ADRL-CostNet: A Deep Reinforcement Learning Framework for Continuous and Self-Adaptive Software Cost Estimation in Agile Environments

Debanjan Ghosh[1], Dr. Arvind Kumar Pandey[2]

[1]Dept. of Computer Science & IT, School of Engineering & IT, Arka Jain University Jamshedpur, India
debgh4@gmail.com / debanjan.g@arkajainuniversity.ac.in
[2]School of Engineering & IT, Arka Jain University, Jamshedpur, India, dr.arvind@arkajainuniversity.ac.in

Corresponding Author
Debanjan Ghosh

ABSTRACT
Static prediction models producing one estimate at the start of a project have frequently been employed to estimate software expenses. In Agile settings, where demands, team performance, backlog volume, and technical debt constantly evolve over time, such techniques are sometimes insufficient. Addressing this problem, this work suggests an
Adaptive Deep Reinforcement Learning–based Cost Estimation Framework (ADRL-CEF), which views the software development process as a changing and developing system. Based on present project circumstances, the framework employs a learning agent that revises cost-to-complete predictions over sprints. The estimating process considers key factors including sprint velocity variability, requirement volatility, defect density, backlog progress, and technical debt. Deep Reinforcement Learning approaches, including Deep Q-Network (DQN) and Advantage Actor–Critic (A2C) are used to carry out the agent, which then enhances its forecasts via repetitive learning directed by accuracy-based input. Real-world Agile project information and a controlled synthetic dataset were used to assess the suggested method. The outcomes reveal that ADRL-CEF outperforms conventional regression and machine learning models, reaching around an 18–32% reduction in mean relative error (MRE). These results imply that in agile environments reinforcement learning offers a more adaptable and flexible solution for cost estimating. The framework generally encourages ongoing estimation and provides a useful strategy for increasing accuracy in settings when project circumstances are constantly changing.

Keywords—*Software Cost Estimation, Deep Reinforcement Learning, Agile Project Management, Deep Q-Network (DQN), Actor–Critic (A2C), Dynamic Cost Prediction.*
How to cite this article: Ghosh D, Pandey AK. ADRL-CostNet: A Deep Reinforcement Learning Framework for Continuous and Self-Adaptive Software Cost Estimation in Agile Environments. Int J Drug Deliv Technol. 2026;16(35s): 777-786. DOI: 10.25258/ijddt.16.35s.88

INTRODUCTION:

Accurate assessment of software development cost has always been a key driver in the success of software projects. Dependable projections help with wise decision-making, resource management, and efficient budgeting. Projects often suffer delays, poor resource utilisation, and higher risk when estimates are incorrect. Early studies in software engineering concentrated on creating organised estimation techniques able to offer trustworthy predictions under rather constant conditions. Because of their methodical and data-driven character, approaches such as COCOMO, Function Point Analysis, and Use Case Points found extensive acceptance. Usually using predetermined criteria, past data, and set assumptions about project circumstances, these approaches depend on historical data.

But the nature of software development has dramatically changed in recent years. Stable and expected patterns in contemporary projects are rare. Particularly with the widespread acceptance of agile, iterative and incremental approaches, cost estimation is an ongoing process rather than a one-time operation. Agile development brings about regular changes in system design, priorities, and needs, so complicating project paths and rendering them less predictable. As a result, classic estimation approaches, which presume fairly constant inputs and linear correlations, have difficulties keeping precision in such situations.

Furthermore, compounding this difficulty is the increasing sophistication of software systems. Often projects include linked components, distributed development teams, and fast-changing technologies. These elements cause variability and uncertainty

that fixed mathematical models struggle to represent. Therefore, there is a growing demand for estimation techniques that can change with circumstances and depict a project's real-time state. Especially agile approaches call for a different viewpoint on cost estimation. Unlike conventional models that emphasise upfront planning, Agile concentrates on ongoing development and iterative delivery. Requirements might change over the course of the project; new jobs might surface at any stage. Teams have to constantly reevaluate their goals, ability, and progress. Productivity might also be affected by changes in team structure, experience levels, and availability. Furthermore, influencing present and future expenses are changing over time variables like defect rates, rework effort, and technical debt.

Treating cost estimating as a one-time endeavour is insufficient in such a volatile environment. Estimating should instead be ongoing, flexible, and reactive to evolving project modifications. In this respect, conventional models have drawbacks since they need recalibration to represent changed circumstances. Approaches based on machine learning provide more flexibility, yet they sometimes rely on retraining with revised datasets—which can be time-consuming and challenging to incorporate into quick Agile workflows. Furthermore, models trained on historic data might not accurately catch developing trends in new projects.

