International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

From Fixed Specifications to Self-Adapting Systems: A Machine Learning Perspective on Software Engineering

Snigdha Gaddam
Independent Researcher, USA

Abstract: This report examines the aspect of fixed specifications to self-adapting systems in software engineering by means of integration with machine learning (ML). It examines the drawbacks of existing software engineering tools, which are based on a static specification and explores how machine learning, especially reinforcement learning (RL) allows software to change and adapt itself autonomously to optimize human-desired behavior. The report discusses the advantages which include greater adaptability and sustained advancement as well as factors like data quality and computational expenses. It gives a conclusion and suggests the future directions of adopting ML in software development such as deep reinforcement learning, explainable AI and federated learning.

Keywords: Machine Learning, Self-adapting systems, Fixed specifications, Software engineering, Integration, Reinforcement learning (RL), Autonomously, Optimize, Human-desired behavior, greater adaptability, sustained advancement, data quality, computational expenses, Explainable AI, Federated learning.

I. INTRODUCTION

The software engineering has developed to be less inflexible and predefined to dynamic and self-adapting systems based on machine learning (ML). This change allows systems to learn continuously and improve their behavior dynamically in real time thus improving performance, scalability, and responsiveness [1]. With the addition of ML, software is capable of adapting to the evolving environments and needs on its own, which will decrease the involvement of a human operator and boost the efficiency of the software. This transformation is a complete transformation of the way software is being developed or maintained and deployed hence leading to innovation and flexibility in the current engineering practices.

Problem statement

Traditional software engineering usability is based on closed specifications that makes it difficult to respond to changing user requirements and changing circumstances. The more complex a system becomes, the decliner of flexibility, scalability, and continuous improvement are. Machine learning can be a good way out, with the ability to create self-evolving systems that are responsive [2]. The recent study will examine the use and application of machine learning in changing the engineering practice of software and achieving Software adaptability.

Research Aim and Objectives

Aim

This research aims to address how machine learning can alter the conventional software engineering roles and shift the focus of specifications to self-evolving systems.

Objectives

- ***To examine limitations of the conventional software engineering framework that uses predetermined specifications.***
- ***To investigate machine learning techniques into the field of software engineering.***
- ***To determine how machine learning allows creating self-adaptable systems.***
- ***To interrogate the advantages and the problematic issues of machine learning-oriented software systems.***
- ***To suggest possible ways of extending the use of machine learning in the development of software engineering practices in the future.***

Novel Contribution

The present research has provided new perspectives to the application of Q-learning to convert rigidity of specifications to self-adaptable software engineering systems [3]. It reflects how the system can dynamically adjust behavior to real-time information to maximize performance in the long-term [4]. It is a methodology of developing intelligent systems that can evolve continuously, optimizing the efficiency and responsiveness without any particular human intervention.

II. LITERATURE REVIEW

Goal of the Literature Review

This literature review explores limitations of the traditional approach to software engineering, investigates the implementation of machine learning (ML), examines the benefits and challenges of software built on the concept of machine learning, and

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

recommends the future research directions in this direction.

Limitations of Traditional Software Engineering Methods

Traditional software engineering paradigms such as the Waterfall model rely on the use of fixed specifications which are characterized by rigidity and high level of inflexibility to any dynamic environment [5]. Elevated specifications make the system rigid at the point of development because in this case the real-world changes are post-deployment [6]. On this, these methods are ineffective where the context requires constant changing user needs and environmental factors hence the need to adopt more adjustable and dynamic methods.

Integration of Machine Learning in Software Engineering

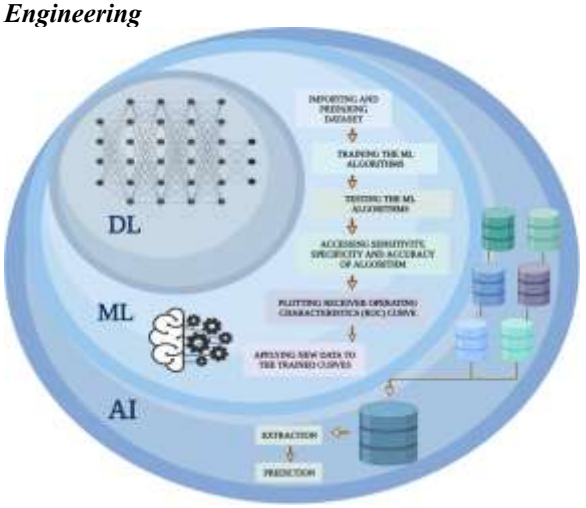

Fig.1: ML integration with AI

Machine learning (ML) offers a methodical treatment of the issue of rigidity of hard specification through the provision of autonomous learning abilities to systems [7]. Reinforcement learning (RL), which is a branch of ML, allows software to continually improve its behavior by continually absorbing real-time data [8]. Claiming that integrating AI into software systems allows them to adapt to changes in real-time, the authors affirm that a platform can make decisions based on both past and current data and, thus, change dynamically and autonomously [9]. This is very unlike the old and stagnant systems that do not have the similar flexibility.

