ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ &
ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Ενισχυτική Μάθηση για Πρόβλεψη Φθοράς Βιομηχανικού Εξοπλισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αναστάσιος Αγγλογάλλος

Επιβλέπων : Γρηγόριος Μέντζας
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ &
ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Ενισχυτική Μάθηση για Πρόβλεψη Φθοράς Βιομηχανικού Εξοπλισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αναστάσιος Αγγλογάλλος

Επιβλέπων : Γρηγόριος Μέντζας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουλίου 2024.

(*Υπογραφή*)	(*Υπογραφή*)	(*Υπογραφή*)
.....	.....	.....
Γρηγόριος Μέντζας	Ιωάννης Ψαρράς	Δημήτριος Ασκούνης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

(*Υπογραφή*)

.....

Αναστάσιος Αγγλογάλλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στον κόσμο της βιομηχανίας, όπου ο στόχος είναι η μείωση των εξόδων και η ελαχιστοποίηση των απωλειών, η συντήρηση, αν και απαραίτητη, συχνά συμβάλλει σε αυτά τα έξοδα και μπορεί να διαταράξει τη διαδικασία παραγωγής. Επομένως, η βελτιστοποίηση των στρατηγικών συντήρησης θα πρέπει να αποτελεί προτεραιότητα.

Οι παραδοσιακές στατιστικές και ντετερμινιστικές προσεγγίσεις για το σχεδιασμό συστημάτων ελέγχου και προγραμματισμού είναι περιορισμένες στην αποτελεσματικότητά τους λόγω της πολύπλοκης, μη γραμμικής φύσης των προβλημάτων συντήρησης.

Παρόλο που η Μηχανική Μάθηση (Machine Learning) προσφέρει πιθανές λύσεις σε αυτές τις πολυπλοκότητες, δεν παύει να αποτελεί πρόκληση η απόκτηση επαρκών δεδομένων από ειδικούς, για την επιβλεπόμενη και μη επιβλεπόμενη μάθηση, οδηγώντας συχνά σε προσεγγιστικές και ανεπαρκής λύσεις.

Η Ενισχυτική Μάθηση (Reinforcement Learning) παρουσιάζει έναν αυτόνομο μηχανισμό μάθησης που μπορεί να ξεπεράσει μερικούς από αυτούς τους περιορισμούς. Η παρούσα διπλωματική επικεντρώνεται στη χρήση διαφόρων αλγορίθμων RL για την αντιμετώπιση ενός προβλήματος Προγνωστικής Συντήρησης (Predictive Maintenance), συγκεκριμένα την πρόβλεψη φθοράς των flutes (φρεζοτρύπανων) σε διάφορες φρέζες CNC υψηλής ταχύτητας. Αυτή η πρόβλεψη βασίζεται σε δεδομένα από δυναμόμετρα, επιταχυνσιόμετρα και αισθητήρες ακουστικής εκπομπής. Όλοι οι αλγόριθμοι θα εκπαιδευτούν και θα αξιολογηθούν χρησιμοποιώντας πραγματικά δεδομένα, και θα διεξαχθεί μια ευρεία ανάλυση για να εξεταστεί η αποτελεσματικότητά τους.

Λέξεις Κλειδιά : Προγνωστική Συντήρηση, Ενισχυτική Μάθηση, MDP, Πρόβλεψη Φθοράς, Μηχανική Μάθηση

Abstract

In the world of manufacturing, where the goal is to cut down on any expense and minimize losses, maintenance, although essential, often contributes to these expenses and can disrupt the production process. Therefore, optimizing maintenance strategies should be a priority.

Traditional deterministic approaches to designing control and planning systems are limited in their effectiveness due to the complex, nonlinear nature of maintenance problems.

While machine learning offers potential solutions to these complexities, the challenge of obtaining sufficient labeled data from experts for supervised and unsupervised approaches often leads to approximate solutions.

Reinforcement learning (RL) presents an autonomous learning mechanism that can potentially overcome some of these limitations. This thesis focuses on employing various RL algorithms to address a predictive maintenance (PdM) problem, specifically the wear prediction of the flutes in cutters of a high-speed CNC milling machine. This prediction is based on data from dynamometer, accelerometer, and acoustic emission sensors. All algorithms will be trained and evaluated using real-world data, and a broad analysis will be conducted to examine their effectiveness.

Keywords: Predictive Maintenance, Reinforcement Learning, MDP, Wear Prediction, Machine Learning

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής εργασίας και ως συνέπεια του κύκλου σπουδών μου, θα ήθελα να εκφράσω τις ειλικρινείς ευχαριστίες μου προς όλους αυτούς, που συνέβαλαν και βοήθησαν σε αυτό το στόχο.

Πρώτα και κύρια, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της διπλωματικής μου εργασίας, Καθηγητή κ. Γρηγόριο Μέντζα, για την εμπιστοσύνη που μου έδειξε με την ανάθεση της εργασίας και την πολύτιμη καθοδήγησή του, καθόλη τη διάρκεια αυτής της διαδρομής. Επίσης, θα ήθελα να ευχαριστήσω τον διδάκτορα κ. Αλέξανδρο Μπουσδέκη και τον υποψήφιο διδάκτορα κ. Στέφανο Κόντο, των οποίων οι συμβουλές, οι προτάσεις και η ανοιχτή στάση απέναντί μου με βοήθησαν να εξελιχθώ ως ερευνητής και να διαμορφώσω την εργασία μου με τον καλύτερο δυνατό τρόπο.

Δεν θα μπορούσα να παραλείψω τους γονείς μου, τα αδέρφια μου Αριστείδη και Κώστα καθώς και την Αγάπη, για την συμπαράσταση και την κατανόηση για όλα τα χρόνια των σπουδών μου. Χωρίς εκείνους σίγουρα δεν θα είχα καταφέρει όσα μπόρεσα.

Ιδιαίτερες ευχαριστίες σε όλους του φίλους μου που ήταν στο πλευρό μου σε όλη την διάρκεια των σπουδών μου και με βοηθήσαν να ξεπεράσω κάθε εμπόδιο. Ακόμη, θα ήθελα να ευχαριστήσω τους εστιακούς μου γείτονες και την τετράποδη συνοδοιπόρο μου Rubia για την συνεχή στήριξη σε αυτό το ταξίδι. Τέλος, νιώθω την ανάγκη να ευχαριστήσω ξεχωριστά όλους όσους περάσαμε ατελείωτες ώρες διαβάσματος μαζί σε αυτά τα πέντε χρόνια, καθώς κάνατε αυτή τη διαδικασία πολύ πιο ευχάριστη και εύκολη.

Είμαι ευγνώμων για τη συνεργασία και την υποστήριξη που έλαβα από όλους και όλες κατά τη διάρκεια αυτής της ερευνητικής προσπάθειας. Αυτή η διπλωματική εργασία είναι αποτέλεσμα της αγάπης σας και σας την αφιερώνω.

Αναστάσιος Αγγλογάλλος
Ιούλιος 2024

Table of Contents

Tables of Tables

Table of Figures

List of Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AI	Artificial Intelligence Τεχνητή Νοημοσύνη
API	Application Programming Interface
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
EDA	Exploratory Data Analysis Διερευνητική Ανάλυση Δεδομένων
HER	Hindsight Experience Replay
MDP	Markov Decision Process Διαδικαστικό Σύστημα Αποφάσεων Markov
ML	Machine Learning Μηχανική Μάθηση
PdM	Predictive Maintenance Προγνωστική Συντήρηση
PHM	Prognostics and Health Management
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement Learning Ενισχυτική Μάθηση
RMS	Root Mean Square
RUL	Remaining Useful Life Υπολειπόμενος Χρήσιμος Χρόνος Ζωής
SAC	Soft Actor Critic
SARSA	State-Action-Reward-State-Action
S-MDP	Semi-Markov Decision Process
S-POMDP	Semi-Partially Observable Markov Decision Process

TD3	Twin Delayed DDPG
TRPO	Trust Region Policy Optimization
UI	User Interface
VPG	Vanilla Policy Gradient

Εκτεταμένη Ελληνική Περίληψη

Θεωρητικό Υπόβαθρο

Προγνωστική Συντήρηση

Η προγνωστική συντήρηση είναι μια στρατηγική προσέγγιση που χρησιμοποιεί την ανάλυση δεδομένων για την ανίχνευση ανωμαλιών και την πρόβλεψη βλαβών εξοπλισμού. Αποτελεί μια μέθοδο προληπτικής στρατηγικής συντήρησης μηχανημάτων ζωτικής σημασίας για την Βιομηχανία.

Βοηθά στη μείωση των κόστους, στην ελαχιστοποίηση του χρόνου αδράνειας και στην ενίσχυση της ασφάλειας, επιτρέποντας παρεμβάσεις συντήρησης κατά τα αρχικά στάδια της φθοράς του εξοπλισμού, όπως απεικονίζεται στην καμπύλη του διαγράμματος P-F (Potential Failure - Functional Failure).

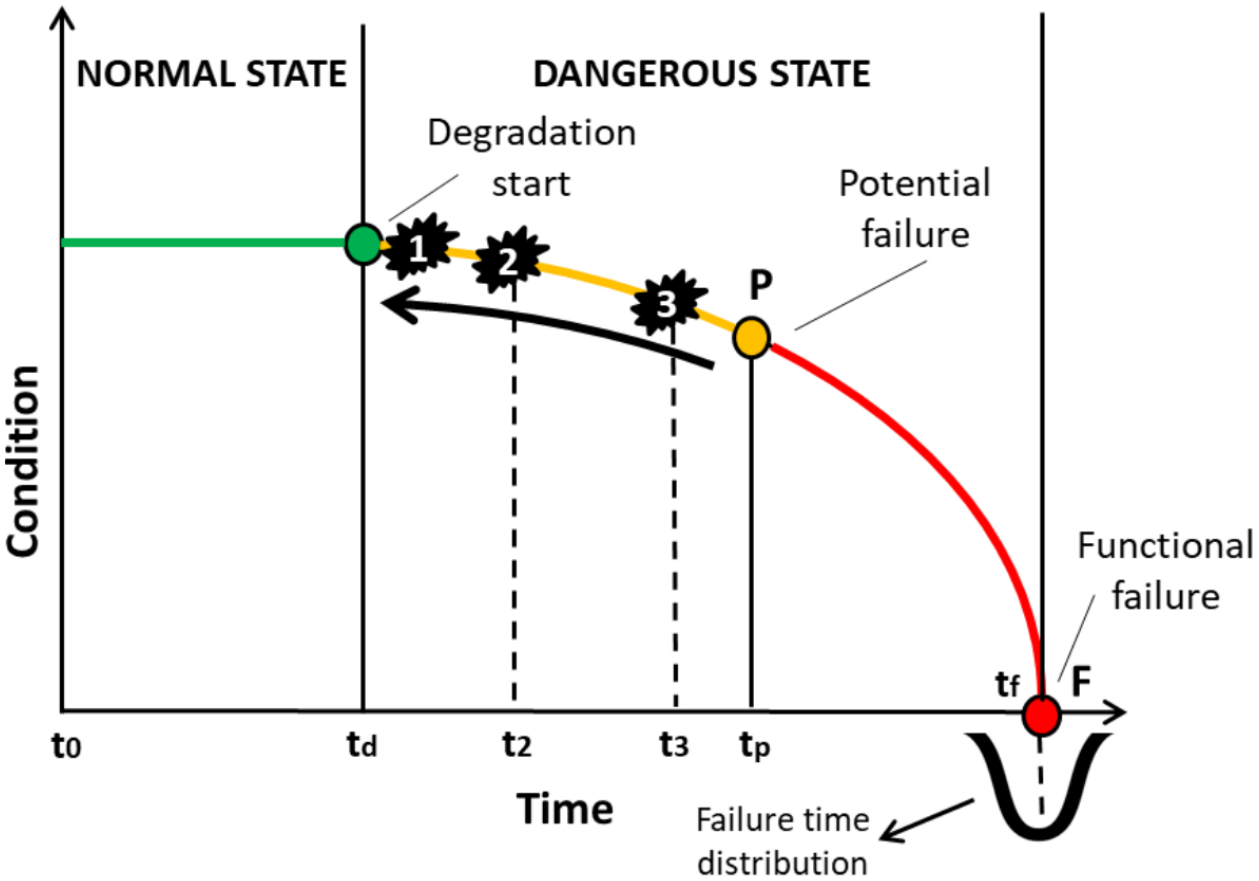


Figure 1 : Προγνωστική συντήρηση στο πλαίσιο της καμπύλης P-F (Bousdekis, Apostolou, & Mentzas, 2020)

Παρά τα οφέλη της, η εφαρμογή της προγνωστικής συντήρησης αντιμετωπίζει προκλήσεις, όπως η ανάγκη απόκτησης νέας τεχνολογίας (όπως καινούργιοι IoT αισθητήρες), σημαντικές αρχικές επενδύσεις και

εξάρτηση από εξειδικευμένο προσωπικό. Απαιτεί επίσης διατμηματική συνεργασία για να είναι αποτελεσματική.

Η ενσωμάτωση της Ενισχυτικής Μάθησης και των MDPs στην προγνωστική συντήρηση μπορεί να βελτιστοποιήσει τη λήψη αποφάσεων υπό αβεβαιότητα και να βελτιώσει τη συνολική αποτελεσματικότητα των στρατηγικών συντήρησης.

Διαδικασία Απόφασης Markov

Ένα διαδικαστικό σύστημα αποφάσεων Markov (MDP) είναι μια διακριτή χρονική διαδικασία στοχαστικού ελέγχου. Παρέχει ένα μαθηματικό πλαίσιο για τη μοντελοποίηση της λήψης αποφάσεων σε καταστάσεις όπου τα αποτελέσματα είναι εν μέρει τυχαία και εν μέρει υπό τον έλεγχο του λήπτη των αποφάσεων. Τα MDP είναι χρήσιμα για τη μελέτη προβλημάτων βελτιστοποίησης που επιλύονται μέσω δυναμικού προγραμματισμού.

Χρησιμοποιούνται σε πολλούς τομείς, συμπεριλαμβανομένων της ρομποτικής, του αυτόματου ελέγχου, της οικονομίας και της βιομηχανίας. Το όνομα των MDP προέρχεται από τον Ρώσο μαθηματικό Αντρέι Μάρκοφ καθώς αποτελούν επέκταση των αλυσίδων Μάρκοφ.

Σε κάθε χρονικό βήμα, η διαδικασία βρίσκεται σε κάποια κατάσταση s, και ο λήπτης των αποφάσεων μπορεί να επιλέξει οποιαδήποτε δράση a είναι διαθέσιμη στην κατάσταση s. Η διαδικασία ανταποκρίνεται στο επόμενο χρονικό βήμα μεταβαίνοντας τυχαία σε μια νέα κατάσταση s' και δίνοντας στον λήπτη των αποφάσεων μια αντίστοιχη ανταμοιβή $R(s, a, s')$.

Η πιθανότητα η διαδικασία να μετακινηθεί στη νέα της κατάσταση s' επηρεάζεται από την επιλεγμένη δράση. Συγκεκριμένα, δίνεται από τη συνάρτηση μετάβασης κατάστασης $P(s'|s, a)$. Έτσι, η επόμενη κατάσταση s' εξαρτάται από την τρέχουσα κατάσταση s και τη δράση του λήπτη των αποφάσεων a, αλλά δεδομένου των s και a, είναι υπό όρους ανεξάρτητη από όλες τις προηγούμενες καταστάσεις και δράσεις· με άλλα λόγια, οι μεταβάσεις κατάστασης ενός MDP ικανοποιούν την ιδιότητα Μάρκοφ.

Τα διαδικαστικά συστήματα αποφάσεων Markov αποτελούν μια επέκταση των αλυσίδων Μάρκοφ· η διαφορά είναι η προσθήκη δράσεων (που επιτρέπουν την επιλογή) και ανταμοιβών (που παρέχουν κίνητρο). Αντιθέτως, αν υπάρχει μόνο μία δράση για κάθε κατάσταση και όλες οι ανταμοιβές είναι ίδιες, ένα διαδικαστικό σύστημα αποφάσεων Markov μειώνεται σε μια αλυσίδα Μάρκοφ.

Συνολικά, ένα MDP είναι ένα 5-tuple (S, A, R, P, ρ_0), όπου:

- S (State Space): είναι το σύνολο όλων των έγκυρων καταστάσεων,
- A (Action Space): είναι το σύνολο όλων των έγκυρων δράσεων,
- $R: S \times A \times S \to R$ είναι η συνάρτηση ανταμοιβής, με $r_t = R(s_t, a_t, s_{t+1})$
- $P: S \times A \to P(S)$ είναι η συνάρτηση πιθανότητας μετάβασης, με $P(s'|s, a)$ να είναι η πιθανότητα μετάβασης στην κατάσταση s' δεδομένου ότι βρισκόμαστε στην κατάσταση s και κάνουμε τη δράση a,

- και ρ_0 είναι η κατανομή της αρχικής κατάστασης, δηλαδή η πιθανότητα να βρεθούμε σε μια συγκεκριμένη s_0.

Ενισχυτική Μάθηση

Μάθηση από Ανταμοιβές

Στην ενισχυτική μάθηση, ένας πράκτορας (agent) μαθαίνει να λαμβάνει αποφάσεις λαμβάνοντας και ερμηνεύοντας σήματα ανταμοιβής (reward) που υποδεικνύουν την ποιότητα των ενεργειών του (action). Ο στόχος είναι να μεγιστοποιηθεί το αθροιστικό άθροισμα των μελλοντικών ανταμοιβών. Αυτή η μέθοδος είναι ιδιαίτερα χρήσιμη σε περιβάλλοντα με έναν τεράστιο αριθμό πιθανών καταστάσεων, πολλές από τις οποίες ο πράκτορας ενδέχεται να μην έχει αντιμετωπίσει προηγουμένως.

Μια αξιοσημείωτη εφαρμογή της ενισχυτικής μάθησης είναι στο σκάκι. Αντί να βασίζεται στη επιβλεπόμενη μάθηση με δεδομένα χαρακτηρισμένα από προηγούμενες παρτίδες για κάθε πιθανή κίνηση, ένας πράκτορας RL μαθαίνει από τα αποτελέσματα των κινήσεών του, προσαρμόζοντας τη στρατηγική του για να μεγιστοποιήσει τις ανταμοιβές, όπως να κερδίσει το κέντρο της σκακιέρας ή γενικότερα τη νίκη στο παιχνίδι. Αυτή η μέθοδος έχει πολλά πλεονεκτήματα σε σύνθετα περιβάλλοντα στα οποία η άμεση διδασκαλία είναι ανέφικτη, καθώς επιτρέπει στον πράκτορα να αναπτύξει στρατηγικές μέσω δοκιμής και σφάλματος βασισμένες στην ανατροφοδότηση των ανταμοιβών του.

Βασικές Έννοιες στην Ενισχυτική Μάθηση

Στην RL, ο πράκτορας αλληλεπιδρά με το περιβάλλον μέσω ενεργειών και λαμβάνει παρατηρήσεις και ανταμοιβές ως αντάλλαγμα. Η **πολιτική** (policy) του πράκτορα, που υπαγορεύει τις ενέργειές του βάσει της τρέχουσας κατάστασης, ενημερώνεται συνεχώς βάσει της ανατροφοδότησης από το περιβάλλον, όπως φαίνεται στο διάγραμμα.

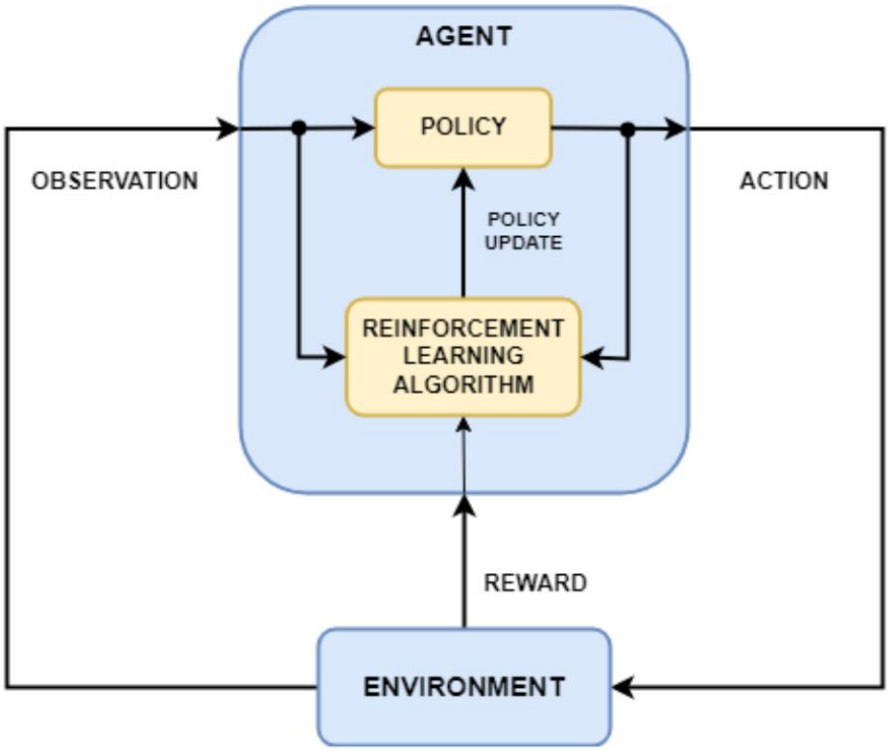


Figure 2 : Γενική Αναπαράσταση Διαδικασίας Ενισχυτικής Μάθησης (MathWorks, 2024)

Συστατικά της Ενισχυτικής Μάθησης

Ένας Αλγόριθμος Ενισχυτικής Μάθησης αποτελείται από :

- **Καταστάσεις και Παρατηρήσεις:** Οι καταστάσεις (States) παρέχουν μια πλήρη περιγραφή του περιβάλλοντος, ενώ οι παρατηρήσεις (Observations) μπορεί να προσφέρουν μερική πληροφορία. Στην βαθιά RL, αυτές αναπαρίστανται με διανύσματα, μήτρες ή ανώτερης τάξης τανυστές.
- **Χώροι Ενεργειών:** Οι ενέργειες μπορεί να είναι διακριτές (πεπερασμένο σύνολο κινήσεων) ή συνεχείς (πραγματικές τιμές διανυσμάτων), επηρεάζοντας την πολυπλοκότητα και τις μεθόδους που χρησιμοποιούνται στους αλγορίθμους RL.
- **Πολιτικές:** Οι πολιτικές καθοδηγούν τις αποφάσεις του πράκτορα και μπορεί να είναι ντετερμινιστικές (σταθερές ενέργειες) ή στοχαστικές (κατανομές πιθανότητας πάνω από ενέργειες). Είναι παραμετροποιημένες και βελτιστοποιούνται για να βελτιώσουν την απόδοση του πράκτορα.
- **Διαδρομές/Επεισόδια:** Μια διαδρομή είναι μια ακολουθία καταστάσεων και ενεργειών που πραγματοποιεί ο πράκτορας. Ο στόχος του πράκτορα είναι να μεγιστοποιήσει την αθροιστική ανταμοιβή σε αυτές τις διαδρομές.
- **Ανταμοιβή και Επιστροφή:** Η συνάρτηση ανταμοιβής παρέχει ανατροφοδότηση για τις ενέργειες του πράκτορα. Η επιστροφή είναι η αθροιστική ανταμοιβή σε μια διαδρομή, η οποία μπορεί να είναι με πεπερασμένο ορίζοντα ή άπειρο ορίζοντα (εξαρτώμενο από ένα παράγοντα γ που αποκαλείται discount factor).

Το κεντρικό πρόβλημα βελτιστοποίησης στην RL είναι να βρεθεί μια πολιτική που μεγιστοποιεί την αναμενόμενη επιστροφή $J(\pi) = \int P(\tau|\pi) \cdot R(\tau) = E_{\tau\sim\pi}[R(\tau)]$. Αυτό περιλαμβάνει την εξερεύνηση και την αξιολόγηση διαφορετικών πολιτικών για να εντοπιστεί αυτή που αποδίδει τις υψηλότερες αθροιστικές ανταμοιβές, η βέλτιστη πολιτική $\pi^* = argmax_\pi J(\pi)$.

Συναρτήσεις Αξίας
Ο παρακάτω πίνακας συνοψίζει τις κύριες συναρτήσεις αξίας στην RL, οι οποίες είναι απαραίτητες για την αξιολόγηση και τη βελτιστοποίηση των πολιτικών.

Συναρτήσεις Αξίας στην Ενισχυτική Μάθηση	
$V^\pi(s) = E_{\tau\sim\pi}[R(\tau)\|s_o = s]$ $V^\pi(s) = E_{a\sim\pi}[Q^\pi(s,a)]$	Η αναμενόμενη απόδοση ξεκινώντας από την κατάσταση s υπό την πολιτική π
$Q^\pi(s,a) = E_{\tau\sim\pi}[R(\tau)\|s_o = s, a_o = a]$	Η αναμενόμενη απόδοση ξεκινώντας από την κατάσταση s, κάνοντας την ενέργεια a, και στη συνέχεια ακολουθώντας την πολιτική π
$V^*(s) = max_\pi V^\pi(s)$	Η μέγιστη αναμενόμενη απόδοση ξεκινώντας από την κατάσταση sυπό την βέλτιστη πολιτική
$Q^*(s,a) = max_\pi Q^\pi(s,a)$	Η μέγιστη αναμενόμενη απόδοση ξεκινώντας από την κατάσταση s, κάνοντας την ενέργεια a, και στη συνέχεια ακολουθώντας την βέλτιστη πολιτική

Table 1 : Συναρτήσεις Αξίας στην Ενισχυτική Μάθηση

Ενισχυτική Μάθηση Χωρίς Μοντέλο & Με Μοντέλο

Η ενισχυτική μάθηση μπορεί να διακριθεί σε δύο κύριες κατηγορίες: **χωρίς μοντέλο** και **με μοντέλο**. Στην ενισχυτική μάθηση με μοντέλο, ο πράκτορας χρησιμοποιεί ένα μοντέλο του περιβάλλοντος για να προβλέψει μεταβάσεις καταστάσεων και ανταμοιβές, επιτρέποντας τον προγραμματισμό και τη βελτιστοποίηση στρατηγικών. Αυτή η προσέγγιση μπορεί να βελτιώσει σημαντικά την αποδοτικότητα των δειγμάτων, όπως αποδεικνύεται από εφαρμογές όπως το AlphaZero (ένα από τα καλύτερο λογισμικά για σκάκι, shogi και go). Ωστόσο, η δημιουργία ακριβούς και αξιόπιστου μοντέλου είναι δύσκολη και χρονοβόρα, και μπορεί να οδηγήσει σε υποβέλτιστη απόδοση λόγω μεροληψίας στο μοντέλο. Αντιθέτως, η ενισχυτική μάθηση χωρίς μοντέλο, αν και χάνει τις δυνατότητες που προσφέρει η χρήση μοντέλου, είναι γενικά ευκολότερη στην υλοποίηση και προσαρμογή, και έχει αναπτυχθεί περισσότερο.

Βελτιστοποίηση της Πολιτικής

Οι μέθοδοι βελτιστοποίησης πολιτικής, όπως οι A2C και PPO, προσαρμόζουν τις παραμέτρους της πολιτικής είτε άμεσα μέσω ανόδου κλίσης (gradient ascent) είτε έμμεσα βελτιστοποιώντας τοπικές προσεγγίσεις της απόδοσης. Αυτές οι μέθοδοι τείνουν να είναι πιο σταθερές και αξιόπιστες, αν και ενδέχεται να είναι λιγότερο αποδοτικές σε δείγματα σε σύγκριση με τις μεθόδους Q-learning που

βελτιστοποιούν έμμεσα την απόδοση του πράκτορα εκπαιδεύοντας τη συνάρτηση Q, χαρακτηριστικό παράδειγμα είναι οι αλγόριθμοι SAC και DDPG.

Ενισχυτική Μάθηση στην Βιομηχανία

Η ενσωμάτωση της ενισχυτικής μάθησης στη συντήρηση περιλαμβάνει τη χρήση πλαισίων όπως MDP, POMDP, και S-MDP για τη μοντελοποίηση του περιβάλλοντος και την ανάπτυξη αποτελεσματικών στρατηγικών συντήρησης. Τόσο οι μέθοδοι ενισχυτικής μάθησης χωρίς μοντέλο όσο και με μοντέλο μπορούν να εφαρμοστούν, με τις μεθόδους χωρίς μοντέλο να είναι πιο πρακτικές σε σύνθετα περιβάλλοντα όπου η ακριβής μοντελοποίηση είναι δύσκολη.

Εισαγωγή

Το Industry 4.0 περιλαμβάνει την ενσωμάτωση του Internet of Things (IoT) για τη δημιουργία ενός συστήματος αλληλεπίδρασης μεταξύ του ψηφιακού και του φυσικού βιομηχανικού κόσμου. Οι συσκευές IoT με ενσωματωμένους αισθητήρες γίνονται όλο και πιο προηγμένες και προσιτές. Σύμφωνα με το Statista (Vailshery, 2024), υπάρχουν περίπου 15,9 δισεκατομμύρια συνδεδεμένες συσκευές IoT από το 2023 και την επόμενη δεκαετία ο αριθμός αυτός αναμένεται να αυξηθεί σε σχεδόν 40 δισεκατομμύρια. Οι αναλύσεις σε πραγματικό χρόνο σε συνδυασμό με τη χρήση αισθητήρων και τεχνητή νοημοσύνη (AI) επιτρέπουν εξελιγμένες πλατφόρμες που βασίζονται στο Cloud. Η επέκταση της συνδεσιμότητας υψηλής ταχύτητας 5G διευκολύνει την υλοποίηση σύνθετων εφαρμογών AI που μπορούν να βελτιώσουν τις βιομηχανικές λειτουργίες και να υποστηρίξουν αποτελεσματικά εξελιγμένα συστήματα.

Σύμφωνα με τους (Burke , Hartigan, & Sniderman , 2017), ένα SmartFactory έχει σχεδιαστεί για να αυτοβελτιστοποιεί την απόδοσή του μαθαίνοντας από τις νέες συνθήκες και προσαρμόζεται σε σχεδόν πραγματικό χρόνο. Το Predictive Maintenance (PdM) περιλαμβάνει την ανάλυση δεδομένων από διάφορες πηγές -όπως αισθητήρες που είναι εγκατεστημένοι σε κρίσιμα μηχανήματα, πληροφορίες από συστήματα προγραμματισμού επιχειρησιακών πόρων (ERP), δεδομένα παραγωγής και συστήματα διαχείρισης συντήρησης- για την πρόβλεψη και την προληπτική αντιμετώπιση βλαβών του εξοπλισμού. Αυτά τα συστήματα διαχείρισης SmartFactory χρησιμοποιούν προηγμένα μοντέλα πρόβλεψης για την πρόβλεψη πιθανών προβλημάτων και τη λήψη προληπτικών μέτρων.

Οι (Ben-Daya, Dufuaa, & Raouf, 2012) διαπίστωσαν ότι το κόστος συντήρησης μπορεί να αντιπροσωπεύει μεταξύ 15% και 40% των εξόδων παραγωγής σε διάφορους κλάδους. Παρά τα οφέλη, πολλοί κατασκευαστές διστάζουν να υιοθετήσουν πρακτικές SmartFactory. Οι (Laape , Dollar, & Cotteleer , 2020) ανέφεραν ότι σχεδόν το 65% των κατασκευαστών που συμμετείχαν στην έρευνα είχαν σημειώσει μικρή ή καθόλου πρόοδο στις πρωτοβουλίες τους για το SmartFactory. Οι πιέσεις κόστους στη μεταποίηση

σημαίνουν ότι η βέλτιστη προληπτική συντήρηση μπορεί να συμβάλει στη μείωση των δαπανών συντήρησης, διασφαλίζοντας παράλληλα τη συνεχή παραγωγή.

Έχουν χρησιμοποιηθεί διάφορες προσεγγίσεις για την αντιμετώπιση των προκλήσεων PdM. Σε αυτές περιλαμβάνονται τεχνικές βελτιστοποίησης μεικτών ακέραιων και πολλαπλών κριτηρίων, όπως η βελτιστοποίηση Pareto (Saydam & Frangopol, 2015) μέθοδοι μηχανικής μάθησης (ML), όπως τα δέντρα αποφάσεων (Frangopol, Lin, & Estes, 1997) τα random forests (Kabir , Foggo, & Yu, 2018), τα gradient boosted δέντρα (XGboost) (Ma, Guo, & Mao, 2020) και τεχνικές ML χωρίς επίβλεψη, όπως η ανάλυση κύριων συνιστωσών (PCA) (Eke, Aka-Ngnui, & Glerc, 2017). Έχουν επίσης εφαρμοστεί μέθοδοι ML με επίβλεψη, όπως η παλινδρόμηση (Susto, Wan, & Pampuri, An adaptive machine learning decision system for flexible predictive maintenance , 2014) και οι μηχανές διανυσμάτων υποστήριξης (SVM) (Ding, He, & Zi , 2008), (Susto , Schirru, & Pampuri, A predictive maintenance system for integral type faults based on support vector machines: an application to ion implantation, 2013).Τεχνικές βαθιάς μάθησης όπως η μακράς βραχυπρόθεσμης μνήμης (LSTM) έχουν χρησιμοποιηθεί για την εκτίμηση της εναπομένουσας ωφέλιμης ζωής (RUL) (Sayyad , Kumar, & Bongale, 2022), (Zheng , Ristovski, & Farahat, 2017) και τα νευρωνικά δίκτυα συνελίξεων (CNN) έχουν χρησιμοποιηθεί για regression (Sateesh Babu, Zhao, & Li, 2016).

Οι πρώτες προσπάθειες επίλυσης τέτοιου είδους προβλημάτων χρησιμοποιούσαν συχνά ντετερμινιστικές προσεγγίσεις, οι οποίες είχαν περιορισμένη αποτελεσματικότητα λόγω της πολυπλοκότητας και της μη γραμμικότητας των βιομηχανικών διεργασιών. Η εισαγωγή της ML βελτίωσε τη μοντελοποίηση των συστημάτων, αλλά η μη επαρκής ποσότητα επισημασμένων δεδομένων από εμπειρογνώμονες αποτελούσε πρόβλημα, οδηγώντας σε προσεγγιστικές λύσεις.

Η ενισχυτική μάθηση (RL), συμπεριλαμβανομένης της βαθιάς ενισχυτικής μάθησης, εμφανίζεται ως μια πολλά υποσχόμενη λύση για αυτούς τους περιορισμούς. Οι παραδοσιακές μέθοδοι βιομηχανικού ελέγχου και σχεδιασμού αδυνατούν να ανταποκριθούν σε πολύπλοκες μη γραμμικές διαδικασίες, αλλά η RL προσφέρει έναν αυτόνομο μηχανισμό μάθησης για την ανάπτυξη βέλτιστων λύσεων χωρίς μοντέλα (Lewis et al. 2012, Sutton and Barto 2018). Δεδομένου ότι η PdM περιλαμβάνει προβλήματα βελτιστοποίησης, η RL αποτελεί πιθανή λύση για την αντιμετώπιση αυτών των προκλήσεων.

Το επίκεντρο της παρούσας διπλωματικής είναι το πρόβλημα πρόβλεψης της φθοράς μηχανών CNC (Computer Numerical Control). Ένα μηχάνημα CNC είναι ένα αυτοματοποιημένο εργαλείο κατασκευής που χρησιμοποιεί προγραμματισμένο λογισμικό υπολογιστή για τον έλεγχο της κίνησης και της λειτουργίας των μηχανημάτων (τρυπάνια, φρέζες, βίντι), επιτρέποντας την ακριβή και αποδοτική παραγωγή σύνθετων εξαρτημάτων από διάφορα υλικά. Τα δεδομένα παρέχονται από την PHM (Prognostics and Health Management) Society και αποτέλεσαν μέρος του 2010 PHM Data Challenge. Ο στόχος είναι να προβλεφθεί η φθορά των flutes (φρεζοτρύπανων) σε διάφορες φρέζες CNC υψηλής ταχύτητας χρησιμοποιώντας μετρήσεις από αισθητήρες δυναμόμετρου, επιταχυνσιόμετρου και ακουστικής εκπομπής.

