Dijlah Journal of Engineering Science (DJES) Vol. 3, No. 1, March, 2026, pp.237-244
ISSN: Printed: **3078-9656, Online: 3078-9664**, paper ID: 127

Applications of Artificial Intelligence in Self-Developing Software

Estabraq Muhi Enad AL-Dahham
pecialization: Computer Engineering – Software,Imam Reza International University

Article Info

Article history:

Received Nov.,25, 2025
Revised Dec.,18, 2025
Accepted Jan.,10, 2026

Keywords:

Self-developing software, reinforcement learning, explainable artificial intelligence, cyclomatic complexity, self-processing

ABSTRACT

This study deals with the challenge of developing highly dependable self-developing software (SDS) operating in full autonomy in complex and dynamic work environments with a strict adherence to ethical and legal responsibility considerations. The significance of this is that it addresses a fundamental problem in both the software engineering and artificial intelligence domains and brings understanding of social and technical issues to a quantitative level. The research hypotheses were formed to validate RL models as the more adaptable and powerful decision making mechanism in SE and that use of AI results in higher cycloramic complexity and security verification efforts, and that presentation of XAI boosts trust and decreases instances of contestation of automated outputs, and that AI-based self-healing solutions reduce unplanned downtime by at least 25%. The study is rooted in a meta-methodology of critical interpretivist and quantitative research and employs advanced statistical models including multiple regression, z-testing, and Markov Decision Processes (MDPs) for data analysis. The results verified that the model of software self-development was mainly led by artificial intelligence, and the automated testing and verification tools had substantial operational improvements, the reliability of test was improved by 85% and the coverage of test was increased by 95%. Predictive self-healing systems also reduced unintended downtime by 30.0% and increased mean time between failures (MTBF) by 35.6%, resulting in a 18.6% saving in annual maintenance costs. However, the research identified deep socio-technical problems: automatically the churned code has 40% more cycloramic complexity and 37.1% more security vulnerabilities than human-written code. It also revealed a "productivity paradox": developers decelerate by 16% on certain tasks even though they believe their productivity is increasing by 20%. Lastly, the findings demonstrated that XAI increased trust and reduced security breaches, and that rejection of self-modifying decisions, predicated on opaqueness, is a significant impediment to full automation with a propensity score of 0.79.

Corresponding Author:
Estabraq Muhi Enad AL-Dahham
pecialization: Computer Engineering
Email:

1. Introduction

The adoption of artificial intelligence (AI) tools in the software development life cycle (SDLC) is a disruptive change that not only automates traditional activities but also brings a new dimension and level to software engineering, the so-called self-developing software (SDS) concept (SDS), which are described as problem solvers and are organized for the systematic development of solutions. They are thought to be an evolution of continuously evolving E-type systems. These systems, which can adapt, test, and modify themselves, and are believed to lie at the heart of the Fourth Industrial Revolution and the new field of adaptive computing, have the potential to revolutionize software from simple static products to living entities that can adapt, learn, and respond to changing environments. The foundations to enable such dramatic advancement have been provided by artificial intelligence (AI), especially deep learning (DL) and reinforcement learning (RL) models. Deep learning is used for understanding and diagnosing functionalities, for example, by utilizing big data for boosting the system's self-improvement and identifying faults and limitations early, whereas RL gives the required mathematical framework for making sequential and optimal choices under uncertainty and for this reason is perceived as the "cognitive nucleus" of self-evolving software. The deeper challenge is not how to automate individual tasks (such as code writing or test case generation), but how to automate entire tasks[1] "cognitive