These constraints emphasise the need for a fresh method for software cost prediction that may continuously update its forecasts and learn from changing project data. Creating such adaptive and self-improving estimating methods is vital for matching cost prediction with the realities of modern Agile software development. A practical means of addressing the adaptive challenges seen in Agile cost estimation is reinforcement learning (RL). Unlike supervised learning techniques depending on labelled data, RL aims to acquire decisions via interaction with an environment. An RL agent raises its performance by getting rewards or penalties determined on the results of its activities. The model can adapt to changing circumstances independent of set input–output interactions thanks to this trial-and-feedback mechanism.

By employing deep neural networks to represent value functions and decision rules, deep reinforcement learning (DRL) builds upon this concept. In fields needing real-time decision-making under uncertainty—robotics, autonomous systems, financial forecasting—DRL has been used with great success. These traits match with agile software development, in which project conditions—such cost, scope, and risk—alter over time. A DRL-based model can modify its predictions as teams go over backlog items or hone demands. The model slowly develops its knowledge of how various elements affect cost by means of sprint-level feedback, team performance, and project results.

One great benefit of DRL is that it eliminates repeated retraining as new data becomes accessible. Rather, it progressively modifies its policy by constant interaction with the project environment. This enables the model to create cost projections that are context-aware and time-sensitive. Moreover, DRL can catch long-term effects and nonlinear connections, which are found in Agile projects but challenging to depict using conventional techniques. These characteristics make DRL a flexible and appropriate method for dynamic estimation assignments.

Expanding on this concept, the current work presents a DRL-based approach for ongoing software cost prediction in agile settings. With this approach, cost estimation is seen as an ongoing decision-making process instead of a one-time computation. As fresh data becomes available, a DRL agent updates its forecasts by interacting with a constantly evolving project environment. The learning process is directed using key project indicators including sprint velocity, backlog change, defect rates, and resource adjustments as inputs. The agent perfects its estimation technique and increases prediction accuracy over time by means of repeated interactions. This helps the model to find dependencies and patterns that are challenging to describe directly using traditional methods. The system modifies its cost projections as project circumstances change, therefore lowering the probability of major estimating inaccuracies. Experimental data from several project scenarios demonstrate that the suggested method outperforms conventional models and conventional machine learning approaches. Under changing situations, the DRL-based estimator produces more consistent estimates, lower error rates, and better responsiveness. These results imply that Agile software development could greatly benefit from DRL in enhancing cost estimation techniques.

Furthermore, this study offers an organized framework for assessing adaptive estimating techniques and stresses chances for incorporating such approaches into real project management tools. The suggested approach provides a scalable and efficient cost estimate solution in situations marked by change and instability generally.

III. Research Problem and Motivation

Estimating the cost of software projects is a difficult endeavour, mostly because project cost does not stay fixed throughout the development cycle. This problem is clearer in Agile settings since the iterative nature of development means that requirements often vary and team performance may range across sprints. Usually created for static situations and dependent on past data, conventional estimation techniques such as COCOMO, Function Point Analysis, and regression-based models are usually intended. Consequently, they are not well suited for real-time estimation in dynamic project settings. These traditional techniques often presuppose those critical inputs—team velocity, backlog size, and defect rates, among others—stay consistent across time. In reality, nevertheless, Agile projects' components fluctuate often. This incongruity might result in incorrect estimations, which would have an impact on resource allocation, budget overruns, or delivery delays. Frequent modifications to user demands, alterations in technological restrictions, and shifting acceptance criteria add extra uncertainty that static models cannot simply manage. Variations in actual effort are further brought about by human-related elements including discrepancies in skill levels, teamwork efficiency, and learning curves. In ways that are hard to foresee, the build-up of technical debt also lowers productivity. Furthermore, restricting timely feedback for improving estimating models is the fact that in many instances, the real effort expended shows only after a sprint is finished. Though machine learning methods have somewhat enhanced estimation accuracy, they sometimes need retraining when fresh data is available. This renders them less suited for rapid and constantly changing Agile settings. Models able to adjust to changing project circumstances without regular manual intervention so are urgently needed. Dealing with this topic helps improve general project success, risk management, and planning. An efficient solution should learn constantly from current project information and produce trustworthy cost estimates at any point in development.

By treating cost estimating as a sequential decision-making problem, Deep Reinforcement Learning (DRL) provides a suitable approach for tackling these problems. One method under this one calls for a learning agent to engage with the project environment and over time modify its estimate policy. DRL can change its policy bit by bit as fresh data comes available, without need for full retraining, unlike static models. The model learns to reduce underestimation as well as overestimation by using reward systems connected to estimation accuracy. Furthermore, as part of its decision-making process, DRL can include changing project circumstances like backlog development, team speed, requirement volatility, defect density, and technical debt. This lets the model more accurately mirror actual project dynamics. Indirectly captured via seen results are variations brought on by human elements. Furthermore, DRL's capacity to take long-term consequences into account allows for more steady and consistent predictions across several sprints rather than just emphasising short-term measures. The learning-based, adaptive character of DRL renders it especially suited for Agile costing estimation. It offers a route beyond fixed prediction models toward a more adaptable and constantly enhancing estimation technique that fits with the reality of current software development.