Στο πλαίσιο της διπλωματικής για την υλοποίηση ενός RL PdM Agent, χρησιμοποιήθηκε μια εξειδικευμένη διάταξη, όπως απεικονίζεται στο παρακάτω διάγραμμα:

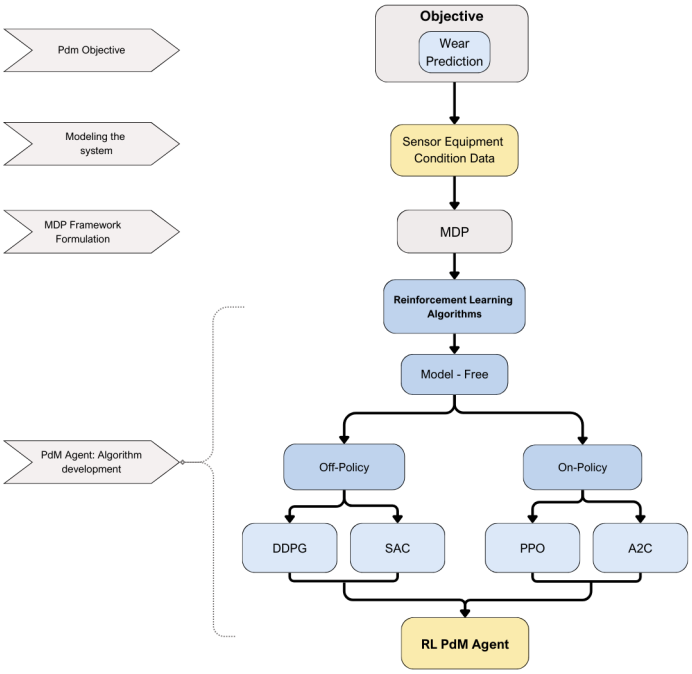


Figure 3: Το διάγραμμα υλοποίησης του Προβλήματος Πρόβλεψης Φθοράς αυτής της διπλωματικής εργασίας.

Σε αυτή την διάταξη, ο πρωταρχικός στόχος είναι η πρόβλεψη της φθοράς, η οποία βασίζεται σε δεδομένα από την παρακολούθηση της κατάστασης του εξοπλισμού με αισθητήρες. Τα δεδομένα αυτά ενσωματώνονται στο πλαίσιο διαδικασίας απόφασης Markov (MDP), το οποίου ο ρόλος είναι η διατύπωση και η μορφοποίηση του προβλήματος με τρόπο που να είναι κατάλληλος για αλγορίθμους ενισχυτικής μάθησης (RL).

Οι αλγόριθμοι ενισχυτικής μάθησης που εφαρμόζονται είναι εξειδικευμένοι αλγόριθμοι που δεν χρησιμοποιούν μοντέλο (model-free). Αυτοί οι αλγόριθμοι κατηγοριοποιούνται σε μεθόδους εκτός πολιτικής (off-policy) και σε μεθόδους εντός πολιτικής (on-policy). Οι off-policy μέθοδοι περιλαμβάνουν τις Deep Deterministic Policy Gradient (DDPG) και Soft Actor-Critic (SAC), ενώ οι on-policy μέθοδοι περιλαμβάνουν τις Proximal Policy Optimization (PPO) και Advantage Actor-Critic (A2C). Ο τελικός στόχος είναι να αναπτυχθεί ένας πράκτορας PdM με βάση το RL, ικανός να λαμβάνει αυτόνομες αποφάσεις για την πρόβλεψη της φθοράς και την αποτελεσματική αποφυγή πιθανών λειτουργικών βλαβών σε μηχανές CNC.

Στις ενότητες που ακολουθούν, αυτή η διάταξη θα αναπτυχθεί λεπτομερέστερα, συζητώντας κάθε συστατικό της και το ρόλο του στη διαμόρφωση ενός αποτελεσματικού RL PdM πράκτορα.

Προσέγγιση

Στην παρούσα ενότητα, θα αναλυθεί η διαδικασία μοντελοποίησης ενός προβλήματος ενισχυτικής μάθησης (RL) χρησιμοποιώντας δεδομένα που συλλέγονται από αισθητήρες. Αρχικά, θα πραγματοποιηθεί η επεξεργασία των δεδομένων αυτών και έπειτα θα μετατροπεί το πρόβλημα σε μια διαδικασία απόφασης Markov (MDP). Στη συνέχεια, μέσω του MDP θα αναπτυχθεί ένα κατάλληλο περιβάλλον για την εφαρμογή διαφόρων αλγορίθμων RL, οι οποίοι θα εκπαιδευτούν στο επεξεργασμένο σύνολο δεδομένων. Ύστερα από την εκπαίδευση, οι αλγόριθμοι θα αξιολογηθούν προκειμένου να καθοριστεί η αποτελεσματικότητά τους. Εάν η επίδοσή τους είναι καλή τότε μπορούν να χρησιμοποιηθούν για την εκτέλεση προβλέψεων σχετικών με την προγνωστική συντήρηση.

Δεδομένα από Αισθητήρες

Τα δεδομένα συλλέγονται από διάφορους αισθητήρες συνδεδεμένους σε μηχανήματα, καταγράφοντας χρονοσειρές παραμέτρων όπως δονήσεις, θερμοκρασία και πίεση. Αυτά τα δεδομένα οργανώνονται σε πίνακες όπου κάθε σειρά αντιστοιχεί σε μια μέτρηση αισθητήρα σε συγκεκριμένο χρονικό διάστημα. Η συνεχής παρακολούθηση αυτών των παραμέτρων επιτρέπει την ανίχνευση ανωμαλιών.

Η κατάλληλη διαμόρφωση και προεπεξεργασία των δεδομένων παίζει σημαντικό ρόλο στην ακρίβεια και αξιοπιστία των προβλέψεων. Αυτό περιλαμβάνει τον καθαρισμό των δεδομένων για την αφαίρεση θορύβου και σφαλμάτων και την κανονικοποίηση για τη διασφάλιση συνέπειας. Η οπτικοποίηση των δεδομένων είναι το πρώτο βήμα στην προεπεξεργασία, καθώς επιτρέπει την εξερεύνηση και κατανόηση των χαρακτηριστικών, των κατανομών και των πιθανών ανωμαλιών, διευκολύνοντας την ανίχνευση σφαλμάτων και την επιλογή των κατάλληλων χαρακτηριστικών για τη μετέπειτα ανάλυση και μοντελοποίηση.

Επεξεργασία Δεδομένων

Η προεπεξεργασία και η εξαγωγή χαρακτηριστικών είναι θεμελιώδη βήματα για την προετοιμασία των δεδομένων για μοντέλα προγνωστικής συντήρησης. Αυτά τα βήματα περιλαμβάνουν εκτός από τον καθαρισμό και την κανονικοποίηση, και την εξαγωγή σχετικών χαρακτηριστικών από τα ακατέργαστα δεδομένα αισθητήρων, προκειμένου να ενισχυθεί η ποιότητα των δεδομένων και να γίνουν πιο κατάλληλα για μοντέλα μηχανικής μάθησης. Η εξαγωγή χαρακτηριστικών βοηθά στη μετατροπή των ακατέργαστων δεδομένων σε μεταβλητές που είναι πιο ενημερωτικές σχετικά με την φθορά του εξοπλισμού ή συμπυκνώνουν την πληροφορία.

Τα χαρακτηριστικά αυτά μπορεί να είναι στατιστικά μεγέθη όπως μέσος όρος και διακύμανση, μετρικές επεξεργασίας σήματος όπως η ρίζα μέσου τετραγώνου (RMS) ή εξειδικευμένοι δείκτες.

Μέσω προσεκτικής προεπεξεργασίας και εξαγωγής χαρακτηριστικών, δημιουργείται ένα πιο διαχειρίσιμο και ενημερωτικό σύνολο δεδομένων, αποφεύγοντας την "κατάρα της διάστασης", η οποία συχνά συναντάται στην μηχανική μάθηση λόγο των πολυάριθμων διαστάσεων των προβλημάτων επίλυσης.

Επειδή τα δεδομένα από τους αισθητήρες είναι όλα χρονοεξαρτώμενα, η καλύτερη κατηγορία χαρακτηριστικών στην προγνωστική συντήρηση είναι τα χαρακτηριστικά στο χρονικό πεδίο. Δηλαδή χαρακτηριστικά που επιτρέπουν την ανάλυση των δεδομένων σε βάθος χρόνου.

Μοντελοποίηση της Διαδικασίας Απόφασης Markov

Για την πρόβλεψη της φθοράς με χρήση ενισχυτικής μάθησης, τα παρακάτω στοιχεία μιας διαδικασίας απόφασης Markov πρέπει να καθοριστούν:

- **Κατάσταση** S_t: Η κατάσταση σε κάθε χρονικό βήμα t αντιπροσωπεύεται από την κατάσταση των εξαρτημάτων της μηχανής και τις τιμές των αισθητήρων που είναι συνδεδεμένοι στη μηχανή. Περιλαμβάνει όλες τις πληροφορίες του περιβάλλοντος που είναι απαραίτητες για την πρόβλεψη της φθοράς.
- **Ενέργεια** a_t: Ο χώρος των ενεργειών αποτελείται από ένα n-διάστατο χώρο που αντιπροσωπεύει τη φθορά των n εξαρτημάτων της μηχανής που παρακολουθούμε. Η ενέργεια είναι η πρόβλεψη της φθοράς κάθε εξαρτήματος σε κάθε χρονικό βήμα.
- **Πιθανότητες Μετάβασης** (P): Οι πιθανότητες μετάβασης καθορίζουν πώς η κατάσταση αλλάζει ως συνέπεια των ενεργειών. Η μετάβαση εξαρτάται από την τρέχουσα κατάσταση και την ενέργεια και όχι από την ιστορία των προηγούμενων καταστάσεων και ενεργειών.
- **Ανταμοιβή** R_t: Η συνάρτηση ανταμοιβής παρέχει ανατροφοδότηση στον πράκτορα βάσει της ακρίβειας των προβλέψεών του. Μπορεί να υπολογιστεί με βάση τη διαφορά μεταξύ της προβλεπόμενης φθοράς και της πραγματικής φθοράς που παρατηρείται, χρησιμοποιώντας σύνθετες συναρτήσεις βαθμολογίας.
- **Αρχική κατανομή καταστάσεων** ρ_0: Καθορίζει την πιθανότητα των αρχικών καταστάσεων από τις οποίες ο πράκτορας ξεκινά τη διαδικασία λήψης αποφάσεων. Αυτή η κατανομή είναι σημαντική για την προαγωγή της ρεαλιστικής εκπαίδευσης και τη βελτίωση της γενίκευσης σε πραγματικά σενάρια.

Αλγόριθμοι Ενισχυτικής Μάθησης

Η ενότητα αυτή εξετάζει πώς η ενισχυτική μάθηση εφαρμόζεται στην πράξη και αναλύει την μαθηματική θεμελίωση του κάθε αλγορίθμου. Περιληπτικά μπορούμε να περιγράψουμε τους αλγορίθμους ως εξής:

- **Vanilla Policy Gradient** (VPG): Προσαρμόζει τις πιθανότητες των ενεργειών για τη μεγιστοποίηση των αποδόσεων, μαθαίνοντας από τις ενέργειες που λαμβάνονται βάσει της τρέχουσας πολιτικής.

- **Trust Region Policy Optimization** (TRPO): Ενημερώνει τις πολιτικές λαμβάνοντας υπόψη τη διαφορά KL για σταθερή και αποτελεσματική μάθηση, αποφεύγοντας μεγάλες διαφορές στην απόδοση.

- **Proximal Policy Optimization** (PPO): Βελτιστοποιεί τις πολιτικές με απλούστερες μεθόδους από το TRPO, διασφαλίζοντας ότι οι νέες πολιτικές δεν αποκλίνουν σημαντικά από τις παλαιότερες μέσω κλιπάρισματος (clipping).

- **Advantage Actor-Critic** (A2C): Συνδυάζει ταυτόχρονη συλλογή δεδομένων με παράλληλους εργαζόμενους για να μειώσει τη διακύμανση και να σταθεροποιήσει τη μάθηση.

- **Deep Deterministic Policy Gradient** (DDPG): Χρησιμοποιεί συνεχή διαστήματα ενεργειών και συνδυάζει την εκμάθηση Q-function και πολιτικής για αποδοτική λήψη αποφάσεων.

- **Twin Delayed DDPG** (TD3): Βελτιώνει την ευστάθεια της μάθησης σε σχέση με τον DDPG, ενσωματώνοντας τεχνικές όπως το Clipped Double-Q Learning, την καθυστερημένη ενημέρωση πολιτικής, και την εξομάλυνση στόχου πολιτικής.

- **Soft Actor-Critic** (SAC): Χρησιμοποιεί μια στοχαστική πολιτική με κανονικοποίηση εντροπίας για να πετύχει μια ισορροπία μεταξύ εξερεύνησης και εκμετάλλευσης. Αυτό επιτρέπει στο SAC να προσαρμόζεται δυναμικά στις μεταβαλλόμενες συνθήκες, προωθώντας τη μάθηση και αποφεύγοντας την πρόωρη σύγκλιση σε υπο βέλτιστες λύσεις.

Κριτήρια Αξιολόγησης

Οι βασικές μετρικές που χρησιμοποιούνται για την αξιολόγηση είναι το **μέσο μήκος επεισοδίου** και η **μέση ανταμοιβή επεισοδίου**.

Το μέσο μήκος επεισοδίου δείχνει τη μέση διάρκεια ενός επεισοδίου πριν φτάσει σε τελική κατάσταση. Στο πλαίσιο του PdM, ένα επεισόδιο μπορεί να αναπαριστά την περίοδο λειτουργίας ενός μηχανήματος πριν από την ανάγκη συντήρησης. Ένα μεγαλύτερο μέσο μήκος επεισοδίου πιθανόν να υποδηλώνει ότι η πολιτική που έμαθε ο αλγόριθμος είναι αποτελεσματική στην αποτροπή βλαβών και στην παράταση του χρόνου λειτουργίας του εξοπλισμού.

Η μέση ανταμοιβή επεισοδίου αντιπροσωπεύει τη μέση αθροιστική ανταμοιβή που λαμβάνεται ανά επεισόδιο. Στο πλαίσιο του PdM, η συνάρτηση ανταμοιβής μπορεί να περιλαμβάνει παράγοντες όπως η αποδοτικότητα του μηχανήματος, το κόστος συντήρησης, ο χρόνος αδράνειας, η συχνότητα εμφάνισης βλαβών και το μέγεθος της φθοράς των υλικών. Μια υψηλότερη μέση ανταμοιβή επεισοδίου υποδεικνύει ότι η πολιτική είναι αποτελεσματική στην εξισορρόπηση αυτών των παραγόντων.

Η διαδικασία αξιολόγησης περιλαμβάνει τη σύγκριση αυτών των μετρικών σε διάφορους αλγορίθμους και περιβάλλοντα. Η ροή εργασιών αξιολόγησης περιλαμβάνει τον ορισμό του περιβάλλοντος αξιολόγησης, την εκπαίδευση πολιτικών με διάφορους αλγορίθμους RL, τη διεξαγωγή πολλαπλών επεισοδίων και τον υπολογισμό των μετρικών και τη σύγκριση τους. Εκτός από το μέσο μήκος επεισοδίου και τη μέση ανταμοιβή επεισοδίου, λαμβάνονται υπόψη και η υπολογιστική αποδοτικότητα, η κλιμακωσιμότητα (scalability) και η προσαρμοστικότητα των αλγορίθμων.

Υλοποίηση & Αποτελέσματα

Τεχνική Υλοποίηση

Η υλοποίηση του προβλήματος χρησιμοποιεί ένα σύνολο τεχνολογιών το οποίο βασίζεται στην Python, αξιοποιώντας διάφορες βιβλιοθήκες και πλαίσια για την επεξεργασία δεδομένων και την ενισχυτική μάθηση.

Οι κύριες βοηθητικές βιβλιοθήκες περιλαμβάνουν τις: NumPy για αριθμητικούς υπολογισμούς, Pandas για αποδοτική διαχείριση δεδομένων και Matplotlib για την οπτικοποίηση δεδομένων.

Η ενισχυτική μάθηση αποτελεί το βασικό συστατικό του έργου. Χρησιμοποιήθηκε η βιβλιοθήκη **Stable Baselines 3**, που βασίζεται στο OpenAI Baselines. Αυτή η βιβλιοθήκη παρέχει αξιόπιστες υλοποιήσεις αλγορίθμων ενισχυτικής μάθησης, όπως οι PPO, A2C, DDPG, και SAC. Το Stable Baselines 3 είναι ιδιαίτερα φιλικό προς τον χρήστη, προσφέροντας ένα API που απλοποιεί τη ρύθμιση, την εκπαίδευση και την αξιολόγηση των μοντέλων RL. Η βιβλιοθήκη περιλαμβάνει επίσης μια σειρά από βοηθητικά εργαλεία που διευκολύνουν τη διαδικασία ανάπτυξης RL.

Η υλοποίηση χρησιμοποιεί διάφορες βοηθητικές λειτουργίες από το Stable Baselines 3, όπως το check_env για τη διασφάλιση της σωστής λειτουργίας των περιβαλλόντων, το VecFrameStack για τη στοίβαξη καρέ (frames), και το evaluate_policy για την αξιολόγηση των εκπαιδευμένων μοντέλων.

Επιπλέον, το TensorBoard χρησιμοποιείται για την παρακολούθηση και οπτικοποίηση των μετρικών εκπαίδευσης μοντέλων σε πραγματικό χρόνο. Η τεχνολογική στοίβα περιλαμβάνει επίσης βιβλιοθήκες όπως os και Pathlib για λειτουργίες συστήματος αρχείων, και το time για τη διαχείριση χρονοσχετικών λειτουργιών.

Σύνολο Δεδομένων

Το σύνολο δεδομένων που χρησιμοποιήθηκε για την εκπαίδευση των μοντέλων, προέρχεται από το 2010 PHM Society Conference Data Challenge. Το Data Challenge εστιάζει στην εκτίμηση του εναπομείναντος χρόνου ζωής (RUL) των φρεζών υψηλής ταχύτητας CNC. Οι συμμετέχοντες βαθμολογήθηκαν με βάση την ικανότητά τους να εκτιμήσουν τον εναπομένοντα χρόνο ζωής ενός κοπτικού εργαλείου από βολφραμικό καρβίδιο. Στην υλοποίηση αυτή, ο στόχος δεν είναι η πρόβλεψη του RUL αλλά της φθοράς των φρεζοτρύπανων των μηχανημάτων.

Το σύνολο δεδομένων περιλαμβάνει έξι αρχεία κοπτικών εργαλείων, από c1 (cutter 1) έως c6. Στο διαγωνισμό τα αρχεία c1, c4 και c6 χρησιμοποιήθηκαν ως εκπαιδευτικά δεδομένα, ενώ τα c2, c3 και c5 ως δεδομένα δοκιμής. Κάθε αρχείο εκπαίδευσης περιέχει ένα αρχείο "wear" που καταγράφει τη φθορά μετά από κάθε κοπή σε χιλιοστά (10^{-3} m) και ένα φάκελο με 315 αρχεία δεδομένων απόκτησης, το καθένα για κάθε κοπή. Γιαυτό το λόγο στην υλοποίηση αυτή, δεδομένου ότι δεν υπήρχε πρόσβαση στα αρχεία φθοράς του συνόλου δοκιμής, χρησιμοποιήθηκαν τα αρχεία των κοπτικών εργαλείων 4 και 6 ως σύνολο δοκιμής. Για την εκπαίδευση των μοντέλων, χρησιμοποιήθηκε το αρχείο c1 που αντιστοιχεί στο κοπτικό εργαλείο 1.

Τα αρχεία δεδομένων κοπής περιέχουν επτά στήλες που αντιστοιχούν σε:

1. Δύναμη (N) στη διάσταση X
2. Δύναμη (N) στη διάσταση Y
3. Δύναμη (N) στη διάσταση Z
-
4. Δόνηση (g) στη διάσταση X
5. Δόνηση (g) στη διάσταση Y
6. Δόνηση (g) στη διάσταση Z
-
7. AE-RMS (V)

Οι παράμετροι του κοπτικού εργαλείου ήταν ταχύτητα ατράκτου 10400 RPM, ταχύτητα προώθησης 1555 mm/min, βάθος κοπής Y (ακτινικό) 0.125 mm και βάθος κοπής Z (αξονικό) 0.2 mm. Τα δεδομένα αποκτήθηκαν με συχνότητα 50 KHz ανά κανάλι. Για περισσότερες πληροφορίες σχετικά με το τρόπο παραγωγής του συνόλου δεδομένων (Li, et al., 2009).

Τα δεδομένα για να χρησιμοποιηθούν στην εκπαίδευση επεξεργάστηκαν κατάλληλα ώστε να μειωθεί ο αριθμός των διαστάσεων σε κάθε κατάσταση.

Επεξεργασία των Δεδομένων

Για την αποτελεσματική χρήση του εκτεταμένου συνόλου δεδομένων που παρέχεται, χρειάζεται να μετασχηματιστούν τα υψηλής διάστασης δεδομένα των χρονοσειρών σε μια διαχειρίσιμη, αποδοτική και χαμηλής διάστασης αναπαράσταση για κάθε κοπή. Αυτή η διαδικασία περιλαμβάνει τα ακόλουθα βήματα:

1. **Καθαρισμός Δεδομένων**:

Το σύνολο δεδομένων που παρέχεται από την PHM Society ήταν ήδη "καθαρό" χωρίς ελλιπείς τιμές.

2. **Εξαγωγή Χαρακτηριστικών**:

Κάθε αρχείο κοπής του συνόλου δεδομένων περιλαμβάνει περίπου 250.000 γραμμές, καθεμία με 7 παραμέτρους δεδομένων, οδηγώντας σε ένα πολύπλοκο χώρο παρατήρησης 1.750.000 διαστάσεων. Για να αντιμετωπιστεί το πρόβλημα της υψηλής διάστασης και να εξαχθούν σημαντικά χαρακτηριστικά, επικεντρωθήκαμε στην εξαγωγή 7 κρίσιμων χαρακτηριστικών χρόνου για κάθε παράμετρο που καταγράφεται από τους αισθητήρες:

Εξαγόμενα Χαρακτηριστικά Χρόνου	
Mean	$\frac{1}{N}\sum_{\iota=1}^{N} x_i$

Root Mean Square (RMS)	$\sqrt{\frac{1}{N}\sum_{i=1}^{N} {x_i}^2}$
Crest Factor	$\frac{max[x_i]}{RMS}$
The average Power	$\frac{1}{N}\sum_{i=1}^{N} {x_i}^2$
Skewness	$\frac{E\,[(x_i - \bar{x})^3]}{RMS^3}$
Kurtosis	$\frac{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^4}{RMS^4}$

Table 2 : Εξαγόμενα Χαρακτηριστικά Χρόνου

Εξάγοντας αυτά τα χαρακτηριστικά για κάθε μια από τις 7 παραμέτρους αισθητήρων (Δύναμη σε X, Y, Z άξονες, Δόνηση σε X, Y, Z άξονες και AE-RMS), η διάσταση των δεδομένων μειώθηκε σε 42 χαρακτηριστικά ανά κοπή.

3. **Κανονικοποίηση**:

Μετά την εξαγωγή των χαρακτηριστικών χρόνου, πραγματοποιήθηκε κανονικοποίηση σε εύρος 0 έως 1 χρησιμοποιώντας min-max scaling, για να εξασφαλιστεί η ομοιομορφία και να βελτιωθεί η απόδοση των αλγορίθμων μάθησης.

4. **Ενσωμάτωση Δεδομένων Φθοράς**:

Ως τελικό βήμα επεξεργασίας των δεδομένων, οι τιμές φθοράς των φρεζών περιλήφθηκαν στο σύνολο δεδομένων, διασφαλίζοντας ένα ολοκληρωμένο σύνολο δεδομένων που περιλαμβάνει τόσο τα εξαγόμενα χαρακτηριστικά όσο και τις μετρήσεις φθοράς.

Πρόσθετη Ανάλυση Δεδομένων:

Πραγματοποιήθηκε διερευνητική ανάλυση δεδομένων (EDA) για την αποκάλυψη προτύπων και σχέσεων εντός του συνόλου δεδομένων. Αυτό περιλάμβανε την οπτικοποίηση των παρεχόμενων χαρακτηριστικών και την εξέταση των συσχετίσεων μεταξύ διαφορετικών χαρακτηριστικών. Επίσης, αναλύθηκαν τα αρχεία φθοράς για την κατανόηση της εξέλιξης της φθοράς μετά από κάθε κοπή, εντοπίζοντας τη μέγιστη διαφορά φθοράς μεταξύ διαδοχικών κοπών, η οποία χρησιμοποιήθηκε ως υπερπαράμετρος σε κάθε μοντέλο MDP.

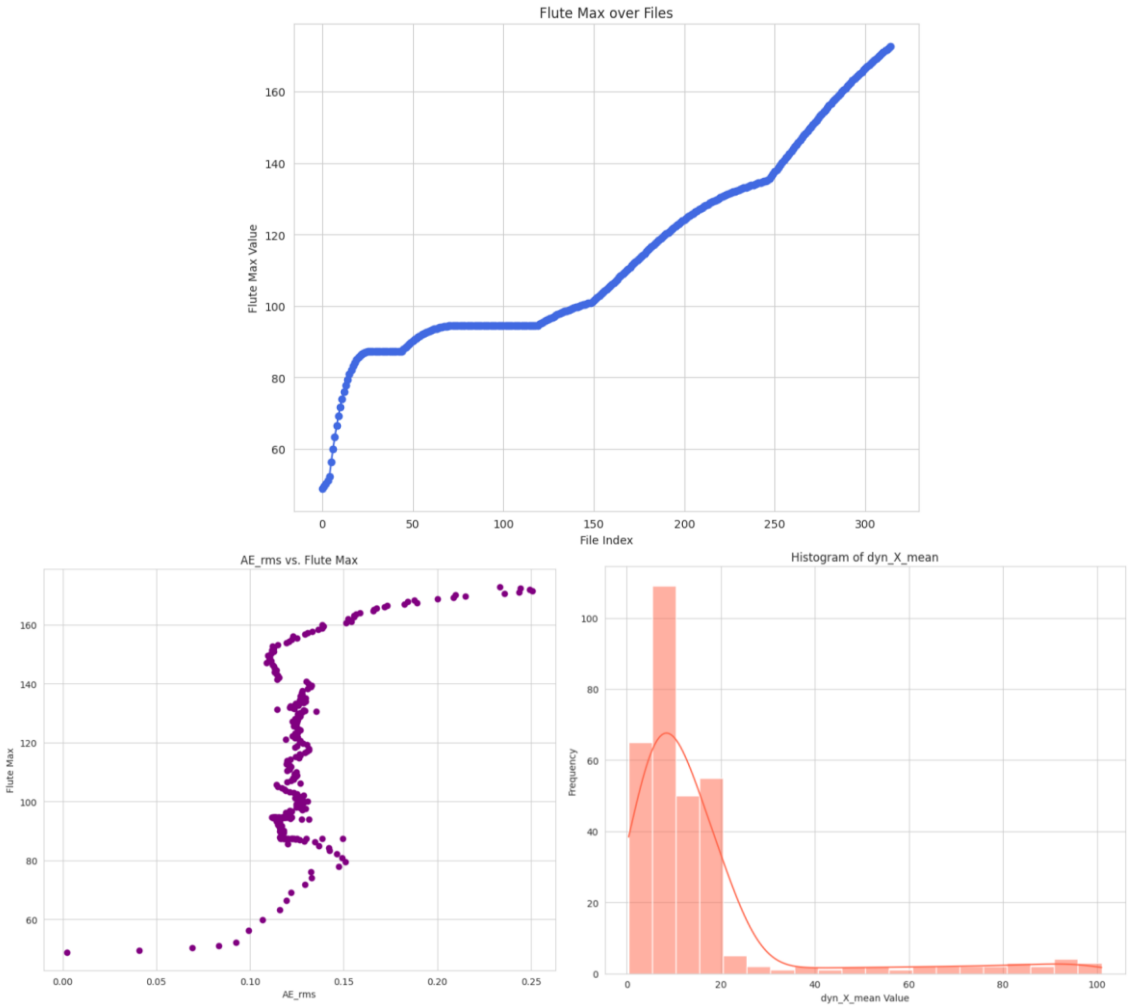


Figure 4 : Παραδείγματα γραφημάτων που δημιουργήθηκαν για σκοπούς διερευνητική ανάλυσης των δεδομένων

Αυτά τα βήματα προετοιμασίας των δεδομένων διασφαλίζουν ότι το σύνολο δεδομένων είναι διαχειρίσιμο και αποδοτικό για την εκπαίδευση των αλγορίθμων ενισχυτικής μάθησης, βελτιώνοντας την ακρίβεια των προβλέψεων φθοράς και την αποδοτικότητα των στρατηγικών συντήρησης.

Τα μοντέλα MDP

Η ενισχυτική μάθηση απαιτεί ένα περιβάλλον προσομοίωσης όπου ο πράκτορας μπορεί να ενεργεί και να λαμβάνει ανταμοιβές. Αυτό το περιβάλλον πρέπει να χρησιμοποιεί το επεξεργασμένο σύνολο δεδομένων και να συμπεριφέρεται με τρόπο που επιτρέπει στον πράκτορα να μάθει να προβλέπει με ακρίβεια τη φθορά της CNC φρέζας. Ένα MDP μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει αποτελεσματικά το περιβάλλον του προβλήματος. Για να οριστεί αυτό το MDP, είναι απαραίτητο να προσδιοριστούν τα εξής: καταστάσεις, ενέργειες, πιθανότητες μετάβασης και ανταμοιβές.

Στην εφαρμογή αυτή, δημιουργούνται τέσσερα περιβάλλοντα τα οποία χωρίζονται σε δύο κύριες κατηγορίες βάσει της μεθόδου πρόβλεψης και του υπολογισμού ανταμοιβών.

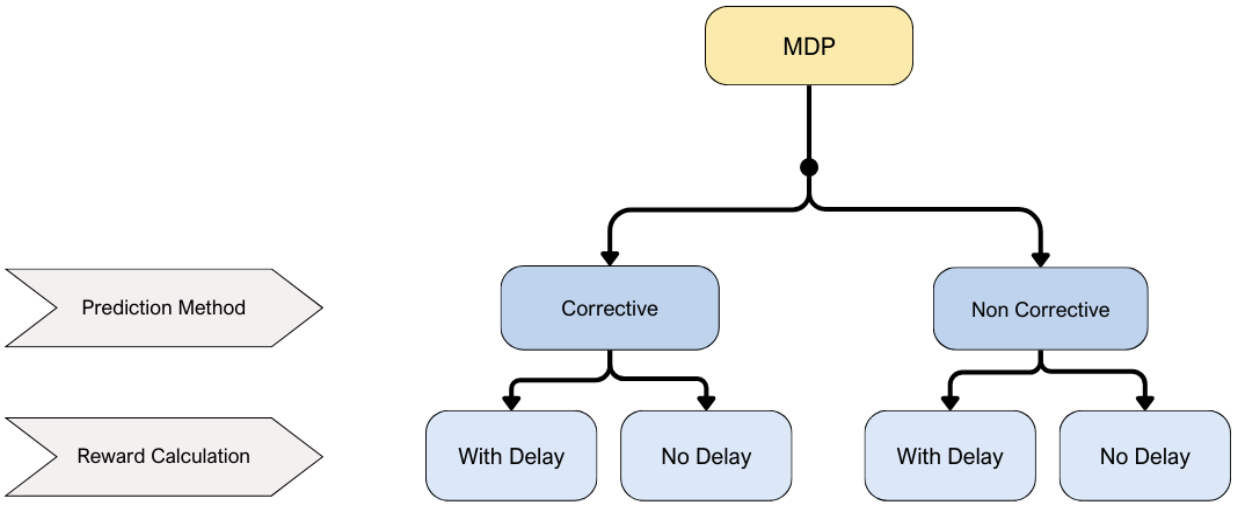


Figure 5 : Σχεδιάγραμμα με τα τέσσερα διαφορετικά Περιβάλλοντα της Υλοποίησης.

1. **Μέθοδος Πρόβλεψης:**
   - Διορθωτική Πρόβλεψη: Οι προβλέψεις του παράγοντα διορθώνονται βάσει των πραγματικών τιμών φθοράς.
   - Μη Διορθωτική Πρόβλεψη: Οι προβλέψεις ενημερώνονται σταδιακά βάσει των προηγούμενων προβλέψεων.

2. **Υπολογισμός Ανταμοιβών:**
   - Χωρίς Καθυστέρηση: Οι ανταμοιβές υπολογίζονται άμεσα βάσει της ακρίβειας πρόβλεψης.
   - Με Καθυστέρηση: Οι ανταμοιβές προσαρμόζονται με έναν τροποποιητή καθυστέρησης για να δώσουν έμφαση στα μακροπρόθεσμα οφέλη.

Κοινά Χαρακτηριστικά:

- Καταστάσεις (S): Η κατάσταση του περιβάλλοντος εκπροσωπείται από τον χώρο παρατήρησης (observation space) και αποτελείται από 42 χαρακτηριστικά χρόνου από τα δεδομένα αισθητήρων.
- Ενέργειες (A): Ο χώρος ενεργειών είναι ένας συνεχής χώρος τριών διαστάσεων (0-1) που εκπροσωπεί τις προβλέψεις του πράκτορα για τη φθορά τριών φρεζών.
- Αρχική Κατανομή Κατάστασης (ρ0): Η αρχική κατάσταση ορίζεται με βάση τις αρχικές τιμές φθοράς που προκύπτουν από τα δεδομένα.

Επιπλέον κοινά στοιχεία:

- Συνάρτηση Επαναφοράς: Επαναφέρει το περιβάλλον στην αρχική κατάσταση.
- Τερματισμός και Αποκοπή: Το περιβάλλον τερματίζεται όταν υπερβαίνεται ο μέγιστος αριθμός βημάτων ή εάν η ανταμοιβή είναι πολύ χαμηλή.
- Συνάρτηση Ανταμοιβής (PHM Score): Υπολογίζει την ανταμοιβή βάσει της ακρίβειας των προβλέψεων, χρησιμοποιώντας την ίδια συνάρτηση που χρησιμοποιήθηκε στο Data Challenge

Τα τέσσερα περιβάλλοντα που προκύπτουν είναι:

Environments | MDP Περιβάλλον	
Corrective with Delay Διορθωτικό με Καθυστέρηση	Οι προβλέψεις διορθώνονται βάσει των πραγματικών τιμών φθοράς και οι ανταμοιβές προσαρμόζονται με καθυστέρηση.
Corrective No Delay Διορθωτικό χωρίς Καθυστέρηση	Οι προβλέψεις διορθώνονται βάσει των πραγματικών τιμών φθοράς και οι ανταμοιβές υπολογίζονται άμεσα.
Non-Corrective with Delay Μη Διορθωτικό με Καθυστέρηση	Οι προβλέψεις ενημερώνονται βάσει των προηγούμενων προβλέψεων και οι ανταμοιβές προσαρμόζονται με καθυστέρηση.
Non-Corrective No Delay Μη Διορθωτικό χωρίς Καθυστέρηση	Οι προβλέψεις ενημερώνονται βάσει των προηγούμενων προβλέψεων και οι ανταμοιβές υπολογίζονται άμεσα.

Ο στόχος του πράκτορα είναι να προβλέψει με ακρίβεια τη φθορά τριών συγκεκριμένων φρεζοτρύπανων με βάση τα δεδομένα από τους αισθητήρες, και η επιτυχία του αξιολογείται βάσει της ακρίβειας των προβλέψεων του. Η ακρίβεια αυτή μετριέται μέσω των ανταμοιβών που λαμβάνει ο παράγοντας, οι οποίες εξαρτώνται από το πόσο κοντά βρίσκονται οι προβλέψεις του στις πραγματικές τιμές φθοράς. Οι ανταμοιβές υπολογίζονται με συγκεκριμένες συναρτήσεις που λαμβάνουν υπόψη τις διαφορές μεταξύ των προβλεπόμενων και των πραγματικών τιμών φθοράς.