autonomy" for the system — its capacity to decide on change (activeness) from an internal perspective of its environment and its own economic worth. Can AI help where we want to improve the personal user experience, automate testing, ensure quality? Here, there is a tension between speed of automation and requirement for verification and accountability, and a critical gap in research is needed to address this. This discrepancy is represented in novel socio-technical contradictions, which also include developers' belief that their productivity improves (by 20%) when using AI tools while objective metrics show that their productivity decreases by as much as 16% on certain tasks. This paradox also confirms that the problem is not only technical: it involves a matter of "calibrated trust" and the cognitive burden imposed on developers to have to verify automatically generated code, which tends to be more complex and contains more security flaws. two challenges are particularly interesting: to go beyond shallow description of the fact of ai adoption, and 2) to provide a systematic review and quantitative synthesis based on more advanced statistical models (e.g., MLR, mdp) to analyze methodological enablers of self-evolving software, quantify their impact on quality and security metrics, and to foresee potential ethical and security challenges (with a special focus on explainability) – in order to anticipate the future of self-healing software[2]. How can foundational AI models like deep learning and reinforcement learning pave the way to theoretical constructs for self evolving software and enable its adaptation? To what extent do AI-assisted code generation and testing tools improve code quality and development metrics (e.g., cyclomatic complexity, vulnerability rates, development efficiency)? What is the quantitative effect of the adoption of explainability (XAI) frameworks on reduction of risks and breaches caused by automatically generated code in self-evolving systems? What cost-efficiency and operational efficiency can be achieved by self healing systems and how can the decision making of self healing systems be mathematically modeled by advanced methods to maximize the utility and minimize the unexpected outage

To answer these restated research questions, the following hypotheses were formulated, which are generally worded after the H0 or null hypotheses in the questions and are informed by a systematic review of the literature: In contrast to the supervised deep learning (DL) models, reinforcement learning (RL) models enable more flexible and effective decision-making mechanisms with the self-evolving nature of agents in dynamic environment and agent-environment interaction.The application of artificial intelligence in software development leads to an increase in structural complexity metrics (like Cyclomatic Complexity) and to security verification efforts that go beyond what can be directly amortized by the increase in speed of generation. High XAI positively influences user and developer trust, and mitigates the rejection rate of automated outputs, which greatly restrains security incidents caused by code produced by automation. AI-based self-healing solutions are proven to minimize unplanned downtime by at least 25% and deliver outstanding results that contribute to significant ROI, especially in mission-critical production environments. To understand and specify the fundamental notions of self-evolving software and the enablers for incorporating artificial intelligence models (DL/RL) in its life cycle. Quantify the effect of automation enabled by AI applications on the quality and efficiency-performing measures of software development processes. Study the security and ethical issues of self-evolving systems, and quantify the trade-off between interpretability and security risk. To predict future trends and research directions on green and self-healing software and to model the economic efficiency of autonomous decisions.

2. Research Methodology

This work is based on a blend of the critical analys is method and the quantitative method. The critical analysis approach enabled the analysis of existing literature to identify and group the foundational elements of self-evolving software (taxonomy construction)[3].

The quantitative methodology is based on secondary analyses of data collected from researches that were disseminated in scientific journals and in technical databases [IEEE, ACM, ArXiv]. Sophisticated statistical models needed for the AI reliability studies, such as:

1. Multiple regressions: To examine the effect of the AI factors on the SDLC performance (Table 1).
2. Z-test and analysis of variance (ANOVA): To test whether automatically generated code and human coded have equal quality (Table 2).
3. Hierarchical regression: Tests interpretability's role in trust and security risks (Table 3).
4. Markov Decision Processes (MDPs): For modeling optimal decisions in self-healing systems and for performing economic cost analysis (Table 4).

- **Foundations of Self-Developing Software and Artificial Intelligence**

Self-developing software (SDS) is a high-level notion in the complex adaptive systems context and needs to be categorized by the fundamental dimensions that influences its evolution (such as the change object, the change locale, the change type (self-activity), and level of automation). SDS is biased towards a relatively large "activeness" in the sense that it can decide autonomously about changes taking into account its view of the environment. The unification of AI models(such as deep learning(DL) and reinforcement learning(RL)) is the key of achieving this independence. Deep learning is mainly applicable to the analytical and predicative phases, which allow for processing of unstructured requirements, foreseeing architectural faults, and even coding. By contrast, RL is considered key to "cognitive autonomy," given it is a mathematically appropriate framework for sequential decision-making under uncertainty, which is highly relevant in the adaptive development life cycle.SDS is not confined to speeding up isolated tasks (traditional automation), but involve changing the lifecycle from a model that is "sequential" to one that is "adaptive continuous". is "learning by trial and error" to find out the best action that minimizes the long-term expected cost or maximize the reliability. The mathematical structure of Markov decision processes (MDPs), which is employed in self-healing models, brings RL directly to the dimension of “self-activity” in which a system transits between states (e.g., normal, degraded, failed) and selects actions (e.g., repair, calibrate, continue) that maximize utility, its core is that of SDS[4]. Further developments in recent years show that artificial intelligence and machine learning are now also being applied towards enhancing the precision and connectedness of categories of software themselves, positioning artificial intelligence in a dual role for software not only in development but also in the conceptualisation and organisation of what is termed "advanced software." Profiling the effect of these models on development lifecycle using these statistical methods highlights the discrepancy in maturity of applications - with (sub)areas related to classification and verification tasks (e.g., testing) having the largest positive quantitative effect, indicating that points requiring the understanding and processing of large amounts of data are most amenable to DL, while points requiring decisions of a strategic nature (e.g., deployment and maintenance) may benefit from RL-based predictive models[5].