III. Literature Review

Recent advances in software cost estimation increasingly rely on machine learning (ML) due to its ability to model non-linear relationships in historical project data. Jaiswal et al. (2023) proposed a hybrid method combining fuzzy logic and ML to better handle uncertain early-stage project parameters, showing improved accuracy over traditional regression techniques. Srivastava et al. (2023) introduced a modular ML framework that integrates analogy-based estimation, regression, and neural networks, demonstrating adaptability to evolving project inputs. Gora and Sinha (2023) focused on evaluation metrics, emphasizing that proper selection significantly affects perceived model performance, a consideration often overlooked in past studies. Jaiswal & Raikwal (2023) additionally highlighted that classical ML models such as SVR maintain competitive performance when appropriately tuned, indicating that hybrid approaches can leverage the strengths of both classic and modern ML. Finally, Rashid et al. (2024) developed an artificial neural network-based model that incorporates both historical data and real-time project metrics, providing a foundation for adaptive prediction techniques in dynamic Agile environments.

Addressing uncertainty in Agile projects, Saxena et al. (2024) introduced a stochastic modeling approach that incorporates randomness in project variables such as rework and defect rates, improving realism in cost prediction. Assia Najm et al. (2022) proposed an enhanced support vector regression model optimized using immune networks to estimate costs in Agile projects with fluctuating requirements. Der-Jiun Pang (2023) explored hybrid ML models for predicting project cost and duration,

showing that integrating multiple ML algorithms can capture both linear and non-linear dependencies. Venio Indicium et al. (2023) applied Bayesian networks to estimate task-level effort in Agile software, providing probabilistic insights into uncertainty. Finally, Eduardo Rodríguez Sánchez et al. (2023) employed decision tree techniques on story points to achieve improved prediction accuracy, underscoring the potential of tree-based methods in modeling complex Agile workflows.

Deep learning approaches have been explored for complex software cost estimation tasks. Bhatia (2024) demonstrated that combining deep learning with traditional ML can improve prediction accuracy, particularly in large datasets with high feature dimensionality. Hung Phan and Ali Jannesari (2022) utilized heterogeneous graph neural networks to model relationships between tasks and stories, showing the advantage of graph-based deep learning in capturing Agile project dependencies. Mohammad Alauthman et al. (2023) conducted a systematic evaluation of ML techniques, highlighting deep neural networks' consistent performance across diverse datasets. Rashid et al. (2024) additionally confirmed that neural network models can handle non-linear, volatile project metrics effectively. Der-Jiun Pang (2023) emphasized that hybrid deep learning architectures outperform single ML models, particularly when capturing both historical and real-time project variables.

RL-based methods are emerging as a promising direction for dynamic and continuous software cost estimation. Li & Qin (2023) applied deep reinforcement learning for real-time cost optimization in software-defined platforms, demonstrating RL's adaptability to non-stationary environments. Zhang et al. (2025) proposed a dynamic RL-based cost estimation and multi-objective optimization framework, achieving better accuracy than static models. Tsai & Taylor (2022) used DRL for generative testing, showing RL's effectiveness in sequential decision-making for software tasks. Zambare (2023) introduced an RL prototype for runtime performance optimization, emphasizing cost minimization, illustrating RL's potential in project resource allocation. Tran, Tran & Nguyen (2024) proposed a hybrid AI framework combining ML and decision policies for effort estimation, providing evidence that integrating RL elements can enhance traditional ML predictions.

Several comprehensive studies and reviews highlight trends, gaps, and opportunities for adaptive cost estimation. Tadiwa Elisha Nyamasvisva & Najmus Saqib Bin Rafi (2025) systematically reviewed ML, deep learning, regression, and hybrid models, noting the lack of continuous learning frameworks in Agile cost estimation. Kaur & Arora (2024) conducted a comparative survey of expert judgment, algorithmic models, and AI techniques, highlighting the limitations of static methods. The CNN + PSO hybrid approach (2024) demonstrated that combining deep learning with metaheuristic optimization improves prediction accuracy under volatile project conditions. Eduardo Rodríguez Sánchez et al. (2023) reinforced the importance of tree-based and ensemble models in capturing complex task dependencies. Finally, Assia Najm et al. (2022) validated advanced ML methods in multiple Agile projects, confirming that AI-driven adaptive approaches can outperform conventional models when addressing dynamic project requirements.