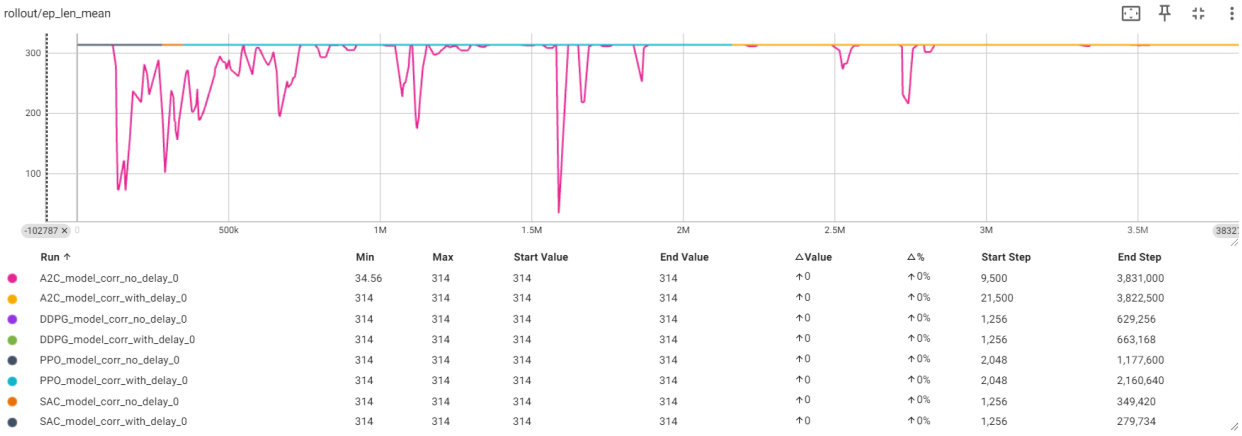
Τα τέσσερα διαφορετικά περιβάλλοντα που δημιουργήθηκαν βοηθούν στην εκτίμηση των διαφορετικών προσεγγίσεων πρόβλεψης και υπολογισμού ανταμοιβών, επιτρέποντας τη σύγκριση και την αξιολόγηση της αποδοτικότητας των αλγορίθμων ενισχυτικής μάθησης που εφαρμόζονται.

Εκπαίδευση Αλγορίθμων Ενισχυτικής Μάθησης

Σε αυτή την ενότητα περιγράφεται η διαδικασία εκπαίδευσης των μοντέλων ενισχυτικής μάθησης. Χρησιμοποιήθηκαν τέσσερις αλγόριθμοι σε τέσσερα διαφορετικά περιβάλλοντα, με αποτέλεσμα να δημιουργηθούν συνολικά δεκαέξι διαφορετικά μοντέλα. Η εκπαίδευση πραγματοποιήθηκε στο επεξεργασμένο σύνολο δεδομένων του κόπτη 1 (train set).

Η διαδικασία εκπαίδευσης εκτελείται σε επαναλήψεις, με κάθε επανάληψη να περιλαμβάνει έναν καθορισμένο αριθμό χρονικών βημάτων. Μετά από κάθε επανάληψη, τα μοντέλα αξιολογούνται και οι μετρικές απόδοσής τους καταγράφονται.

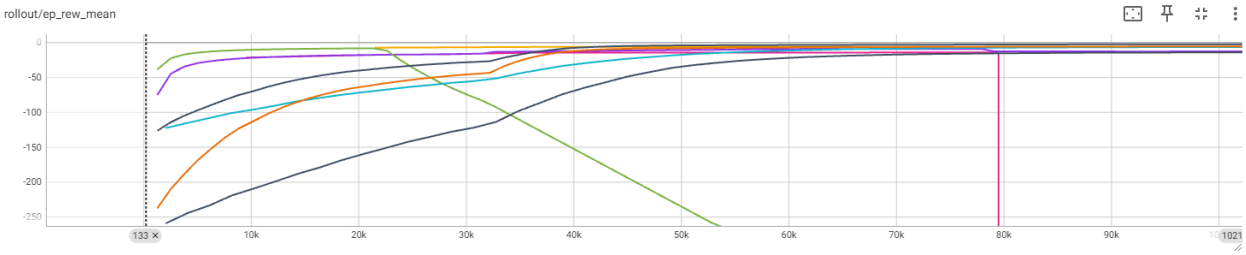
Διορθωτικό Περιβάλλον



Run	Min	Max	Start Value	End Value	ΔValue	Δ%	Start Step	End Step
A2C_model_corr_no_delay_0	34.56	314	314	314	↑0	↑0%	9,500	3,831,000
A2C_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	21,500	3,822,500
DDPG_model_corr_no_delay_0	314	314	314	314	↑0	↑0%	1,256	629,256
DDPG_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	1,256	663,168
PPO_model_corr_no_delay_0	314	314	314	314	↑0	↑0%	2,048	1,177,600
PPO_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	2,048	2,160,640
SAC_model_corr_no_delay_0	314	314	314	314	↑0	↑0%	1,256	349,420
SAC_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	1,256	279,734

Figure 6 : Μέση Διάρκεια Επεισοδίου ανά timestep για όλους τους Αλγόριθμους στο Διορθωτικό Περιβάλλον με / χωρίς καθυστέρηση

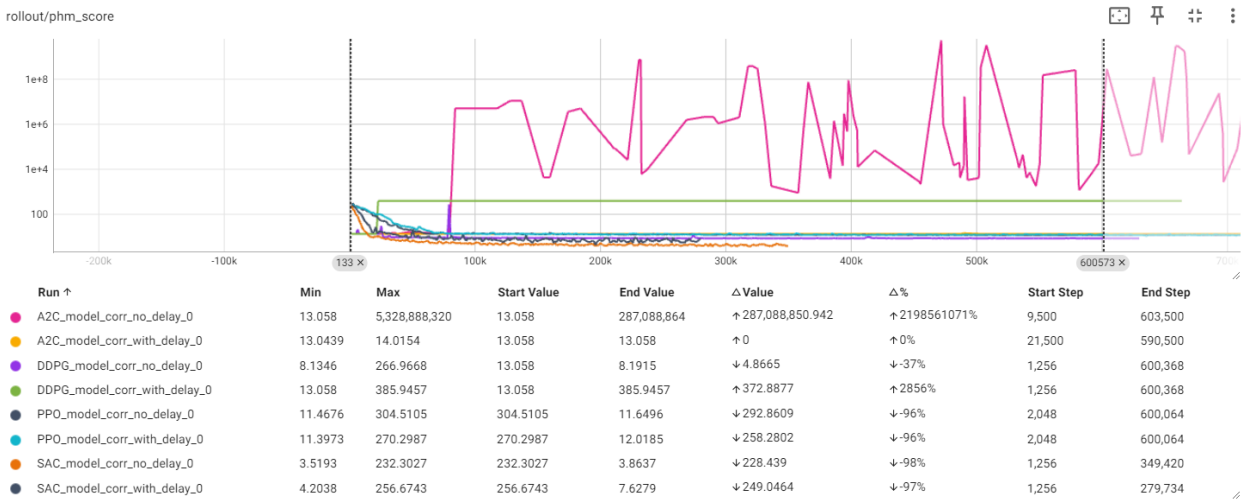
Το γράφημα δείχνει την τάση του μοντέλου A2C "χωρίς καθυστέρηση" να παρουσιάζει σημαντικές διακυμάνσεις (η επιλογή εύρους είναι γύρω στα 3,8 εκατομμύρια βήματα). Ταυτόχρονα, το διάγραμμα δείχνει ότι όλοι οι άλλοι αλγόριθμοι διατηρούν μια σταθερή μέση διάρκεια επεισοδίου στη μέγιστη τιμή (314), η οποία είναι επιθυμητή καθώς έχουν κορεστεί.



Run	Min	Max	Start Value	End Value	ΔValue	Δ%	Start Step	End Step
A2C_model_corr_no_delay_0	-15,418,906	-14.0199	-21.417	-15,418,906	↓15,418,884.583	↑71993550%	9,500	115,000
A2C_model_corr_with_delay_0	-7.2217	-6.2164	-7.2217	-6.2981	↑0.9236	↓-13%	21,500	115,000
DDPG_model_corr_no_delay_0	-75.178	-9.0253	-75.178	-12.6649	↑62.5131	↓-83%	1,256	101,736
DDPG_model_corr_with_delay_0	-265.5265	-8.0956	-38.5167	-265.5265	↓227.0098	↑589%	1,256	101,736
PPO_model_corr_no_delay_0	-259.0568	-14.0944	-259.0568	-14.0944	↑244.9624	↓-95%	2,048	102,400
PPO_model_corr_with_delay_0	-122.3658	-6.3482	-122.3658	-6.3482	↑116.0176	↓-95%	2,048	102,400
SAC_model_corr_no_delay_0	-237.4886	-5.4398	-237.4886	-5.4474	↑232.0412	↓-98%	1,256	101,884
SAC_model_corr_with_delay_0	-126.2699	-2.3179	-126.2699	-2.3202	↑123.9497	↓-98%	1,256	101,884

Figure 7 : Μέση Ανταμοιβή Επεισοδίου ανά timestep για όλους τους Αλγόριθμους στο Διορθωτικό Περιβάλλον με / χωρίς καθυστέρηση

Το γράφημα δείχνει καθαρά την προσπάθεια του μοντέλου A2C "χωρίς καθυστέρηση" για μεγάλη εξερεύνηση (exploration). Ενώ τα περισσότερα μοντέλα τελικά σταθεροποιούνται, το καθένα δείχνει διαφορετικά επίπεδα βελτίωσης της απόδοσης. Το μοντέλο A2C "χωρίς καθυστέρηση" αντιμετωπίζει σημαντικές αρνητικές ανταμοιβές, αντανακλώντας τη δυσκολία του στην εκμάθηση αρχικά. Με τον καιρό, άλλα μοντέλα, ιδιαίτερα αυτά με διόρθωση καθυστέρησης, δείχνουν πιο σταθερές βελτιώσεις στις ανταμοιβές.

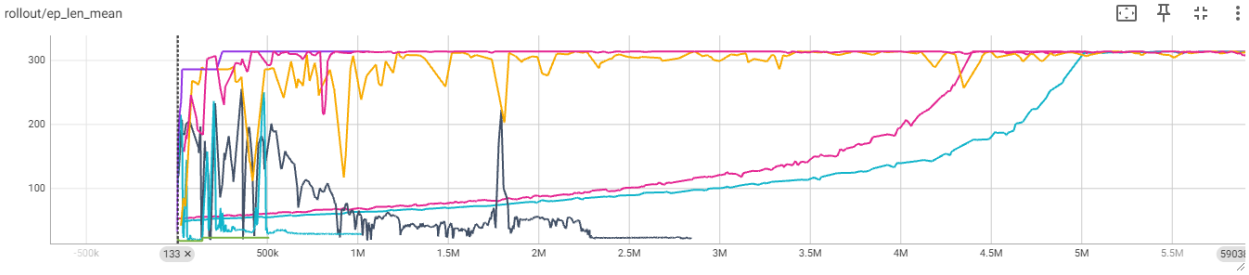


Run	Min	Max	Start Value	End Value	Δ Value	Δ %	Start Step	End Step
A2C_model_corr_no_delay_0	13.058	5,328,888,320	13.058	287,088,864	↑ 287,088,850.942	↑ 2198561071%	9,500	603,500
A2C_model_corr_with_delay_0	13.0439	14.0154	13.058	13.058	↑ 0	↑ 0%	21,500	590,500
DDPG_model_corr_no_delay_0	8.1346	266.9668	13.058	8.1915	↓ 4.8665	↓ -37%	1,256	600,368
DDPG_model_corr_with_delay_0	13.058	385.9457	13.058	385.9457	↑ 372.8877	↑ 2856%	1,256	600,368
PPO_model_corr_no_delay_0	11.4676	304.5105	304.5105	11.6496	↓ 292.8609	↓ -96%	2,048	600,064
PPO_model_corr_with_delay_0	11.3973	270.2987	270.2987	12.0185	↓ 258.2802	↓ -96%	2,048	600,064
SAC_model_corr_no_delay_0	3.5193	232.3027	232.3027	3.8637	↓ 228.439	↓ -98%	1,256	349,420
SAC_model_corr_with_delay_0	4.2038	256.6743	256.6743	7.6279	↓ 249.0464	↓ -97%	1,256	279,734

Figure 8 : Βαθμολογία PHM ανά timestep για όλους τους Αλγόριθμους στο Διορθωτικό Περιβάλλον με / χωρίς καθυστέρηση

Για άλλη μια φορά το A2C "χωρίς καθυστέρηση" παρουσιάζει μεγάλες διακυμάνσεις, ενώ το A2C "με καθυστέρηση" παραμένει σταθερό. Το μοντέλο DDPG "χωρίς καθυστέρηση" αρχικά παρουσιάζει διακυμάνσεις αλλά στη συνέχεια παραμένει σταθερό, και το DDPG με καθυστέρηση δείχνει αυξημένη εκμετάλλευση (exploitation) παραμένοντας σχεδόν σταθερό μετά την αρχική εξερεύνηση. Και τα δύο μοντέλα PPO διατηρούν σταθερά χαμηλές βαθμολογίες, υποδεικνύοντας ισχυρή απόδοση. Τα μοντέλα SAC επίσης αποδίδουν καλά με σταθερές, χαμηλές βαθμολογίες. Συνολικά, τα PPO και SAC είναι τα πιο σταθερά, με τη διόρθωση καθυστέρησης να βελτιώνει τη σταθερότητα για τα A2C και DDPG.

Μη Διορθωτικό Περιβάλλον



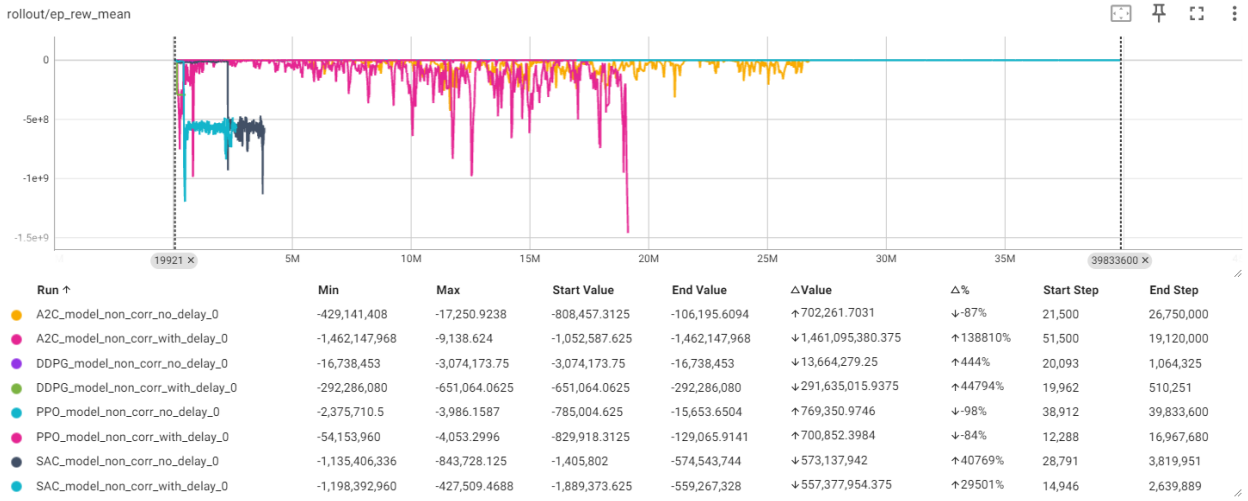
Run	Min	Max	Start Value	End Value	Δ Value	Δ %	Start Step	End Step
A2C_model_non_corr_no_delay_0	41.87	314	42.81	311.87	↑ 269.06	↑ 628%	21,500	5,880,000
A2C_model_non_corr_with_delay_0	157.28	314	157.28	308.91	↑ 151.63	↑ 96%	39,500	5,910,000
DDPG_model_non_corr_no_delay_0	32.25	314	33.25	314	↑ 280.75	↑ 844%	133	1,064,325
DDPG_model_non_corr_with_delay_0	18	23	20.75	23	↑ 2.25	↑ 11%	498	510,251
PPO_model_non_corr_no_delay_0	48.26	314	48.26	313.29	↑ 265.03	↑ 549%	38,912	5,883,904
PPO_model_non_corr_with_delay_0	53.33	314	53.33	314	↑ 260.67	↑ 489%	12,288	5,902,336
SAC_model_non_corr_no_delay_0	18.81	255.64	118.9868	22.51	↓ 96.4768	↓ -81%	9,043	2,844,187
SAC_model_non_corr_with_delay_0	18.41	250.26	84.0833	28.69	↓ 55.3933	↓ -66%	2,018	1,019,921

Figure 9 : Μέση Διάρκεια Επεισοδίου ανά timestep για όλους τους Αλγόριθμους στο Μη Διορθωτικό Περιβάλλον με / χωρίς καθυστέρηση

Από την παραπάνω σύγκριση προκύπτουν τα εξής:

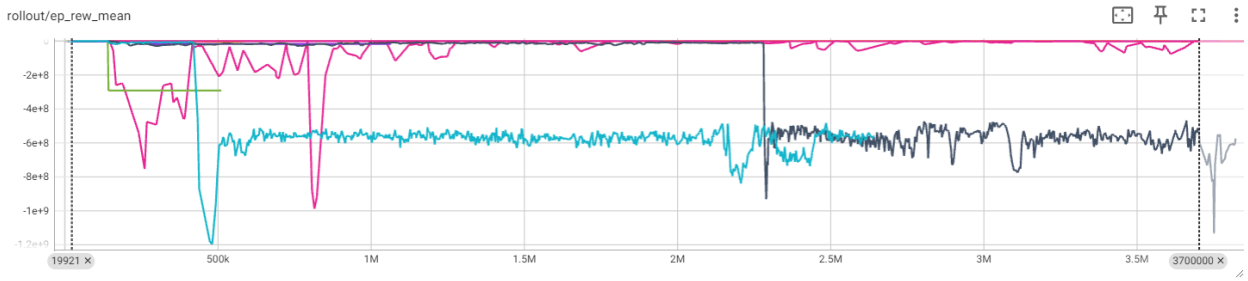
- **PPO**: Εμφανίζει την πιο σταθερή και ισχυρή απόδοση και στις δύο εκδόσεις (με και χωρίς διόρθωση καθυστέρησης), φτάνοντας τη μέγιστη διάρκεια επεισοδίου των 314, γεγονός που τον καθιστά τον πιο αξιόπιστο αλγόριθμο για αυτή την εργασία.

- **A2C**: Παρουσιάζει σημαντικές βελτιώσεις στη διάρκεια επεισοδίου. Η έκδοση χωρίς διόρθωση καθυστέρησης έχει ελαφρώς καλύτερη απόδοση, αλλά η διόρθωση καθυστέρησης βοηθά στη γρήγορη σταθεροποίηση.
- **DDPG**: Το μοντέλο χωρίς διόρθωση καθυστέρησης επιτυγχάνει τη μέγιστη διάρκεια επεισοδίου (314), ενώ το μοντέλο με διόρθωση καθυστέρησης παρουσιάζει ελάχιστη βελτίωση, υποδεικνύοντας ότι η διόρθωση καθυστέρησης δεν ωφελεί.
- **SAC**: Και τα δύο μοντέλα (με και χωρίς διόρθωση καθυστέρησης) έχουν σημαντική μείωση στη διάρκεια επεισοδίου, υποδεικνύοντας ότι ο αλγόριθμος αυτός δεν εκπαιδεύεται κατάλληλα.



Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	-429,141,408	-17,250.9238	-808,457.3125	-106,195.6094	↑702,261.7031	↓-87%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	-1,462,147,968	-9,138.624	-1,052,587.625	-1,462,147,968	↓1,461,095,380.375	↑138810%	51,500	19,120,000
DDPG_model_non_corr_no_delay_0	-16,738,453	-3,074,173.75	-3,074,173.75	-16,738,453	↓13,664,279.25	↑444%	20,093	1,064,325
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-651,064.0625	-292,286,080	↓291,635,015.9375	↑44794%	19,962	510,251
PPO_model_non_corr_no_delay_0	-2,375,710.5	-3,986.1587	-785,004.625	-15,653.6504	↑769,350.9746	↓-98%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	-54,153,960	-4,053.2996	-829,918.3125	-129,065.9141	↑700,852.3984	↓-84%	12,288	16,967,680
SAC_model_non_corr_no_delay_0	-1,135,406,336	-843,728.125	-1,405,802	-574,543,744	↓573,137,942	↑40769%	28,791	3,819,951
SAC_model_non_corr_with_delay_0	-1,198,392,960	-427,509.4688	-1,889,373.625	-559,267,328	↓557,377,954.375	↑29501%	14,946	2,639,889

Figure 10 : Μέση Ανταμοιβή Επεισοδίου ανά timestep για όλους τους Αλγόριθμους στο Μη Διορθωτικό Περιβάλλον με/χωρίς καθυστέρηση

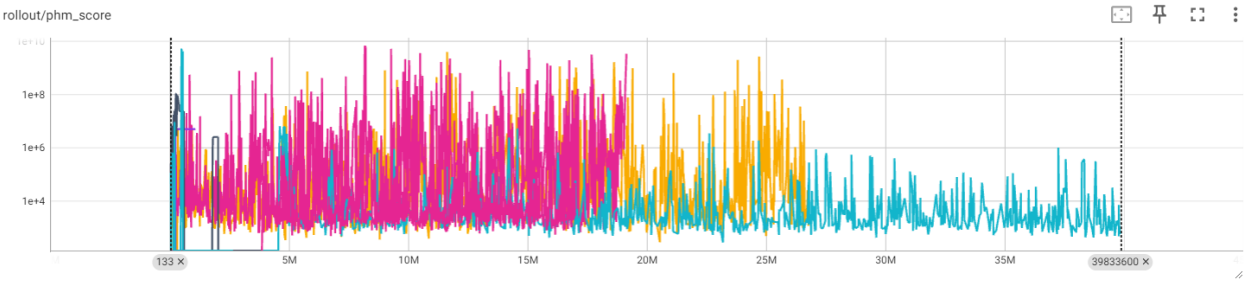


Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	-3,790,772.25	-47,070.875	-808,457.3125	-602,791.1875	↑205,666.125	↓-25%	21,500	3,691,500
A2C_model_non_corr_with_delay_0	-989,238,400	-34,740.9297	-1,052,587.625	-1,186,827.75	↓134,240.125	↑13%	51,500	3,700,000
DDPG_model_non_corr_no_delay_0	-16,738,453	-3,074,173.75	-3,074,173.75	-16,738,453	↓13,664,279.25	↑444%	20,093	1,064,325
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-651,064.0625	-292,286,080	↓291,635,015.9375	↑44794%	19,962	510,251
PPO_model_non_corr_no_delay_0	-1,434,418.5	-785,004.625	-785,004.625	-1,280,917.75	↓495,913.125	↑63%	38,912	3,741,696
PPO_model_non_corr_with_delay_0	-1,823,630.75	-768,119.625	-829,918.3125	-1,742,058.375	↓912,140.0625	↑110%	12,288	3,698,688
SAC_model_non_corr_no_delay_0	-933,118,208	-843,728.125	-1,405,802	-525,046,880	↓523,641,078	↑37249%	28,791	3,699,952
SAC_model_non_corr_with_delay_0	-1,198,392,960	-427,509.4688	-1,889,373.625	-559,267,328	↓557,377,954.375	↑29501%	14,946	2,639,889

Figure 11 : Επιλογή Μικρότερου Εύρους της Μέσης Ανταμοιβής Επεισοδίου ανά timestep για όλους τους Αλγόριθμους στο Μη Διορθωτικό Περιβάλλον με / χωρίς καθυστέρηση

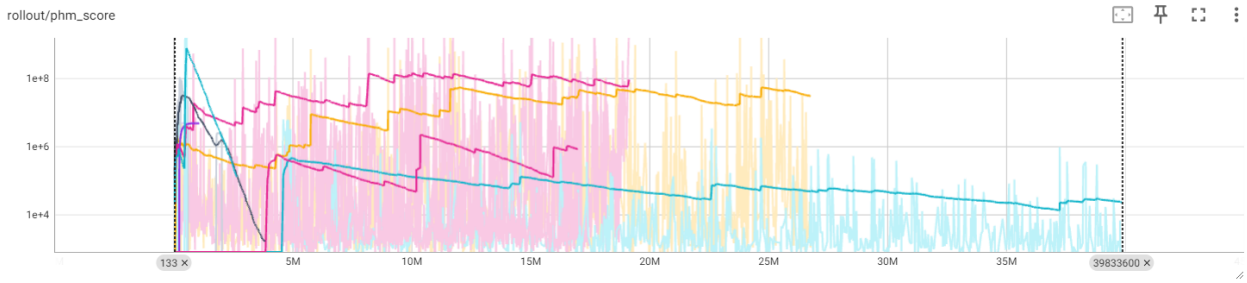
Οι τρεις παραπάνω εικόνες απεικονίζουν τη Μέση Ανταμοιβή Επεισοδίου ανά timestep. Η πρώτη εικόνα δείχνει το πλήρες εύρος της εκπαίδευσης, δείχνοντας την απόδοση των οκτώ μοντέλων συνολικά, με πολλές διακυμάνσεις στην κλίμακα των αποτελεσμάτων. Η δεύτερη εικόνα εστιάζει σε μικρότερο εύρος, παρέχοντας μια πιο λεπτομερή εικόνα της αρχικής απόδοσης και των φάσεων σταθεροποίησης για τα

μοντέλα, με τα A2C και SAC να παρουσιάζουν σημαντική μεταβλητότητα. Η τρίτη εικόνα των μοντέλων αποκαλύπτει ότι τα μοντέλα όπως το A2C και το DDPG παρουσιάζουν δραστικές πτώσεις στις ανταμοιβές πιθανώς λόγω της εξερεύνησης, όπως παρατηρήθηκε στα προηγούμενα γραφήματα, ενώ τα PPO και SAC είναι σχετικά σταθερά αλλά δείχνουν σημαντική μεταβλητότητα.



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	3,935,591,168	0	30,639.4805	↑30,639.4805	↑Infinity%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	474.9217	6,790,013,952	21,823.1406	3,374,888,704	↑3,374,866,880.8594	↑15464625%	51,500	19,120,000
DDPG_model_non_corr_no_delay_0	0	4,959,841	0	4,959,841	↑4,959,841	↑Infinity%	133	1,064,325
DDPG_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	498	510,251
PPO_model_non_corr_no_delay_0	0	6,386,266.5	0	565.6765	↑565.6765	↑Infinity%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	0	206,020,384	0	2,777.8508	↑2,777.8508	↑Infinity%	12,288	16,967,680
SAC_model_non_corr_no_delay_0	0	107,543,288	252,349.75	0	↓252,349.75	↓-100%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	0	5,341,618,176	115,093.0938	0	↓115,093.0938	↓-100%	14,946	2,639,889

Figure 12 : Βαθμολογία PHM ανά timestep για όλους τους αλγόριθμους στο μη διορθωτικό περιβάλλον με / χωρίς καθυστέρηση.



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	55,360,212.9076	0	30,636,905.3972	↑30,636,905.3972	↑Infinity%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	21,823.1406	147,271,019.9556	21,823.1406	93,594,384.123	↑93,572,560.9823	↑428777%	51,500	19,120,000
DDPG_model_non_corr_no_delay_0	0	4,954,853.8202	0	4,954,853.8202	↑4,954,853.8202	↑Infinity%	133	1,064,325
DDPG_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	498	510,251
PPO_model_non_corr_no_delay_0	0	472,978.1049	0	24,131.2842	↑24,131.2842	↑Infinity%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	0	2,295,272.6857	0	845,879.8488	↑845,879.8488	↑Infinity%	12,288	16,967,680
SAC_model_non_corr_no_delay_0	1,689.2414	32,404,227.9729	252,349.75	1,689.2414	↓250,660.5086	↓-99%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	22,558.2696	784,422,693.061	115,093.0937	133,538.6964	↑18,445.6027	↑16%	14,946	2,639,889

Figure 13 : "Smoothed" γράφημα της βαθμολογίας PHM ανά timestep για όλους τους αλγόριθμους στο μη διορθωτικό περιβάλλον με / χωρίς καθυστέρηση.

Ο στόχος είναι να ελαχιστοποιηθεί η Βαθμολογία PHM, αλλά λόγω της αρχικοποίησης της μεταβλητής phm_score ως μηδενική, η πραγματική ελάχιστη τιμή δεν εμφανίζεται εδώ. Ωστόσο, είναι εύκολο να εντοπιστεί γραφικά στο ομαλοποιημένο γράφημα ποια μοντέλα αποδίδουν καλύτερα. Το χειρότερο μοντέλο είναι αναμφισβήτητα το μοντέλο DDPG "με καθυστέρηση", το οποίο ποτέ δεν κατάφερε να φτάσει σε μήκος επεισοδίου 314 και συνεπώς δεν έλαβε ποτέ βαθμολογία PHM. Όλα τα άλλα μοντέλα έχουν φτάσει το μέγιστο μήκος επεισοδίου κατά τη διάρκεια της εκπαίδευσης. Το DDPG "χωρίς καθυστέρηση" έχει μια σταθερά υψηλή βαθμολογία PHM, η οποία είναι ανεπιθύμητη. Τα A2C, SAC και PPO εμφανίζουν μεγάλη μεταβλητότητα και υψηλή συχνότητα χαμηλών βαθμολογιών ακολουθούμενες από υψηλές κορυφώσεις. Αν και υπάρχει συνεχής βελτίωση σε όλες τις περιπτώσεις, η μεγάλη αστάθεια υποδηλώνει ότι τα μοντέλα εξακολουθούν να είναι σε κάποιο βαθμό ατελή.

Καλύτερα Μοντέλα Εκπαίδευσης

Όσον αφορά την εκπαίδευση, ο **PPO** είναι ο καλύτερος αλγόριθμος λόγω του γρήγορου χρόνου εκπαίδευσης ανά βήμα χρόνου και της συνολικής αποδοτικότητάς του. Είναι απλός στην υλοποίηση, απαιτεί λιγότερη υπολογιστική ισχύ και επιτυγχάνει σταθερά υψηλές και σταθερές ανταμοιβές, καθιστώντας τον την πιο αξιόπιστη επιλογή για το συγκεκριμένο πρόβλημα ενισχυτικής μάθησης (αυτό συμβαίνει κατά την εκπαίδευση, τα αποτελέσματα στα δοκιμαστικά σύνολα δεδομένων δεν θα ακολουθούν το ίδιο μοτίβο). Το **SAC** είναι ο δεύτερος καλύτερος αλγόριθμος, δείχνοντας εξαιρετική απόδοση με τον ταχύτερο χρόνο σύγκλισης και τον λιγότερο αριθμό βημάτων που απαιτούνται για τη σύγκλιση. Ωστόσο, είναι πιο υπολογιστικά απαιτητικό και πιο περίπλοκο στην υλοποίηση. Από την άλλη πλευρά, το DDPG είναι αναμφισβήτητα ο χειρότερος αλγόριθμος για αυτό το σενάριο χρήσης. Υποφέρει από ανεπαρκή εξερεύνηση, οδηγώντας σε κακά αποτελέσματα και υψηλότερες υπολογιστικές απαιτήσεις, καθιστώντας το ακατάλληλο για το συγκεκριμένο σύνολο δεδομένων και πρόβλημα φθοράς.

Αξιολόγηση των Αλγορίθμων

Κάθε μοντέλο που δημιουργείται χωρίζεται σε έναν αριθμό υπομοντέλων. Αυτά τα υπομοντέλα παράγονται μετά την εκπαίδευση του αλγορίθμου ενισχυτικής μάθησης (RL) σε ένα συγκεκριμένο περιβάλλον. Κατά τη διάρκεια της εκπαίδευσης, το πιο πρόσφατο υπομοντέλο φορτώνεται και δημιουργείται ένα νέο περίπου κάθε 10.000 βήματα. Για παράδειγμα, το μοντέλο PPO Corrective No Delay έχει συνολικά 1.117.000 χρονικά βήματα, με αποτέλεσμα να αποτελείται από σχεδόν 112 υπομοντέλα.

Για κάθε ένα από τα 16 μοντέλα που δημιουργήθηκαν, κάθε υπομοντέλο αξιολογείται χρησιμοποιώντας μια ειδικά κατασκευασμένη για το πρόβλημα συνάρτηση `Custom_Evaluate_Policy`. Από τα αποτελέσματα που επιστρέφονται, επιλέγεται το υπομοντέλο με την καλύτερη απόδοση. Οι παρακάτω πίνακες παρουσιάζουν τα αποτελέσματα αυτής της διαδικασίας.

Corrective Environment | RL Algorithm Performance

	Περιβάλλον (Delay) & Σύνολο Δεδομένων (Μηχάνημα - Κόπτης)		Max Reward		Min Phm Score		Χρόνος Εκπαίδευσης & Συνολικά Timesteps
			Τιμή	Timestep	Τιμή	Timestep	
PPO	No	c4	-17.21	40,000	16.99	40,000	3 ώρες 1,117,000 steps
		c6	-16.21	70,000	18.00	70,000	
	With	c4	-8.98	20,000	15.73	20,000	2 ώρες 2,160,000 steps
		c6	- 8.23	1,460,000	17.85	670,000	
SAC	No	c4	-37.37	90,000	45.76	90,000	6.2 ώρες

		c6	-14.80	30,000	16.36	30,000	349,420 steps
	With	c4	-10.94	20,000	23,07	20,000	5.3 ώρες 279,374 steps
		c6	-8,21	110,000	16,22	110,000	
DDPG	No	c4	-18.84	20,000	19.32	20,000	12.67 ώρες 629,256 steps
		c6	-17.04	250,000	18.55	520,000	
	With	c4	-10.75	20,000	19.36	20,000	11.5 ώρες 452,160 steps
		c6	-8.83	20,000	18.12	20,000	
A2C	No	c4	-17.84	700,000	18.33	700,000	13.7 ώρες 19,120,000 steps
		c6	-16.33	7,000,000	18.12	7,000,000	
	With	c4	-10.75	14,640,000	19.36	14,640,000	17.2 ώρες 14,640,000 steps
		c6	-8.83	14,640,000	18.12	14,640,000	

Τα καλύτερα αποτελέσματα παρατηρούνται κυρίως σε συγκεκριμένα χρονικά βήματα παρά στο τέλος της εκπαίδευσης, κάτι που μπορεί να οφείλεται στο γεγονός ότι τα μοντέλα υπερεκπαιδεύονται πάνω στα δεδομένα του test set. Αυτό έχει ως αποτέλεσμα το μοντέλο να μαθαίνει πολύ καλά τα δεδομένα εκπαίδευσης, τα οποία δεν γενικεύονται σε νέα δεδομένα όπως τα σύνολα δεδομένων cutter 4 και cutter 6.

Ο PPO έχει βελτιωμένη απόδοση με καθυστέρηση και οι υψηλότερες ανταμοιβές υποδηλώνουν ότι μπορεί να προσαρμόσει την πολιτική του μέσω του περιβάλλοντος για πιο μακροπρόθεσμες ανταμοιβές πιο αποτελεσματικά. Είναι επιρρεπές σε υπερεκπαίδευση και παρατηρούμε ότι οι υψηλότερες βαθμολογίες συμβαίνουν στα αρχικό στάδιο της εκπαίδευσης.

Ο SAC δείχνει σημαντική βελτίωση με καθυστέρηση, βρίσκοντας βέλτιστα αποτελέσματα σε διαφορετικά χρονικά βήματα, υποδεικνύοντας μια αναντιστοιχία στην απόδοση των διαφορετικών συνόλων δεδομένων δοκιμής. Η ικανότητα του SAC να βελτιστοποιεί γρήγορα ευθυγραμμίζεται με την εξερεύνηση βάσει εντροπίας του, καθιστώντας το λιγότερο επιρρεπές σε υπερεκπαίδευση και πιο προσαρμόσιμο σε καθυστερημένες ανταμοιβές.

Το DDPG επιτυγχάνει τα καλύτερα αποτελέσματα γρήγορα και παρόλο τις μακροχρόνιες εκπαιδευτικές περιόδους, κάτι που υποδηλώνει ότι μπορεί να λειτουργεί υποβέλτιστα. Η σταθερή απόδοση σε

διαφορετικές συνθήκες υποδεικνύει ανθεκτικότητα, αν και γνωρίζοντας τη διαδικασία εκπαίδευσης, το μοντέλο δεν βελτιώνεται με την πάροδο του χρόνου και είναι κολλημένο σε ένα τοπικό μέγιστο.

Ο A2C έχει χρονικά μεγάλες εκπαιδευτικές περιόδους και υψηλά χρονικά βήματα, υποδεικνύοντας αναποτελεσματικότητα στη σύγκλιση αλλά συνεπή απόδοση. Η βελτίωση με καθυστέρηση υποδηλώνει ότι μπορεί να βελτιστοποιήσει τις μακροπρόθεσμες ανταμοιβές, αλλά απαιτεί σημαντικούς υπολογιστικούς πόρους. Η συνέπεια σε διαφορετικές συνθήκες υποδεικνύει ανθεκτικότητα, ενώ είναι το μόνο μοντέλο όπου σε όλες τις δοκιμαστικές περιπτώσεις η εκπαιδευτική διαδικασία ενισχύει την απόδοση. Δεν υπάρχουν περιπτώσεις που το καλύτερο αποτέλεσμα να βρίσκεται σε πρώιμα χρονικά βήματα, κάτι που σημαίνει ότι το μοντέλο μαθαίνει προς στη σωστή κατεύθυνση.