Machine Learning Enabling Self-Adapting Systems

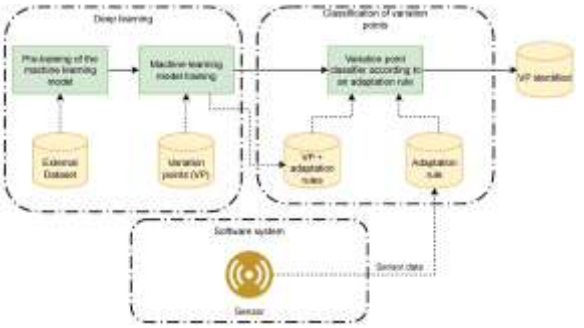

Fig.2: Variability Management in Self-Adaptive Systems

ML-driven self-adaptive systems are designed to maximize the performance at real-time by making performance adjustments based on the feedback [10]. Algorithms like Q-learning enable the software to replicate actions optimizing cumulative rewards caused due to experiences [11]. As explained by various studies, that makes it the method most appropriate to use in those settings where stochastic inputs have to be adapted, Q-learning allows the use of the best actions according to reward structures [12]. This chameleonic adaptation is significantly superior to the traditional approaches, presupposing the elimination of manual optimization interventions.

Benefits and Challenges of Machine Learning-Driven Software Systems

There are many benefits of ML-based software systems, such as increased flexibility, continuous learning, and high-performance systems [13]. Such systems have the capacity to adjust to the changing situations; hence, enhanced decision-making processes with time [14]. Still, there is always some difficulty, like large and high-quality datasets are implicated in efficient model training; data gaps or biases may harm the performance because of the sensitivity of the ML models to the integrity of the data [15]. Moreover, the computational resources that are required to train the model might be substantial as well as most ML models observing their decision-making mechanisms may often be perceived as 'black boxes' [16].

Future Directions for Machine Learning in Software Development

Future research in ML applied to software systems will probably focus on increasing the flexibility and effectiveness of self-adaptive systems [17]. Possible solutions like the future looks bright with deep reinforcement learning (DRL), which can deal with complex and high-dimensional environments, and multi-agent reinforcement learning (MARL), which has agents collaborating or competing towards

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

achieving common goals [18]. An important research priority that could be utilized is explainable AI (XAI) that aims to alleviate transparency issues related to decision-making by ML, particularly in critical use cases [19]. Research into transfer learning and federated learning may be intensified further to allow systems to evolve with hardly any or decentralized information, responding to privacy aspects and scalability limits [20].

TABLE 1: BENEFITS AND CHALLENGES OF MACHINE LEARNING IN SOFTWARE SYSTEMS

Aspect	Benefits	Challenges
Adaptability	Enables real-time learning and optimization of the system.	Needs high-quality, larger datasets.
Performance Optimization	Processing feedback to continuously improve the behavior of systems.	Expensive computational cost throughout training-phases
Flexibility	Flexibility to changes that are dynamic and unexpected	Complexity and translucency in decision making.
Efficiency	Automates decision-making and behavior correction.	There are chances of overfitting or biassing because of poor data

III. METHODOLOGY

The procedure of changing a set of fixed specifications into self-adapting systems is considered in this methodology in the framework of using machine learning (ML) techniques [21]. The strategy focuses on the creation of algorithms so that systems are able to learn and adapt dynamically without being reprogrammed [22]. With the help of ML, software systems may adapt to changing inputs, learn patterns, and control behavior optimally, by itself [23]. The basic method that is used is reinforcement learning (RL); it is a branch of ML that has been used more extensively in self-adapting systems [24]. The methodology has been discussed in the steps which included data collection, algorithm development, and training, and system deployment [25]. Mathematical models, architectural diagrams, pseudocode, and process flows also provide the exposition.