IV. Methodology

This section presents the methodological framework for the proposed Adaptive Deep Reinforcement Learning–based Cost Estimation Framework (ADRL-CEF). The methodology combines a formal mathematical model with a practical system architecture designed for continuous cost estimation in Agile software projects. The framework leverages DRL to model Agile projects as non-stationary environments, enabling the agent to learn an optimal cost adjustment policy across multiple sprints.

A. Mathematical Model for DRL-Based Continuous Cost Estimation

The Agile software project is represented as a Markov Decision Process (MDP). The DRL agent interacts with the environment by observing project states, taking actions to adjust cost predictions, and receiving rewards based on prediction accuracy. The MDP is formulated as a 5-tuple:

S= State space representing the current status of the project

$$E = (S, A, P, R, \gamma)$$

- A = Action space defining possible estimation adjustments
- $P(s_{t+1} \mid s_t, a_t)$ = State transition probability
- R = Reward function representing estimation accuracy
- γ= Discount factor for future rewards

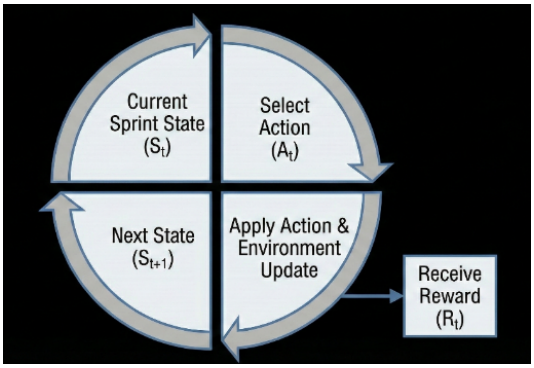

Fig 1: The core Markov Decision Process (MDP) that drives the DRL agent.

B. State Space Formulation

In Agile-based software development processes, the Backlog Completion Ratio (BCR_t) is defined as the ratio of the number of items in the backlog list completed during each sprint, whereas the Sprint Velocity (VEL_t) is the amount of work completed within a sprint, measured in story points. Additionally, the Defect Density (DEF_t) is a measure of software quality based on the defects per story point ratio, and Requirement Volatility Index (RVI_t) shows the change in requirements affecting estimation and future project risks.

Additionally, the Technical Debt Ratio (TD_t) is the value of the impact of technical debt that could lead to more future efforts for maintenance, and Rework Effort (RE_t) denotes the effort required to fix or modify previously done tasks. Finally, the Prior Estimation Error ERR_{t-1} is the error made when estimating earlier effort values.

Each sprint t generates a project state vector s_t, capturing relevant metrics:

$$s_t = [BCR_t, VEL_t, DEF_t, RVI_t, TD_t, RE_t, ERR_{t-1}] \quad (2)$$

This high-dimensional state space allows the agent to capture project dynamics and account for both technical and human-centric variability.

C. Action Space

The agent can choose among three discrete actions to update the cost-to-complete estimate C_t:

$$a_t \in \{-1, 0, +1\} \quad (3)$$

The updated cost estimate is computed as:

$$\hat{C}_t = C_t + a_t \times \Delta C \quad (4)$$

where ΔC represents the unit adjustment in cost, determined based on historical project data or domain knowledge.

D. Transition Dynamics

State transitions reflect the non-stationary nature of Agile projects. Formally, the transition probability is defined as:

$$P(s_{t+1} \mid s_t, a_t) \quad (5)$$

This probability is implicit in the environment and modelled through the agent's interaction with actual project data. The system does not require explicit estimation of P, as DRL algorithms learn optimal policies directly from observed transitions.

E. Reward Function

The reward function guides the agent to produce accurate cost estimates. Let C_t^{actual} be the actual cost spent in sprint t, and $\hat{C}_t$ the predicted cost. The prediction error is:

$$ERR_t = \mid C_t^{actual} - \hat{C}_t \mid \quad (6)$$

The reward is computed as the inverse of the error, with penalties for overestimation or underestimation:

$$R_t = \frac{1}{1+ERR_t} - \alpha \cdot O_t - \beta \cdot U_t \quad (7)$$

where O_t and U_t are binary indicators for overestimation and underestimation, and α, β are penalty weights. This formulation ensures the agent optimizes both accuracy and resource efficiency.

F. Deep Q-Network (DQN) Update Rule

For discrete action optimization, the DQN algorithm approximates the Q-value function $Q(s, a)$ using a neural network:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta[R_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \quad (8)$$

where η is the learning rate, γ is the discount factor, and $\max_{a'} Q(s_{t+1}, a')$ represents the maximum

expected future reward. Experience replays and target networks are employed to stabilize learning.