Non-Corrective Environment | RL Algorithm Performance

	Περιβάλλον (Delay) & Σύνολο Δεδομένων (Μηχάνημα - Κόπτης)		Max Reward		Min Phm Score		Saturated (Yes / No)	Χρόνος Εκπαίδευσης & Συνολικά Timesteps
			Value	Timestep	Value	Timestep		
PPO	No	c4	-25,255,715	38,730,000	35,497,029	38,730,000	Yes	17 hours 39,833,600 steps
		c6	-163,762,286	20,000	233,645,404	20,000		
	With	c4	-544,135,308	20,000	726,430,945	20,000	Yes	19.2 hours 16,967,680 steps
		c6	-583,253,362	20,000	876,304,975	20,000		
SAC	No	c4	-643	400,000	752	390,000	No	43.3 ώρες 3,819,951 steps
		c6	-5,679	410,000	3140	460,000		
	With	c4	-23,455	200,000	11,418	200,000	No	32.1 ώρες 2,639,889 steps
		c6	-15,048	470,000	7,712	190,000		
DDPG	No	c4	-563,092,188	1,050,000	735,266,738	1,050,000	Yes	19.5 hours 1,064,325 steps
		c6	-637,838,860	1,050,000	910,064,743	1,050,000		
	With	c4	DNF	-	DNF	-	No	13.5 hours 510,251 steps
		c6	DNF	-	DNF	-		

A2C	No	c4	-50,834,200	30,000	3,469,607	30,000	Yes	26,1 hours 26,750,000 steps
		c6	-28,469,419	30,000	39,536,971	30,000		
	With	c4	-550,755,354	19,110,000	735,266,790	19,110,000	Yes	22,8 hours 19,120,000 steps
		c6	-606,030,553	19,110,000	910,064,743	19,110,000		

Στα Μη Διορθωτικά περιβάλλοντα, το SAC δείχνει την καλύτερη απόδοση στο μοντέλο χωρίς καθυστέρηση, ενώ το PPO και το A2C παρουσιάζουν σημαντικές δυσκολίες. Το DDPG αποτυγχάνει να συγκλίνει αποτελεσματικά, ιδιαίτερα με καθυστέρηση.

Όσον αφορά το PPO, οι χρόνοι εκπαίδευσης και τα συνολικά χρονικά βήματα είναι αρκετά και επαρκή. Τα καλύτερα αποτελέσματα εμφανίζονται νωρίς στην εκπαίδευση, υποδηλώνοντας υπερεκπαίδευση και αναποτελεσματικότητα. Ο κορεσμός που παρατηρείται στην εκπαίδευση δείχνει ότι το PPO φτάνει σε ένα σημείο όπου δεν υπάρχει περαιτέρω βελτίωση, πιθανώς λόγω κακής προσαρμογής σε αυτόν τον τύπο περιβάλλοντος ή πιθανής κακής συνάρτησης ανταμοιβής.

Το SAC είναι μακράν το καλύτερο μοντέλο και ξεπερνά κάθε άλλο αλγόριθμο. Δείχνει καλύτερη προσαρμοστικότητα λόγω της εξερεύνησης που βασίζεται στην εντροπία, η οποία βοηθά στη διαχείριση των καθυστερημένων ανταμοιβών και στην αποφυγή υπερεκπαίδευσης. Η καλύτερη απόδοση δίνεται στο σημείο που παρατηρείται σταθερότητα στη διαδικασίας εκπαίδευσης, το οποίο δείχνει σωστή παραμετροποίηση.

Το DDPG αντιμετωπίζει δυσκολίες σύγκλισης στο μη διορθωτικό περιβάλλον, ειδικά με καθυστέρηση. Είναι το μόνο μοντέλο που δεν κατάφερε να ολοκληρώσει και να φτάσει το κόψιμο 315, καθώς τα επεισόδια σταματούν πριν φτάσουν στο τελικό κόψιμο λόγω εξαιρετικά χαμηλής ανταμοιβής. Οι υψηλές αρνητικές ανταμοιβές και οι βαθμολογίες Phm υποδεικνύουν κακή απόδοση, πιθανώς λόγω ανεπαρκούς εξερεύνησης.

Το A2C παρουσιάζει αναποτελεσματικότητα με μεγάλους χρόνους εκπαίδευσης και έντονα αρνητικές ανταμοιβές, υποδηλώνοντας ότι οι συγχρονισμένες ενημερώσεις του δεν είναι κατάλληλες για το μη διορθωτικό περιβάλλον, οδηγώντας σε υπερεκπαίδευση και κακή γενίκευση.

Συνολικά, συγκρίνοντας κάθε αλγόριθμο σε όλα τα διαφορετικά περιβάλλοντα, είναι σαφές ότι ο αλγόριθμος **SAC** αποδίδει καλύτερα στο Μη Διορθωτικό Περιβάλλον, ενώ ο **PPO** υπερισχύει στο Διορθωτικό Περιβάλλον. Όσον αφορά την καθυστέρηση, το περιβάλλον "με καθυστέρηση" επιτυγχάνει καλά αποτελέσματα πιο γρήγορα κατά την εκπαίδευση. Ωστόσο, στη φάση των δοκιμών, η απόδοση του περιβάλλοντος "με καθυστέρηση" συνήθως ξεπερνιέται από το περιβάλλον "χωρίς καθυστέρηση", το οποίο επιτυγχάνει καλύτερα αποτελέσματα αλλά χρειάζεται περισσότερο χρόνο εκπαίδευσης.

Συμπεράσματα

Ο στόχος αυτής της διπλωματικής ήταν η χρήση διαφόρων αλγορίθμων ενισχυτικής μάθησης για την επίλυση ενός προβλήματος προγνωστικής συντήρησης (PdM), συγκεκριμένα την πρόβλεψη φθοράς των φρεζών σε φρέζες υψηλής ταχύτητας CNC. Αυτή η πρόβλεψη βασίστηκε σε δεδομένα από δυναμόμετρα, επιταχυνσιόμετρα και αισθητήρες εκπομπής ακουστικής. Όλοι οι αλγόριθμοι εκπαιδεύτηκαν και αξιολογήθηκαν χρησιμοποιώντας δεδομένα από πραγματικές πειράματα και διεξήχθη ευρεία ανάλυση για να εξεταστεί η αποτελεσματικότητά τους.

Τα αποτελέσματα ανέδειξαν την αποτελεσματικότητα αυτής της προσέγγισης, υπογραμμίζοντας ότι οι τεχνικές RL μπορούν να είναι αποτελεσματικές και να παράγουν αξιόπιστα αποτελέσματα χωρίς να απαιτείται μοντέλο της εξεταζόμενης μηχανής.

Για να βελτιωθεί περαιτέρω η απόδοση του RL PdM Agent, μπορούμε είτε να αυξήσουμε την ποσότητα των δεδομένων που χρησιμοποιούνται για την εκπαίδευση, κάτι που θα γίνει πιο εφικτό καθώς αυξάνεται συνεχώς ο αριθμός των συσκευών IoT και η βιομηχανία συνεχίζει να υιοθετεί νέες τεχνικές χρησιμοποιώντας περισσότερη τεχνολογία IoT. Μια άλλη προσέγγιση είναι να προσπαθήσουμε να μοντελοποιήσουμε το περιβάλλον του CNC συνεργαζόμενοι με μηχανικούς υλικών και μηχανολόγους μηχανικούς για να δημιουργήσουμε ένα μοντέλο που να προσεγγίζει την πραγματικότητα. Αν και αυτή η προσέγγιση δεν εγγυάται την επιτυχία, θα απαιτούσε λιγότερα δεδομένα.

Συνολικά, αν υπάρχουν επαρκή δεδομένα, η προσέγγιση χωρίς μοντέλο είναι προτιμότερη λόγω της καθολικότητάς της και της έλλειψης ανάγκης για λεπτομερή γνώση της συγκεκριμένης χρήσης και των αλληλεπιδράσεων της μηχανής που εξετάζεται.

Μελλοντική Έρευνα

Η μελλοντική έρευνα θα μπορούσε να περιλαμβάνει τη σύγκριση διαφορετικών προσεγγίσεων μηχανικής μάθησης με την προσέγγιση RL και την πιθανή συνδυαστική τους χρήση. Επιπλέον, η απόκτηση του πλήρους συνόλου δεδομένων από το PHM Data Challenge του 2010 θα επέτρεπε περαιτέρω βελτίωση της απόδοσης των μοντέλων.

Ένας άλλος ενδιαφέρον δρόμος για μελλοντική έρευνα είναι η αντιμετώπιση ενός διαφορετικού στόχου PdM για την ίδια μηχανή χρησιμοποιώντας το ίδιο σύνολο δεδομένων. Συγκεκριμένα, η πρόβλεψη της Υπολειπόμενης Χρήσιμης Ζωής (RUL) των φρεζών, που ήταν ο στόχος του διαγωνισμού PHM το 2010, θα μπορούσε να είναι μια χρήσιμη επέκταση αυτής της δουλειάς ενσωματώνοντας τεχνικές RL για την επίλυση αυτού του προβλήματος.

1.Introduction

Industry 4.0 encompasses the integration of the Internet of Things (IoT) to create a feedback loop between the digital and physical worlds. IoT devices with built-in sensors are becoming more advanced and affordable. According to Statista (Vailshery, 2024) there are approximately 15.9 billion connected IoT devices as of 2023 and in the next decade this number is expected to grow to almost 40 billion. Real-time analytics combined with edge sensors and artificial intelligence (AI) are enabling sophisticated cloud-based platforms. The rollout of high-speed 5G connectivity opens the door for the implementation of complex AI applications that can improve industrial operations and support effective cyber-physical systems.

According to (Burke , Hartigan, & Sniderman , 2017), a SmartFactory is designed to self-optimize its performance by learning from new conditions and adapting in near-real time. **Predictive maintenance** (PdM) involves analyzing data from various sources—such as sensors installed on critical machinery, information from enterprise resource planning (ERP) systems, production data, and maintenance management systems—to predict and proactively address equipment failures. These SmartFactory management systems utilize advanced predictive models to forecast potential issues and take preventive action.

(Ben-Daya, Dufuaa, & Raouf, 2012) found that maintenance costs can account for between 15% and 40% of manufacturing expenses across different industries. Despite the benefits, many manufacturers are hesitant to adopt SmartFactory practices. (Laape , Dollar, & Cotteleer , 2020) reported that nearly 65% of surveyed manufacturers had made little or no progress on their SmartFactory initiatives. Cost pressures in manufacturing mean that optimal predictive maintenance can help reduce maintenance expenses while ensuring continuous production.

A variety of approaches have been used to address PdM challenges. These include mixed-integer and multi-criteria optimization techniques like Pareto optimization (Saydam & Frangopol, 2015), machine learning (ML) methods like decision trees (Frangopol, Lin, & Estes, 1997), random forests (Kabir , Foggo, & Yu, 2018), gradient-boosted trees (XGboost) (Ma, Guo, & Mao, 2020), and unsupervised ML techniques like Principal Component Analysis (PCA) (Eke, Aka-Ngnui, & Glerc, 2017). Supervised ML methods such as regression (Susto, Wan, & Pampuri, An adaptive machine learning decision system for flexible predictive maintenance , 2014) and support vector machines (SVM) (Ding, He, & Zi , 2008), (Susto, Schirru & Pampuri, 2013) have also been applied. Deep learning techniques like long short-term memory (LSTM) have been used for estimating remaining useful life (RUL) (Sayyad , Kumar, & Bongale, 2022), (Zheng , Ristovski, & Farahat, 2017), and convolutional neural networks (CNNs) have been used for regression (Sateesh Babu, Zhao, & Li, 2016).

Early methods often used deterministic approaches, which limited effectiveness due to the complexity and nonlinearity of industrial processes. The introduction of ML improved the modeling of systems, but sufficient labeled data from experts remains a challenge, leading to approximate solutions.

Reinforcement learning (RL), including deep reinforcement learning, is emerging as a promising solution to these limitations. Traditional control and planning methods can struggle with complex nonlinear

processes, but RL offers an autonomous learning mechanism to develop optimal, model-free solutions (Lewis , Vrabie, & Vamvoudakis, 2012), (Sutton & Barto, Reinforcement learning: an introduction, 2018). Given that PdM involves optimal planning, RL has gained attention as a potential solution for addressing PdM challenges.

The focus of this study is the CNC Machine Wear Prediction Problem. A CNC (Computer Numerical Control) machine is an automated manufacturing tool that uses pre-programmed computer software to control the movement and operation of machinery, allowing for precise and efficient production of complex parts. The data is provided by the PHM (Prognostics and Health Management) Society and was part of the 2010 PHM Data Challenge. The goal is to predict the wear of the flutes in cutters of a high-speed CNC milling machine using measurements from dynamometer, accelerometer, and acoustic emission sensors.

For the predictive maintenance (PdM) task in this context, a specialized taxonomy has been employed, as illustrated in the diagram below:

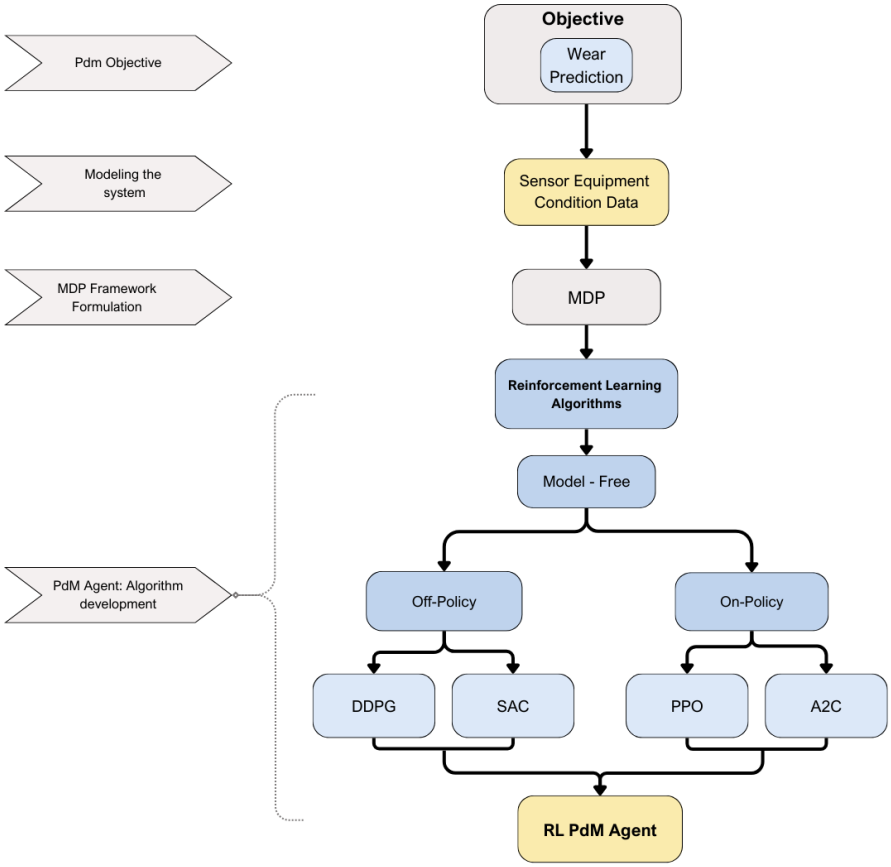


Figure 14 : A taxonomy of Reinforcement Learning for the Wear Prediction Problem of this thesis

In this taxonomy, the primary objective is Wear Prediction, which is based on data from sensor equipment condition monitoring. This data is integrated into the Markov Decision Process (MDP) framework, which is crucial for formulating the problem in a manner that is suitable for RL algorithms.

The reinforcement learning algorithms applied are specifically model-free algorithms. These algorithms are categorized into off-policy and on-policy methods. Off-policy methods include Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC), while on-policy methods encompass Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C). The end goal is to develop an RL-based PdM agent capable of making autonomous decisions to predict wear and avoid potential and functional failures in CNC machines effectively.

In the sections that follow, this taxonomy will be elaborated upon in greater detail, discussing each component and its role in formulating a reliable RL PdM Agent.

2. Theoretical Background & Related Work

2.1 Predictive Maintenance

Predictive maintenance is a proactive maintenance strategy that uses data analysis tools and techniques to detect anomalies in equipment operations and predict future failures. It is an important enabler for Industry 4.0, providing significant benefits such as reduced costs, minimized downtime, and enhanced safety. However, successful implementation requires overcoming technical and managerial challenges, integrating it into a broader digital strategy, and fostering a culture that supports innovation and data-driven decision-making.

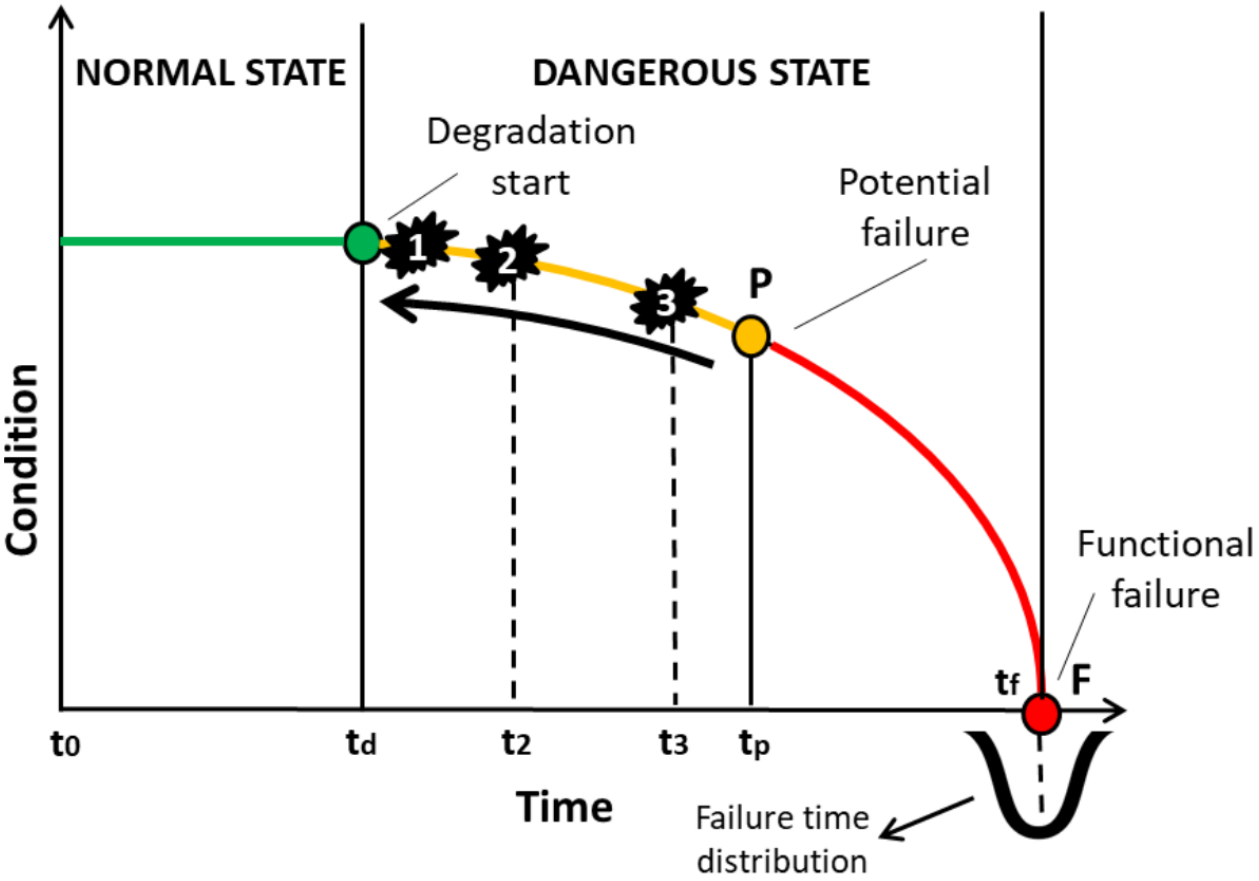


Figure 15 : Predictive maintenance in the context of the P-F curve (Bousdekis, Apostolou, & Mentzas, 2020)

The figure above displays the P-F curve (Potential Failure - Functional Failure), showing the transition from a normal state to a dangerous state, highlighting key points such as the degradation start, potential failure, and functional failure. This curve represents the concept that failure is a process that develops over time, providing a window for predictive maintenance to "intervene".

By addressing issues in this time window before they escalate into major failures, the overall lifespan of equipment is extended, leading to increased equipment lifespan. This approach ensures that maintenance is performed only when necessary, which in turn reduces labor and material costs, contributing to reduced

maintenance costs. Furthermore, by preventing unexpected breakdowns through proactive maintenance, predictive maintenance minimizes downtime and ensures continuous operation, thereby boosting productivity. Additionally, early detection of potential failures enhances workplace safety by reducing the risk of accidents. The continuous monitoring and data analysis involved in predictive maintenance also lead to enhanced knowledge about equipment behavior and performance, enabling better decision-making and operational improvements.

Predictive maintenance, despite its benefits, presents several challenges. Implementing it requires additional equipment, such as new sensors and monitoring devices, which can be costly and complex to integrate with existing systems. There is also a significant initial budget required to transition to Industry 4.0, including investments in hardware, software, and personnel training. Specialized personnel with expertise in data science, IT, and maintenance engineering are necessary to install and operate the systems, which can be a barrier for some organizations. Additionally, effective predictive maintenance requires cross-field collaboration between IT professionals, data scientists, and maintenance engineers, necessitating strong communication and teamwork across different departments.

To effectively implement a predictive maintenance strategy, integrating Markov Decision Processes (MDPs) can be beneficial, as MDPs provide a robust framework for optimizing maintenance decisions under uncertainty, further improving the efficiency and effectiveness of maintenance strategies.

2.2 Markov Decision Process

2.2.1 MDP | Sequential Decision Problem

A sequential decision problem in a fully observable, stochastic environment with a Markovian transition model, (Russel & Norvig, 2021) and additive rewards is called a **Markov Decision Process** (**MDP**) and consists of the following:

- **State Space** (with initial state s_0): This is the set of all possible states that the system can be in. The initial state, usually denoted by s_0, is the state from which the decision-making process begins.

- **Action Space** $A(s)$: This is the set of available actions that the agent can take in each state. This collection of actions may depend on the state.

- **Transition Model** $\boldsymbol{P(s'|s,a)}$: This is the model that describes the probability of transitioning from one state s to another state s' under the influence of action a. This model is essential for calculating the expected return of each action.

- **Reward Function** $R(s,a,s')$: This function determines the reward that the agent receives when performing action a and transitioning from state s to state s'. The reward can be positive, negative, or zero, depending on the goal the agent is pursuing.

- **Starting state distribution $\boldsymbol{\rho_0}$** : It represents the probability distribution over the initial state from which the decision-making process begins. It defines how likely each possible starting state is, setting the initial conditions for the agent's interaction with the environment.

With these elements, a variety of decision-making problems can be modeled, and algorithms can be developed to exploit the structure of Markov Decision Processes (MDPs) to find optimal decision-making strategies.

Methods for solving MDPs typically involve dynamic programming, which simplifies a problem by recursively breaking it down into smaller segments while retaining the optimal solutions of these segments.

Regarding the solution to the problem, no fixed sequence of actions can resolve the problem, as the agent may end up in a state different from the goal state. Thus, the solution must specify the agent's actions for any state it may encounter.

A solution of this kind is called a **policy**

The policy is denoted by π, and $\pi(s)$ is the action that policy π recommends for state s. Regardless of the outcome of the action, the resulting state will belong to the policy, and the agent will know what to do next. Each time a given policy is executed starting from the initial state, the stochastic nature of the environment may lead to a different history of the environment. Thus, the quality of a policy is measured in terms of the expected utility of the possible histories of the environment generated by the policy.

An optimal policy is a policy that gives the highest expected utility, denoted by π^*. Given the policy π^*, the agent decides what to do by examining its current perception, which indicates the current state s, and then performs the action $\pi^*(s)$. The policy explicitly represents the agent function and is thus a description of a simple reflexive agent, computed from the information used for a utility-based agent.

The introduction of uncertainty brings MDPs closer to the real world than causal search problems. For this reason, MDP solving algorithms have been studied in various fields such as AI, Operations Research, Economics and Control Theory.

The introduction of uncertainty is a key element that makes MDPs (Markov Decision Processes) more representative of the real world compared to causal search problems. This is due to the fact that the real world is often complex and uncertain, with many variables and fuzzy parameters. By introducing uncertainty, MDP solving algorithms can take into account various scenarios and possible developments, making them more flexible and realistic.

For this reason, MDP solving algorithms have been applied in many different fields, including artificial intelligence, operations research, economics and control theory. The flexibility and adaptability of these algorithms make them valuable tools for tackling real-world problems where uncertainty is present.

- In the field of Artificial Intelligence, MDP solving algorithms are used to develop systems that can make decisions under uncertainty, such as robots and autonomous vehicles.

- In Operations Research, MDPs help analyze and optimize processes involving random factors, such as inventory management or workflow in a factory.

- In finance, MDP algorithms can be used for portfolio management and investment selection, taking into account market uncertainty and price volatility.

- In control theory, MDPs help to model and design systems that must respond to unpredictable changes or disturbances.

The use of MDPs in so many different domains highlight the need for decision-making methods that take uncertainty into account, making them a valuable tool for addressing real-world challenges.

2.2.2 Utility in terms of time

It is essential to determine whether the decision-making process has a finite or infinite horizon. A finite horizon indicates a fixed time N after which subsequent events are irrelevant, such as the end of a game (examples are checkmate in chess or machine failure in PdM). Therefore

$$\boldsymbol{Value_h = U([s_o, a_o, s_1, a_1, \ldots, s_{n+k}]) = V_h([s_o, a_o, s_1, a_1, \ldots, s_n])}$$

For each k $>$ 0

In the finite horizon case, the optimal energy in a given state may depend on the remaining time. A time-dependent policy is called a non-stationary policy

In the case of a non-stationary time boundary, there is no reason to have different behavior in the same state at different times. So, an optimal action depends only on the current state, and the optimal policy is stationary. Thus, policies for the infinite horizon case are simpler than those for the finite horizon case. However, the infinite horizon does not necessarily mean that all sequences of states are infinite, it just means that there is some fixed deadline. That is, there may be finite sequences of states in an infinite horizon MDP containing a terminal state.

To be able to calculate the utility Value_h for the sequences of states we will use **addictive discounted rewards**.

The utility V_h is calculated as follows:

$$V_h([s_o, a_o, s_1, a_1, \ldots]) = \boldsymbol{R(s_o, a_o, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3)} + \ldots$$

The utility is calculated by summing up the rewards of each agent's action after multiplying it by a discount factor.

Where the **discount factor** γ is a number between 0 and 1. The discount factor describes the agent's preference for current rewards over future rewards. When γ is close to 0, then far-future rewards are

considered insignificant. When γ is close to 1, then an agent is more willing to wait for long-term rewards. When γ is exactly 1, discounted rewards are restricted to the special case of purely **additive rewards**.

Some reasons why additive discounted rewards are very useful for solving the problem are empirical, economic, and the most important and practical of all uncertainty. Both humans and animals seem to place more value on short-term rewards than on rewards in the distant future. As far as economics is concerned, if the rewards are monetary, then it is indeed better to receive them sooner rather than later, because the early rewards can be invested and produce returns at the time we wait for the later ones.

A third reason is uncertainty. Uncertainty about the actual rewards that may never materialize for many for many reasons that are not accounted for in the transition model

Finally, a fourth reason arises from a natural property of preferences relative to history. In the terminology of multicriteria utility theory, each transition $s_t \underset{\boldsymbol{a_t}}{\rightarrow} s_{t+1}$ can be viewed as a property of the history $[\boldsymbol{s_o}, \boldsymbol{a_o}, \boldsymbol{s_1}, \boldsymbol{a_1}, \dots]$. Theoretically, the utility function could depend on these properties in arbitrarily complex ways. There is, however, a very plausible preference independence assumption that can be made, namely that the agent's preferences between state sequences are stationary.

Suppose that two histories $[\boldsymbol{s_o}, \boldsymbol{a_o}, \boldsymbol{s_1}, \boldsymbol{a_1}, \dots]$ and $[\boldsymbol{s'_o}, \boldsymbol{a'_o}, \boldsymbol{s'_1}, \boldsymbol{a'_1}, \dots]$ start with the same transition like $s_0 = s'_0, a_0 = a'_0 \; and \; s_1 = s'_1$. Then the stationarity for preferences means that the two histories should be ordered in terms of preference in the same way as the histories $[s_1, a_1, s_2, \dots]$and $[s'_1, a'_1, s'_2, \dots]$. This means that if you prefer a particular future to another future starting tomorrow, then you should still prefer that future even if it were to start today. Stagnation is a fairly "innocent" assumption, but one that is only satisfied by additive discounting, which is the only form of utility for histories.

One such reason for using discounted rewards is that they conveniently eliminate certain unpleasant "infinities". With infinite horizons there is a potential difficulty: if the environment does not contain a terminal state, or if the agent never reaches it, then all historical environments will be infinite, and utilities with additive undiscounted rewards will generally be infinite. Although we can agree that $+\infty$ is better than $-\infty$, comparing two sequences of states with $+\infty$ utilities is more difficult. There are three solutions of which we have already seen two:

With **discounted rewards**, the utility of an infinite sequence is finite. To be precise, if $\gamma < 1$ and the rewards are blocked by $\pm R_{max}$, then:

$$V_h([\boldsymbol{s_o}, \boldsymbol{a_o}, \boldsymbol{s_1}, \boldsymbol{a_1}, \dots]) = \sum_{\boldsymbol{t=0}}^{\infty} \boldsymbol{\gamma^t R}(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}}) \leq \sum_{\boldsymbol{t=0}}^{\infty} \boldsymbol{\gamma^t R_{max}} = \frac{\boldsymbol{R_{max}}}{\boldsymbol{1-\gamma}}$$

Using the standard formula for the sum of an infinite geometric sequence.

If the environment contains terminal states and if the agent is guaranteed to reach a perfect state at some point, then we will never need to compare infinite sequences. A policy that is guaranteed to supervene a

terminal state is called a **proper policy**. With proper policies we can use $\gamma = 1$ (additive undiscounted rewards). The existence of improper policies can make the usual algorithms for solving MDP fail in the case of additive rewards, and thus provides a good reason for using discounted rewards.

Infinite sequences can be compared in terms of the average reward obtained per time step. The average reward is a useful criterion for some problems, but the analysis of average reward algorithms is complicated.

2.2.3 Optimal Policies and the utilities of the situations

Knowing that the utility of a given history is the sum of the discounted rewards, policies can be compared by evaluating the expected utilities obtained during their execution. Assuming that the agent is in an initial state s, S_t(a random variable) is defined as the state that the agent achieves at time t when executing a particular policy π. Obviously, $S_0 = s$, the current state of the agent. The probability distribution for the state sequences $S_1, S_2, \ldots$ is determined by the initial state s, the policy π, and the transition model for the environment.

The expected utility obtained by executing policy π starting from state s is given by the formula

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right]$$

Where the expectation E depends on the probability distribution over the sequences of states defined by s and π. Now of all the policies that the agent could choose to execute starting from state s, one (or more) will have higher expected utilities than all the others. The optimal policy will be denoted by ${\pi^*}_s$:

$${\pi^*}_s = argmax_\pi V_\pi(s)$$

A remarkable effect of using discounted utilities with infinite horizons is that the optimal policy is independent of the initial state. (Of course, the sequence of actions will not be independent - the policy is a function that specifies an action for each state.) This fact seems reasonably obvious: if policy ${\pi^*}_1$ is optimal starting from state s_1 and policy ${\pi^*}_2$ is optimal starting from state s_2 then, when they reach a third state s_3, there will be no disagreement between the policies (${\pi^*}_1$ and ${\pi^*}_2$ or with a policy ${\pi^*}_3$) for the agent's next move. The proof follows from the uniqueness of the utility function with respect to the states of the environment. For this reason, we can denote the optimal policy as π^*.

According to the above, the actual utility of a state is simply V_{π^*}, the expected sum of the discounted rewards that we will have if the agent executes an optimal policy.

Therefore, the utility function can be written as $V_{\pi^*}(s) = V(s)$

2.2.3.1 Bellman's equation

The utility function U(s) allows the agent to choose actions using the maximum expected utility principle, to choose the action that maximizes the reward for the next step plus the expected utility of the subsequent state of the environment:

$$\pi^*(s) = argmax_{a\epsilon A(s)} \sum_{s\prime} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

From the definition of utility, it follows that there is a direct relationship between the utility of one state and the utility of its neighboring states. The utility of a state is the expected reward for the subsequent transition plus the discounted utility of the subsequent state, assuming that the agent always chooses the optimal action.

The utility is also defined as:

$$V(s) = max_{a\epsilon A(s)} \sum_{s\prime} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

This equation is called the Bellman equation in honor of Richard Bellman (1957)

The utilities of the states defined by the equation $V_\pi(s) = E\ [\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})]$, as the expected utility of the consequent sequential states, are solutions of the set of Bellman equations.

2.2.3.2 Function Q

Another useful tool for solving MDP problems is the energy-utility function or function Q. $Q(s,a)$ is the expected utility from performing a given action in a given state. The function is related to utilities in the following way:

$$V(s) = max_a Q(s,a)$$

Furthermore, from the function Q the optimal policy can be derived as follows:

$$\pi^*(s) = argmax_a Q(s,a)$$

We can also develop a Bellman equation for functions Q, noting that the expected total reward for performing an action is its immediate reward plus the discounted utility of the outcome state, which in turn can be represented by the function Q:

$$Q(s,a) = \sum_{s\prime} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$$
$$Q(s,a) = \sum_{s\prime} P(s'|s,a)[R(s,a,s') + \gamma max_{a\prime} Q(s',a')]$$

Solving Bellman's equations for V (or for Q) provides what is to find an optimal policy. The function Q appears all the time in MDP process solving algorithms, and the following definition (pseudocode) is used:

Table 3 : The Pseudocode of Function Q

```
Function Q_Value(MDP, States, Actions, V) :
    Initialize Q_value to 0
    For each next state in MDP.States:
        transition_prob = MDP.Transition_Prob(state, action, next_state)
        reward = MDP.Reward(state, action, next_state)
        Q_Value += transition_prob * (reward + MDP.Discount * V(next_state))
    End For
    Return Q_value
End Function
```

2.3 Reinforcement Learning

2.3.1 Learning from Rewards

In RL a decision agent learns from a sequence of reward signals that provide some indication of the quality of its behavior. The goal is to optimize the sum of future rewards.

The usefulness of reinforcement learning is seen in applications where the set of possible situations - states in the environment is very large and the agent is likely to be led into a situation that has not been seen or "taught" before.

A typical example and success story is the use of RL in chess. So, let's consider the problem of learning chess from an agent and let's say we use supervised learning to achieve our goal. Given that the chess-playing agent's function takes as input a position on the chessboard and returns a move, we train this function by providing it with examples of positions on the chessboard and returns a move, each of which is labeled with the correct move. At the same time, we have a database of several million grandmaster games, each of which is a sequence of positions and moves. The moves made by the winner are, with a few exceptions, considered good, if not always perfect. Thus, we have a promising training dataset. The problem is that there are relatively few examples (10^8) compared to the space of all possible positions on the board (10^{40}).

In a new game, we soon encounter positions that are significantly different from those in the database, and the agent training function is likely to fail badly, mainly because the agent does not know what he is trying to achieve with his moves (check or even mate) or, in even more detail, the effect his moves have on the positions of the pieces. Chess is a miniature version of the real world. In more realistic problems, we would need much larger databases of grandmaster games, which simply don't exist. For this reason, Yann LeCun (Turing Award 2018) points out that "the AI revolution will not be supervised".

From the perspective of an AI system designer, providing a reward system to the agent is usually easier than providing characterizing examples of how the agent should behave in a multitude of situations.

The reward function is often very comprehensive and easy to define as it requires only a few lines of code to tell the agent playing chess whether it has won or lost the game. We also do not need to be experts and able to perform the right actions in every situation (we do not need to be grandmasters), as would be the case if we tried to implement supervised learning.

In practice, a little experience can make a significant contribution to reinforcement learning. The rewards for winning and losing in chess are very **sparse rewards**, because in the vast majority of situations the agent receives no informational reward signal (one per game). For this reason, we need to build a reward function that accounts for other elements of the environment such as the value of each piece the agent holds and its position (such as extra positive score for capturing a center). These intermediate rewards make learning much easier.