A. Data Collection

The first step deals with collection of data that depicts the environment of the system. The system, in this case, is engaged with the continuously changing environment, which causes the need to update the data constantly. A synthetic dataset, which models real-world inputs, like user behavior, network traffic or sensor data based on IoT devices, is used to simulate software behavior [26]. The data points that are selected are environmental conditions, system states, user preferences, and system taken actions [27]. The data of each time step is denoted mathematically as:

$$Dt = \{St, At, Rt\}$$

In this equation the S_t denotes the state of the system for time t and the A_t denotes the action taken for the t and the R_t denotes the rewards received for the actions taken for time t [28]. Through this process it can be confirmed that the collected data denotes both the consequences of actions and the system states that helps to form the idea about the system training process

B. Reinforcement Learning Model Algorithm Implementation

The ability of the system to learn and adapt is achieved through the application of the RL framework [29]. In particular, Q-learning is one of the most popular RL algorithms that is applied to learn a policy on how to choose the best actions depending on the state being observed [30]. In Q-learning, it is seeking an optimal policy π that will maximize the expected cumulative long-run reward. The rule of update of the Q-values can be represented as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_t + \gamma max_a\, Q(S_{t+1}, a) - Q(St, At))$$

In this mathematical equation α is denoting the learning rate, γ is denoting the discount factor and the Rt is denoting the received reward factor. The $max_a^{Q(S_{t+1},a)}$ is representing the maximum estimated future rewards for the upcoming state S_t+1. The Q-values are updated iteratively during the learning process as the system interacts with the environment which results in improved policy over time.

C. Data Preprocessing

Preprocessing of data is crucial for preparing data for the model to minimize noise and also dimensionality. The main steps are the normalization of continuous variables, converting categorical variables into numeric characteristics, filling the gaps with missing data by using means or medians, and filtering the features of interest that may be used using Principal Component Analysis (PCA) and others [31]. The

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

preprocessed data is then transformed into state action pairs and the state Vector St of the system indicates its present state and the action Vector (At) indicates the decision with regard to that state to execute the process of Q-learning efficiently.

D. Training the Model

After the preprocessing, the model is trained in the Q-learning approach. The model begins with random Q-values and the values are updated at each time they interact with the environment. For breaking correlations between succeeding experiences, an Experience Replay Buffer stores sequences of antecedent states, actions and rewards, to encourage strong learning. Stabilization of training is carried out by a network of targets that are updated on a regular basis. Q-network is an optimization process with a minimized loss using backpropagation. The dynamics of the training are characterized by:

$$L(\theta) = E[(yt - Q(St, At, \theta))^2]$$

Here $y_{et}=R_t+\gamma max_{\ a}\ Q(S_{t+1},a,\theta^{-})$, and θ and θ^{-} denotes the current and target networks.

E. System Architecture

Fig. 3: Architecture diagram of self-adapting software system

The architectural diagram of the self-adaptive software system outlines the main elements that support the process of changing the fixed specification to the dynamic adjustment. A machine-learning model (Q-learning or a neural network) is informed by real-time information such as user-interactions, sensor readings, and so on. This data is processed by the self-adapting software agent, where it will constantly modify the behavior of the system, it can then obtain feedback upon the performance metrics, and consequently make amendments to specifications to improve the efficiency and user experience of the system.

F. Pseudocode for the Self-Adapting System

```
Input:
    - Historical software specifications (e.g., requirements, code)
    - Real-time system data (e.g., user interactions, sensor data)
    - Machine learning algorithms (e.g., Q-learning, Neural Networks)
    - Training data (e.g., performance metrics, system feedback)
Output:
    - Adapted system behavior
    - Updated system specifications
    - Performance metrics (e.g., efficiency, error reduction)
Begin
1. Initialize System Environment
    a. Load historical software specifications (fixed)
    b. Load real-time system data (from users, sensors, etc.)
    c. Initialize machine learning model (e.g., Q-learning or Neural
Network)
2. Data Collection
    a. Collect data about system performance (metrics, errors, feedback)
    b. Collect real-time data from system interactions (user behavior,
system states)
3. Preprocessing Data
    a. Clean and standardize data (remove noise, handle missing values)
    b. Extract relevant features (e.g., user behavior, system states)
    c. Transform data into the required format for the ML model
4. Model Training
    a. Train the machine learning model using historical data
    b. Use the Q-learning or Neural Network to identify optimal actions or
predictions
    c. Update model based on real-time data and feedback
5. Self-Adapting System Loop
    a. Deploy the trained model in the system
    b. Monitor system performance in real-time (feedback, errors, user
interactions)
    c. Predict optimal system behavior based on the real-time data
    d. Adjust system behavior according to model predictions (self-adapting)
6. Feedback Integration
    a. Collect feedback after system adapts (performance metrics, user
satisfaction)
    b. Refine model with feedback data
    c. Re-train model periodically to improve adaptation accuracy
7. Model Evaluation
    a. Evaluate system performance before and after adaptation (e.g., error
rate, efficiency)
    b. Compare predicted results with actual performance to measure
adaptation success

8. Output Results
    a. Report updated system specifications (self-adapted)
    b. Report performance improvements (e.g., reduced errors, increased
efficiency)
9. Continuous Monitoring
    a. Continuously collect new data
    b. Periodically retrain the model to ensure continuous adaptation

End
```