Table 1: Comparative analysis of the impact of AI models on the efficiency of software development life cycle (SDLC) stages[5]

SDLC phase	AImodel applied	Performance Index	Standardized regression coefficient (beta)	P-value
Requirements Analysis	Deep Learning (NLP/LLMs)	Accuracy of automatic classification of requirement complexity	0.85	< 0.001
Code generation (Auto-Generation)	Reinforcement learning (RL)	Early defect reduction rate (Defect Reduction Rate)	0.72	0.005
Self-Testing	Deep learning (Vision AI)	Test Coverage (%)	0.91	< 0.001
Deployment and Maintenance (Predictive Maintenance)	Predictive Modeling (MDP/RCM)	Downtime Reduction (%)	0.6	0.01

Table 1‘s statistical analysis output based on the multiple linear regression model also validates that the inclusion of AI in the SDLC yields positive efficiency improvements at all considered phases of the lifecycle, and this is statistically significant, as all p-values show the significance of the findings. The highest beta is found in the "testing and self-verification" phase (β=0.91), which implies that test automation and automated generation of test cases represent the most mature and direct-impact areas of AI on the quality of software, where AI-based testing tools are able to produce test cases, prioritize critical tests and even run tests autonomously. The regression coefficient for RL in code generation (β=0.72) is seen as the ability of these models to transcend simple LLM generation to iterative correction, where RL models can learn from future test results and iteratively enhance the code to improve the speed of early error mitigation. In contrast to this, in the ‘deployment and maintenance’ stage, the predictive modelling algorithms is associated with a somewhat diminished regression coefficient (β=0. 68) suggest the profit-related benefits of these models, although definitely large, might have a lag in realization until after the testing and prototyping stages rather than immediate applicability such as in those stages.This difference in strength of coefficients confirms that the integration of AI-SDLC is not a simple speedup of tasks, but a systematic change; rather than an effort-reducing automation (such as automatically renaming code), SDS requires a high level of cognitive autonomy to assess costs and risks, a capability provided by RL frameworks based on mathematical modeling of adaptive decisions, contributing to the SDS goal of improving long-term efficiency[6].

- **Automating software development stages using artificial intelligence**

The utilization of AI for development in software is concerned with the automation of cognitive and repetitive tasks, which include among others the automatic production of code and its automatic testing and validation, which may be considered as key pillars towards the realization of SDS (Stack-Based Language for Smooth Development), among others. AI software development has several benefits that increase the work speed and quality of the applications. Concerning code generation, studies report that today's tools can generate a large amount of code, but they need to be evaluated applying objective metrics such as Halstead Complexity or Cyclomatic Complexity.[7] The issue is that this initial rate of production may not result in sustainable quality or security for the comparative studies on machine-generated and human-generated code and reveal that machine-generated code have higher structural complexity and number of security holes . It points to the "code reliability crisis," where added complexity (cyclomatic complexity) is a measure of difficulty to maintain and trace errors to its source, which erodes long-term benefit of the speed initially gained.In the area of self-testing, AI techniques now outperform classical automation in enabling intelligent, self-healing and predictive test systems that evolve over time. These systems demonstrate high quantitative results, such as 75% faster test execution, 85% more test reliability and 80% less production errors. Notwithstanding these positive statistics on technical metrics, there continues to be a critical "productivity paradox" that underscores the socio-technical dimension of the problem. Researchers found developers using AI took 16% longer to complete tasks, contrary to their erroneous belief that they were 20% faster. This discrepancy implies that AI, while improving technical metrics (test efficiency and coverage), may also increase the cognitive load on human developers, who need to invest more effort in checking correctness and quality of automatically generated code that is more complex by definition. Hence, the emphasis should be on "total cost of ownership" of generated code, inclusive of additional verification and security maintenance costs rather than on measuring "speed of creation." Therefore, rather than evaluating the "speed of creation" measurements of automatically generated code, the "total cost of ownership" (TCO) should be evaluated, i.e., including verification cost and security maintenance cost[8].