As long as, we can provide the right reward signal, RL provides a very general way to build AI systems. This is especially true for simulated environments, where the opportunities for gaining experience are limitless. There are hundreds of different RL algorithms and many of them can be used as tools for a wide range of learning methods.

2.3.2 Key Concepts in RL

In this section, the mathematics behind RL and its practical applications will be briefly analyzed.

2.3.2.2 RL Components

RL is the study of agents and how they learn by trial and error. It formalizes the idea that rewarding or punishing an agent for its behavior makes it more likely to repeat or forego that behavior in the future.

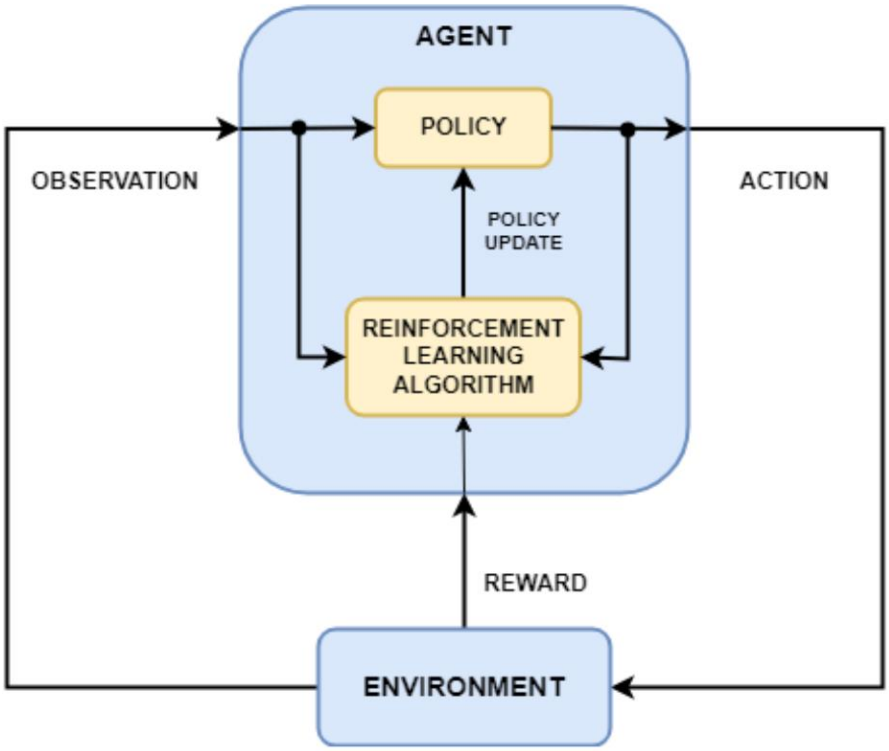


Figure 16 : General Representation of Reinforcement Learning Scenario (MathWorks, 2024)

In RL, the main characters are the **agent** and the **environment**. The environment represents the world in which the agent resides and with which it interacts. During each interaction step, the agent observes a (potentially partial) view of the world's state and then decides on an action to take. The environment changes in response to the agent's actions but can also change independently.

The agent receives a reward signal from the environment, which measures the value of the current state. The agent's objective is to maximize its cumulative reward, known as the return. Reinforcement learning methods provide ways for the agent to learn behaviors that achieve this goal.

To explore the specifics of RL, additional terminology must be introduced. This involves discussing key concepts and components that support the RL processes:

States and Observations
The complete description of the world is captured by a state s, ensuring all information is visible. In contrast, an observation o provides only a partial description, potentially omitting certain details.

In deep RL, states and observations are represented by a real-valued vector, matrix, or higher-order tensor. When the agent has access to the complete state, the environment is considered fully observed. If the agent can only access partial information, the environment is called partially observed.

Action Spaces
Different environments allow different kinds of actions. The set of all valid actions in a given environment is often called the action space. Some environments, like Atari and Go, have discrete action spaces, where

only a finite number of moves are available to the agent. Other environments, like where the agent controls a robot in a physical world, have continuous action spaces. In continuous spaces, actions are real-valued vectors.

The collection of all actions that are allowed within a particular environment is referred to as the action space. For instance, environments like Chess and Go feature discrete action spaces, limiting the agent to a finite set of moves (the legal moves on the board). Conversely, environments where the agent operates a robot in the physical world have continuous action spaces, characterized by real-valued vectors for actions.

This difference significantly impacts methods in deep RL. Certain algorithm families are tailored for one type of action space and would require extensive modifications to function in the other.

Policies

A policy guides an agent in selecting actions and can be either deterministic or stochastic. A deterministic policy is usually represented by μ:

$$\alpha_t = \mu(s_t),$$

A stochastic policy, on the other hand, is typically denoted by π:

$$\alpha_t = \pi(\cdot|s_t).$$

The policy acts as the agent's decision-making mechanism, so "policy" and "agent" are often used interchangeably, such as saying, *"The policy is trying to maximize reward."*

In deep RL, policies are **parameterized**, meaning their outputs are functions dependent on adjustable parameters (e.g., the weights and biases of a neural network). These parameters can be fine-tuned using optimization algorithms to alter the agent's behavior.

Parameters of such policies are frequently denoted by θ or φ, and this notation is included as a subscript on the policy symbol to indicate their role:

$$\alpha_t = \mu_\theta(s_t)$$
$$\alpha_t \sim \pi_\theta(\cdot|s_t)$$

Trajectories / Episodes

A trajectory τ consists of a sequence of states and actions

$$\tau = (s_0, a_0, s_1, \alpha_1, \ldots)$$

The initial state s_0 is randomly drawn from the start-state distribution, often represented by ρ_0:

$$s_0 \sim \rho_0(\cdot)$$

State transitions, which describe changes in the world from state s_t at time t to state s_{t+1}at time $t+1$, are influenced by the most recent action a_t. These transitions can be deterministic:

$$s_{t+1} \sim f(s_t, a_t)$$

or stochastic,

$$s_{t+1} \sim P(\cdot \mid s_t, a_t)$$

Actions are determined by the agent's policy.

Trajectories are often referred to as **episodes** or **rollouts**

Reward and Return

In reinforcement learning, the reward function *R* plays the role of a trainer. It relies on the current state, the action taken, and the subsequent state:

$$r_t = R(s_t, a_t, s_{t+1})$$

However, this is often simplified to depend solely on the current state: $r_t = R(s_t)$, or state-action pair $r_t = R(s_t, a_t)$.

The agent's objective is to maximize the cumulative reward over a trajectory. This cumulative reward is $R(\tau)$.

One type of return is the finite-horizon undiscounted return, which is the total sum of rewards obtained over a fixed number of steps:

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Another type is the infinite-horizon discounted return, which considers all rewards the agent ever receives, but discounts them based on how far into the future they occur. This is expressed with a discount factor γ_t in (0,1):

$$R(\tau) = \sum_{t=0}^{T} (\gamma_t \cdot r_t)$$

The inclusion of a discount factor is both intuitively appealing and mathematically convenient. Intuitively, immediate rewards are often preferred over future rewards. Mathematically, without a discount factor, the infinite-horizon sum of rewards may not converge to a finite value, making it difficult to handle in equations. By applying a discount factor and under reasonable conditions, the infinite sum converges, simplifying the calculations.

The RL Problem

The goal in Reinforcement Learning is to select a policy which maximizes expected return when the agent acts according to it. No matter the choice of the reward function (return measure) and the policy the goal is the same.

To discuss expected return, understanding probability distributions over trajectories is essential.

Assuming both environment transitions and the policy are stochastic, the probability of a T - step trajectory is given by:

$$P(\tau|\pi) = \rho_o(s_o) \cdot \prod_{t=o}^{T-1} P(s_{t+1}|s_t, a_t) \cdot \pi(a_t|s_t)$$

τ : This represents a trajectory (episode), which is a sequence of states and actions over T steps. A trajectory can be written as $\tau = (s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T)$

π : This is the policy that the agent is following. It defines the probability distribution over actions given a state, $\pi(a_t|s_t)$, for each time step t, this term represents the probability of the agent taking action a_t given the state s_t according to the policy π.

$\rho_o(s_o)$: This is the initial state distribution, that is the probability distribution over the initial state s_o. Meaning the probability the trajectory starts in the initial state s_o. In some environments (like chess) this probability is 1.

$P(s_{t+1}|s_t, a_t)$: This is the transition probability, that is the probability of transitioning to state s_{t+1} given the current state s_t and action a_t. For each time step t we calculate this term and we multiply it by $\pi(a_t|s_t)$.

In essence, the formula $P(\tau|\pi)$ combines the probabilities of starting in the initial state, taking actions according to the policy, and transitioning between states according to the environment's dynamics, to give the overall probability of the entire trajectory occurring under the policy π.

The **expected return** (for whichever measure), denoted by $J(\pi)$, is then:

$$J(\pi) = \int P(\tau|\pi) \cdot R(\tau) = E_{\tau\sim\pi}[R(\tau)]$$

The expected return $J(\pi)$ is a measure of how good a policy π is in terms of the cumulative reward it can achieve over time.

$P(\tau|\pi)$: As mentioned above, this is the probability of a trajectory τ occurring under the policy π. It encapsulates the dynamics of the environment and the behavior of the policy.

$R(\tau)$: This is the return of the trajectory τ. The return is the cumulative reward obtained over the trajectory. Depending on the specific problem, this can be the sum of rewards, a discounted sum of rewards (γ), or some other measure of cumulative reward.

$\int P(\tau|\pi) \cdot R(\tau)$: This represents the expected return over all possible trajectories τ, weighted by their probabilities under the policy π.

Expectation $E_{\tau\sim\pi}[R(\tau)]$: This is another way of expressing the integral. It means taking the expectation of the return $R(\tau)$ over trajectories τ that are sampled according to the probability distribution induced by the policy π.

The **expected return** $J(\pi)$ quantifies the performance of a policy π by averaging the cumulative rewards (returns) over all possible trajectories that the policy can generate.

The **central optimization problem** in RL can then be expressed by:

$$\pi^* = argmax_\pi J(\pi)$$

The central optimization problem in reinforcement learning is to find the policy π that maximizes the expected return $J(\pi)$.

π^* : This denotes the optimal policy, the policy that yields the highest expected return.

$argmax_\pi J(\pi)$: This expression means we are looking for the policy π that maximizes the function $J(\pi)$.

The goal of the agent is to find the best policy π that maximizes its expected cumulative reward. This involves exploring the space of possible policies and evaluating their performance to find the one that provides the highest expected reward.
This is a classic optimization problem where the objective function is $J(\pi)$, the expected return, and the variable to optimize, is the policy π.

Value Functions

Understanding the value of a state or state-action pair is essential. The value refers to the expected return when starting in a specific state or state-action pair and subsequently following a particular policy indefinitely. Value functions are a key component to nearly every RL algorithm.

There are four main functions of note here.

1. The On-Policy Value Function, $V^\pi(s)$, represents the expected return when starting in state s and always following policy π:

$$V^{\pi}(s) = E_{\tau\sim\pi}[R(\tau)|s_o = s]$$

$E_{\tau\sim\pi}$: This denotes the expected value over trajectories τ that are generated by following the policy π.

$R(\tau)$: This is the return of the trajectory τ. The return is the sum of rewards obtained over the trajectory. Depending on the specific problem, it can be the total reward or the discounted sum of rewards.

$s_o = s$: This specifies that the trajectory starts from a specific state

The **on-policy value function $\boldsymbol{V^{\pi}(s)}$** gives the expected return starting from state s and then following policy π. It is an estimate of how good it is to be in a particular state s if you act according to policy π. By knowing $\boldsymbol{V^{\pi}(s)}$**,** an agent can improve its policy by choosing actions that lead to states with higher values.

2. The On-Policy Action-Value Function, $Q^{\pi}(s,a)$, Represents the expected return when starting in state s, taking action a (which may not originate from the policy), and then always following policy π:

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}[R(\tau)|s_o = s, a_o = a]$$

From the above, it can be deduced that the state-value function $V^{\pi}(s)$ can be expressed as the expectation of the action-value function $Q^{\pi}(s,a)$ over all actions a taken according to policy π. Mathematically, this can be represented as:

$$V^{\pi}(s) = E_{a\sim\pi}\,[Q^{\pi}(s,a)]$$

This equation makes clear that $V^{\pi}(s)$ is the expected return if the agent starts in state s and thereafter follows policy π, considering all possible actions weighted by their probabilities under the policy.

3. The Optimal Value Function, $V^{*}(s)$, represents the expected return when starting in state s and always following the optimal policy in the environment

$$V^{*}(s) = max_{\pi}E_{\tau\sim\pi}[R(\tau)|s_o = s] = max_{\pi}V^{\pi}(s)$$

4. The Optimal Action-Value Function, $Q^{*}(s,a)$, Represents the expected return when starting in state s, taking action a, and then always following the optimal policy in the environment:

$$Q^{*}(s,a) = max_{\pi}E_{\tau\sim\pi}[R(\tau)|s_o = s, a_o = a] = max_{\pi}Q^{\pi}(s,a)$$

Table 4 : Summary of Value Functions used in RL

Value Functions	
$V^{\pi}(s)$	Expected return starting from state s under policy π
$Q^{\pi}(s,a)$	Expected return starting from state s, taking action a, and then following policy π.
$V^{*}(s)$	Maximum expected return starting from state s under the optimal policy.
$Q^{*}(s,a)$	Maximum expected return starting from state s, taking action a, and then following the optimal policy

The Optimal Q - Function and the Optimal Action

A link exists between the optimal action-value function $Q^{*}(s,a)$ and the action chosen by the optimal policy. By definition, $Q^{*}(s,a)$ represents the expected return for starting in state s, taking an arbitrary action a, and then following the optimal policy indefinitely.

In state s, the optimal policy will choose the action that maximizes the expected return from that state. Therefore, given $Q^{*}(s,a)$, the optimal action $a^{*}(s)$ can be directly determined as:

$$a^{*}(s) = argmax_{a}\, Q^{*}(s,a)$$

There might exist multiple actions that maximize $Q^{*}(s,a)$. In such cases, all of them are considered optimal, and the optimal policy may randomly select any of them. However, there always exists an optimal policy that deterministically selects an action.

Bellman Equations

The value functions follow special self-consistency equations known as **Bellman equations**. These equations are grounded in the concept that the value of the starting point comprises the expected reward from that point and the value of the subsequent point reached.

Bellman equations provide a recursive framework that decomposes the value of a state or state-action pair into the immediate reward and the expected value of the next state or state-action pair. These equations are essential in RL as they illustrate how value functions change over time.

For the on-policy value functions, the Bellman equations are:

$$V^{\pi}(s) = E_{a\sim\pi, s'\sim P}\left[r(s,a) + \gamma V^{\pi}(s')\right]$$
$$Q^{\pi}(s,a) = E_{s'\sim P}\left[r(s,a) + \gamma E_{a\sim\pi}\left[Q^{\pi}(s',a')\right]\right]$$

Where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s,a)$, indicating that the next state s' is sampled from the environment's transition rules, $a \sim \pi$ is shorthand for $a \sim \pi(\cdot|s)$ and $a' \sim \pi$ is shorthand for $a' \sim \pi(\cdot|s')$

The Bellman equation, in both cases $V^{\pi}(s)$ and $Q^{\pi}(s,a)$, decompose the value into the immediate reward r(s,a) and the expected value of the next state or state-action pair s' or (s',a') under the current policy π. Lastly, γ is the discount factor, which scales future rewards to reflect their diminished importance relative to immediate rewards.

The Bellman equations for the optimal value functions are:

$$V^{*}(s) = max_{a} E_{s' \sim P}[r(s,a) + \gamma V^{*}(s')]$$
$$Q^{*}(s,a) = E_{s' \sim P}[r(s,a) + \gamma max_{a'} Q^{*}(s',a')]$$

In the case of optimal value functions, the Bellman equation considers the maximum over actions max_{a} or action-value pairs $max_{a'}$, reflecting the fact that the agent aims to choose the action that maximizes the expected return.

This inclusion of the maximum reflects the optimal decision-making process, where the agent selects actions that lead to the highest possible value.

Advantage Functions

In RL, it's often more useful to understand how an action compares to others rather than assessing its absolute value. This relative measure is captured by the advantage function.

The advantage function $A^{\pi}(s,a)$ for a policy π describes how much better it is to take a specific action a in state s, compared to randomly selecting an action according to $\pi(\cdot|s')$, assuming you act according to π forever after. Mathematically, the advantage function is defined by:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

This function helps to highlight the benefit of choosing a particular action over the expected value of actions in that state. This can help in speed and the computational complexity of the algorithm.

2.3.2.2 A Taxonomy of RL Algorithms

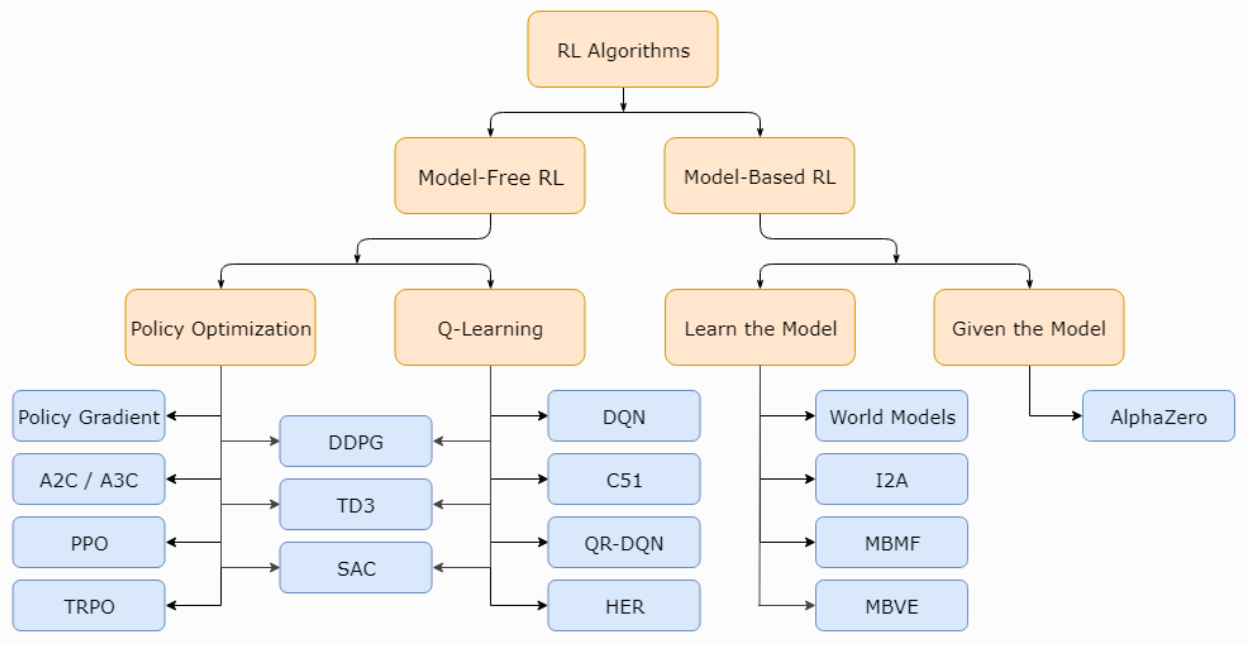


Figure 17 : A non-exhaustive, but useful taxonomy of algorithms in modern RL. (Open AI, Kind of RL Algorithms, 2024)

A taxonomy of algorithms in modern RL is challenging to do due to the adjustable nature and uniqueness of these algorithms.

This tree structure displays the basic design choices in RL algorithms regarding the learning objectives and how to achieve them and to showcase the trade-offs involved in these choices.

Model-Free & Model-Based RL

A major branching point in RL algorithms is whether the agent has access to or learns a model of the environment. A model of the environment is a function that predicts state transitions and rewards.

The main advantage of having a model is that it enables the agent to plan by forecasting the outcomes of various possible choices and explicitly deciding among them. This planning can be formulated into a learned policy. A well-known example of this approach is **AlphaZero** by DeepMind, which mastered chess, shogi, and go (arguably the hardest board games). This method can significantly improve sample efficiency over methods that do not use a model.

The main disadvantage is that agents usually lack a ground-truth model of the environment. To use a model, agents must learn it from experience, which presents challenges. The most significant challenge is that bias in the model can be exploited by the agent, leading to suboptimal or poor performance in the real environment despite good performance with the learned model. **Model learning** is difficult by nature, and substantial effort and resources may not always yield successful results.

Algorithms utilizing a model are called **model-based methods**, while those that do not are known as **model-free methods**. Although model-free methods miss out on potential gains in sample efficiency from using a model, they are generally easier to implement and fine-tune. As a result, model-free methods are more popular and have been more extensively developed and tested than model-based methods.

Learning Objectives in RL

Another critical branching point in an RL algorithm is the question of what the agent should learn.

Learning Objectives in Model-Free RL

There are two main approaches to representing and training agents with model-free RL:

1. Policy Optimization. Methods in this category explicitly represent a policy as $\pi_\theta(a|s)$. They optimize the parameters θ either directly through gradient ascent on the performance objective $J(\pi_\theta)$, or indirectly, by maximizing local approximations of $J(\pi_\theta)$. This optimization is typically performed on-policy, meaning each update only uses data collected while following the most recent version of the policy. Policy optimization often includes learning an approximator $V_\varphi(s)$ for the on-policy value function $V^\pi(s)$,which helps determine how to update the policy.

Examples of policy optimization methods that will be used for the CNC Wear predictor model are:

- **A2C / A3C**, which performs gradient ascent to directly maximize performance,
- and **PPO**, whose updates indirectly maximize performance by optimizing a surrogate objective function that provides a conservative estimate of how much $J(\pi_\theta)$ will change as a result of the update.

2. Q-Learning. This approach focuses on approximating the optimal action-value function $Q^*(s,a)$ with $Q_\theta(s,a)$. Typically, an objective function based on the Bellman equation is used. The optimization is generally performed **off-policy**, allowing updates to utilize data collected at any time during training, regardless of the agent's exploration strategy when the data was obtained. The corresponding policy is derived from the relationship between Q^* and π^*, this means that the actions taken by the Q-learning agent are determined by:

$$a(s) = argmax_a Q_\theta(s,a)$$

Trade-offs Between Policy Optimization and Q-Learning.
Policy optimization methods are good in their own approach, directly optimizing for the desired outcome. This direct optimization tends to enhance stability and reliability. In contrast, Q-learning methods indirectly optimize agent performance by training Q_θ to satisfy a self-consistency equation. This indirect approach introduces various potential failure modes, making Q-learning less stable. However, Q-learning methods are significantly more sample-efficient when effective, as they can reuse data more efficiently than policy optimization techniques.

Interpolating Between Policy Optimization and Q-Learning.
Policy optimization and Q-learning are not mutually exclusive and can sometimes be equivalent. A range of algorithms exists that blend the two approaches, allowing for a careful balance between their respective strengths and weaknesses. Examples of such algorithms include:

- **DDPG**, an algorithm which concurrently learns a deterministic policy and a Q-function by using each to improve the other,

- and **SAC**, a variant which uses stochastic policies, entropy regularization, and a few other tricks to stabilize learning and score higher than DDPG on standard benchmarks.

Learning Objectives in Model-Based RL

Model-based RL encompasses a variety of methods that resist simple categorization into a few clusters. Contrasting model-free RL, there are many unique approaches to using these models.

In the case of the CNC Wear Prediction Model, the environment is complex, so model-based reinforcement learning is not a suitable choice because constructing an accurate and reliable model of the environment is highly challenging. Complex environments involve numerous variables, intricate dynamics, and stochastic elements that are difficult to capture accurately. Any inaccuracies or biases in the model lead to poor decision-making and suboptimal policies, as the model's predictions will not accurately reflect the real-world environment. Additionally, developing and maintaining such a model would be computationally intensive and time-consuming, making model-based approaches less practical for complex environments compared to model-free methods that directly learn from interactions with the environment using our Phm Dataset.

2.3.2.3. Policy Optimization

The mathematical basis of policy optimization algorithms will be briefly analyzed in this section.

Deriving the Simplest Policy Gradient

Here, we consider the case of a stochastic, parameterized policy, π_θ. We aim to maximize the expected return.
Consider a stochastic, parameterized policy π_θ with the goal of maximizing the expected return:

$$J(\pi) = \int P(\tau|\pi) \cdot R(\tau) = E_{\tau\sim\pi}[R(\tau)]$$

For this derivation, $R(\tau)$ represents the finite-horizon undiscounted return. The process for deriving the infinite-horizon discounted return is nearly identical.

Optimizing the policy through gradient ascent involves:

$$\theta_{k+1} = \theta_k + a\,\nabla_\theta J(\pi_\theta|_{\theta_k})$$

The gradient of policy performance, $\nabla_\theta J(\pi_\theta)$, is called the policy gradient, and algorithms that optimize the policy this way are called policy gradient algorithms. Examples include Vanilla Policy Gradient and TRPO. PPO (in an abstract way)

To implement this algorithm, an expression for the policy gradient that can be computed numerically is required. This process involves two main steps:

- Deriving the analytical gradient of policy performance, which has the form of an expected value.
- Creating a sample estimate of that expected value, computed using data from a finite number of agent-environment interaction steps.

Key facts that are useful for deriving the analytical gradient:

1. **Probability of a Trajectory**. The probability of a trajectory $\tau = (s_0, a_0, \ldots, s_{T+1})$ given that actions are drawn from π_θ is:

$$P(\tau|\theta) = \rho_o(s_o) \cdot \prod_{t=o}^{T} P(s_{t+1}|s_t, a_t) \cdot \pi_\theta(\alpha_t|s_t)$$

2. **The Log-Derivative Trick**. This method leverages a basic calculus principle: the derivative of $log(x)$ with respect to x is $\frac{1}{x}$. When rearranged and combined with chain rule, we get:

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\,\nabla_\theta log(P(\tau|\theta))$$

3. **Log-Probability of a Trajectory**. The log-prob of a trajectory is:

$$log(P(\tau|\pi)) = log(\rho_o(s_o)) + \sum_{t=0}^{T} (log(P(s_{t+1}|s_t, a_t) + log(\pi_\theta(\alpha_t|s_t)))$$

4. **Gradients of Environment Functions**. The environment has no dependence on θ, so gradients of $\rho_o(s_o)$, $P(s_{t+1}|s_t, a_t)$, and $R(\tau)$ are zero.

5. **Grad-Log-Prob of a Trajectory**. The gradient of the log-prob of a trajectory is:

$$\nabla_\theta log(P(\tau|\theta)) = \cancel{\nabla_\theta log(\rho_o(s_o))} + \sum_{t=0}^{T} \left(\cancel{\nabla_\theta log(P(s_{t+1}|s_t, a_t))} + \nabla_\theta log(\pi_\theta(\alpha_t|s_t))\right)$$

$$\nabla_\theta log(P(\tau|\theta)) = \sum_{t=0}^{T} \nabla_\theta log(\pi_\theta(a_t|s_t))$$

Combining the above, we derive the following:

Derivation for Basic Policy Gradient:

The derivation for the basic policy gradient starts with the objective to find the gradient of the expected return with respect to the policy parameters θ:

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta E_{\tau\sim\pi}[R(\tau)]$$

This expectation can be represented as an integral over all possible trajectories τ:

$$\nabla_\theta E_{\tau\sim\pi}[R(\tau)] = \nabla_\theta \int P(\tau|\theta)R(\tau)$$

The gradient can be moved inside the integral:

$$\int \nabla_\theta P(\tau|\theta)R(\tau)d\tau$$

Next, apply the log-derivative trick, which utilizes the identity $\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta logP(\tau|\theta)$:

$$\int P(\tau|\theta)\nabla_\theta logP(\tau|\theta)R(\tau)d\tau$$

This expression can be reformulated back into expectation form:

$$E_{\tau\sim\pi}[\nabla_\theta logP(\tau|\theta)R(\tau)]$$

Recognizing that the probability of a trajectory $P(\tau|\theta)$ under policy π_θ can be decomposed into the product of the probabilities of individual actions given states:

$$P(\tau|\theta) = P(s_o)\prod_{t=0}^{T-1} \pi_\theta(a_t|s_t)P(s_{t+1}|s_t, a_t)$$

Where $\pi_\theta(a_t|s_t)$ represents the policy's probability of taking action a_t in state s_t, the log-probability of the trajectory is given by:

$$logP(\tau|\theta) = \sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t)$$

Substituting this back into the expectation:

$$E_{\tau\sim\pi}\left[\nabla_\theta \sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t)R(\tau)\right]$$

Since the gradient operator is linear, it can be moved inside the summation:

$$E_{\tau\sim\pi}\left[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)\right]$$

The final expression for the gradient of the expected return $J(\pi_\theta)$ is:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi}\left[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)\right]$$

This represents the basic policy gradient theorem, indicating that the gradient of the expected return can be estimated by sampling trajectories, computing the gradient of the log-probabilities of the actions taken, and weighting these gradients by the returns. This method allows for the optimization of policy parameters in RL by iteratively adjusting them to maximize the expected return.

To estimate the final expression, which is an expectation, a sample mean can be used. By collecting a set of trajectories $D = \{\tau_i\}_{i=1,\dots,Z}$, where each trajectory results from the agent acting in the environment with policy π_θ, the policy gradient is estimated as follows:

$$\hat{g} = \frac{1}{|D|}\sum_{t\epsilon D}\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)$$

In this formula, $|D|$ denotes the number of trajectories in D (in this case, Z).

This is the most straightforward version of the desired computable expression. Assuming the policy is represented in a manner that allows calculation of $\nabla_\theta log\pi_\theta(a_t|s_t)$, and the policy can be executed in the environment to gather trajectory data, then the policy gradient can be computed, and an update step can be performed.

2.4 Reinforcement Learning in Maintenance

RL has significant potential in the field of maintenance, particularly in predictive and preventive maintenance strategies. In maintenance, RL algorithms can be used to optimize the scheduling of maintenance activities to minimize downtime and extend the life of equipment. By learning from data on equipment performance and failure patterns, an RL agent can develop policies that determine the best times to perform maintenance tasks, balancing the costs of maintenance with the risks of equipment failure.

One key application is predictive maintenance, where RL models analyze real-time data from sensors to predict when a machine is likely to fail and recommend precautionary actions. This approach helps in avoiding unexpected breakdowns and reduces the overall maintenance cost. Additionally, RL can optimize inventory management for spare parts, ensuring that the right parts are available when needed without overstocking or understocking which results in downtime.

The application of RL in predictive maintenance is modeled in a structured manner to effectively address the complexities of maintenance strategies. The structured approach proposed below outlines the key steps and methodologies involved in implementing RL for predictive maintenance, ensuring a comprehensive and efficient solution. The following taxonomy, adapted from various expert sources and demonstrated in the paper (Siraskar, Kumar, Patil, Bongale, & Kotecha, 2023), illustrates the detailed process and categorization of RL techniques used in predictive maintenance.

The model approach of RL in predictive maintenance is displayed below:

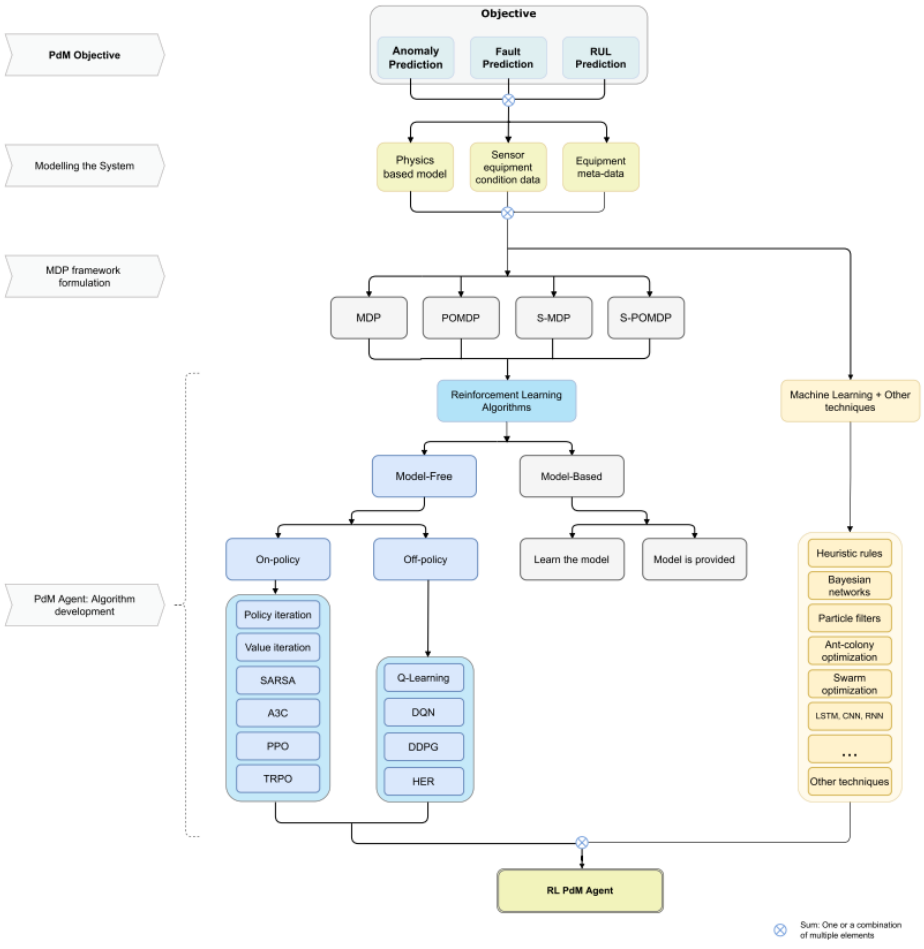


Figure 1 A taxonomy of Reinforcement Learning in Predictive Maintenance

The diagram displays the approach for implementing RL in PdM, structured into key sections.

The primary objectives include anomaly prediction, fault prediction, and estimating the remaining useful life (RUL) of equipment.

To achieve these goals, the system can be modeled using physics-based models, real-time sensor data, and historical equipment meta-data.

Various frameworks for formulating the problem are employed, such as MDP, Partially Observable Markov Decision Processes (POMDP), Semi-Markov Decision Processes (S-MDP), and Semi-Partially Observable Markov Decision Processes (S-POMDP).

Reinforcement learning algorithms are categorized into model-free and model-based methods. Model-free methods, which do not rely on an environment model, are divided into on-policy and off-policy approaches. On-policy methods optimize based on data from the current policy and include techniques like policy iteration, value iteration, SARSA **(**State–action–reward–state–action**)**, A3C, PPO, and TRPO (Trust Region Policy Optimization). Off-policy methods, such as Q-learning, DQN, DDPG, and HER (Hindsight Experience Replay), use data from various policies.

Model-based methods utilize an environment model, which can either be learned from data or provided beforehand. These methods allow the agent to plan by simulating different scenarios and optimizing its actions based on the predicted outcomes.

In addition to RL algorithms, other techniques can be employed to enhance predictive maintenance capabilities. The sum of these processes results in the creation of an RL PdM Agent.

The integration of various techniques within this structured approach results in an RL PdM Agent capable of making informed, data-driven decisions, ultimately extending the lifespan of machinery and improving overall operational efficiency. The proposed approach will be discussed in detail in the following chapters, highlighting its potential benefits and implementation strategies in the context of predictive maintenance.

3. Approach

3.1 The objective

This chapter analyzes the process of modeling a reinforcement learning (RL) problem using data collected from sensors. Initially, this data will be processed and subsequently transformed into a Markov Decision Process (MDP). Through the MDP, a suitable environment will be developed for the implementation of various RL algorithms, which will be trained on the processed data set. The last step is, after the training, the algorithms will be evaluated to determine their effectiveness. If their performance is satisfactory, they can be utilized to perform predictions related to predictive maintenance.