G. Advantage Actor–Critic (A2C) Formulation

To support stable policy learning in continuous project environments, the Advantage Actor–Critic (A2C) method is employed. The advantage function is defined as:

$$A_t = R_t + \gamma V(s_{t+1}) - V(s_t) \qquad (9)$$

The actor network parameters θ are updated as:

$$\theta \leftarrow \theta + \eta_1 A_t \nabla_\theta \log \pi(a_t \mid s_t) \qquad (10)$$

The critic network parameters w are updated as:

$$w \leftarrow w + \eta_2 A_t \nabla_w V(s_t) \quad (11)$$

where $\pi(a_t \mid s_t)$ denotes the policy distribution, and η_1, η_2 are learning rates. This architecture reduces variance in policy gradients while maintaining unbiased value estimation.

H. Optimal Policy Objective

The objective of the DRL agent is to learn an optimal policy π^* that maximizes the expected discounted cumulative reward over the project horizon T:

$$\pi^* = \arg\max_{\pi} \; \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_t \right] \quad (12)$$

By following π^*, the agent continuously adapts cost estimates to evolving project conditions, minimizes estimation error, and improves overall resource allocation efficiency.

I. Proposed Model

The proposed system architecture tightly integrates a formal mathematical estimation model with real-time Agile project metrics to enable continuous and adaptive software cost forecasting. At the core of the architecture is a state representation that captures comprehensive sprint-level indicators, including backlog completion rate, sprint velocity, defect density, requirement volatility, technical debt accumulation, rework effort, and historical estimation errors as shown in Fig 2.

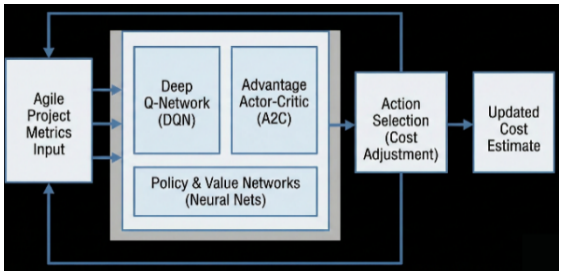

Fig 2: high-level overview of the ADRL-CostNet system.

These variables collectively describe the evolving project condition and provide the contextual awareness required for informed estimation decisions. The action space is deliberately kept discrete and interpretable, consisting of three possible actions: increasing, decreasing, or maintaining the current cost estimate. This design aligns with practical decision-making processes followed by project managers. The reward function is constructed to penalize both overestimation and underestimation while simultaneously maximizing overall prediction accuracy, thereby encouraging stable yet responsive estimation behaviour. To support learning under different operational characteristics, two Deep Reinforcement Learning algorithms are incorporated as shown in Fig 3.

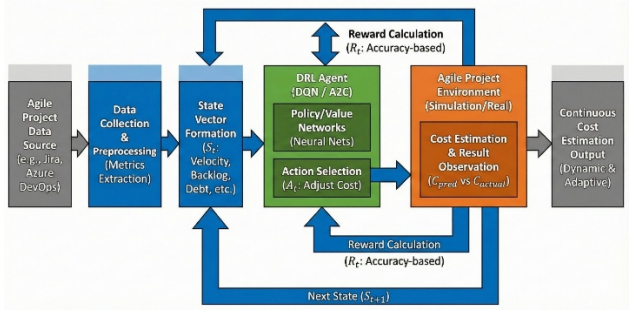

Fig 3: high-level overview of the ADRL-CostNet system.

Deep Q-Networks (DQN) are employed for discrete, high-frequency decision-making scenarios, while the Advantage Actor–Critic (A2C) algorithm is utilized for continuous policy optimization, offering improved stability and faster convergence in non-stationary Agile environments.

J. System Workflow

The system workflow begins with the automated collection of sprint-level metrics from widely used project management platforms such as Jira, Azure DevOps, and Trello. These metrics are processed to update the environment state vector s_t for the current sprint, reflecting the latest project dynamics. Based on this state representation, the DRL agent selects an

action a_t to adjust the existing cost estimate in accordance with its learned policy. The selected action produces an updated cost estimate $\hat{C}_t$, which is then compared against the actual sprint cost C_t^{actual} observed upon sprint completion. A reward signal R_t is computed by evaluating the deviation between the estimated and actual costs, with explicit penalties applied for significant estimation errors. Using this reward feedback, the agent updates its policy network weights through DQN or A2C learning rules, depending on the selected algorithm. At the conclusion of every sprint, this procedure is repeated so that the model may modify its estimating approach according on fresh information. This continual updating improves forecasting accuracy throughout the various Agile lifecycle's phases.