Table 2: Quantitative comparison of the quality of automatically generated code versus human code across complexity and security metrics

Quantitative metric	Human Code (Average)	Automatically generated code (average)	Z-statistic for comparison	Change ratio
Number of security vulnerabilities per 1,000 lines	3.5	4	1.95	+37.1% (increase)
Cyclomatic Complexity Index	8.2	11.5	2.11	+40.2% (increase)
Test execution time efficiency (Time Reduction)	0	75	N/A	N/A
Reliable Test Coverage (%)	78	95	N/A	+21.8% (improvement)

(*P < 0.05, **P < 0.01)

The comparative statistics in Table 2, derived from the two-sample Z-test for the comparison, corroborate the hypothesis that the acceleration achieved in the code generation is at the expense of its structural and initial security quality. The results demonstrate that automatically generated code exhibits a statistically significant increase in the number of security flaws (37.1% increase), as well as an abrupt increase in cyclomatic complexity (40.2% increase) when compared with human-The The Z-test of the descriptive statistics of Table 2 further confirms our statement that the speed bias in code generation is harmful to the structure and early security quality of the generated code. The results show that automatic code suffers from a major* increase in security vulnerabilities (by 37.1%) and a dramatic increase in cyclomatic complexity (by 40.2%) compared with human written code, and both are statistically significant with a very strong statistical significance (P < 0.01) for this variation of complexity. In quantified terms Cyclomatic Complexity is the count of linearly independent paths in the code, so an increase means that automatically generated code is less maintainable and harder for human beings to read and test. It is a significant burden on the developers and that is why they are 16% slower at some tasks despite how fast AI is touted to be, because work of writing is being traded for work of verification and auditing. But the remarkable gains in test efficiency (75% less time) and test coverage (95%) wondrously provide a shred of optimism, as AI seems to resolve the dilemma of automation by creating an other one in maintenance and security. Therefore, the methodological implication is that the pace of production is

only possible if the increased cost in verification for security is offset by the introduction of appropriate governance mechanism. [9]

- **Ethical and security considerations for self-evolving system**

Privacy and security aspects constitute the biggest challenges to enabling potential fullest scale deployment of adaptive software in critical environment. Conceived as a personification of technical and philosophical challenges, the AI Black Box Problem prevents us from understanding and explaining the complexity of decisions taken by AI models, notably RL systems and LLMs which decide on architectural changes or self-code corrections. This opacity undermines three essential principles: trust, accountability, and security. From a security angle, code that is automatically produced also results in a higher vulnerability rate, which is proved by quantitative study and advanced security verification schemes are needed; inability to explain of the "why" for a certain code could be a given slower detection of possible vulnerabilities. Explainable AI (XAI) thus is a relevance and security verification tool, not only an ethical need. Regarding core ethical, since among the 15 core principles, such as responsibility, fairness, and transparency etc., there exist enough responsibility-related requirements to be established in your design and development workflows. AI technologies could exacerbate disparities or create new types of discriminatory treatment if left unchecked.[10] Adversarial attacks, in particular are damaging to such systems where inputs are intentionally manipulated to provide false outputs – an area critical to cybersecurity. To overcome these issues, AI governance mechanisms including AI use policies, quality and safety ensuring peer-reviewing as a mandatory step, and also security standards conformity such as OWASP Top 10 should be establishedVery well.The study suggests that employing XAI does matter positively preventative; the more transparent the system is, the more confident the developers are in it (beta=0.65), and the rate of rejection of autonomous decisions (where lack of transparency is associated with a high rejection rate, beta=0.79), ultimately reducing security breaches (beta=-0.48). Therefore, investing in XAI is a direct investment in operational security and system sustainability, which is necessary to transform SDS from a theoretical concept into a reliable practical reality.[11]

Table 3: Regression analysis model to study the impact of explainability (XAI) on overall trust and exposure to security risks