This process is displayed in the figure below:

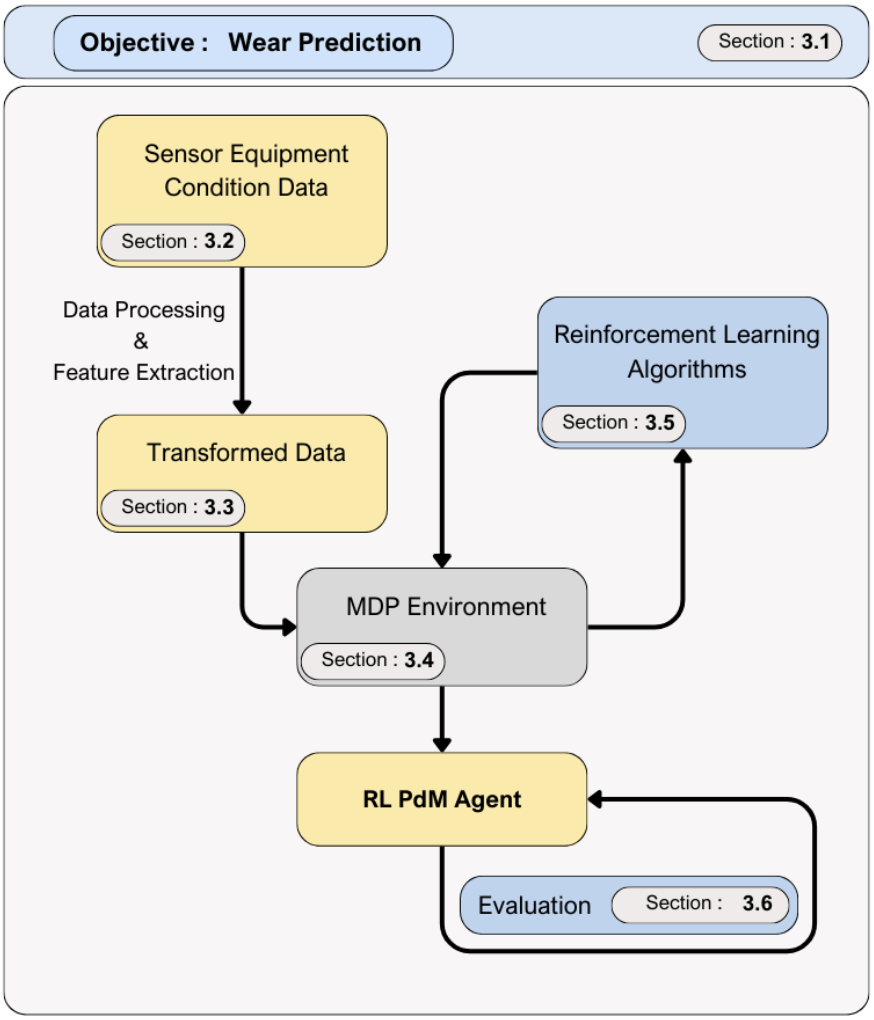


Figure 18 : Structure of the Proposed Approach

Table 5 : Input & Output of each step of the Proposed Approach

Section	Step	Input	Output
3.2	Data Structure	Sensor Signals	Structured Data
3.3	Data Processing	Structured Sensor Data	Transformed Data
3.4	MDP	Transformed Data Reinforcement Learning Algorithm	Observation Space, Action Space, Reward Function, Episode Information
3.5	RL Algorithm	Observation, Reward	Action
3.6	Evaluation	Trained RL Model	Evaluation Metrics

3.2 Data structure and input parameters

In the context of predictive maintenance and degradation prediction, understanding the data structure and input parameters is crucial. The dataset typically consists of time-series data collected from various sensors attached to machinery. The data gathered from the sensors is formulated in a table form with each row corresponding to a sensor value at a specific timeframe. These sensors capture critical parameters such as vibrations, temperature, and pressure over time. By continuously monitoring these parameters, it becomes possible to detect anomalies and predict equipment failures before they occur. The integrity and quality of this data are paramount, as they form the foundation for subsequent analysis and predictive modeling.

The input parameters include not only the raw sensor readings but also derived features that help in capturing the underlying patterns and trends in the data. These features can be statistical metrics, frequency domain features, or other complex transformations that reveal more about the machine's operational state. Properly selecting and engineering these features is a key step in building effective predictive models. Features must be chosen based on their relevance to the wear and tear of the machinery, ensuring they provide meaningful insights into the equipment's condition.

Proper configuration and preprocessing of these parameters are vital to ensure the accuracy and reliability of the predictions. This process involves cleaning the data to remove noise and errors, normalizing it to ensure consistency, and possibly augmenting it to create a more robust dataset. By meticulously preparing the data, it is possible to enhance the performance of predictive algorithms, leading to more accurate and reliable maintenance schedules. Thus, understanding and structuring the data appropriately is the first critical step in the journey towards effective predictive maintenance.

In order to preprocess the data, it is essential to know what needs to be cleaned. Therefore, the first step is to visualize the data. Visualization allows for the exploration and understanding of the data's characteristics, distributions, and potential anomalies. By creating plots and charts, such as time series plots, histograms, and scatter plots, patterns, trends, and outliers can be identified. This initial exploratory data analysis (EDA) helps in recognizing any inconsistencies or irregularities that need to be addressed during preprocessing. Visualization not only aids in detecting errors and noise but also provides insights into the data's structure, which is crucial for effective feature engineering and subsequent modeling steps.

3.3 Pre-processing & Feature Extraction

Pre-processing and feature extraction are fundamental steps in preparing the data for predictive maintenance models. These steps involve transforming raw sensor data into meaningful features that capture the essential characteristics of the equipment's operational state derived from the sensor's feedback. The transformation process typically includes cleaning, normalization, and extraction of relevant features. This is done to enhance the quality of the data, making it more suitable for machine learning models. By reducing noise and correcting for any inconsistencies, pre-processing ensures that the data accurately reflects the true condition of the machinery (the true value of the sensors).

Feature extraction plays a critical role in this process by deriving new variables from the raw data that are more informative for predicting equipment wear and failures. These features can be statistical summaries like mean and variance, signal processing metrics like root mean square (RMS), or domain-specific indicators. The goal is to distill the raw data into a set of features that provide the most insight into the machine's health, thereby improving the predictive power of the models. This step is essential for handling the high-dimensional nature of sensor data and focusing on the most relevant aspects.

Through careful pre-processing and feature extraction, it becomes possible to create a more manageable and informative dataset. This dataset not only improves the accuracy of predictive models but also reduces computational complexity. Efficient feature extraction techniques can significantly tackle the **curse of dimensionality** and enhance the ability to detect early signs of wear and predict failures, leading to more effective maintenance strategies. As such, pre-processing and feature extraction are irreplaceable steps in the workflow of predictive maintenance.

3.3.1 Curse of Dimensionality

The "**curse of dimensionality**" is a concept that describes the challenges and negative effects that arise when working with high-dimensional data in various fields like machine learning and other fields. It was first popularized by Richard Bellman in his book "Dynamic Programming" (1957). As the number of dimensions in a dataset increases, various issues emerge, complicating analysis and modeling.

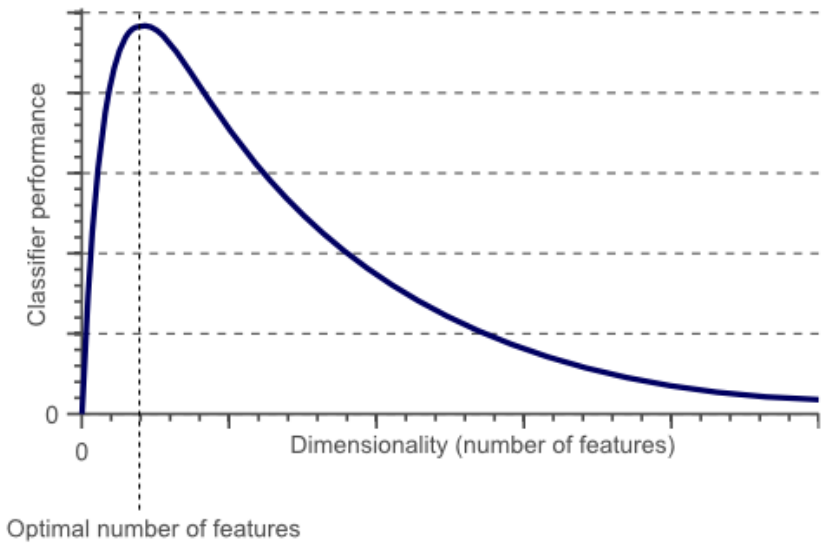


Figure 19 : Curse of Dimensionality Graph

First, **computational complexity** escalates dramatically with the increase in dimensionality. This complexity arises from the exponential growth in the number of possible combinations or configurations of data points as you add more dimensions. This exponential increase leads to higher demands on computational resources, making tasks like searching, sorting, or performing clustering in high-dimensional space increasingly time-consuming and resource-intensive.

Relevant Research done by (Beyer & Goldstein, 1999) demonstrated that common indexing techniques used in low-dimensional spaces fail to work efficiently in high-dimensional spaces due to the exponential increase in search complexity.

Another significant issue is the inherent **sparsity** that comes with high-dimensional data. When you expand into more dimensions, data points are spread out over a larger space, resulting in fewer points within any given volume. This sparsity makes it challenging to identify meaningful patterns or relationships among the data. It also complicates statistical analysis, as you need more data points to achieve statistical significance in a high-dimensional space. (Bellman, 1961) in "Adaptive Control Processes: A Guided Tour" discussed how adding more dimensions makes it exponentially harder to estimate functions due to sparse data. Also, (Aggarwal, Hineeburg, & Keim, 2002) illustrated how data becomes sparse in high-dimensional spaces, affecting nearest-neighbor algorithms and clustering.

The **distance measures and concentration of distances** is another problem within the curse of dimensionality. As the number of dimensions grows, the distances between data points tend to converge, which can render distance-based metrics like Euclidean distance less effective. When distances become similar, traditional methods for clustering or identifying nearest neighbors lose their discriminative power, making it difficult to distinguish between similar and dissimilar data points.

In addition to computational complexity, sparsity, and the concentration of distances, high-dimensional data often leads to **overfitting** in machine learning models. Overfitting occurs when a model learns the specific noise or random fluctuations in the training data rather than capturing general patterns. With more

dimensions, models become highly flexible, capable of fitting data points precisely, but at the cost of generalization. This risk of overfitting can result in poor performance on new or unseen data. (Beyer & Goldstein, 1999) also analyzed how distances converge in high-dimensional data, leading to concentration, which reduces the effectiveness of distance-based methods like k-Nearest Neighbors and clustering. (Pestov, 2000) in his paper “On the geometry of similarity search: Dimensionality curse and concentration of measure” discussed how traditional distance measures lose discriminative power in high-dimensional spaces.

Lastly, High-dimensional data is hard to visualize, making exploratory data analysis more difficult.

Overall, the curse of dimensionality encompasses the various difficulties that arise when the dimensionality of a dataset is very high, to tackle this problem the **Time domain Features** are a possible solution.

3.3.2 Time domain Features

Figure 20 : Time Domain Data Analysis

Time domain features are crucial in industry, particularly for predictive maintenance, where the goal is to anticipate equipment failures and minimize downtime. In industrial settings, machinery and equipment are monitored continuously, generating vast amounts of time-series data from sensors that measure vibrations, temperature, pressure, and other critical parameters. Time domain features allow engineers and data scientists to analyze this data efficiently, extracting meaningful patterns and identifying early signs of wear or impending failure. Features such as mean, variance, root mean square (RMS), and zero-crossing rate offer insights into the operational health of equipment. By monitoring these features over time, maintenance teams can detect anomalies, such as an increase in vibration or a shift in temperature patterns, which often indicate that a component is nearing failure. This early detection enables proactive maintenance, reducing unscheduled downtime and extending the lifespan of machinery.

Time domain features refer to characteristics derived from analyzing time-series data - data points collected or observed at different time intervals. These features are crucial for understanding the underlying patterns, trends, and relationships in the data and are used in a wide range of applications.

Types of Time Domain Features

Time domain features can be categorized based on their function and purpose:

Basic Statistical Time Domain Features:

Mean: The average value over a given time period. It provides a sense of central tendency.

$$Mean\ =\ \mu\ =\frac{1}{N}\sum_{\iota=1}^{N}x_i$$

Median: The middle value when data is arranged in ascending order, it is a good measure against outliers.

$$Median\ =\ middle\ value\ of\ sorted\ \{x_1,x_2,\ldots,x_N\}$$

Standard Deviation: Reflects the dispersion or spread of data around the mean, it is indicating variability.

$$Standard\ Deviation=\sigma\ =\ \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$

Variance: Represents the average squared deviation from the mean, providing a sense of the distribution's spread.

$$Variance=\sigma^2=\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$$

Range: The difference between the maximum and minimum values in a time series, indicating the total span of the data.

$$Range\ =\ x_{max}\ -\ x_{min}$$

Signal Processing Features:

Root Mean Square (RMS): This measures the square root of the mean of the squares of all values in the time series. RMS is often used in engineering and signal processing to determine the magnitude or energy of a signal, providing a more accurate measure of a signal's power compared to simple averages.

$$RMS\ =\ \sqrt{\frac{1}{N}\sum_{i=1}^{N}{x_i}^2}$$

Zero-Crossing Rate: This feature indicates how frequently a signal changes sign, which is useful in identifying high-frequency components or rapid changes in a signal. It is commonly used in audio

and speech processing to differentiate between voiced and unvoiced speech segments, among other applications.

$$Zero - Crossing\ Rate\ = \{X(t) < 0\ and\ X(t+1) > 0\}\ \boldsymbol{or}\ \{X(t) < 0\ and\ X(t+1) > 0\}$$
$$|X(t) - X(t+1)|\ \geqslant\ \epsilon\ ,$$

where ϵ is a threshold to avoid miscounting zero crossing due to noise.
(Torres-Garcia, Mendoza, Reyes-Garcia, & Villasenor-Pineda, 2022)

Autocorrelation: This measures the correlation of a time series with its lagged version. It helps identify repeating patterns, trends, or cyclic behavior in a dataset. Autocorrelation is valuable in analyzing seasonality in time-series data or detecting hidden cycles.

$$Autocorrelation(\tau) = \frac{1}{N\ -\ \tau}\sum_{i=1}^{N-\tau}(x_i \cdot x_{i+\tau})$$

Energy: This is a measure of the total energy contained in a time-series signal. It provides insights into the intensity or activity level of a signal, commonly used in vibration analysis and fault detection.

$$Energy\ =\ \sum_{i=1}^{N} x_i^2$$

Area Under the Curve (AUC): This is the total area between the time series and the x-axis over a specified interval. AUC is useful for understanding the cumulative effect of a time series, often applied in physiological data analysis to measure total activity or response over time.

$$AUC\ =\ \int_{t_1}^{t_2} x(t)dt$$

Crest Factor: This is the ratio between the peak value and the RMS value, indicating the dynamic range of a signal. It is commonly used in mechanical engineering to detect anomalies in vibration patterns.

$$Crest\ Factor\ =\ \frac{max[x_i]}{RMS}$$

Higher-Order Statistical Features:

Skewness: Indicates the asymmetry of the data distribution. A skewed time series may suggest trends or biases.

$$Skewness\ =\frac{E\left[\left(x_i\ -\ \underline{x}\right)^3\right]}{RMS^3}$$

Kurtosis: Reflects the "tailedness" of the data distribution, which can reveal the presence of outliers or extreme events.

$$Kurtosis = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_i - \underline{x})^4}{RMS^4}$$

These time domain features serve as an essential tool for making sense of time-series data. They provide a concise yet comprehensive way to understand the underlying patterns, detect anomalies, and reduce data complexity, contributing to better insights and more effective applications across a wide range of fields.

Understanding and structuring the data appropriately is the first critical step in the journey towards effective predictive maintenance. Once the data has been cleaned, visualized, and features have been extracted, the next phase involves modeling the problem as a Markov Decision Process (MDP) to enable the application of reinforcement learning (RL) algorithms.

3.4 Modeling an MDP

In order to structure the Markov Decision Process, we need to carefully define the environment in which the agent will take actions.

As it was mentioned in 2.2.1 MDP | Sequential Decision Problem, a MDP consists of the following components: States Space, Action Space, Transition Model, Reward Function and the starting state distribution ρ_0.

Therefore, an MDP is a 5-tuple, (S, A, R, P, ρ_0), where:

- S is the set of all valid states,
- A is the set of all valid actions,
- $R: S \times A \times S \to R$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1})$
- $P: S \times A \to P(S)$ is the transition probability function, with $P(s'|s,a)$ being the probability of transitioning into state s' if you start in state s and take action a,
- and ρ_0 is the starting state distribution.

In the context of the degradation prediction problem, the above parameters of the 5-tuple MDP need to be configured and mapped to the following:

State S_t : The state at any time step t is represented by the condition of the machine parts and the value of the sensors attached to the machines at t. The intent is to predict the wear overtime so specific information need to be observed in order to induce the further wear of the machine. The vector containing all the information of the environment is called the state of the environment.

The set S contains all the valid states $S_t, \forall t$ of the environment, therefore it contains all the different wear states and values the examined equipment may attain.

Action a_t : The action space consists of a n-dimensional space representing the wear of the n component of the machine we are monitoring.

$a_t = [action_1, action_2, \ldots, action_n]$, where:

$action_1$ $(prediction_1)$: The prediction of the agent for the additional wear of the component 1 at timestep t

$action_2$ $(prediction_2)$: The prediction of the agent for the additional wear of the component 2 at timestep t

$action_n$ $(prediction_n)$: The prediction of the agent for the additional wear of the component n at timestep t

The action is the corresponding additional wear that the agent predicts for the component. This vector can be either discrete or continuous depending on the PdM Objective. In the case of wear prediction, a continuous action space is more fitting, since the wear is not a discrete metric.

If the objective was a Remaining Useful Life Prediction the action space would be different, and the vector might be of a different type (float, integer). However, the model would operate in a similar manner.

Transition Probabilities (P): The transition probabilities define how the state changes in response to actions. Depending on how the machine operates the resulting induced wear is a stochastic phenomenon. Because it is a stochastic phenomenon, the way predictions are managed is through calculating the probability of the given wear occurring given the current state. The name Markov Decision Process refers to the fact that the system obeys the **Markov property**, which means that transitions only depend on the most recent state and action, and not on prior history. The probability of transitioning to state s_{t+1} given the current state s_t and action a_t is denoted by $P(s_{t+1}|s_t, a_t)$

Reward R_t : The reward function is designed to provide feedback to the agent based on the accuracy of its predictions. Depending on the objective a different reward function will be used, penalizing bad behavior and rewarding good behavior-actions. Specifically for wear prediction, the reward can be computed based on the difference between the predicted wear and the actual wear observed. Although this is a credible method, more advanced ways of scoring the reward are usually used called score functions. The score functions behave differently in the case of a bad action and a good action in order to help the agent understand. The general formula of a reward can be:

$$R_t = Score_f(\, predicted\ value\ - actual\ value\,)$$

Starting state distribution ρ_0 defines the probability distribution over the initial states from which the agent begins its decision-making process in the environment. This distribution is crucial as it sets the initial conditions and significantly influences the early stages of learning and exploration. In the context of the degradation prediction problem, ρ_0 represents the likelihood of various initial wear conditions and sensor readings of the machine at the start of the monitoring period. The initial state could be determined based on historical data, reflecting common starting conditions observed in past machine operations. Alternatively, in a model-based approach it could be initialized randomly within a feasible range defined by the physical and operational constraints of the machinery.

A well-defined ρ_0 ensures that the agent experiences a realistic range of initial conditions during training, promoting robust learning and better generalization to real-world scenarios. It is important to capture the variability and distribution of initial states accurately, as this affects how the agent learns to handle different wear patterns and predict maintenance needs effectively.

To solve the degradation prediction problem with RL, it is essential to carefully design the environment in which the agent will operate. The following sections will elaborate on how RL works in practice, and the potential algorithms that can be used for this problem.

3.5 Solving with RL

In this section, we will explore how RL works in practice and examine the potential algorithms we can use for this problem.

3.5.1 VPG | Vanilla Policy Gradient

The Vanilla Policy Gradient (VPG) algorithm (Sutton, McAllester, Singh, & Mansour, 2000) is founded on the principle of adjusting the probabilities of actions to maximize returns. Specifically, the algorithm increases the probabilities of actions that yield higher returns and decreases the probabilities of actions that lead to lower returns. This iterative process continues until the optimal policy is achieved.

VPG is an on-policy algorithm, meaning it learns from actions taken based on the current policy. It is versatile and can be applied to environments with either discrete or continuous action spaces.

The mathematical formulation of VPG involves defining a policy π_θ, with parameters θ and the expected finite-horizon undiscounted return of the policy, $J(\pi_\theta)$. The gradient of $J(\pi_\theta)$ is given by:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\, \pi_\theta(a_t|s_t) \cdot A^{\pi_\theta}(s_t, a_t)\right]$$

where τ is a trajectory and A^{π_θ} is the advantage function for the current policy.

The policy gradient algorithm updates the policy parameters through stochastic gradient ascent on policy performance:

$$\theta_{k+1} = \theta_k + a\nabla_\theta J(\pi_{\theta_k})$$

In practice, policy gradient implementations often estimate the advantage function based on the infinite-horizon discounted return, although they use the finite-horizon undiscounted policy gradient formula.

VPG trains a stochastic policy (Schulman, Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs, 2016) in an on-policy manner, meaning it explores the environment by sampling actions according to the latest version of its stochastic policy. The randomness in action selection depends on the initial conditions and the training procedure. As training progresses, the policy typically becomes less random, as the update rule encourages the exploitation of rewards that have already been found. This can sometimes lead to the policy becoming trapped in local minimum, balancing the trade-off between exploration and exploitation.

Pseudocode

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

Figure 21 : Pseudocode of the VPG Algorithm Implementation by (Open AI, Vanilla Policy Gradient, 2024)

Explanation of Pseudocode in the Displayed Steps

1: Initialization of parameters, the parameters are used to initialize the policy and the value function that will be optimized.

2: The algorithm runs for k = 0,1,2, …, looping and improving the policy.

3: A set of trajectories is collected by running the current policy in the environment. Each trajectory is a sequence of states, actions and rewards obtained by following the policy.

4: Calculate the rewards-to-go for each time step in the trajectory. The rewards-to-go represent the cumulative future rewards from a given state.

5: Compute the advantage estimates using any method of advantage estimation, based on the current value function. Advantage estimates indicate how much better or worse an action is compared to the expected value from that state.

6: The policy gradient $\hat{g}_k$ is estimated as the average of the gradients of the log-probabilities of the actions taken, weighted by the advantage estimates.

7: Update the policy parameters θ using any gradient ascent algorithm

8: The objective is to fit the value function by performing regression on the mean-squared error between the value function predictions and the rewards-to-go. "Fit the value function" means to update the parameters of the value function by minimizing the mean-squared error between the predicted values and the actual rewards observed, ensuring the value function accurately reflects the expected returns.

9: The process repeats, iteratively refining the policy and value function parameters to maximize the expected return.

3.5.2 TRPO | Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015) updates policies by taking the largest step possible to improve performance while ensuring the new and old policies remain close. This closeness is measured using **KL-Divergence**, which quantifies the difference between probability distributions.

Unlike standard policy gradient methods that keep policies close in parameter space, TRPO focuses on the performance space. Small differences in parameters can lead to significant performance variations, making large steps risky in vanilla policy gradients. TRPO avoids such pitfalls, ensuring more stable and efficient learning.

TRPO is an on-policy algorithm suitable for both discrete and continuous action spaces.

Mathematical Formulation

Let π_θ denote a policy with parameters θ. The theoretical TRPO update is:

$$\theta_{k+1} = \ argmax_\theta \mathcal{L}(\theta_k, \theta) \ \ such\ that \ \ \overline{D}_{KL}(\theta||\theta_k) \leq \delta$$

Here $\mathcal{L}(\theta_k, \theta)$ is the surrogate advantage, measuring how the new policy π_θ performs relative to the old policy π_{θ_k}using data from the old policy π_{θ_k}:

$$\mathcal{L}(\theta_k, \theta) \;=\; E_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right]$$

$\underline{D}_{KL}(\theta||\theta_k)$ is an average KL-divergence between policies across states visited by old policy, the KL-divergence constraint is:

$$\overline{D}_{KL}(\theta||\theta_k) = \; E_{s\sim\pi_{\theta_k}}\left[D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)\right)\right]$$

The theoretical TRPO update poses practical challenges, approximations are a good trick for quick solutions. To address this, TRPO employs a Taylor expansion of the objective and constraint around θ_k:

$$\mathcal{L}(\theta_k, \theta) \;\approx\; g^T(\theta - \theta_k)$$
$$\overline{D}_{KL}(\theta||\theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)$$

This results in an approximate optimization problem,

$$\theta_{k+1} = argmax_\theta \; g^T(\theta - \theta_k) \;\; such\; that \;\; \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \le \delta$$

Solving this with Lagrangian duality (Boyd & Vandenberghe, 2009) gives:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

A problem is that, due to the approximation errors introduced by the Taylor expansion, this may not satisfy the KL constraint, or actually improve the surrogate advantage. TRPO adds a modification to this update rule: a backtracking line search,

However, due to approximation errors from the Taylor expansion, this may not always satisfy the KL constraint or improve the surrogate advantage. If we were to stop here, and just use this final result, the algorithm would be exactly calculating the Natural Policy Gradient (Kakade, 2001). TRPO refines this approach with a backtracking line search:

$$\theta_{k+1} = \theta_k + a^{\,j}\sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

where $a \in (0,1)$ is the backtracking coefficient and j is the smallest nonnegative integer that meets the KL constraint and produces a positive surrogate advantage.

Computing and storing the matrix inverse H^{-1} is computationally expensive, especially with neural network policies involving thousands or millions of parameters. TRPO avoids this issue by using the conjugate gradient algorithm to solve $H_x = g$ for $x = H^{-1}g$. This approach requires only a function to compute the matrix-vector product H_x rather than the entire matrix H. This is achieved by setting up a symbolic operation to calculate

$$H_x = \nabla_\theta((\nabla_\theta \bar{D}_{KL}(\theta||\theta_k))^T x)$$

This formula returns the current output without computing the whole matrix.

Exploration vs. Exploitation

TRPO is an algorithm that trains a stochastic policy following an on-policy method, exploring by sampling actions according to the latest version of its stochastic policy. The randomness in action selection decreases over time as the policy exploits known rewards. This gradual reduction in randomness encourages the policy to leverage the rewards it has found, though it may sometimes get trapped in local minimum.

Pseudocode

Algorithm 1 Trust Region Policy Optimization

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

for $k = 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Figure 22 : Pseudocode of the TRPO Algorithm Implementation by (Tensorflow, 2024)

Explanation of Pseudocode in the Displayed Steps

1: The algorithm starts by initializing the policy parameters θ_0 and the value function parameters φ_0. These are the starting points for the policy and value function that will be iteratively updated.

2: The KL-divergence limit δ, backtracking coefficient a, and the maximum number of backtracking steps K. These hyperparameters control the extent of updates and ensure that changes to the policy do not diverge too drastically.

3: For each iteration k, the algorithm performs a series of steps to update the policy.

4: The current policy is used to run the environment and collect a set of trajectories.

5: The rewards-to-go are calculated for each time step. This involves summing future rewards, providing an estimate of the total expected reward from each state-action pair.

6: Advantage estimates are computed using the current value function V_{φ_k}. The advantage function provides a measure of how much better an action is compared to the average action at a given state.
7: The policy gradient is estimated by computing the gradient of the log probability of actions taken, weighted by the advantage estimates. This gradient points in the direction of improving the policy.
8: To avoid the computational expense of directly inverting the Hessian matrix, the conjugate gradient algorithm is used to approximate $H^{-1}g$.
9: The policy is updated using a backtracking line search to ensure the new policy satisfies the KL-divergence constraint. This step involves iteratively adjusting the step size to find the maximum feasible step that improves the policy.
10: The value function parameters are updated by minimizing the mean-squared error between the predicted values $V_{\varphi}(s_t)$ and the rewards-to-go $\widehat{R_t}$. This regression step helps the value function accurately predict future rewards.
11: The loop continues until the algorithm converges or the maximum number of iterations is reached. The policy and value function are iteratively improved through this process.

3.5.3 PPO | Proximal Policy Optimization

Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) is designed to take the largest possible improvement step on a policy using the available data without risking performance collapse. Unlike TRPO, which uses a complex second-order method, PPO employs simpler first-order methods with additional techniques to ensure new policies remain close to old ones, achieving similar performance with greater implementation ease.

PPO combines elements from A2C (using multiple workers) and TRPO (Trust Region for actor improvement). The main idea is to keep the new policy not too far from the old policy after an update, using clipping to prevent large updates.

PPO is an on-policy algorithm suitable for environments with both discrete and continuous action spaces. There are two primary variants of PPO:

> **PPO-Penalty** addresses the KL-constrained update similarly to TRPO but takes a different approach by incorporating a penalty for KL-divergence directly into the objective function. Instead of enforcing a hard constraint, it dynamically adjusts the penalty coefficient during training to ensure it is appropriately scaled, promoting stable and efficient policy updates.
>
> **PPO-Clip** avoids using a KL-divergence term or any hard constraints in the objective function. Instead, it employs specialized clipping to prevent the new policy from deviating significantly from the old policy. This method effectively maintains policy stability without the need for complex calculations. Our implementation focuses exclusively on PPO-Clip due to its simplicity and effectiveness within our Tech Stack (stable-baselines 3 library).

Mathematical Formulation
PPO-clip policies update with this formula:

$$\theta_{k+1} = argmax_{\theta} E_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)]$$

This involves taking multiple steps of Stochastic Gradient Descent (SGD) (Bottou, 1991) to maximize the objective. Here, L is given by:

$$L(s,a,\theta_k,\theta) = min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a)\right)$$

where ε is a small hyperparameter that limits how far the new policy can deviate from the old one.

The clipping function is defined as:

$$clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon\right)$$

This function ensures that the ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$is constrained within the interval $[1-\varepsilon, 1+\varepsilon]$. If the ratio exceeds this range, it is "clipped" to stay within the bounds. This constraint prevents excessive updates to the policy, maintaining the new policy close to the old one, and thus avoids large, potentially destabilizing updates. This is crucial for the stability and reliability of policy updates in PPO.

The formula $L(s,a,\theta_k,\theta)$ is complex and it essentially helps to keep the new policy close to the old policy by constraining the probability ratio. A simplified version of this objective (Achiam , 2018), which is the one implemented in the code, is:

$$L(s,a,\theta_k,\theta) = min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g(\varepsilon, A^{\pi_{\theta_k}}(s,a))\right)$$

Where $g(\varepsilon, A) = \begin{cases} (1+\varepsilon)\cdot A & , A \geq 0 \\ (1-\varepsilon)\cdot A & , A < 0 \end{cases}$

To understand this approach, it is useful to consider a single state-action pair (s,a) and examine different scenarios.

Advantage is positive: If the advantage is positive for a given state-action pair, the contribution to the objective simplifies to:

$$L(s,a,\theta_k,\theta) = min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right)A^{\pi_{\theta_k}}(s,a)$$

Since the advantage is positive, the objective increases if $\pi_\theta(a|s)$ increases. However, the minimum function limits this increase. When $\pi_\theta(a|s) > (1+\varepsilon)\pi_{\theta_k}(a|s)$, the minimum function caps the term at $(1+\epsilon)A^{\pi_{\theta_k}}(s,a)$. Therefore, the new policy does not gain by deviating significantly from the old policy.

Advantage is negative: If the advantage for a given state-action pair is negative, the contribution to the objective simplifies to:

$$L(s, a, \theta_k, \theta) = max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

Since the advantage is negative, the objective increases if $\pi_\theta(a|s)$ decreases. However, the maximum function limits this increase. When $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$, the maximum function caps the term at $(1-\epsilon)A^{\pi_{\theta_k}}(s, a)$. Therefore, the new policy does not benefit from deviating significantly from the old policy.

Clipping acts as a regularizer by eliminating incentives for dramatic policy changes, with the hyperparameter ϵ controlling the permissible deviation from the old policy while still benefiting the objective.

Although clipping helps ensure reasonable policy updates, a new policy might still deviate excessively. Different PPO implementations use various techniques to prevent this. In our implementation, a simple method called early stopping is employed. If the mean KL-divergence between the new and old policies exceeds a threshold, gradient steps are halted to maintain stability.

Exploration vs. Exploitation

The PPO algorithm trains a stochastic policy in an on-policy method, meaning it explores by sampling actions based on the current stochastic policy. The randomness in action selection depends on initial conditions and the training process. Over time, the policy becomes less random as the update rule encourages exploiting previously discovered rewards. This reduction in randomness can lead to the policy getting trapped in local minimum.

Pseudocode

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Figure 23 : Pseudocode of the PPO Algorithm Implementation by (Open AI, Proximal Policy Optimization, 2024)

Explanation of Pseudocode in the Displayed Steps

1: θ_0,φ_0 are initialized. These parameters will be updated in every loop to improve the police and value function

2: For each iteration k, the algorithm executes the following steps to refine the policy.

3: The current policy is used to interact with the environment, generating a set of trajectories.

4: Calculate the rewards-to-go for each time step in the trajectories. This involves summing the future rewards to estimate the total expected reward from each state-action pair.

5: Using the current value function, compute the advantage estimates. The advantage function indicates how much better an action is compared to the average action at a given state

6: The policy is updated by optimizing the PPO-Clip objective function. This involves maximizing the expected advantage while ensuring that the new policy does not deviate too much from the old policy. The objective is to find the parameter θ that balance exploration and exploitation, ensuring stable policy improvement.

7: The value function parameter, φ, is updated by minimizing the mean-squared error between the predicted values and the computed rewards-to-go. This regression step ensures that the value function accurately predicts the expected rewards, improving the quality of the advantage estimates used for policy updates.

8: The loop continues until a certain criterion is met, such as a set number of iterations or convergence of the policy and value function parameters. This iterative process ensures continuous improvement of the policy and value function, leading to better performance over time.

3.5.4 A2C | Advantage Actor Critic

Advantage Actor-Critic (A2C) (Mnih, et al., 2016) is a synchronous reinforcement learning algorithm that enhances the Asynchronous Advantage Actor-Critic (A3C) by synchronizing data collection across multiple parallel workers. This approach mitigates the high variance and noisy gradients seen in vanilla policy gradients by incorporating a baseline, typically the value function, which stabilizes learning.

In A2C, the Actor updates the policy distribution based on feedback from the Critic, who estimates the value function. The advantage function $A(s,a) = Q(s,a) - V(s)$ quantifies how much better taking action a in state s is compared to the average action. This advantage function helps calculate the policy gradient, ensuring policy parameters are updated to maximize expected returns. The policy gradient formula is given by:

Reducing Variance with a Baseline
One way to reduce variance is by subtracting a baseline from the cumulative reward, making the gradients smaller and more stable. The state-value function $V(s)$ is often used as the baseline, which helps stabilize the learning process.

$$\nabla_\theta J_\theta(\pi_\theta) = E_{\tau\sim\pi_\theta} \sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)A(s_t,a_t)$$

The synchronous nature of A2C ensures efficient and stable training, suitable for various complex environments. This method effectively balances exploring new actions with exploiting known rewards, leading to robust and efficient policy learning.

Pseudocode for A2C
This code is an unofficial implementation of A2C unlike all the pseudocodes provided for all other algorithms.

Algorithm: Advantage Actor - Critic (A2C)		
Steps		
1.	Initialize policy parameters θ and value function parameters φ	
2.	**for** each episode:	
	a.	Initialize state s from the environment
	b.	Initialize lists to store log probabilities, values, and rewards
	c.	**for** each step in the episode:

		i.	Compute value $V(s)$ and policy distribution $\pi(a\|s)$ using the Actor-Critic network
		ii.	Sample action a from policy distribution $\pi(a\|s)$
		iii.	Execute action a in the environment to get next state s', reward r, and termination flag
		iv.	Store log probability $log\pi(a\|s)$, value $V(s)$, and reward r
		v.	Update state s to s'
		vi.	**if** termination flag is true, break the loop
	d.	Compute rewards-to-go $\widehat{R_t}$ for each time step	
	e.	Compute advantage estimates $\hat{A}_t = \widehat{R_t} - V(s_t)$	
	f.	Update policy parameters θ:	
		i.	Compute policy gradient $\nabla_\theta = \frac{1}{D}\sum_t \nabla_\theta log\pi(\alpha_t\|s_t)\hat{A}_t$
		ii.	Apply gradient ascent: $\theta = \theta + \alpha_\theta \nabla_\theta$
	g	Update value function parameters φ:	
		i.	Compute value function loss $L_v = \frac{1}{D}\sum_t \left(V_{s_t} - \hat{R}_t\right)^2$
		ii.	Apply gradient descent: $\varphi = \varphi - \alpha_\varphi \nabla_\varphi L_v$
3.	**end for**		

Table 6 : A2C Pseudocode

Explanation of the Pseudocode

Initialization: The algorithm starts by initializing the policy parameters θ and the value function parameters φ. These parameters will be updated iteratively throughout the learning process.

Episode Loop: For each episode, the algorithm resets the environment to get the initial state s. It initializes lists to store log probabilities, values, and rewards collected during the episode.