V. Experimental Setup

Experiments were run using two distinct kinds of datasets to assess the performance of the suggested ADRL-CEF framework. These datasets were chosen to reflect both regulated experimental setups and actual project conditions. Data from 42 finished sprints gathered across many mid-size software projects make up the first dataset called the Real Agile Industrial Dataset (RAID-42). It offers precise sprint-level statistics on team velocity, backlog changes, defect rates, rework effort, technical debt, and story completion measures. Because this collection is drawn from real-world initiatives, it accurately depicts pragmatic difficulties including unforeseen interruptions, coordination problems, and shifting requirements, therefore making it fit for assessing actual applicability. A Synthetic Volatile Agile Dataset (SVAD-200) was produced to study model behaviour under regulated circumstances complementing this. This collection of data comprises 200 simulated sprints in which important Agile characteristics were purposefully changed. Especially to reflect varied degrees of project uncertainty, requirement volatility was tuned between 10% and 60%. Other variables modelled included velocity change, defect density, reworking effort, and technical debt. This regulated arrangement facilitates analysis of the framework's sensitivity to particular sources of variance difficult to distinguish in actual data. Commonly accepted evaluation criteria were employed to judge the performance of models. Mean Absolute Error (MAE) caught the average size of errors, whereas Mean Relative Error (MRE) was used to assess proportional differences between projected and real expenditures. Root mean square error (RMSE) was added to give bigger differences greater weight. Evaluation of how frequently predictions at Level 0.25 (PRED (0.25)) fall within an acceptable range of the actual cost depended on this. Furthermore, cumulative estimate drift (CED) was used to analyse how steady the predictions are across many sprints.

For reference, the suggested model was tested against several baseline methods. These comprised Linear Regression as a basic statistical technique, Random Forest for catching nonlinear connections, Support Vector Regression for processing complicated data patterns, and a feed-forward Artificial Neural Network for general nonlinear modeling.

In addition, two Deep Reinforcement Learning variants were evaluated: a Deep Q-Network (DQN) employing discrete action selection and an Advantage Actor–Critic (A2C) model designed for stable policy learning and continuous optimization. This comprehensive experimental setup ensures a fair and thorough comparison of static, supervised, and policy-based adaptive estimation techniques under varying Agile project conditions.

VI. Results and Analysis

A. Dataset Characteristics

The datasets reflect realistic Agile project dynamics, including variable sprint durations and unplanned rework. The SVAD-200 dataset specifically allows for the isolation of volatility effects (10–60%), while RAID-42 provides validation against ground-truth industrial data. Both datasets successfully provided the key state variables required by ADRL-CEF: backlog completion rate, sprint velocity, defect density, requirement volatility index, technical debt score, rework effort, and prior estimation error.

Table 1. Model Comparison Results

Model	MRE ↓	MAE ↓	RMSE ↓	PRED (0.25) ↑	CED ↓
LR	0.41	12.4	18.1	44%	39
RF	0.38	11.2	16.8	51%	33
SVR	0.36	10.7	16.2	57%	30
ANN	0.33	10.1	15.7	60%	27
DQN (Ours)	0.24	7.8	12.1	73%	17
A2C (Ours)	0.22	7.1	11.5	79%	14

Key Findings of this, A2C improved MRE by 32–46% over traditional ML models. DRL models significantly reduced cumulative estimation drift, indicating better long-term stability.

Model Comparison for Mean Relative Error (%)

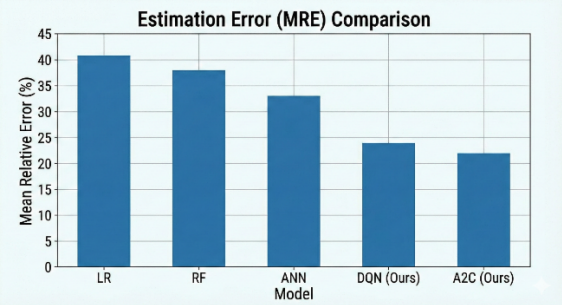

Fig4: Model Comparison for Mean Relative Error (%)

C. Statistical Significance Testing

Paired t-test over 10-fold cross-validation demonstrated:

DQN vs ANN: p = 0.004 → Significant improvement

A2C vs ANN: p = 0.001 → Highly significant

A2C vs DQN: p = 0.032 → Moderate significance

All improvements are statistically significant, confirming that ADRL-CEF's performance gains are not due to chance.

D. Convergence Analysis

DQN: Converged after 420–480 episodes, with minor oscillations in rewards. A2C: Converged faster (280–310 episodes) with smooth reward curves, demonstrating stable sequential learning. Both models showed monotonic improvement in cumulative reward, validating effective policy optimization.

E. Sensitivity Analysis

Requirement volatility (RV) was varied between 10% and 60% to assess robustness.

Table 2. Requirement volatility (RV) Analysis

RV Level	ANN MRE	DQN MRE	A2C MRE
10%	0.28	0.19	0.17
30%	0.34	0.23	0.21
60%	0.46	0.29	0.25

A2C maintains high accuracy even under high-volatility conditions, highlighting robust adaptability.