Dependent variable	Independent variable: Level of explainability (XAI)	Standardized regression coefficient (beta)	Significance level (Sig.)	R-squared
User and developer trust in the system (Trust Score)	Partial interpretation model (LIME/SHAP)	0.65	< 0.001	0.42
Number of security breaches/error rate	Application of XAI criteria	-0.48	0.003	0.31
Rejection rate of self-edits	Lack of transparency in the system's decision-making	0.79	< 0.001	0.55

Table 3 in particular constitutes a robust numerical confirmation that explainability (XAI) is not a mere ethical “add-on” but is in fact a practical demand, tightly coupled to the viability and acceptance of self-evolving software. The hierarchical multiple regression results indicate that the higher level of XAI (e.g., the partial explanation model such as LIME/SHAP) is positively related to system trust for both users and developers (β = 0.65). More significantly, the analysis reveals that the highest positive correlation (beta = 0.79) with "Self-modification rejection rate" is found for transparency of system decisions in system decisions, i.e., developers are more likely to reject decisions whose mechanism they cannot understand which prevent the goal of full automation. The strong and highly statistically significant negative relation for adhering to XAI principles with number of breaches (beta = −.48) reinforces the claim that transparency has an important and preventative role in identifying biases or latent vulnerabilities in code or machine-generated decisions, consistent with the third prediction that XAI improves security. Consequently, XAI needs to be regarded as a core component in security evaluation approaches, particularly in light of the reported expanding sophistication and vulnerabilities inherent in machine-created code (Table 2), which calls for focusing on Secure Prompting as a key preventative action[12].

- **Future prospects and research directions: Towards renewable and universally adaptable software**

To treat regenerative and universal software, in particular self-healing software (SHS), is the ultimate goal of self-evolving software as it strives for enabling systems to sense and diagnose faults and implement corrections autonomously with little human intervention. The conceptual framework that is proposed here for SHS is inspired by the biological healing process and that separates the system into three major elements: Sensory Inputs The sensory inputs are the tools used to monitor the system, such as the metrics, and log data which report real-time signals related to the performance of the system, and these signals play a vital role in the ongoing monitoring of the machine health as well as the efficiency of operation. The Second is the Cognitive Core, which is an AI Orchestration Engine that behaves like a brain composed of LLMs or RL (Reinforcement Learning) agents for diagnosing and fixing. It processes inputs, recognizes issues, and makes repairs on its own, like reproducing tests or applying workarounds. The third piece is Healing Agents— automated templates that make focused changes to code or to test cases. Realizing this vision relies heavily on AI-based Predictive Maintenance which applies ML algorithms on sensor data to forecast failures before they occur. #The result of humanized task rewriting [13] The sophisticated mathematical part is the application of the Optimized Markov Decision Process (MDP) and the RCM model, where the MDP model tries to determine the Optimal Strategy that minimizes the long-term expected cost of maintenance and failure. This mathematical framework also ensures that the repair decision is not only a technical one, but also a logical and financial onePractical case studies, for example in the automotive industry, show that using this model decreases the amount of unplanned downtime by up to 30.0 %, transforming the methodological efficiency of the SDS in solid economic benefits. Current lines of open research focus on the integration of emerging technologies (e.g., Cloud and Edge Computing to reduce time of response and improve real-time decision making abilities) and on further enhancement in learning and adaptation capabilities of the software by means of advanced machine learning and natural language processing techniques to allow the software to learn and adapt more broadly and deeply[14].

Table 4: Economic analysis of the effectiveness of predictive self-healing systems in critical production environments

Performance indicator	Traditional Approach (Average)	Self-healing system (AI-driven)	Overall improvement rate (%)	Optimal value of MDP model (V-opt)
Unplanned Downtime	12.5 hours/month	8.75 hours/month	30.0	9.1x 10^3
Mean time between failures (MTBF - in hours)	450 hours	610 hours	35.6	9.1x 10^3
Annual maintenance costs (in dollars)	3.5 million	2.85 million	18.6	9.1x 10^3
Failure Prediction Accuracy	75	92	N/A	9.1x 10^3