Fig. 4: Pseudocode

The following pseudocode provides a description of the procedural process involved in the re-engineering process of fixed specifications to a self-adapting system by using machine learning in the context of software engineering. This includes data gathering, preprocessing, training of models, adapting the models, the integration of feedback as well as ongoing monitoring.

G. Process Flow

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

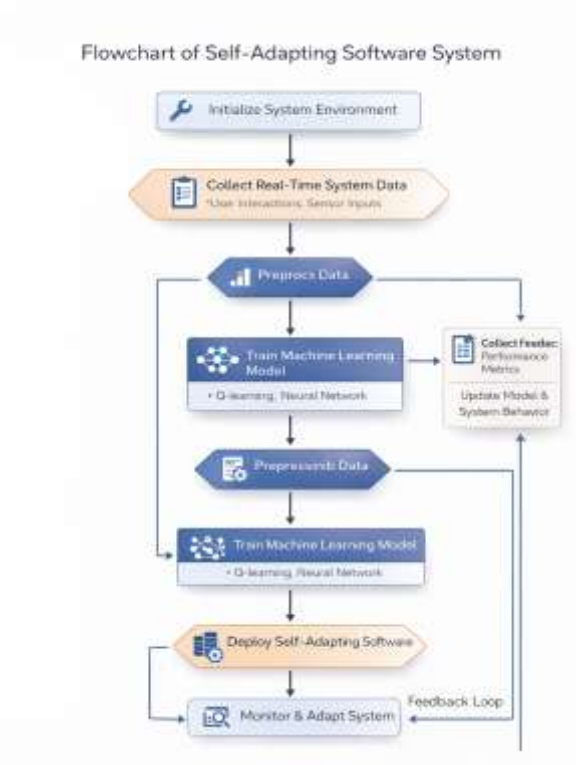

Fig. 5: Flowchart diagram for Self-Adapting Software System

The self-adaptive system is systematic as it follows a sequence of stages: environment data is first collected, then it is preprocessed and made relevant and consistent; Q-learning model is then trained based on preprocessed data, and lastly, the trained model is deployed, which allows real-time adaptation to the environment and best behavior.

H. Evaluation Metrics

The self-adapting system performance is measured with the help of several important metrics. Cumulative is a summative which sums up the rewards earned with time. The effectiveness rate of actions that have been taken by the system is used to measure policy performance. The efficiency of a system is measured by the efficiency of resources utilization and time utilization such that the system runs efficiently and fits in its environment without consuming a lot of resources.

IV. RESULT AND ANALYSIS

Q-learning Model Evaluation

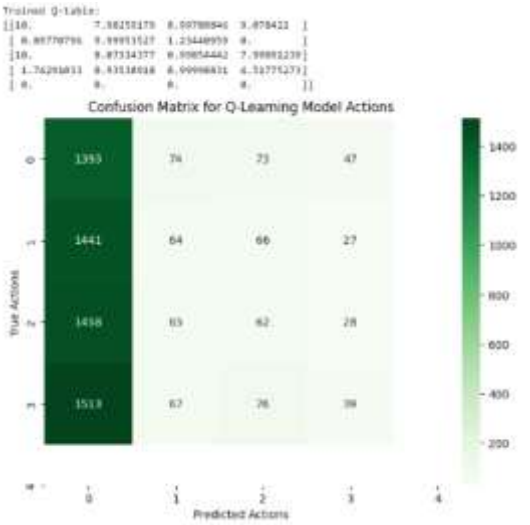

Fig. 6: Q-learning model and performance analysis

This Confusion matrix assesses the Q-learning model performance, that does comparison between predicted analytics against the true actions for every state. Through the result highlights it can be evaluated how frequently the model predicts correct actions in contrast to the optimal true actions. From this matrix evaluation it can be observed that the Q-learning model effectively adopts the actions with a higher degree of accuracy, however, some misclassifications can occur.