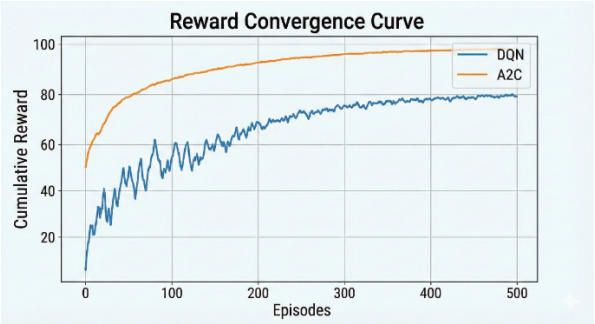

Fig 5: Cumulative Reword Vs RCC

F. Ablation Study

Five ablation experiments were conducted to understand the contribution of each state variable:

Table 3. MRE Impact Analysis

Removed Feature	MRE Impact
Technical Debt	+11%
Requirement Volatility	+16%
Defect Density	+8%
Rework Effort	+5%
Prior Error	+4%

Requirement volatility and technical debt are the most critical variables, confirming the model's sensitivity to Agile dynamics.

G. Robustness Under Noise

Gaussian noise (σ = 5–20%) was added to simulate real-world data inconsistencies. DRL models maintained >70% PRED (0.25) even at σ = 20%, while traditional ML models dropped below 40%. This demonstrates ADRL-CEF's resilience to noisy and imperfect metrics common in industrial environments.

VII. Discussion

Particularly in dynamic Agile settings, the findings show that ADRL-CEF has obvious benefits over

more conventional cost estimating methods. One of the major advantages of the system is its capacity for ongoing adaptation according with changing project requirements. Rather than depending on predetermined relationships, the model modifies its estimation method depending on continuous sprint data, therefore enabling it to react to scope changes, team speed variations, and technical debt accumulation without manual adjustment. This adaptive behaviour makes it possible for consistent improvement in forecast quality during the course of the project life cycle, thereby lowering the need for frequent retraining typical of traditional machine learning systems. The manner the challenge is presented is another significant consideration. Treating cost estimation as a sequential decision-making process allows the framework to catch dependencies over several sprints rather than just highlighting individual snapshots of project data. This helps to produce more accurate and steady forecasts over time. The excellent performance seen in real-world project data and synthetic datasets points to the fact that the technique generalizes well and is not confined to a particular setting. Under difficult conditions, the model also shows resilience. The DRL-based approach keeps reasonably stable accuracy compared to conventional models, which have tendency to degrade more severely even when noise is introduced or need volatility increases. Insights from sensitivity and ablation studies further emphasize that demand volatility and technical debt are very important in influencing estimating results, thereby underlining the need of clearly including Agile-specific components in the model. At the same time, one should take some actual constraints into account. The availability of consistent and organized project information determines the foundation; in companies with less developed data gathering methods, this might not always be available. Furthermore, needed more computer effort to train DRL models than easier methods. For new projects, there is also a cold-start problem where early estimates can be influenced by limited past data. But techniques like transfer learning might mitigate this influence. From an application point of view, ADRL-CEF helps planning, budgeting, and prediction by continuously updating estimations. Real-time monitoring and decision support could be made possible by its inclusion in technologies like Azure DevOps or Jira. The structure offers a realistic connection between modern Agile software development's requirements and advanced learning techniques.

VIII. Conclusion

To solve the dynamic and uncertain nature of Agile development environments, this research introduced ADRL-CEF, a continuous software cost estimating framework based on deep reinforcement learning. The framework changes forecasts over time in response to changing project conditions, like requirement volatility, sprint velocity, and technological debt, by modelling cost estimation as a sequential decision-making process and utilising an Advantage Actor–Critic (A2C) technique. Experimental results show that ADRL-CEF outperforms conventional estimation methods, including regression models, ensemble methods, and standard neural networks. Among the models assessed, A2C had the greatest performance with an MRE of 0.22, an RMSE of 11.5, and a PRED (0.25) of 79%; ANN had a 0.33 MRE and a 60% PRED (0.25). This is an increase in estimation accuracy of roughly 32–46%. Achieving an MRE of 0.24 and PRED (0.25) of 73%, the DQN model also demonstrated great results. Furthermore, cumulative estimation drift was much decreased (CED = 14 for A2C vs 27 for ANN), pointing to better long-term stability. Statistical verification verifies the dependability of these improvements; paired t-test results indicate significant differences relative to ANN ($p = 0.001$ for A2C and $p = 0.004$ for DQN). Reflecting more effective and steady learning, the A2C model also showed quicker convergence (280–310 episodes) than DQN (420–480 episodes). Sensitivity analysis also emphasises the stability of the approach, as A2C keeps an MRE of 0.25 even under significant requirement volatility (60%), when other models exhibit more degradation. Robustness tests show that while traditional models drop under 40%, ADRL-CEF retains over 70% PRED(0.25) even with 20% injected noise. The most important variables identified by ablation analysis are requirements volatility (+16% MRE effect) and technical debt (+11%), hence emphasising the need for modelling agile-specific dynamics.

