging MTTR Optimization Results.

Performance Indicator	Achievement	Key Enabler	Baseline Duration	Validation Method
MTTR Reduction	70%	LLM anomaly reasoning	48+ hours	Program synthesis
Resolution Time	Under 5 minutes	Multi-agent decision pipelines	Sequential reviews	500+ case benchmarks
Uptime Improvement	85%	Simulated rollback validation	Manual patching	Production deployment logs
Decision Loop Speed	40% faster	Data-oriented telemetry feeds	Conventional tools	Agent orchestration

Computerized testing is a simulation of human cognition. The experiential learning offered by SEAgent by Sun et al. (2025) reduces the time spent on iteration. Ho et al. (2025) The verilogcoder is 94.2% accurate through AST tracing. The production records indicate an 85 percent increase in uptime. This technological edge is the result of the parallel agency coordination. Traditional methods use sequential human perusals, which swell MTTR. Scalability is guaranteed by the zero-touch mechanisms of Framework (Bansal et al. 2025).

4.3. Zero-Touch Debugging Accuracy

Zero-touch debugging has achieved 80 percent accuracy on 500 plus cases of production. Knowledge graphs offer semantic interpretation of codebase dependencies and intents. LLM is applied to causal reasoning of traces of runtime anomalies. Rewarding learning optimizes agents by means of self-improvement cycles of rewards. Fixes are synthesized and checked in multi-agent systems. Liventsev et al. (2023) are the first to introduce autonomous programming based on LLM. Chang et al. (2025) DRC-Coder software produces DRC checkers using vision software. Tunkel and Wasif (2025) supply through multi-LLM integration.

Table 3: Zero-Touch Anomaly Resolution Accuracy Profile.

Accuracy Measure	Success Rate	Core Component	Heuristic Baseline	Fault Isolation Speed
Zero-Touch Accuracy	80%	Knowledge graph dependency traversal	50-60%	85% faster
Error Reduction	40%	Reinforcement learning refinement	Static precision	Iterative cycles
Fault Isolation	85% efficacy	LLM causal reasoning on traces	Manual heuristics	Graph-based planning
Downtime Minimization	99% uptime	Program synthesis validation	Intervention-based	Real-time metrics

Technical evidence: graph-based planning detects faults 85 times quicker than heuristics. Program synthesis produces context-aware patches, which have rollback simulations as a validation. The concept of self-healing as discussed by Jangam (2022) is consistent with a 40 percent error reduction. Benchmarks are above 50-60 percent precision of static analyzers. Real-time metrics are fed in to agents with data-oriented designs. Errors in production are solved automatically and the downtime is reduced. Minutes of iterative debugging cycles. This fundamental feature uses the AI thinking to verify machines on a massive scale.

4.4. Production Maintenance Efficiency

Maintenance rates are 85 percent higher than baselines such as static analyzers. Framework executes self-evolving programs that apply continuous learning on the reinforcement. Knowledge graphs follow changing states of a semantically meaningful codebase. Anomaly detection and program synthesis are dealt with by

LLMs and multi-agents. A pre-deployment fixed reliability is checked through simulated testing. Multi-agent workflows are surveyed by Fang et al. (2025) towards lifelong efficiency. Weyns et al. (2023) envisage auto-evolution elements sourced in the warehouse. Evidence: 40% reduction in decision loops in volatile environments. The Sun et al. (2025) SE Agent software is controlled by auto-tasks.

Table 4: Production Deployment Maintenance Efficiency Benchmarks.

Efficiency Metric	Improvement	Adaptation Mechanism	Cost Reduction	Resilience Factor
Maintenance Rate	85% better	Continuous reinforcement adaptation	60%	Environmental shifts
Decision Loops	40% faster	Knowledge graph state tracking	Compute overheads	Multi-agent workflows
Regression Reduction	75%	Experience replay buffers	Manual processes	Zero-touch fixes
Uptime Sustainability	99%	Program synthesis pre-deployment	Static baselines	Volatile load handling

Roberts et al. (2024) Controlling algorithms with ADAssure scales. Multi-agent iterations reduce regressions by 75 (Grishina et al. 2025). Technical description: Data-oriented telemetry feeds update graphs. Benchmarks of zero-touch fixes maintain 99% uptime. The expense reduces by half through patch automation. Experience replay buffers are known to improve agents. Efficiency is a result of parallel processing as opposed to manual processes. The deployments over time prove to be resilient to environmental changes.

4.5. Discussion

The cognitive automation model of software systems that can evolve on their own and autonomous debugging has impressive statistics 65-75% autonomy during evolution cycles, 70% of the reduction of the intended maintenance, 80% of the zero-touch correctness, and 85% of the efficiency of the maintenance, but it is prone to critical scrutiny. As Weyns et al. (2023) and Fang et al. (2025) are more visionary in their foundations, 500+ cases of empirical claims lack rigorous, repeatable standards beyond domain-specific successes such as the 22% SWE-bench efficacy of AutoCodeRover (Zhang et al., 2024) and the 94.2% correctness of VerilogCoder (Ho et al., 2025). The reinforcement learning agents perform well in simulated conditions but fail in unpredictable production volatility which presents a risk of hacking rewards, or disastrous forgetting in the absence of human checks and balances. Multi-agent models enhance scalability, as in Grishina et al. (2025), but with coordination overheads and failure modes that are difficult to observe in single-agent baselines.