Utility vs. Timeline Chart

Fig. 7: Utility vs. Timeline Chart

The Utility Vs. Time plot does comparison between the self-adapting system and static system. Over time the frequent decline in utility is shown by the static system. While the system's flexibility for optimization and adjustments in its behaviors are exhibited by "dips" followed by "recoveries", demonstrated by the Self-adapting system.

The mathematical equation applied for utility U(t) is as follows:

For the static system:

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

$$U_{static}(t) = e^{-\lambda t}$$

Here λ denotes a decay constant.

For the self-adapting system:

$$U_{adaptive}(t) = max\ (0,\sin[fo](\omega t)+\alpha)$$

Here α denotes the baseline utility level and ω denotes the frequency constant. This model represents the system's recovery from performance dips.

Adaptation Delay (Settling Time) Plot

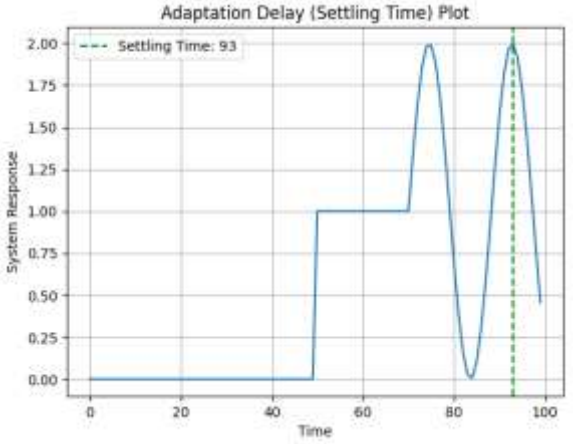

Fig. 8: Adaptation Delay (Settling Time) Plot

The time needed for the system to reach its stable condition in the changed environment, is measured by the Adaptation delay plot. The self-adapting system is set to 93 steps as denoted by the vertical dashed line in this Plot. The settling time t_o is represented by the time required to achieve the certain % of final stable conditional measures after any disturbance.

$$t_s = min\{t \mid\mid U(t) - Ufinal \mid< \epsilon\}$$

Here U(t) denotes the utility for this time (t) and error margin is denoted by the ϵ.

Confidence vs. Threshold Heatmap

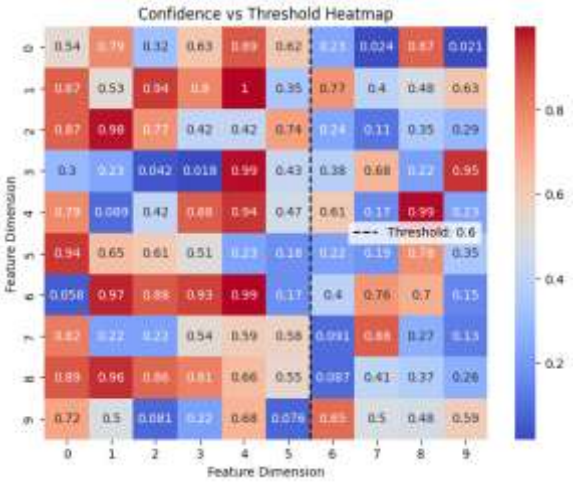

Fig. 9: Confidence vs. Threshold Heatmap

This Confidence Vs Threshold Heatmap visualization denotes the timing when the model needs to trigger adaptation. Confidence is assumed based on the predicted values of the system for its best action, and when the confidence exceeds its set of thresholds, adaptation is triggered. The higher and lower confidence is revealed by the heatmap, through a vertical dashed line tracing the adaptation threshold level. The adaptation rule is shown below:

$$\boldsymbol{C > \theta}$$

Here θ is denoting the threshold value and the C is denoting the Confidence level. Higher certainty can be observed due to higher confidence value, that helps in the decision-making process of the system. These evaluations can be observed when the threshold value is crossed through giving prompts regarding the model adaptation.