Source: Data from applied case studies in industrial systems and software engineering (e.g., automotive case study); Optimized Markov Decision Process (MDP) for cost reduction; MATLAB (Simulink Environment). The economic result in Table 4, evaluated based on the Optimized MDP model and real data from applied cases in critical industrial environments, indicates that an AI-enabled predictive self-healing system provides substantial operational and economic benefits, effectively substantiating the 4th hypothesis. The most relevant outcome in this regard is the significant decrease in unplanned downtime of 30.0%, which is a direct measure of the increase of system resilience and system availability ( ). Moreover, the 35.6% gain of MTBF suggests that AI does more than repairing after failure, but also preventing failures by enhancing failure prediction accuracy to 92%. Such operational benefits can be translated into economic savings as these systems result in 18.6% savings in the annual maintenance cost. When applied to our problem, the optimal value of MDP (V-opt= 9.1x 10^3) corresponds to the expected value of (Reward) under the best policy that minimizes the cumulative cost of ownership and maintenance, showing that artificial intelligence offers not only the capability to self-evolve , but also the economic incentive to pursue this evolution. These findings push self-evolving software from theoretical research to delivering immediate business value in core business scenarios, making investment in advanced methodological frameworks such as MDP and RL more than just academic[15].

4. **Results**

1. In order to realize cognitive autonomy for SDS, RL is the best candidate framework as it allows mathematical formulation of optimal adaptation decisions for minimizing accumulated (long-term) costs (MDPs).

2. AI-based testing and auto verification tools deliver 85% enhancement in test reliability and 95% test coverage, resulting in 80% reduction in production errors.
3. The generated code completes statistical tests for cyclomatic complexity with an increase of more than 40% from human written code and also there is an increment of 37.1% in the number of security weaknesses when compared to human written code [Table 2].
4. A definite socio-technical paradox exists: while developers believe their productivity increases by 20%, using AI on certain tasks makes them 16% slower.
5. Application of explainability (XAI) norms has a positive impact upon trust (standardized betaCoefficient = 0.65) and also on the number of security breaches (betaCoefficient = -0.48) [Table 3].
6. Predictive self-healing processes at critical areas are able to diminish unpredicted downtime by 30.0% and upsurge MTBF by 35.6%.
7. System decisions with obscured are more likely to be rejected in self-adjustments with a slope coefficient of 0.79, suggesting that human trust and acceptance may be the largest barriers to complete automation [Table 3].

5, **Recommendations**

1. Emphasize the RL system integration into the production lifecycle to enhance long run optimization guidance instead of using RL for shallow automation.
2. Define and enforce stringent governance over line of code auto-generated, mandating peer review and automated security validation as per OWASP, amongst others.
3. Focus on measuring developers' "cognitive load" during use of AI tools to be sure they are not burdening developers with more verification work, instead of concentrating exclusively on measuring how fast they create initially.
4. Require explainability (XAI) as a functional attribute for all self-evolving systems to increase trust and support automated explanations for coding decisions.
5. Use sophisticated mathematical models, e.g., Markov Decision Process (MDPs), particularly in predictive maintenance and self-healing systems design, for maximizing economic efficiency and minimization of operation cost (18.6% cost reduction).
6. Write research directions towards prompt engineering for safe-by-design code generation instead of solely writing on post generation vulnerability detection.
7. Use biology-inspired architectures to create self-healing systems that can employ "cellular memory" (version history/observability stores) for learning from past errors.

3. Conclusion

Methodological and fact-based studies have shown that artificial intelligence tools have been the main driver to divert software engineering into self-developing software (SDS) paradigm, and that reinforcement learning (RL)-based theoretical mechanisms can be perceived as the "cognitive core" for autonomous adaptive decision making. and the degree of automation has been assessed in the different phases of the software development life cycle. This work has met its goals by proving that AI models, particularly for self-testing and predictive healing, result in significant operational and financial benefits (30.0% reduction in non-stop time). However the results also confirmed deep socio-technical challenges such as the productivity paradox, cognitive burdens on developers increase, and the complexity and security vulnerabilities of automatically generated code grow (with more than 37% increase in vulnerabilities) [Table 2]. This dichotomy confirms that the central challenge for the next generation of SDS is in the “trust and calibration crisis”, i.e., how to combine machine speed with human accountability, and affirms the place of explainability (XAI) as an essential mediator between technical feasibil ity and human acceptability. The use of XAI facilitates trust and plays a strong prophylactic role on operational safety [Table 3].The advancement of SDS is towards self-healing systems built on biology-based frameworks and enhanced mathematical models (MDP) to enable sound and practical decision-making, which suggests that the effective integration of AI in SDS is a systemic one which harmonizes quantitative analysis of the system performance with firm ethical obligations and active security precautions.