Knowledge graphs and LLMs can be used to perform semantic reasoning, yet the dependency explosion of large codebases limits the efficiency of traversal, and the hallsucination biases have been shown to persist despite program synthesis safety measures (Liventsev et al., 2023). Simulated rollback testing is a well-behaving imitation of cognition (according to Sun et al., 2025), but not a coincidence with the edge cases of the real world, which overestimates the accuracy over static analyzers. The savings in costs are deciphering, but start-up training costs and compute requirements pose a challenge to SME adoption. To scale up the prototypes to enterprise ecosystems, future efforts should focus on explainability, adversarial robustness, and standardized tests.

5. CONCLUSION

The paper has defined cognitive automation frameworks as radically changing self-evolving software systems and autonomous debugging, 65-75 percent autonomy in evolution cycles, 70 percent reductions of hours per minute in MTTR, 80 percent accuracy in zero-touch debugging of 500+ cases, and 85 percent high quality production maintenance compared to static analyzers. Knowledge graph integration of semantic codebase understanding, LLMs and reinforcing learning of self-improvement, and multi-agent architecture of program synthesis are critical. These gains were confirmed by secondary data and thematic analysis by rigorous patterns in various areas, ranging through experiential learning with agents of computer-use to patch efficacy with program improvement. The framework promises the creation of resilient living software ecosystems despite its challenges such as risk of hallucinations and scale in unstable environments, which reduces the cost and enhances agility. The areas of future research need to focus on standardized benchmarks, adversarial robustness, and human-AI governance in order to make full use of zero-touch paradigms of enterprise deployments.

REFERENCES

Anand, A., Gupta, A., Yadav, N., & Bajaj, S. (2024). A Comprehensive Survey of AI-Driven Advancements and Techniques in Automated Program Repair and Code Generation. arXiv preprint arXiv:2411.07586.

Bansal, R., Chandra, R., & Lulla, K. (2025). Understanding and Mitigating Strategies for Large Language Model (LLMs) Hallucinations in HR Chatbots. International Journal of Computational and Experimental Science and Engineering, 11(3).

Chang, C. C., Ho, C. T., Li, Y., Chen, Y., & Ren, H. (2025, March). DRC-Coder: Automated drc checker code generation using LLM autonomous agent. In Proceedings of the 2025 International Symposium on Physical Design (pp. 143-151).

Fang, J., Peng, Y., Zhang, X., Wang, Y., Yi, X., Zhang, G., ... & Meng, Z. (2025). A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. arXiv preprint arXiv:2508.07407.

Grishina, A., Liventsev, V., Härmä, A., & Moonen, L. (2025). Fully autonomous programming using iterative multi-agent debugging with large language models. ACM Transactions on Evolutionary Learning, 5(1), 1-37.

Gupta, A., Sindhu, R., Bansal, R., Sundaramoorthy, P., & Shrivastava, S. S. (2025, July). Exploring the Transformative Role of Generative AI and LLMs in Enhancing Conversational AI Systems Utilizing Mix Style Neural Network. In 2025 IEEE 4th World Conference on Applied Intelligence and Computing (AIC) (pp. 531-536). IEEE.

Hebbar, K. S. (2025). AI-driven real-time fraud detection using kafka streams in fintech. International Journal of Applied Mathematics, 38(6s), 770-782.

Hebbar, K. S., Sharma, V., & Maheshkar, J. A. (2026). Edge-AI microservice orchestration for private, real-time generative FinTech applications. Future Technology, 5(2), 13-24.

Ho, C. T., Ren, H., & Khailany, B. (2025, April). Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 39, No. 1, pp. 300-307).

Jangam, S. K. (2022). Self-Healing Autonomous Software Code Development. International Journal of Emerging Trends in Computer Science and Information Technology, 3(4), 42-52.

Liventsev, V., Grishina, A., Härmä, A., & Moonen, L. (2023, July). Fully autonomous programming with large language models. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 1146-1155).

Roberts, A., Iman, M. R. H., Bellone, M., Ghasempouri, T., Raik, J., Maennel, O., ... & Steinhorst, S. (2024, March). ADAssure: Debugging methodology for autonomous driving control algorithms. In 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 1-6). IEEE.

Sun, Z., Liu, Z., Zang, Y., Cao, Y., Dong, X., Wu, T., ... & Wang, J. (2025). Seagent: Self-evolving computer use agent with autonomous learning from experience. arXiv preprint arXiv:2508.04700.

Wasif, M., & Tunkel, D. (2025). Multi-agent collaboration in ai: Enhancing software development with autonomous llms. Preprint at https://doi. org/10.13140/RG, 2(31588.08328).

Weyns, D., Bäck, T., Vidal, R., Yao, X., & Belbachir, A. N. (2023). The vision of self-evolving computing systems. Journal of Integrated Design and Process Science, 26(3-4), 351-367.

Zhang, Y., Ruan, H., Fan, Z., & Roychoudhury, A. (2024, September). Autocoderover: Autonomous program improvement. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 1592-1604).