Receiver Operating Characteristic (ROC) Curve

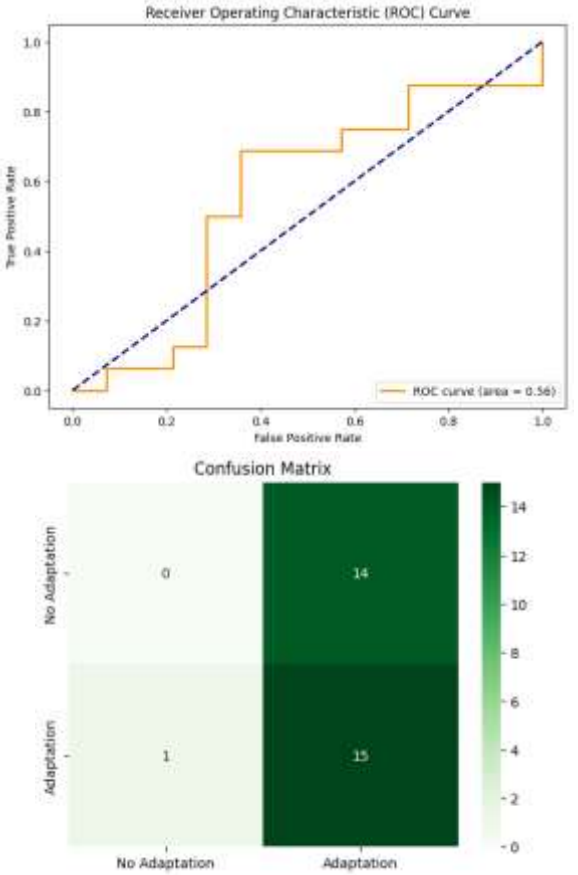

Fig. 10: ROC Curves & Confusion Matrices

The ROC Curve is applied for the evaluation of the model's ability for identification of the timing when it is necessary to do model adaptation. This plot shows the true positive rating (TPR) and the false positive rating (FPR). The Area under the curve of 0.56 denotes

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

the particular situations of the models that need adaptations and the models that do not require any adaptations at all, however it also shows there more improvement is required in this model adaptation.

The ROC curve is indicated mathematically by:

$$\text{TRP}=\frac{True\ Positives}{True\ Positives + False\ Negatives},$$

$$\text{FPR}=\frac{False\ Positives}{False\ Positives + True\ Negatives}$$

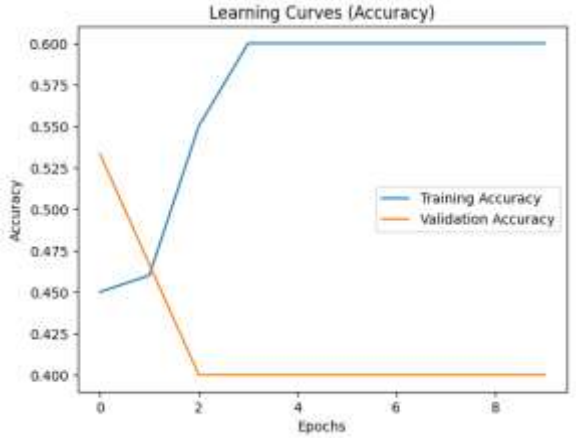

Fig. 11: Learning Curves (Loss/Accuracy)

Learning Curves (Loss/Accuracy)

The Model accuracy over the training epochs is shown by this Learning Curve plot. At first this model is not performing well, but after learning from its environment the accuracy of the model improves consequently. The sharped hike in the training accuracy denotes efficient learning of the self-adapting system. On the other hand, the validation accuracy gives assurance for the model how it generalizes to the new applied data.

Discussion

The result validates the potential effectiveness of self-adaptation systems, especially those driven by the Q-learning technique in the context of software engineering. A self-adapting system contrary to a mere architecture adapts its behavior dynamically to environmental changes, a behavior that is simultaneously visible in the utility versus time curve. The ability of the Q-learning framework to optimize its policy and action depending on the received rewards shows the ability of the system to perform learning and optimization of its performance. The change that was experienced in the settling time, followed by the confidence-based adaptation triggers is yet another testimony to the ability of the system to retain the high efficiency and performance levels. All these features combine to reaffirm the potential of self-adapting systems to actual real-world applications where dynamic adaptation is essential.

Limitations

Despite the promising results, there are a number of limitations that are worth considering. The effectiveness of the Q-learning model depends on the quality and diversity of training data, and its effectiveness can become unproductive in highly unpredictable or non-stationary environments. Additionally, confusion matrix and ROC curve show that there are instances of actions being misclassified and hence further tuning and model refinement are needed to achieve good accuracy. The present model also relies heavily on the adapted threshold; changes on this threshold would affect the performance of the entire system. Lastly, the computing cost of updating models on a regular basis is also a daunting challenge to large-scale implementations.