Step Loop: Within each episode, the algorithm iterates through a series of steps:

1. **Compute Value and Policy**: The Actor-Critic network is used to compute the value of the current state $V(s)$ and the policy distribution $\pi(a|s)$.
2. **Sample Action**: An action α is sampled from the policy distribution.
3. **Execute Action**: The action is executed in the environment, resulting in a new state s', a reward r, and a termination flag indicating whether the episode has ended.
4. **Store Values**: The log probability of the action, the value of the current state, and the reward are stored.
5. **Update State**: The current state is updated to the new state (s').
6. **Check for Termination**: If the termination flag is true, the step loop is terminated (break).

Compute Rewards-to-Go: After the episode ends, the rewards-to-go $\widehat{R_t}$ are computed for each time step. This represents the cumulative future rewards from each state.

Compute Advantage Estimates: The advantage estimates $\widehat{A_t}$ are calculated by subtracting the state value from the rewards-to-go.

Update Policy Parameters: The policy gradient ∇_θ is computed using the advantage estimates. The policy parameters θ are then updated using gradient ascent.

Update Value Function Parameters: The value function loss L_v is computed as the mean squared error between the estimated values and the rewards-to-go. The value function parameters φ are updated using gradient descent.

The episode loop repeats until the desired number of episodes is completed, continuously refining the policy and value function to improve the agent's performance. This structured approach ensures robust and efficient learning in complex environments.

3.5.5 DDPG | Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) (Silver , et al., 2014) (Lillicrap, et al., 2016) is an algorithm that simultaneously learns a Q-function and a policy. Using off-policy data and the Bellman equation it is learning the Q-function, and this Q-function is then used to optimize the policy.

DDPG is closely related to Q-learning, sharing the motivation that if the optimal action-value function $Q^*(s,a)$ is known, the optimal action $a^*(s)$ in any given state can be found by solving:

$$a^*(s) = argmax_a Q^*(s,a)$$

DDPG interleaves learning an approximator for $Q^*(s,a)$ with learning an approximator for $a^*(s)$, specifically designed for environments with continuous action spaces. In such environments, computing the maximum over actions in $max_a Q^*(s,a)$ becomes complex due to the continuous nature of the action space. DDPG addresses this by directly optimizing the policy using the gradients of the Q-function, enabling efficient learning and decision-making in continuous action spaces, such as those in predictive maintenance involving sensors in a machine.

Because the action space is continuous, the function $Q^*(s,a)$ is presumed to be differentiable with respect to the action argument. This allows us to set up an efficient, gradient-based learning rule for a policy $\mu(s)$ which exploits that fact. Then, instead of running an expensive optimization subroutine each time we wish to compute $max_a Q(s,a)$, we can approximate it with $max_a Q(s,a) \approx Q(s,\mu(s))$

Given that the action space is continuous, we assume that the function $Q^*(s,a)$ is differentiable with respect to the action variable. This assumption enables us to establish an efficient, gradient-based learning rule for a policy $\mu(s)$ that leverages this property. Consequently, instead of executing a costly optimization subroutine each time we want to compute $max_a Q(s,a)$, we can approximate it using $max_a Q(s,a) \approx Q(s,\mu(s))$.

DDPG is an off-policy algorithm specifically designed for environments with continuous action spaces. It can be considered as deep Q-learning adapted for continuous action spaces, making it highly effective in such contexts.

Mathematical Formulation
The two components of DDPG, we'll delve into the mathematics behind this section: is learning a Q function and learning a policy.

The Q-Learning Side of DDPG
First, a small revision on the Bellman equation describing the optimal action-value function, $Q^*(s,a)$. It's given by:

$$Q^*(s,a) = E_{s\prime\sim P}\left[r(s,a) + \gamma max_{a\prime} Q^*(s',a')\right]$$

Here $s' \sim P$ indicates that the next state, s', is sampled by the environment form a distribution $P(\cdot\,|s,a)$

This Bellman equation is the starting point for learning an approximator to $Q^*(s,a)$. Suppose the approximator is a neural network $Q_\varphi(s,a)$, with parameters φ, and that we have collected a set D of transitions (s,a,r,s',d) (where d indicates whether state s' is terminal). We can set up a mean-squared Bellman error (MSBE) function, which tells us roughly how closely Q_φ comes to satisfying the Bellman equation:

$$L(\varphi,D) = E_{(s,a,r,s\prime,d)\sim D}\left[\left(Q_\varphi(s,a) - \left(r + \gamma(1-d)\cdot max_{a\prime} Q_\varphi(s',a')\right)\right)^2\right]$$

The evaluation $(1-d)$, follows the Python convention, where `True` evaluates to 1 and `False` evaluates to 0. When d is True it indicates that s' is a terminal state—the Q-function should reflect that the agent receives no additional rewards after the current state.

Function approximator-based Q-learning algorithms, such as DQN (and its variants) and DDPG, focus heavily on minimizing the MSBE loss function. These algorithms utilize two key techniques, which are essential to understand. Additionally, there is a specific detail unique to DDPG that is noteworthy.

Replay Buffers.
Training a deep neural network to approximate $Q^*(s,a)$ typically involves using an experience replay buffer, denoted as D. This buffer stores a set of previous experiences and should be sufficiently large to encompass a diverse range of experiences. However, it is important to balance the size of the buffer. Relying solely on the most recent data can lead to overfitting and instability, while using too much past experience can slow down learning. Achieving the right balance may require some tuning.

DDPG, being an off-policy algorithm, leverages this replay buffer to include older experiences, even those collected with outdated policies. This is possible because the Bellman equation applies universally to all transition tuples, regardless of how the actions were chosen or what occurred after the transitions. The optimal Q-function

Target Networks
Q-learning algorithms utilize target networks to **stabilize training**. The term:

$$\left(r + \gamma(1-d) \cdot max_{a'} Q_{\varphi}(s', a')\right)$$

is known as the **target**. Minimizing the MSBE loss involves making the Q-function approximate this target. However, since the target depends on the same parameters φ that are being trained, this can lead to instability. To address this, a second network, called the target network, is employed. This network, with parametersφ_{targ}, lags behind the main network, thereby providing a more stable target.

In DQN-based algorithms, the target network's parameters are copied from the main network at fixed intervals. In DDPG-style algorithms, the target network is updated continuously using polyak averaging:

$$\varphi_{targ} \leftarrow \rho\varphi_{targ} + (1-\rho)\varphi$$

where ρ is a hyperparameter between 0 and 1, typically close to 1.

DDPG Detail: Calculating the Max Over Actions in the Target.
In continuous action spaces, computing the maximum over actions in the target is challenging. DDPG addresses this by using a target policy network to compute an action that approximately maximizes $Q_{\varphi_{targ}}$. The target policy network is maintained similarly to the target Q-function, through polyak averaging of the policy parameters during training.

As mentioned earlier: computing the maximum over actions in the target is a challenge in continuous action spaces. DDPG deals with this by using a target policy network to compute an action which approximately maximizes $Q_{\varphi_{targ}}$. The target policy network is found the same way as the target Q-function: by polyak averaging the policy parameters over the course of training.

Summing it up, Q-learning in DDPG is achieved by minimizing the following MSBE loss with stochastic gradient descent:

$$L(\varphi, D) = E_{(s,a,r,s\prime,d)\sim D}\left[\left(Q_\varphi(s,a) - \left(r + \gamma(1-d)Q_{\varphi_{targ}}(s', \mu_{\theta_{targ}}(s'))\right)\right)^2\right]$$

Where $\mu_{\theta_{targ}}$ is the target policy.

The policy Learning Side of DDPG

Policy learning in DDPG is fairly simple. We want to learn a deterministic policy $\mu_\theta(s)$ which gives the action that maximizes $Q_\varphi(s,a)$.Because the action space is continuous, and we assume the Q-function is differentiable with respect to action, we can just perform gradient ascent (with respect to policy parameters only) to solve

Policy learning in DDPG aims to develop a deterministic policy $\mu_\theta(s)$ that outputs the action maximizing $Q_\varphi(s,a)$. Given the continuous action space and the assumption that the Q-function is differentiable with respect to the action, gradient ascent can be applied to optimize the policy parameters. The objective is to solve:

$$max_\theta\, E_{s\sim D}\left[Q_\varphi(s, \mu_\theta(s))\right]$$

In this context, the Q-function parameters φ are treated as constants.

Exploration vs Exploitation

In DDPG, a deterministic policy is trained in an off-policy manner. Due to its deterministic nature, the policy might not initially explore a sufficient range of actions to gather valuable learning signals if explored on-policy. To enhance exploration, noise is added to the actions during training. The authors of the DDPG paper (Lillicrap, et al., 2016) recommended using time-correlated OU noise (Ornstein-Uhlenbeck process) (Uhlenbeck, George, Ornstein, & Leonard, 1930). However, more recent findings indicate that uncorrelated, mean-zero Gaussian noise is equally effective and simpler to implement, making it the preferred choice. To improve the quality of training data, the noise scale can be gradually reduced as training progresses.
During testing, no noise is added to the actions, allowing the policy to fully exploit what it has learned.

Pseudocode

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Figure 24 : Pseudocode of the DDPG Algorithm Implementation by (OpenAI, Deep Deterministic Policy Gradient, 2024)

Explanation of Pseudocode in the Displayed Steps

1: Initialization of parameters

2: The target parameters for the policy $\theta_{targ}$and Q-function φ_{targ} are set equal to the main parameters.

3: Main loop

4: For the current state, the action a is selected by adding noise ϵ to the deterministic policy $\mu_\theta(s)$, ensuring exploration

5: The selected action aaa is executed in the environment.

6: The next state s', reward, and done signal are observed.

7: The transition is stored.
8: If the new state is a terminal state, the environment is reset to start a new episode.
9: If it's time to update, proceed to the next steps
-
11: A batch of transitions is randomly sampled from the replay buffer.
12: For each transition in the batch the target y
13: The Q-function parameter φ is updated by minimizing the loss between predicted Q-values and computed targets using gradient descent
14: The policy parameter θ is updated by maximizing the expected Q-value using gradient ascent
15: The target networks are updated using a soft update mechanism
16: The update steps are repeated for the specified number of updates
-
18: Continue the main loop until the convergence criteria is met.

3.5.6 TD3 | Twin Delayed DDPG

TD3 (Fujimoto, Hoof, & Meger, 2018) is an off-policy algorithm designed for environments with continuous action spaces and it is a variant of DDPG specifically designed to address some of the limitations and instability issues. Although DDPG can achieve impressive results, it is often sensitive concerning hyperparameters and other tuning factors. A typical failure mode involves the Q-function significantly overestimating Q-values, resulting in a flawed policy that exploits these inaccuracies. Twin Delayed DDPG (TD3) avoids this issue by incorporating three essential techniques:

Clipped Double-Q Learning. By maintaining two Q-functions and using the smaller value to form the targets in the Bellman error loss functions, TD3 reduces overestimation bias.

"Delayed" Policy Updates. TD3 stabilizes training by updating the policy and target networks less frequently than the Q-function, typically performing one policy update for every two Q-function updates.

Target Policy Smoothing. To avoid the policy's tendency to exploit Q-function errors, TD3 introduces noise to the target action, smoothing out the Q-values with respect to action changes.

These techniques collectively enhance TD3's performance significantly compared to the baseline DDPG.

Mathematical Formulation
TD3 improves upon DDPG by concurrently learning two Q-functions, Q_{φ_1} and Q_{φ_2}, through the minimization of mean square Bellman error. This process is almost identical to how DDPG learns a single Q-function. To explain TD3's approach and highlight its differences from DDPG, we will analyze the loss function from the innermost part outward.

One key aspect is target policy smoothing. The actions used to form the Q-learning target are derived from the target policy $\mu_{\theta_{targ}}$ with added clipped noise in each action dimension. After incorporating the noise,

the target action is clipped to remain within the valid action range $[a_{Low}, a_{High}]$. The target actions are thus given by:

$$a'(s') = clip(\mu_{\theta_{targ}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High}), \epsilon \sim N(0, \sigma)$$

To prevent bad behavior in DDPG, target policy smoothing serves as a regularizer for the algorithm. In DDPG, if the Q-function approximator develops a false sharp peak for certain actions, the policy may rapidly exploit this peak, leading to unstable or incorrect behavior. Target policy smoothing avoids this issue by smoothing out the Q-function over similar actions, preventing the policy from exploiting these incorrect peaks.

Clipped double-Q learning is another important technique. In this method, both Q-functions use a single target, calculated using the smaller value from the two Q-functions:

$$y(r, s', d) = r + \gamma(1 - d) min_{i=1,2} Q_{\varphi_{i,targ}}(s', a'(s'))$$

Each Q-function then regresses towards this target:

$$L(\varphi_1, D) = E_{(s,a,r,s',d)\sim D}\left[\left(Q_{\varphi_1}(s, a) - y(r, s', d)\right)^2\right]$$
$$L(\varphi_2, D) = E_{(s,a,r,s',d)\sim D}\left[\left(Q_{\varphi_2}(s, a) - y(r, s', d)\right)^2\right]$$

By using the smaller Q-value for the target and regressing towards it, this approach helps avoid overestimation in the Q-function.

The policy in TD3 is learned by maximizing Q_{φ_1}:

$$max_\theta \, E_{s\sim D}\left[Q_{\varphi_1}(s, \mu_\theta(s))\right]$$

This process remains largely unchanged from DDPG. However, TD3 updates the policy less frequently than the Q-functions, which helps reduce the volatility typically seen in DDPG due to policy updates affecting the target.

Exploration vs. Exploitation

Training a deterministic policy in TD3 is done in an off-policy way. Due to the deterministic nature of the policy, if the agent were to explore on-policy initially, it would likely not attempt a sufficient variety of actions to obtain valuable learning signals. To improve exploration, noise is added to the actions during training, typically in the form of uncorrelated mean-zero Gaussian noise. Adjusting the noise scale throughout training can help in acquiring higher-quality training data, although some implementations, like ours, keep the noise scale constant.

During testing, noise is not added to the actions, allowing the policy to fully exploit the learned behavior.

Pseudocode

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

      end if
    end for
  end if
until convergence

Figure 25 : Pseudocode of the TD3 Algorithm Implementation by (OpenAI, Twin Delayed DDPG, 2024)

Explanation of Pseudocode in the Displayed Steps:

1: Parameter initialization.

2: Target values initialization.

3: Main loop.
4: For the current state, select an action and use the policy with added noise for exploration.
5: Execute the selected action in the environment.
-
9: If it's time to update, proceed with the following steps.
-
11: Randomly sample a batch of transitions from the replay buffer.
12: Compute the target action using the target policy and clipped noise for stability.
13: Compute the target value y using the minimum of the two target Q-functions.
14: Update both Q-function parameters φ_1 and φ_2 by minimizing the loss between the predicted Q-values and the computed targets using gradient descent.
15: If the update step is at the specified delay interval, proceed to update the policy.
16: Update the policy parameters θ by maximizing the expected Q-value from Q_{φ_1}
17: Update the target networks using a soft update mechanism.
-
21: Continue the main loop until convergence criteria are satisfied.

3.5.7 SAC | Soft Actor Critic

Soft Actor Critic (SAC) (Haarnoja, Zhou, Abbeel, & Levine, 2018), (Haarnoja, et al., 2019) optimizes a stochastic policy using an off-policy approach, creating a bridge between stochastic policy optimization and DDPG-style methods. Although SAC and TD3 were published around the same time and are not direct successors of one another, SAC incorporates the clipped double-Q trick. Additionally, the stochastic nature of SAC's policy benefits from an effect similar to target policy smoothing.

Entropy regularization is a key feature of SAC. The policy is trained to balance expected return with entropy, which measures the randomness in the policy. This balance is closely related to the exploration-exploitation trade-off: higher entropy promotes more exploration, potentially accelerating learning in later stages. Additionally, it helps prevent the policy from prematurely converging to suboptimal solutions.

SAC, the successor to Soft Q-Learning (SQL), incorporates the double Q-learning technique from TD3. A distinguishing feature of SAC is its training objective, which maximizes a balance between expected return and entropy, a measure of randomness in the policy. SAC is an off-policy algorithm. The version of SAC implemented here is designed for environments with continuous action spaces. An alternative version of SAC, with a slightly modified policy update rule, can handle discrete action spaces. An alternate version of SAC, which slightly changes the policy update rule, can be implemented to handle discrete action spaces.

Mathematical Formulation
To understand Soft Actor Critic (SAC), the concept of entropy-regularized reinforcement learning must first be introduced. In this context, the equations for value functions are slightly modified.
.
Entropy-Regularized Reinforcement Learning
Entropy quantifies the randomness of a random variable. In entropy-regularized RL, this measure is incorporated into the learning process to balance exploration and exploitation more effectively.

Given a random variable x with a probability mass or density function P. The entropy H of x is computed from its distribution P according to:

$$H(P) = E_{x\sim P}[-logP(x)]$$

In entropy-regularized reinforcement learning, the agent receives a bonus reward at each time step proportional to the entropy of the policy at that timestep. This modifies the RL problem to:

$$\pi^* = argmax_\pi E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t(R(s_t,a_t,s_{t+1}+aH(\pi(\cdot\,|s_t)))\right]$$

Here $a > 0$ serves as the trade-off coefficient. It is important to note that an infinite-horizon discounted setting is assumed for this formulation, and this assumption will be maintained throughout the explanation. Consequently, the value functions in this setting are slightly modified. The value function $V^\pi(s)$ is redefined to include the entropy bonuses from each timestep.

$$V^\pi(s) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t(R(s_t,a_t,s_{t+1})+aH(\pi(\cdot\,|s_t)))\mid s_0 = s\right]$$

Q^π is changed to include the entropy bonuses from every timestep except the first:

$$Q^\pi(s,a) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + a\sum_{t=1}^{\infty}\gamma^t H(\pi(\cdot\,|s_t))\mid s_0 = s, a_0 = a\right]$$

With these definitions V^π and Q^π are connected by:

$$V^\pi(s) = E_{a\sim\pi}[Q^\pi(s,a)] + aH(\pi(\cdot\,|s))$$

And the Bellman equation for Q^π is:

$$Q^\pi(s,a) = E_{s\prime\sim P,a\prime\sim\pi}[R(s,a,s') + \gamma(Q^\pi(s',a') + aH(\pi(\cdot\,|s')))]$$

$$Q^\pi(s,a) = E_{s\prime\sim P}[R(s,a,s') + \gamma V^\pi(s')]$$

SAC simultaneously learns a policyπ_θ and two Q-functions $Q_{\varphi1}$, $Q_{\varphi2}$. There are two standard variants of SAC: one with a fixed entropy regularization coefficient α and another that adjusts α throughout training to enforce an entropy constraint. For simplicity, the fixed entropy regularization coefficient version is used, though the entropy-constrained variant is generally preferred by practitioners.

The Q-functions in SAC are learned similarly to those in TD3, but with some notable differences. Like TD3, both Q-functions are trained using MSBE minimization by regressing to a single shared target. This

shared target is computed using target Q-networks, which are obtained by polyak averaging the Q-network parameters over the course of training. Additionally, the shared target employs the clipped double-Q trick, similar to TD3.

However, there are several differences. Unlike TD3, SAC's target includes a term from entropy regularization. Furthermore, the next-state actions used in the target come from the current policy rather than a target policy, as is done in TD3. Finally, SAC does not use explicit target policy smoothing. While TD3 trains a deterministic policy and adds random noise to next-state actions to achieve smoothing, SAC trains a stochastic policy, where the inherent noise from stochasticity provides a similar effect.

Understanding how entropy regularization contributes is crucial before presenting the final form of the Q-loss. We begin by revisiting the recursive Bellman equation for the entropy-regularized Q^{π} and slightly modifying it using the definition of entropy

$$Q^{\pi}(s,a) = E_{s\prime\sim P, a\prime\sim\pi}\left[R(s,a,s') \; + \; \gamma(Q^{\pi}(s',a') + aH(\pi(\cdot\,|s')))\right]$$

This can be rewritten with the definition of entropy as:

$$Q^{\pi}(s,a) = E_{s\prime\sim P, a\prime\sim\pi}\left[R(s,a,s') \; + \; \gamma(Q^{\pi}(s',a') - a\cdot log\pi(a'|s'))\right]$$

This is an expectation over next states (sampled from the replay buffer) and next actions (sampled from the current policy, rather than the replay buffer). Since this is an expectation, it can be approximated with samples.

$$Q^{\pi}(s,a) \approx \; r \; + \gamma(Q^{\pi}(s',\bar{\alpha}') - a\cdot log\pi(\bar{\alpha}'|s'))\,, \qquad \bar{\alpha}' \sim \pi(\cdot\,|s')$$

The notation for the next action is switched to $\bar{\alpha}'$ to emphasize that the next actions must be sampled directly from the policy, while r and s' are obtained from the replay buffer.

SAC establishes the MSBE loss for each Q-function using this sample approximation for the target. The only remaining question is which Q-function is used to compute the sample backup. Like TD3, SAC employs the clipped double-Q trick and selects the minimum Q-value between the two Q approximators.

Combining all of the above, the loss functions for the Q-networks in SAC are:

$$L(\varphi_i, D) = E_{(s,a,r,s\prime,d)\sim D}\left[\left(Q_{\varphi_i}(s,a) - y(r,s',d)\right)^2\right]$$

where the target is given by:

$$y(r,s',d) = \; r + \gamma(1-d)\left(min_{j=1,2,\ldots}Q_{\varphi_{targ,j}}(s',\bar{\alpha}') - alog\pi_{\theta}(\bar{\alpha}'|s')\right), \qquad \bar{\alpha}' \sim \pi_{\theta}(\cdot\,|s')$$

Leaning the policy

In each state, the policy should maximize the expected future return plus the expected future entropy, effectively maximizing $V^{\pi}(s)$. This can be expanded as follows:

$$V^{\pi}(s) = E_{\alpha\sim\pi}[Q^{\pi}(s,a)] + aH(\pi(\cdot\,|s))$$

$$V^{\pi}(s) = E_{\alpha\sim\pi}[Q^{\pi}(s,a) - alog\,\pi(\alpha|s)]$$

The way we optimize the policy makes use of the reparameterization trick, in which a sample from $\pi_{\theta}(\cdot\,|s)$ is drawn by computing a deterministic function of state, policy parameters, and independent noise. To illustrate: following the authors of the SAC paper, we use a squashed Gaussian policy, which means that samples are obtained according to

To optimize the policy, the reparameterization trick is used. This involves drawing a sample from $\pi_{\theta}(\cdot\,|s)$ by computing a deterministic function of the state, policy parameters, and independent noise. Following the authors of the SAC paper (Haarnoja, Zhou, Abbeel, & Levine, 2018), a squashed Gaussian policy is used, where samples are obtained according to:

$$\bar{\alpha}_{\theta}(s,\xi) \;=\; tanh(\mu_{\theta}(s) + \sigma_{\theta}(s) \odot \xi)\,, \xi \sim N(0,I)$$

The equation describes how to sample actions from the policy. Here, $\mu_{\theta}(s)$ is the mean of the Gaussian distribution, which is a function of the state and policy parameters, while $\sigma_{\theta}(s)$ represents the standard deviation, also dependent on the state and policy parameters. The variable ξ is noise sampled from a standard normal distribution $N(0,I)$. The $tanh$ function is applied to squash the output, ensuring that the actions remain within a bounded range, typically $[-1,1]$.

By doing the reparameterization trick, the expectation over actions, which is challenging due to the dependence on policy parameters, is converted into an expectation over noise, thus removing this dependency:

$$E_{a\sim\pi_{\theta}}[Q^{\pi_{\theta}}(s,a) - alog\pi_{\theta}(a|s)] = E_{\xi\sim N}[Q^{\pi_{\theta}}(s,\bar{\alpha}_{\theta}(s,\xi)) - \alpha log\pi_{\theta}(\bar{\alpha}_{\theta}(s,\xi)|s)]$$

This technique closely aligns with DDPG and TD3 policy optimization, but it incorporates the min-double-Q trick, accounts for stochasticity, and includes an entropy term.

Exploration vs Exploitation

A stochastic policy with entropy regularization is used in SAC, enabling on-policy exploration. The entropy regularization coefficient a plays an important role in managing the exploration-exploitation tradeoff: higher values of a lead to increased exploration, while lower values encourage more exploitation. Identifying the appropriate coefficient, which ensures the most stable and highest-reward learning, can change across environments and usually requires meticulous tuning.

Pseudocode

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence

Figure 26 : Pseudocode of the TD3 Algorithm Implementation by (OpenAI, Soft Actor Critic, 2024)

Explanation of Pseudocode in the Displayed Steps:

1: Initialization of parameters.
2: Initialization of target parameters
3: Main loop
-
9: If it's time to update the networks, proceed with the following steps.
-
11: Randomly sample a batch of transitions from the replay buffer.

12: Compute the target actions $a'(s)$ using the target policy with added noise. Compute the target value $y(r, s', d)$ using the minimum of the two target Q-functions and the entropy term
13: Update both Q-function parameters φ_1 and φ_2 by minimizing the loss between the predicted Q-values and the computed targets using gradient descent
14: Update the policy parameters θ by maximizing the expected Q-value with the entropy term for exploration. The term, $\bar{a}(s)$, is a sample from $\pi_\theta(a|s)$, differentiable with respect to θ.
15: Update the target networks using a soft update mechanism
16: End of Policy Update Check
17: End of Update Check
18: Continue the main loop until the convergence criteria are satisfied

3.5.8 Algorithms and the PdM task

In the context of Predictive Maintenance (PdM), advanced reinforcement learning algorithms such PPO, A2C, DDPG, TD3, and SAC are suited for optimizing maintenance strategies and decision-making processes. These algorithms enable the creation of intelligent systems that can learn from historical and real-time data to predict equipment failures and recommend optimal maintenance actions.

Algorithms like DDPG and TD3 are particularly well-suited for environments with continuous action spaces, such as the fine-tuning of maintenance schedules and resource allocation in complex industrial systems. Their ability to handle continuous actions allows for precise adjustments and efficient management of maintenance tasks.

SAC's incorporation of entropy regularization makes sure that a balanced exploration-exploitation trade-off, which is essential for adapting maintenance policies to varying operational conditions. This balance helps in discovering new maintenance strategies while still exploiting known effective actions, leading to improved overall system performance.

The stochastic nature of algorithms like PPO and A2C allows for robust learning in dynamic environments, where the conditions and data patterns continually evolve. This adaptability is important in PdM tasks where equipment behavior and environmental factors can change over time, requiring flexible and resilient maintenance policies.

By leveraging these RL techniques, PdM systems can significantly enhance predictive accuracy, optimize maintenance interventions, reduce downtime, and ultimately extend the lifespan of critical machinery. These improvements lead to cost savings, increased operational efficiency, and more reliable production processes.

3.6 Evaluation Metrics

Evaluating and comparing the performance of different RL algorithms is the key to understanding the effectiveness of the PdM approach. Several evaluation metrics are commonly used to assess and compare

the quality of the learned policies. The two universal metrics considered are mean episode length and mean episode reward.

Mean Episode Length

Mean episode length is a metric that indicates the average duration of an episode before a terminal state is reached. In the context of PdM, an episode could represent the operational period of a machine before a maintenance intervention is required. A longer mean episode length could suggest that the learned policy is effective at preventing failures and prolonging the operational time of the equipment.

For example, in PdM tasks, an RL algorithm that learns a policy resulting in longer mean episode lengths would usually imply that the system is more reliable (this depends on the environment formulation). Evaluating the mean episode length involves running multiple episodes with the trained policy and calculating the average duration across these episodes.

Mean Episode Reward

Mean episode reward is another critical metric used to evaluate the performance of reinforcement learning algorithms. It represents the average cumulative reward obtained per episode. In the context of PdM, the reward function can be designed to encapsulate various factors such as operational efficiency, maintenance costs, downtime, and the occurrence of failures.

A higher mean episode reward indicates that the policy is effective in balancing the trade-offs between these factors, leading to more optimal maintenance strategies. For example, in a PdM setting, the reward could be higher for policies that achieve minimal maintenance costs while ensuring high operational efficiency and minimal downtime. The mean episode reward is calculated by averaging the total rewards obtained over multiple episodes using the trained policy.

Evaluating Policies

Evaluating the policies learned by different algorithms involves comparing these metrics across various runs and environments. The evaluation process workflow is:

1. **Define the Evaluation Environment**: Set up an MDP environment that reflects the real-world operational conditions of the machinery. This includes defining the state space, action space, and reward function relevant to the PdM task.
2. **Train Policies**: Use different RL algorithms (like PPO, A2C, DDPG, TD3, SAC) to train policies within the defined environment. It is good to make sure that the training process is consistent across algorithms in order to make fair comparisons. This can be done by using time limits or timestep limits.
3. **Run Multiple Episodes**: For each trained policy, run a significant number of episodes in the evaluation environment. This helps in capturing the performance over various scenarios and conditions.
4. **Compute the Metrics**: Calculate the mean episode length and mean episode reward for each policy by averaging the results across all episodes. Also compute other metrics from a unique score

function to the algorithm’s mathematical running core (like KL Divergence, Actor Loss, Critic Loss, etc.). These metrics provide insights into the effectiveness and efficiency of the policies.

5. **Compare Metrics**: Compare the mean episode length and mean episode reward across different algorithms. Policies with longer mean episode lengths and higher mean episode rewards are generally considered more effective. In algorithms that share similar features like on-policy and off-policy algorithms it is possible to compare additional computed metrics.
6. **Analyze Variability and Robustness**: In addition to mean values, it’s important to analyze the variability and robustness of the policies. This involves examining the standard deviation and distribution of the episode lengths and rewards to understand the consistency and reliability of the policies. This is usually done by observing the graphs that were generated and looking for spikes.

Practical Considerations

While mean episode length and mean episode reward are essential metrics, other factors may also be considered in an overall evaluation, such as:

- **Computational Efficiency**: The time and resources required to train the policy.
- **Scalability**: The algorithm's ability to handle larger and more complex environments.
- **Adaptability**: How well the policy adapts to changes in the environment or operational conditions.

By thoroughly evaluating these metrics, engineers can make informed decisions about which RL algorithms are best suited for their PdM tasks, leading to improved maintenance strategies and enhanced operational performance.

4. Implementation, Evaluation & Results

4.1 Tech Stack

The implementation of the CNC Machine Wear Problem utilizes a tech stack centered around Python, using various libraries and frameworks to perform data manipulation and reinforcement learning.

At its core, the project is implemented in Python, which is a multipurpose scripting language and is known for its extensive support in scientific computing and machine learning. The primary helper libraries used for general tasks include NumPy for numerical computations and array handling, Pandas for efficient data manipulation and analysis of table data, and Matplotlib for data visualization.

Figure 27 : Stable Baselines Logo

Reinforcement learning is a key component of the project, for which Stable Baselines 3 is employed. Stable Baselines 3, a library based on OpenAI Baselines, is a robust library that provides reliable (stable) implementations of RL algorithms in Python. It is built on top of the PyTorch framework and offers a big arsenal of tools designed to help with the development and deployment of RL models. One of the primary reasons for using Stable Baselines 3 is the standardized implementation of up-to-date RL algorithms, such as Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC). These implementations have been fully tested and are known for their performance and stability.

The library is particularly user-friendly, featuring an API that simplifies the setup, training, and evaluation of RL models. The ease of use makes it accessible while at the same time being reliable. Moreover, Stable Baselines 3 is designed with extensibility in mind, allowing users to modify existing algorithms or

implement new ones with minimal effort. This flexibility is crucial for custom projects, like this one, that often need to adapt algorithms to specific use cases or experiment with new approaches.

In addition to its algorithm implementations, Stable Baselines 3 offers a range of utilities that streamline the RL development process. These include tools for environment checking, vectorized environment handling, policy evaluation, and results plotting. Such utilities help ensure that the models are correctly implemented, efficiently trained, and effectively evaluated, providing comprehensive support throughout the RL model lifecycle. Overall, Stable Baselines 3 stands out as a valuable resource for anyone working in reinforcement learning, combining reliability, usability, and flexibility in a single, cohesive package.

Additionally, the project makes use of several utility functions from Stable Baselines 3. The check_env utility ensures that the custom environments adhere to the Gym API, while VecFrameStack is used for stacking frames in environments with image observations. The evaluate_policy function is responsible for the evaluation of the trained models, and DummyVecEnv allows for the parallel execution of multiple environment instances. Last but not least, the plot_results and X_TIMESTEPS functions are utilized for visualizing training progress, and the Monitor class is used to log information about the environment during training.

To support file system operations, the project uses the os library for interacting with the operating system and Pathlib for handling file paths for object-oriented purposes. The time library is also included to manage time-related functions and measure execution time.

List of Libraries Imports	
Helper Libraries	matplotlib.pyplot
	numpy
	pandas
	os
	time
Environment Building	gymnasium Env gymnasium.spaces Box

RL	stable_baselines3 PPO A2C DDPG SAC stable_baselines3.common.env_checker check_env stable_baselines3.common.vec_env VecFrameStack stable_baselines3.common.evaluation evaluate_policy stable_baselines3.common.vec_env DummyVecEnv stable_baselines3.common.results_plotter plot_results X_TIMESTEPS stable_baselines3.common.monitor Monitor
	torch

TensorBoard was used for the real-time monitoring and visualization of model training metrics. Tensorboard is a visualization toolkit included with TensorFlow, it is created by Google and is designed to help users understand and debug machine learning models. It allows the visualization of various training metrics and provides insights into model performance helping in the optimization process.

TensorBoard operates through a logging process during model training. The training data are logged using TensorFlow's “Summary Writer”. This data is stored in log files which are then visualized through TensorBoard’s web interface, enabling users to monitor the training progress of their models in real-time.

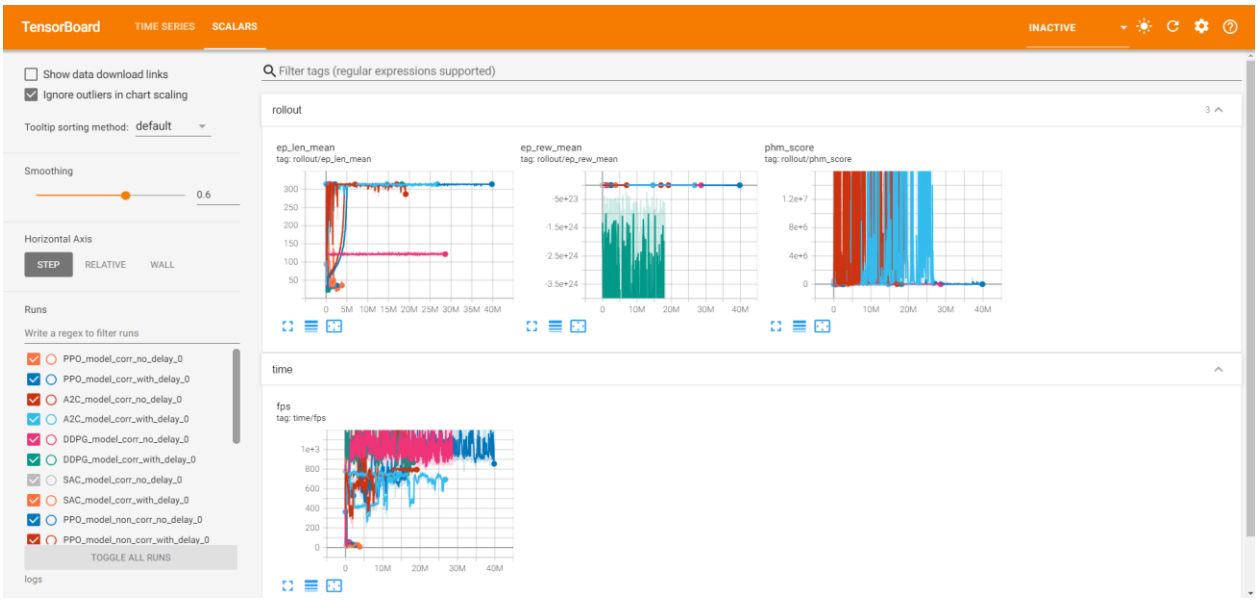


Figure 28 : Scalars Tab Tensorboard UI

The TensorBoard user interface (UI) comprises several dashboards and tools for the detailed monitoring and analysis of model training. The Scalars Dashboard displays plots of metrics such as average reward and average episode length over time, which is useful for tracking the model’s improvement with respect to the number of training iterations.