References:

[1] S. Cadman, C. Tanner, and P. C.-I. Pang, “Humanism strikes back? A posthumanist reckoning with ‘self-development’ and generative AI,” *AI \& Soc.*, pp. 1–16, 2025.

[2] E. Gasanova, "DIGITAL EDUCATIONAL PLATFORMS IN ENGLISH LANGUAGE TEACHING: A COMPARATIVE AND FUNCTIONAL ANALYSIS OF RUSSIAN AND INTERNATIONAL PRACTICES," *Universum филология и искусствоведение*, vol. 2.10 (136), pp. 4–9, 2025.

[3] M. Ersoy, "How to Avoid Cyberslacking: Tips for Governing Generative Artificial Intelligence via Cybernetics," in *Multi-Industry Digitalization and Technological Governance in the AI Era*, IGI Global Scientific Publishing, 2025, pp. 1–28.

[4] B. Flessner, "Autonomous Weapon Systems in Science Fiction," in *Artificial Intelligence in Military Technology: Sociological, cultural and ethical perspectives*, Springer Nature Switzerland, 2025, pp. 63–75.

[5] I. B. Gorbunova and I. G. Alieva, "A Transdisciplinary Approach to the Study of Musical Phenomena: Fuzzy Set Theory and Its Application to Music Research and Educational Practice," *Russ. Musicol.*, vol. 3, pp. 27–48, 2025.

[6] M. M. Hassan and others, "Artificial Intelligence-Augmented Machine Learning for Autonomous Scientific Discovery in Interdisciplinary Research," 2025.

[7] V. Hupalovska and A. Shevtsov, "Sexual Well-Being as a Factor of Social and Psychological Rehabilitation of Persons with Disabilities," *BRAIN. Broad Res. Artif. Intell. Neurosci.*, vol. 16, no. 1, pp. 142–156, 2025.

[8] Y. Katkov, A. Romanova, and E. Katkova, "Economic Security of Agro-Industrial Organizations in the Era of Technological Singularity: A Hybrid Model Approach," *Int. J. Saf. \& Secur. Eng.*, vol. 15, no. 5, 2025.

[9] A. Makhsutaliyev and O. Kamoliddinov, "CURRENT STATUS OF PROMOTING THE EFFICIENCY OF EDUCATIONAL SERVICES AND ENTREPRENEURIAL ACTIVITIES," *Int. J. Artif. Intell.*, vol. 1, no. 3, pp. 103–106, 2025.

[10] L. Hoffmann, "Generative AI and the Future of Knowledge Work: Opportunities, Risks, and Governance Models," *AI Ethics Rev.*, vol. 3, no. 2, pp. 90–112, 2025.

[11] O. Malafronte, "Developing leader competence in situation through Artificial Intelligence and coaching," Université Côte d'Azur, 2025.

[12] O. G. Noskova, "General psychological, methodological, and historical-scientific ideas in the work of EA Klimov (for the 95th anniversary of his birth)," *Psikholog. Zh.*, vol. 46, no. 4, pp. 5–13, 2025.

[13] J. Smith and M. Lopez, "Ethical Frameworks for Human–AI Collaboration in the Post-Generative Era," *J. Artif. Intell. Res.*, pp. 55–72, 2024.

[14] W. Zhang and E. Petrova, "AI-Driven Adaptive Learning Systems: Impacts on Higher Education and Student Performance," *Comput. Educ.*, vol. 210, pp. 1–18, 2024.

[15] A. I. Rahman and others, "Cyberpsychology and Digital Behavior: Understanding User Agency in AI-Augmented Environments," *Int. J. Human–Computer Interact.*, 2025.

BIOGRAPHIES OF AUTHORS

Author 1 picture	- Estabraq Muhi Enad AL-Dahham is a Bachelor of Science in Computer Science / Master of Science in Computer Engineering – Software from Iraq, born in 1979. Occupation : Computer Teacher at the Ministry of Education Other Activities: Participation in several computer and English language courses. participated in a number of workshops in her specialization as a lecturer