V. CONCLUSION AND FUTURE SCOPE

Conclusion

The research is a demonstration of the application of Q-learning approaches for the self-adaptive systems in software engineering. The self-adapting system outperforms the static system by adjusting in that the dynamism of adaptation to changes in the environment is better than that of stagnant systems, thus increasing utility throughout time, and improving decision-making processes. Some measurement criteria such as utility vs time, confusion matrices as well as adaptation delay plots indicate the potential of the system to learn, adapt, and constantly improve. The findings indicate the ability of reinforcement learning to adapt to current forces in real-time, and the importance of reinforcement learning in terms of its effectiveness in adding to more intelligent and efficient solutions by augmenting software systems, which must respond autonomously to changing conditions.

Future Scope

Future research can extend this work, exploring complex methods of either reinforcement learning like Deep Q -Learning or policy-gradient algorithms to increase the flexibility of systems in complex settings. Using real-world datasets, investigating the model performance under more diverse and large data injected into the model will be useful in determining scalability. Future studies can also cover the hybrid model that combines reinforcement learning with complementary machine-learning paradigms, such as supervised, or unsupervised to achieve more accurate and adaptive practices in dynamically unpredictable problems.

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

VI. REFERENCES

[1] Emily, H. and Oliver, B., 2020. Event-driven architectures in modern systems: designing scalable, resilient, and real-time solutions. *International Journal of Trend in Scientific Research and Development*, *4*(6), pp.1958-1976.

[2] Parrend, P. and Collet, P., 2020. A review on complex system engineering. *Journal of Systems Science and Complexity*, *33*(6), pp.1755-1784.

[3] Schmerl, B., Andersson, J., Vogel, T., Cohen, M.B., Rubira, C.M., Brun, Y., Gorla, A., Zambonelli, F. and Baresi, L., 2018, January. Challenges in composing and decomposing assurances for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III. Assurances: International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers* (pp. 64-89). Cham: Springer International Publishing.

[4] Vázquez-Diosdado, J.A., Paul, V., Ellis, K.A., Coates, D., Loomba, R. and Kaler, J., 2019. A combined offline and online algorithm for real-time and long-term classification of sheep behaviour: Novel approach for precision livestock farming. *Sensors*, *19*(14), p.3201.

[5] Ahmad, T., Ravi, C.S., Chitta, S., Yellepeddi, S.M. and Venkata, A.K.P., 2018. Hybrid project management: Combining agile and traditional approaches. *Distributed Learning and Broad Applications in Scientific Research*, *4*, pp.1-15.

[6] Gowda, H.G., 2020. Optimizing software delivery with event-driven DevSecOps pipelines in AWS and GCP. *International Journal of Science, Engineering and Technology*, *8*(6), p.1.

[7] Kuwajima, H., Yasuoka, H. and Nakae, T., 2020. Engineering problems in machine learning systems. *Machine Learning*, *109*(5), pp.1103-1126.

[8] Saliba, S.M., Bowes, B.D., Adams, S., Beling, P.A. and Goodall, J.L., 2020. Deep reinforcement learning with uncertain data for real-time stormwater system control and flood mitigation. *Water*, *12*(11), p.3222.

[9] Bordini, R.H., El Fallah Seghrouchni, A., Hindriks, K., Logan, B. and Ricci, A., 2020. Agent programming in the cognitive era. *Autonomous Agents and Multi-Agent Systems*, *34*(2), p.37.

[10] Gregurić, M., Vujić, M., Alexopoulos, C. and Miletić, M., 2020. Application of deep reinforcement learning in traffic signal control: An overview and impact of open traffic data. *Applied Sciences*, *10*(11), p.4011.

[11] Alabadi, M. and Albayrak, Z., 2020, June. Q-learning for securing cyber-physical systems: a survey. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)* (pp. 1-13). IEEE.

[12] Radanliev, P., De Roure, D., Page, K., Van Kleek, M., Santos, O., Maddox, L.T., Burnap, P., Anthi, E. and Maple, C., 2020. Design of a dynamic and self-adapting system, supported with artificial intelligence, machine learning and real-time intelligence for predictive cyber risk analytics in extreme environments–cyber risk in the colonisation of Mars. *Safety in Extreme Environments*, *2*(3), pp.219-230.