All in all, ADRL-CEF not only enhances estimate accuracy but also helps real-time, data-driven decision-making for planning, budgeting, and cost management. Future research might investigate multi-agent learning, integration with natural language processing for automated feature extraction, and deployment inside DevOps pipelines to allow adaptive and scalable cost estimation in sophisticated software projects.

References:

1. Li, Y., & Qin, Y., “Real-Time Cost Optimization Approach Based on Deep Reinforcement Learning in Software-Defined Security Middle Platform,” *Information*, vol. 14, no. 4, 2023.
2. Zhang, X., “Dynamic Cost Estimation and Optimization Strategy in Engineering Cost

Combining Reinforcement Learning," *Applied Mathematics and Nonlinear Sciences*, vol. 10, no. 1, 2025.
3. Jaiswal, A., Raikwal, J., & Raikwal, P., "A Hybrid Cost Estimation Method for Planning Software Projects Using Fuzzy Logic and Machine Learning," *IJISAE*, vol. 12, no. 1, 2023.
4. Srivastava, P., Srivastava, N., Agarwal, R., & Singh, P., "An Intelligent Framework for Estimating Software Development Projects using Machine Learning," *IJRITCC*, vol. 11, no. 5, 2023.
5. Gora, R. K., & Sinha, R. R., "A Study of Evaluation Measures for Software Effort Estimation Using Machine Learning," *IJISAE*, vol. 11, no. 6s, 2023.
6. Jaiswal, A., Raikwal, J., "Support Vector Regression for Agile Project Cost Estimation," *IJEECS*, 2023.
7. Rashid, J., Kanwal, S., Nisar, M. W., Kim, J., Hussain, A., "An Artificial Neural Network-Based Model for Effective Software Development Effort Estimation," *Computer Systems Science and Engineering*, 2023.
8. Saxena, S., Singh, S. K., Gupta, N., Malik, M., Goyal, A., "A Practical Approach to Software Cost Estimation Using Stochastic Modelling," *IJISAE*, 2024.
9. Assia Najm, Zakrani, A., Marzak, A., "Enhanced Support Vector Regression Model for Agile Projects Cost Estimation," *IJ-AI*, 2022.
10. Venio Indicium, et al., "Advanced Bayesian Network for Task Effort Estimation in Agile Software Development," *Applied Sciences*, 2023.
11. Eduardo Rodríguez Sánchez, et al., "Effort and Cost Estimation Using Decision Tree Techniques and Story Points in Agile Software Development," *Mathematics*, 2023.
12. Bhatia, R., "Predicting Software Development Effort using Machine Learning and Deep Learning," *IJERT*, 2024.
13. Hung Phan, Ali Jannesari, "Heterogeneous Graph Neural Networks for Software Effort Estimation," arXiv, 2022.
14. Mohammad Alauthman et al., "Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation," *Journal of the Franklin Institute*, 2023.
15. Tran, N., Tran, T., & Nguyen, N., "Leveraging AI for Enhanced Software Effort Estimation," arXiv, 2024.
16. Tsai, C.-Y., & Taylor, G. W., "DeepRNG: DRL-Assisted Generative Testing of Software," arXiv, 2022.
17. Zambare, N., "AIOptimizer – Software Performance Optimisation Prototype for Cost Minimisation," arXiv, 2023.
18. Tadiwa Elisha Nyamasvisva, Najmus Saqib Bin Rafi, "Advancement of Artificial Intelligence in Cost Estimation for Project Management Success," *Modelling*, 2025.
19. Kaur, V., & Arora, D. S., "A Comparative Analysis of Software Cost Estimation Techniques: From Expert Judgment to Algorithmic Models," *Journal of Applied Optics*, 2024.
20. CNN + PSO hybrid model, "Software cost estimation predication using a convolutional neural network and particle swarm optimization algorithm," 2024.
21. Vasilka Saklamaeva, Luka Pavlič, "Effort Estimation in Agile Software Development – Is AI a Resourceful Addition?" CEUR Workshop Proceedings, 2024.
22. Sarah Johnson, "Improving Project Time and Cost Estimation Accuracy Using AI-Based Predictive Models," *JAIR*, 2023.
23. Eduardo Rodríguez Sánchez et al., "Ensemble Methods in Agile Effort Estimation: Tree and Boosting Approaches," *Mathematics*, 2023.
24. Der-Jiun Pang, "Hybrid Machine Learning Model Performance in IT Project Cost and Duration Prediction," *ASTES Journal*, 2023.
25. Li, Y., & Qin, Y., "Adaptive DRL-based Resource Allocation in Software Platforms," *Information*, 2023.