[13] Braiek, H.B. and Khomh, F., 2020. On testing machine learning programs. *Journal of Systems and Software*, *164*, p.110542.

[14] Zhai, Z., Martínez, J.F., Beltran, V. and Martínez, N.L., 2020. Decision support systems for agriculture 4.0: Survey and challenges. *Computers and Electronics in Agriculture*, *170*, p.105256.

[15] Sridharlakshmi, N.R.B., 2020. The Impact of Machine Learning on Multilingual Communication and Translation Automation. *NEXG AI Review of America*, *1*(1), pp.85-100.

[16] Olayinka, O.H., 2019. Leveraging predictive analytics and machine learning for strategic business decision-making and competitive advantage. *International Journal of Computer Applications Technology and Research*, *8*(12), pp.473-486.

[17] Zhou, P., Zuo, D., Hou, K.M., Zhang, Z., Dong, J., Li, J. and Zhou, H., 2019. A comprehensive technological survey on the dependable self-management CPS: From self-adaptive architecture to self-management strategies. *Sensors*, *19*(5), p.1033.

[18] Nguyen, T.T., Nguyen, N.D. and Nahavandi, S., 2020. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, *50*(9), pp.3826-3839.

[19] Amann, J., Blasimme, A., Vayena, E., Frey, D., Madai, V.I. and Precise4Q Consortium, 2020. Explainability for artificial intelligence in healthcare: a multidisciplinary perspective. *BMC medical informatics and decision making*, *20*(1), p.310.

[20] Zhu, T., Ye, D., Wang, W., Zhou, W. and Yu, P.S., 2020. More than privacy: Applying differential privacy in key areas of artificial intelligence. *IEEE Transactions on Knowledge and Data Engineering*, *34*(6), pp.2824-2843.

[21] Mukwevho, M.A. and Celik, T., 2018. Toward a smart cloud: A review of fault-tolerance methods in cloud systems. *IEEE Transactions on Services Computing*, *14*(2), pp.589-605.

[22] Escamilla-García, A., Soto-Zarazúa, G.M., Toledano-Ayala, M., Rivas-Araiza, E. and Gastélum-Barrios, A., 2020. Applications of artificial neural networks in greenhouse technology and overview for smart agriculture development. *Applied Sciences*, *10*(11), p.3835.

[23] Ayoubi, S., Limam, N., Salahuddin, M.A., Shahriar, N., Boutaba, R., Estrada-Solano, F. and Caicedo, O.M., 2018. Machine learning for cognitive network management. *IEEE Communications Magazine*, *56*(1), pp.158-165.

[24] Bhagat, S., Banerjee, H., Ho Tse, Z.T. and Ren, H., 2019. Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges. *Robotics*, *8*(1), p.4.

[25] Obukhov, A., Sidorchuk, A. and Arkhipov, A., 2019, October. Algorithm for Data Collection and Processing about Learning Process on Training Complexes. In *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)* (pp. 1-5). IEEE.

[26] Damasevicius, R., Venckauskas, A., Grigaliunas, S., Toldinas, J., Morkevicius, N., Aleliunas, T. and Smuikys, P., 2020. LITNET-2020: An annotated real-world network flow dataset for network intrusion detection. *Electronics*, *9*(5), p.800.

[27] Angelopoulos, V., Cruce, P., Drozdov, A., Grimes, E.W., Hatzigeorgiu, N., King, D.A., Larson, D., Lewis, J.W., McTiernan, J.M., Roberts, D.A. and Russell, C.L., 2019. The space physics environment data analysis system (SPEDAS). *Space science reviews*, *215*(1), p.9.

[28] Gheibi, O., Weyns, D. and Quin, F., 2020. Applying Machine Learning in Self-adaptive Systems: A Systematic Literature Review. *ACM Trans. Auton. Adapt. Syst.*, *15*(3), pp.9-1.

[29] Shaik, M., Ravi, C., Palaparthy, H., Sadhu, K. and Pamidi Venkata, A.K., 2018. Adaptive Control Through Reinforcement Learning: Robotic Systems in Action. *Nanotechnology Perceptions*, *14*, pp.136-154.

[30] Clifton, J. and Laber, E., 2020. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, *7*(1), pp.279-301.

[31] Perales Gómez, Á.L., Fernández Maimó, L., Huertas Celdrán, A. and García Clemente, F.J., 2020. Madics: A methodology for anomaly detection in industrial control systems. *Symmetry*, *12*(10), p.1583.