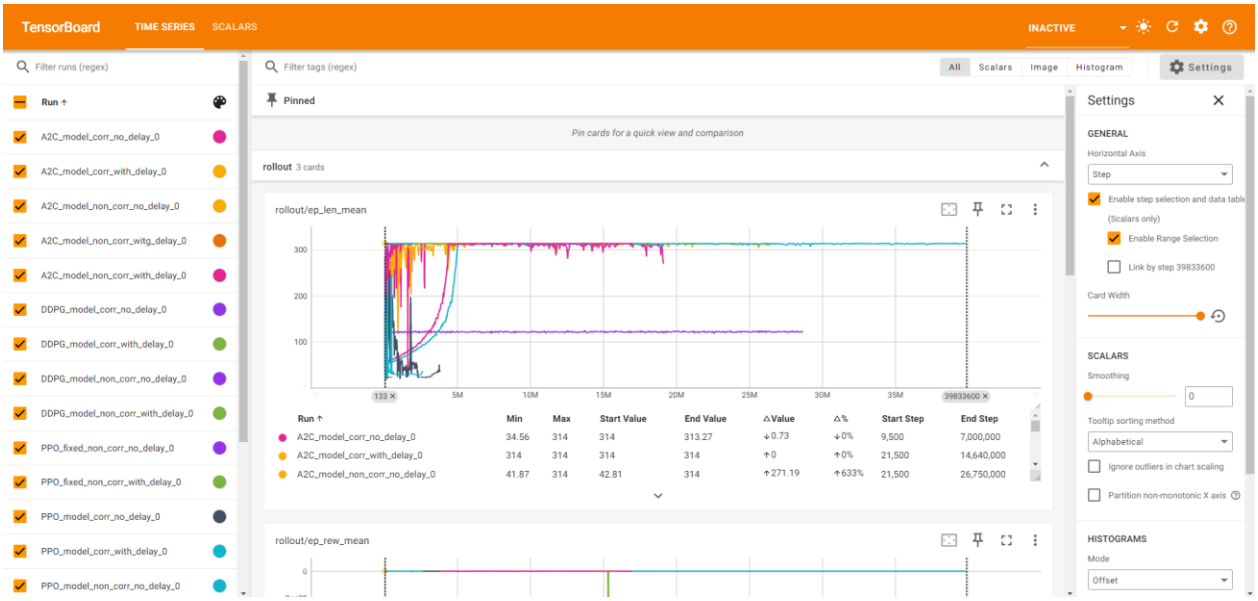


Figure 29 : Time Series Tab Tensorboard UI

Additionally, the Time Series tab in TensorBoard allows for a more detailed examination of metrics over time, providing thorough insights into how specific values change throughout the training process. This can be particularly useful for identifying trends, anomalies, and areas where the model may require further tuning or adjustment.

4.2 Data structure and input parameters

The dataset used to train the models is part of the 2010 PHM Society Conference Data Challenge dataset.

The PHM Data Challenge was a competition open to all conference attendees of the PHM society. In 2010 the challenge was focused on RUL estimation for high-speed CNC milling machine cutters using dynamometer, accelerometer, and acoustic emission data. Both Student and Professional teams could enter the competition.

Participants were scored based on their ability to estimate the remaining useful life of a 6mm ball nose tungsten carbide cutter.

The dataset provided by the PHM Society is six individual cutter records, c1 to c6. Records c1, c4 and c6 are training data, and records c2, c3, and c5 are test data:

Each training record contains one “wear” file that lists wear after each cut in 10^-3 mm, and a folder with 315 individual data acquisition files (one for each cut). The data acquisition files are in .csv format, with seven columns, corresponding to:

Column 1: Force (N) in X dimension
Column 2: Force (N) in Y dimension
Column 3: Force (N) in Z dimension
—
Column 4: Vibration (g) in X dimension
Column 5: Vibration (g) in Y dimension
Column 6: Vibration (g) in Z dimension
—-
Column 7: AE-RMS (V)

The spindle speed of the cutter was 10400 RPM; feed rate was 1555 mm/min; Y depth of cut (radial) was 0.125 mm; Z depth of cut (axial) was 0.2 mm. Data was acquired at 50 KHz/channel. Background on the apparatus and experimental setup can be found on the paper (Li, et al., 2009)

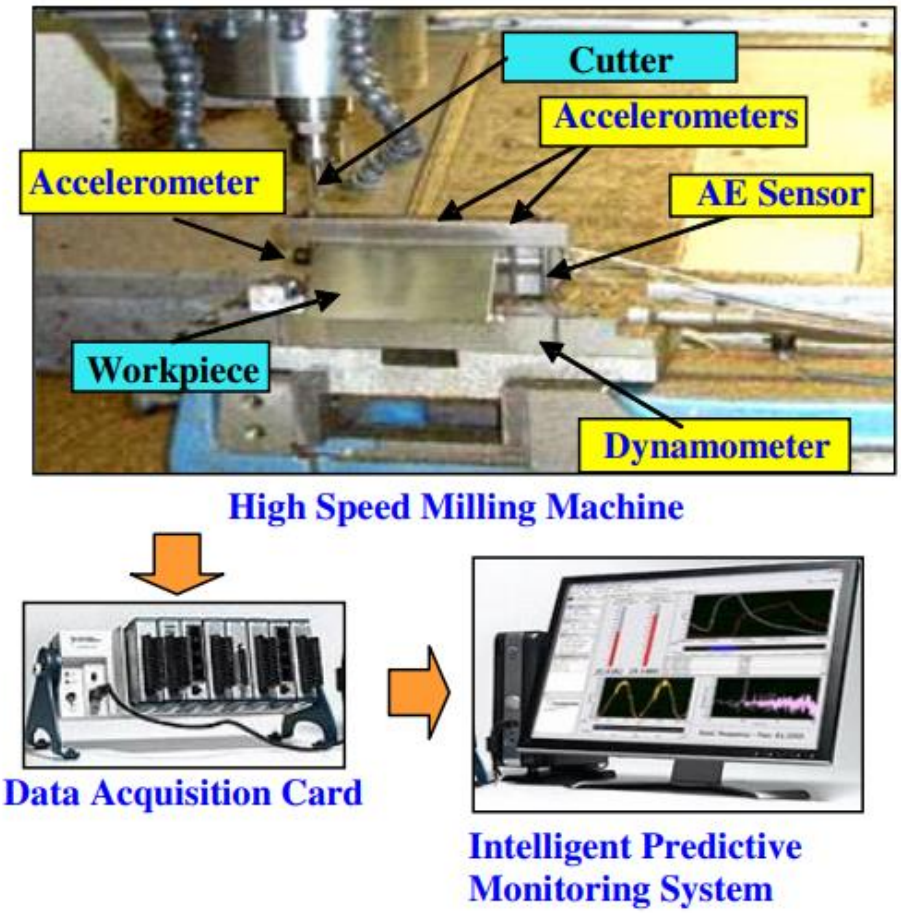


Figure 30 : Tool Conditions Monitoring in high-speed Milling Process (Li, et al., 2009)

For the training of the models, the file c1 was used corresponding to cutter 1, which is part of the training set of the competition. Since we did not have access to the test set's 'wear' data files we had to use cutter 4 and 6 as the test set

The input parameters of our dataset were the same as the ones provided from the competition's dataset. We preprocessed the accelerometer, dynamometer and Acoustic Emission data and used the dataset we created.

4.3 Preprocessing & Feature Extraction

To effectively use the extensive dataset provided, we need to transform the high-dimensional time-series data into a manageable, efficient, and low-dimensional representation for each cut. This process involves several steps including data cleaning, feature extraction and normalization.

Data Cleaning

The dataset provided by the PHM Society was already "clean" and contained no missing values. This significantly reduced the preprocessing effort required, allowing the focus to be on feature extraction and other important steps in the data preparation process.

Feature Extraction

Each cut file in the dataset consists of approximately 250,000 rows, with each row containing 7 data parameters, leading to a highly complex observation space of 1,750,000 dimensions for the agent to work with. To address the curse of dimensionality and extract meaningful features, the focus was on deriving 7 critical time-domain features for each input parameter recorded by the sensors.

The time-domain features extracted are as follows:

Extracted Time Domain Features	
Mean	$\frac{1}{N}\sum_{\iota=1}^{N} x_i$
Root Mean Square (RMS)	$\sqrt{\frac{1}{N}\sum_{i=1}^{N} {x_i}^2}$
Crest Factor	$\frac{max[x_i]}{RMS}$
The average Power	$\frac{1}{N}\sum_{i=1}^{N} {x_i}^2$
Skewness	$\frac{E\left[(x_i - \bar{x})^3\right]}{RMS^3}$
Kurtosis	$\frac{\frac{1}{N}\sum_{i=1}^{N}(x_{\ i} - \bar{x})^4}{RMS^4}$

By extracting these features for each of the 7 sensor parameters (Force in X, Y, Z; Vibration in X, Y, Z; and AE-RMS), the dimensionality of the data was reduced to 42 features per cut. This reduction in dimensionality is crucial for making the dataset manageable and ensuring the efficiency of the learning algorithms.

Normalization

After extracting the time-domain features, normalization was performed to a range of 0 to 1 to ensure uniformity and improve the performance of the learning algorithms. Normalization was conducted using min-max scaling, which adjusts the values in each feature to fall within the specified range, enhancing the convergence rate of the training process.

Integration of Wear Data

As a final step, the wear flute values were included into the dataset, ensuring a complete dataset that includes both the extracted features and the wear measurements. This all-in-one dataset is more practical for the training process as it contains all relevant information needed by the model.

Additional Data Analysis

An exploratory data analysis (EDA) was conducted to uncover patterns and relationships within the dataset. This included visualizing the features provided and examining correlations between different features. EDA helped gain a deeper understanding of the dataset's structure and its relevance to the model performance.

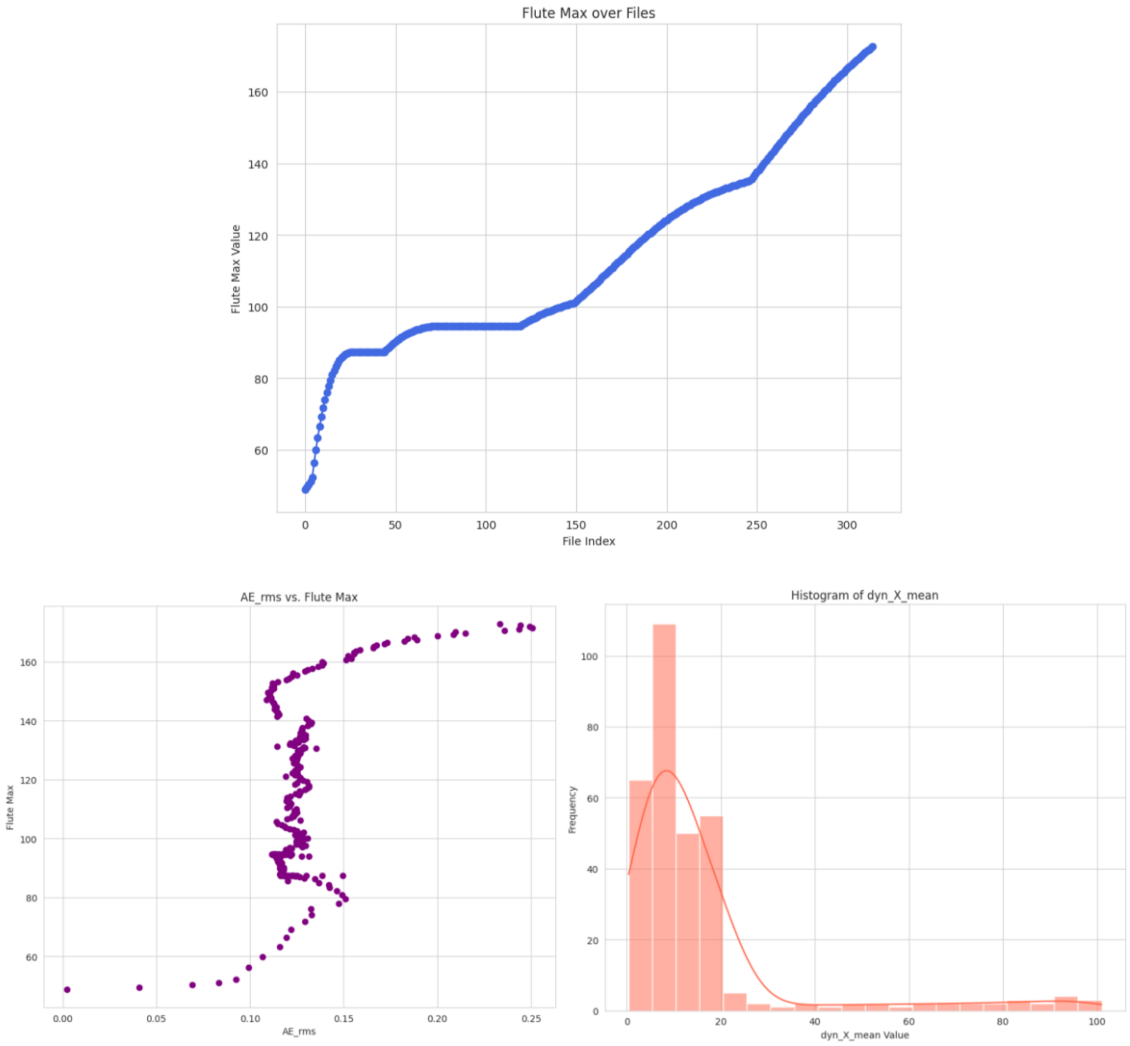


Figure 31 : Part of the Graphs that were created during EDA

Additionally, to better understand the data and the progression of flute degradation after each cut, the wear files provided in the dataset were analyzed. A function was developed to traverse each cutter machine's wear files and identify the maximum wear difference between sequential cuts. This information is crucial and will be utilized in the prediction process of the reinforcement learning agent. The maximum wear difference recorded was used as a hyperparameter in every MDP model.

4.4 MDP Model

After creating the dataset, the next step is to model the problem. For the Reinforcement Learning (RL) algorithm to function, a simulated environment is required where the agent can act and receive rewards. This simulated environment should utilize the created dataset and behave in a manner that enables the agent to learn to predict the wear of the CNC flute accurately.

A Markov Decision Process (MDP) can effectively model the CNC Machine Wear Problem environment. To define this MDP, it is necessary to specify the following components within the context of the problem: states, actions, transition probabilities, and rewards. Following the guidelines outlined in Chapter 3.3, the 5-tuple of the MDP will be tailored to the specific characteristics of the problem and the dataset created during preprocessing.

4.4.1 Components of the MDP

Using the Gymnasium (formerly OpenAI Gym) library we design a custom environment for solving the RL optimization problem. It simulates a system where an agent makes predictions about the wear on certain components (referred to as "flutes") in a manufacturing process. The agent aims to predict wear accurately based on historical processed data, with rewards and penalties assigned based on prediction accuracy.

In the context of Markov Decision Process (MDP) theory, the environment aligns with the MDP framework, which is defined by a 5-tuple (S, A, R, P, ρ_0).

The environments created for the training of the RL algorithm in this implementation are four. They are split into two major categories based on the Prediction Model and the Reward Calculation.

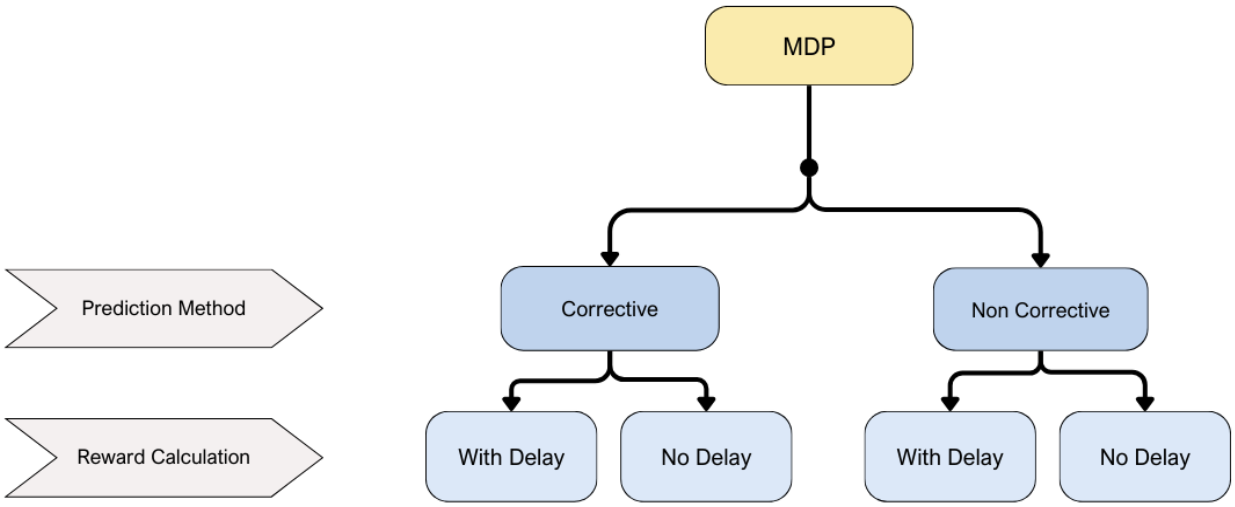


Figure 32 : The four different environments

All the environments contain some similar features but differ in the two major categories mentioned. The similar features are the State Space, Action Space and the Initial State Distribution:

Similar features

States (S):

- The State Space is completely represented by the observation space (fully observable environment) and consists of 42 time-domain features derived from sensor data. All the data is derived from the dataframe (df) that we provide to the environment class as an argument, which is the processed dataset we have created after completing the steps of section 4.2.

```
self.observation_space = gymnasium.spaces.Box(low=0, high=1,
                         shape=(42,), dtype=np.float64)
```

- Observations are constructed from the current step's sensor data using the custom `_get_obs()` method:

```
def _get_obs(self):
    obs = self.df.loc[self.current_step,
            time_domain_features_columns].to_numpy(dtype=np.float64)
    return obs
```

An observation is part of a row of the dataframe (df) which is the PHM dataset we have processed. For each step in every episode, meaning for each cut we examine, we pick from the data frame all the time domain features corresponding to the cut (the current step).

Actions (A):

- The action space is a continuous space with three dimensions, each ranging from 0 to 1. These actions represent the agent's predictions for the additional wear on three flutes.

- The action space is defined as :

```
low_bound = np.array([0, 0, 0], dtype=np.float64)
high_bound = np.array([1, 1, 1], dtype=np.float64)
self.action_space = gymnasium.spaces.Box(low=low_bound, high=high_bound,
dtype=np.float64)
```

The prediction of the additional wear of each flute will be used to calculate the overall wear

Initial State Distribution ρ_0 :

- The initial state is set up by the **`_state_s0()`** method, initializing the environment with the initial wear values:

```
def _state_s0(self):
    self.current_step = 0

    self.flute_1 = (2*self.df.loc[0, "flute_1"]) -
                    self.df.loc[1, "flute_1"]
    self.flute_2 = (2*self.df.loc[0, "flute_2"]) -
                    self.df.loc[1, "flute_2"]
    self.flute_3 = (2*self.df.loc[0, "flute_3"]) -
                    self.df.loc[1, "flute_3"]
    self.flute_max = (2*self.df.loc[0, "flute_max"]) -
                    self.df.loc[1, "flute_max"]

    self.flute_1_pred = self.flute_1
    self.flute_2_pred = self.flute_2
    self.flute_3_pred = self.flute_3
    self.flute_max_pred = max(self.flute_1, self.flute_2, self.flute_3)

    self.pred_dif_1 = 0
    self.pred_dif_2 = 0
    self.pred_dif_3 = 0
    self.pred_dif_max = 0

    self.score = 0
```

For the initial flute wear values since there is no information about the initial conditions on the data provided by the PHM, we set the value of every flute as:

flute-wear = wear_value(cut 0) - (wear_value(cut 1) - wear_value(cut 0))
= 2*wear_value(cut 0) - wear_value(cut 1)

Also, the variables of flute predictors (flute_i_pred) and flute prediction difference (flute_pred_dif_i) are initialized at certain values which will be useful for the reward calculation process.

This approach ensures a consistent starting state across different runs.

Rewards (R):

- The reward function evaluates the accuracy of the agent's predictions. The closer the predictions are to the actual wear values, the higher the reward.
- The reward is calculated using the **`score_function`** method described below, which are called in the **`step`** method:

```
    def step(self, action):
        self._take_action(action)

        reward_flute_1 = self.score_function(self.pred_dif_1)
        reward_flute_2 = self.score_function(self.pred_dif_2)
        reward_flute_3 = self.score_function(self.pred_dif_3)
        reward_flute_max = self.score_function(self.pred_dif_max)

        reward = (REW_WEIGHT[0]*reward_flute_1) +
                 (REW_WEIGHT[1]*reward_flute_2) +
                 (REW_WEIGHT[2]*reward_flute_3) +
                 (REW_WEIGHT[3]*reward_flute_max)

        #delay_modifier = (self.current_step / (MAX_STEPS-2))
        #reward = reward_without_delay * delay_modifier

    return observation, reward, terminated, truncated, info
```

Additional Components

Each one of the environments created also has these additional components that are not mentioned in the MDP theory but are required for the proper execution of the RL Algorithm:

- **Reset Function**:
    - The `reset` method reinitializes the environment to the starting state and returns the initial observation and information:

```
    def reset(self, seed=None, options=None):
        super().reset(seed=seed)
        self._state_s0()
        self.current_step = 0
        observation = self._get_obs()
        info = self._get_info()
        return observation, info
```

- **Termination and Truncation:**
    - The environment terminates when the current step exceeds the maximum number of steps (`MAX_STEPS`). This is done to ensure that we don't get a segmentation fault and the index is out of range.

    - The environment can also be truncated if the episode reward is smaller than -200 million. This is done in order to speed up the training process by cutting short poorly performing training episodes, allowing the agent to focus on more productive actions.

```
def step(self, action):
    ...
    if self.current_step > MAX_STEPS - 2:
    # MAX_STEPS - 2 = 315 - 2 = 313
        terminated = True

    if reward < -200000000: #200 million
        truncated = True
    return observation, reward, terminated, truncated, info
```

- **Score function:**

```
def score_function(self, delta):
    if delta < 0:
        return 1 - np.exp(-delta/10)
    else:
        return 1 - np.exp(delta/4.5)
```

Where δ represents the difference between the predicted maximum wear and the actual maximum wear.
(δ = Wear_Prediction_Value – Wear_Actual_Value)

- The score function is the negation of the original score function used in the 2010 PHM Data Challenge competition. The reason why the negation version of the function is used is that the agent wants to maximize its reward and the original version is a minimization function. It is designed to provide feedback to the agent based on the accuracy of its predictions, specifically focusing on penalizing overestimations more severely than underestimations.

  Analysis of the Score Function

  1. Underestimation (Negative δ):
     - When $\delta < 0$, the agent has underestimated the wear. The function $1 - e^{(-\delta/10)}$ is used, which grows slowly as δ becomes more negative.

       For example, if $\delta = -2$:

       $$S(-2) = 1 - e^{(-\delta/10)} = 1 - e^{(2/10)} \simeq -0.2214$$

     - The negative penalty is relatively small, reflecting a smaller punishment for underestimation.
  2. Overestimation (Positive δ):
     - When $\delta > 0$, the agent has overestimated the wear. The function $1 - e^{(\delta/4)}$ is used, which grows faster as δ becomes more negative.

For example, if δ = 2:

$$S(2) = 1 - e^{(\delta/4.5)} = 1 - e^{(2/4.5)} \simeq -0.5596$$

- The negative penalty is larger, returning a more significant punishment for overestimation

The score function effectively penalizes overestimations more heavily than underestimations. This design ensures that the agent learns to avoid predicting excessively high wear values, which can be more costly and disruptive. The use of exponential functions provides a smooth gradient for learning, allowing the agent to adjust its predictions incrementally and effectively.

Differences Between the Four Environments

The four environments are structured around two major categories: **Prediction Method** and **Reward Calculation**. Each combination of these categories defines a distinct environment within the Markov Decision Process (MDP) framework.

4.4.1.1 Prediction Method

- **Corrective Prediction**:
  - In this approach, the agent's predictions are corrected based on the actual wear values observed. This means that the agent adjusts its predictions by considering the current actual wear, providing a more accurate and real-time adjustment to the state of the environment.

```
self.flute_1_pred = self.flute_1 + (action[0] * MAX_DIFF)
self.flute_2_pred = self.flute_2 + (action[1] * MAX_DIFF)
self.flute_3_pred = self.flute_3 + (action[2] * MAX_DIFF)
self.flute_max_pred = max(self.flute_1_pred, self.flute_2_pred,
                          self.flute_3_pred)
```

- **Non-Corrective Prediction**:
  - This method involves updating the predictions incrementally based on previous predictions without immediate correction based on actual wear values. It relies on the agent's previous predictions to inform future predictions.

```
self.flute_1_pred += (action[0] * MAX_DIFF)
self.flute_2_pred += (action[1] * MAX_DIFF)
self.flute_3_pred += (action[2] * MAX_DIFF)
self.flute_max_pred = max(self.flute_1_pred, self.flute_2_pred,
self.flute_3_pred)
```

In both cases the action of the agent is multiplied by a MAX_DIFF variable which is provided by the analysis we have done in the preprocessing phase of the implementation on the wear files of the PHM

dataset. It is the max additional wear between two cuts in all of the cutters. By multiplying the max difference by a value on the range of 0 to 1 we can create the additional wear that should be predicted

4.4.1.2. Reward Calculation

- **No Delay**:
  - The reward is calculated directly based on the difference between predicted and actual wear values, without any scaling factor related to the current step. This method focuses on immediate accuracy without considering the long-term impact of actions.

```
reward = (REW_WEIGHT[0]*reward_flute_1) +
         (REW_WEIGHT[1]*reward_flute_2) +
         (REW_WEIGHT[2]*reward_flute_3) +
         (REW_WEIGHT[3]*reward_flute_max)
```

The REW_WEIGHT values are equal for every environment and are 0,1 for each flute and 0,7 for the flute_max.

- **With Delay**:
  - The reward is adjusted by a delay modifier, which scales the reward based on the current step within the episode. This method aims to emphasize actions that provide long-term benefits by rewarding actions that have a positive effect later in the episode.

```
delay_modifier = (self.current_step / (MAX_STEPS-2))
reward = reward_without_delay * delay_modifier
```

4.4.2. Summary of the Four Environments

Combining the two categories, the four distinct environments can be described as follows:

Environments	
Corrective with Delay	The agent's predictions are corrected based on actual wear values, and the rewards are adjusted by a delay modifier to emphasize long-term benefits.

Corrective No Delay	The agent's predictions are corrected based on actual wear values, and the rewards are calculated directly based on immediate prediction accuracy.
Non-Corrective with Delay	The agent's predictions are incrementally updated based on previous predictions, and the rewards are adjusted by a delay modifier to emphasize long-term benefits.
Non-Corrective No Delay	The agent's predictions are incrementally updated based on previous predictions, and the rewards are calculated directly based on immediate prediction accuracy.

In conclusion, the environments simulate the CNC machine's wear prediction problem. The agent's objective is to accurately predict the wear on three specific components (flutes) based on curated historical sensor data. The state space consists of 42 time-domain features from sensors, offering a detailed manufactured snapshot of the machine's condition after each cut. The action space is a three-dimensional continuous vector, where each dimension (ranging from 0 to 1) corresponds to the agent's prediction for the wear on one of the flutes. Rewards are calculated based on the accuracy of these predictions, with the reward function favoring closer predictions and penalizing larger deviations.

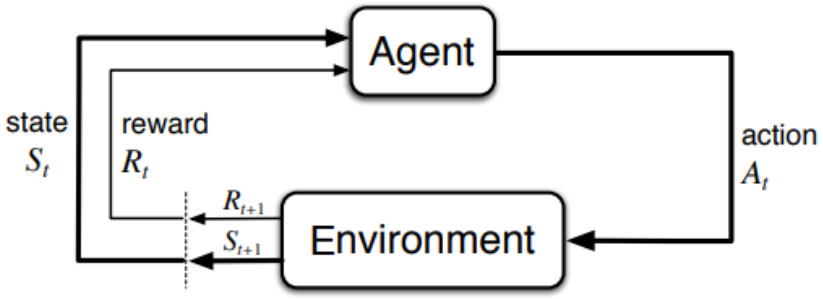


Figure 33 : The MDP Model

State transitions are governed by the `_take_action()` method, which updates the environment based on the agent's predictions. The initial state is set by the `_state_s0()` method, initializing the environment with predefined wear values. The environment resets to its starting state at the beginning of each episode, ensuring a consistent baseline for the agent's learning process. The step function executes one time step in the environment, updating the state, calculating the reward, and determining whether the episode should terminate or truncate. This setup provides a controlled simulation for training reinforcement learning agents to optimize their prediction accuracy, illustrating a practical application of Markov Decision Process (MDP) theory.

4.5 RL Model Training

In this section, the training process of the reinforcement learning (RL) models is described in detail. Four algorithms were utilized within four different environments, resulting in a total of sixteen distinct models. The training was conducted on the processed dataset of cutter 1 (train set).

The training process begins by creating an instance of the simulated environment using the preprocessed dataset. Each environment allows the RL agent to interact with it and learn to predict the wear of the machine.

The next step is to initialize the environment and set up a logging mechanism to track the progress and performance of the models. A key aspect of this setup is defining the number of timesteps for training and creating a directory structure to store the training logs and model files. A callback function was implemented to log the score during training and to monitor the performance of the agent at various steps.

To ensure efficient and organized training, a function was developed to manage the creation and naming of model files. This function also can load previously trained models, enabling the continuation of training from the last saved state. During the training loop, models are periodically saved, and their performance is evaluated to ensure they are learning effectively.

The training process is executed in iterations, with each iteration involving a specified number of timesteps. After each iteration, the models are evaluated, and their performance metrics are logged.

Parameters

A compilation of constant parameters are used in training:

```
# Global Parameters for CNC Custom Env
MAX_STEPS = len(df.loc[:, 'File'].values) #len = 315

MAX_DIFF = 20

REW_WEIGHT = [0.1 , 0.1 , 0.1 , 0.7] # flute_1-2-3 and flute_max

TIMESTEPS = 10000

logdir = "logs"

time_domain_features_columns = ['dyn_X_mean', 'dyn_X_rms', 'dyn_X_crest',
'dyn_X_avg_power', 'dyn_X_skewness', 'dyn_X_kurtosis', 'dyn_Y_mean', 'dyn_Y_rms',
'dyn_Y_crest', 'dyn_Y_avg_power', 'dyn_Y_skewness', 'dyn_Y_kurtosis', 'dyn_Z_mean',
'dyn_Z_rms', 'dyn_Z_crest', 'dyn_Z_avg_power', 'dyn_Z_skewness', 'dyn_Z_kurtosis',
'acc_X_mean', 'acc_X_rms', 'acc_X_crest', 'acc_X_avg_power', 'acc_X_skewness',
'acc_X_kurtosis', 'acc_Y_mean', 'acc_Y_rms', 'acc_Y_crest', 'acc_Y_avg_power',
'acc_Y_skewness', 'acc_Y_kurtosis', 'acc_Z_mean', 'acc_Z_rms', 'acc_Z_crest',
```

```
'acc_Z_avg_power', 'acc_Z_skewness', 'acc_Z_kurtosis', 'AE_mean', 'AE_rms',
'AE_crest', 'AE_avg_power', 'AE_skewness', 'AE_kurtosis']
flute_wear_collumns = ['flute_1','flute_2', 'flute_3', 'flute_max']
```

The **MAX_STEPS** parameter represents the maximum number of steps an agent can do in the environment, and it is equal to the length of the data frame created in section 4.3.

The **MAX_DIFF** parameter sets the maximum allowable difference in the wear measurements, helping to constrain the model's predictions within a realistic range.

The **REW_WEIGHT** parameter is a list of weights assigned to the reward components for different flutes. The weights [0.1, 0.1, 0.1, 0.7] indicate the relative importance of the wear measurements for flute 1, flute 2, flute 3, and the maximum wear among the flutes, respectively. This helps prioritize the most critical wear measurement in the reward calculation.

The training timesteps (**TIMESTEPS**) are constant for all models and the value is set to 10,000 steps. The **logdir** parameter is the directory where the training logs are stored.

The **time_domain_features_collumns** is a list that contains the names of the columns representing the time-domain features extracted from the dynamometer, accelerometer, and Acoustic Emission (AE) data. These features are used as input to the RL model.

Lastly, the **flute_wear_collumns** is a list that includes the names of the columns representing the wear measurements for the three individual flutes and the maximum wear among them. These measurements are necessary for calculating agent’s rewards.

Also, the learning rate is constant in all models depending on the algorithm:

Learning Rate of RL Algorithms	
PPO	0.0003
SAC	0.0003
A2C	0.0007
DDPG	0.001

Table 7 : Learning Rate of RL Algorithms

4.5.1. Non-Corrective Environments

4.5.1.1. PPO No Delay

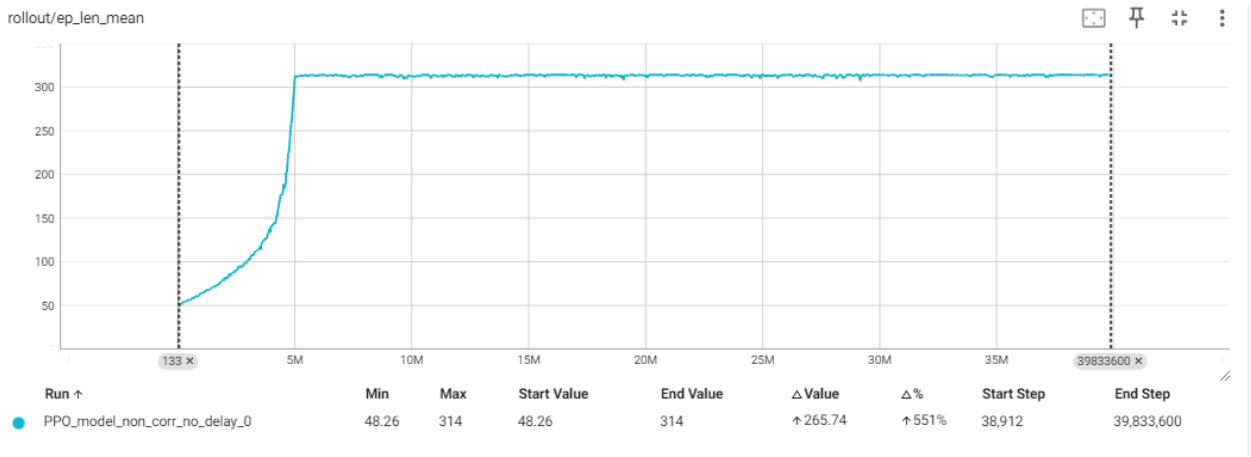


Figure 34 : Episode Length Mean per Timestep of PPO Non-Corrective No Delay Model

Initially, the episode length increases rapidly indicating the model's learning and improving performance. After about 5 million steps, the episode length stabilizes around 314, suggesting that the model has converged.

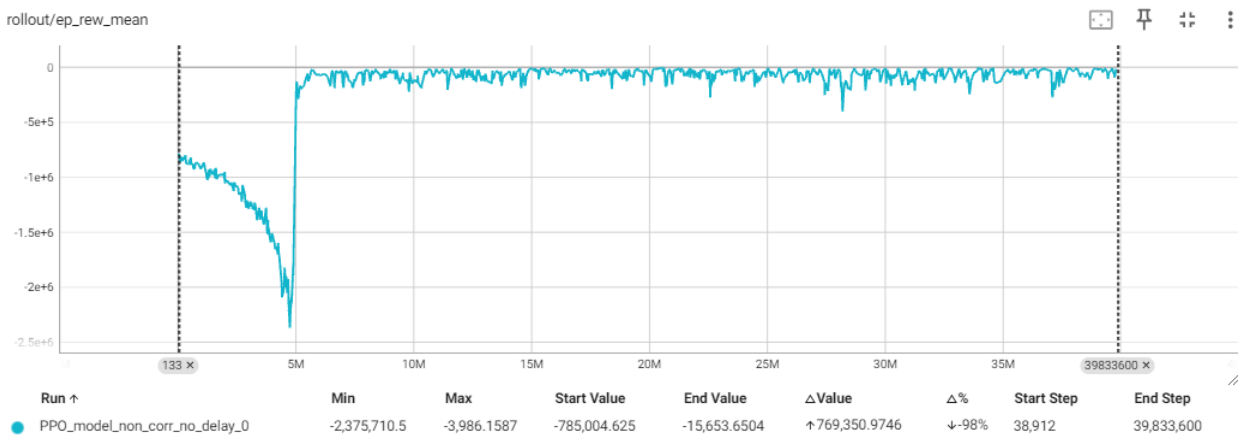


Figure 35 : Episode Reward Mean per Timestep of PPO Non-Corrective No Delay Model

At first, the rewards decrease sharply, indicating a period of learning and exploration. Around 5 million steps, the rewards stabilize and maintain a consistent value with some fluctuations, suggesting that the model has converged.

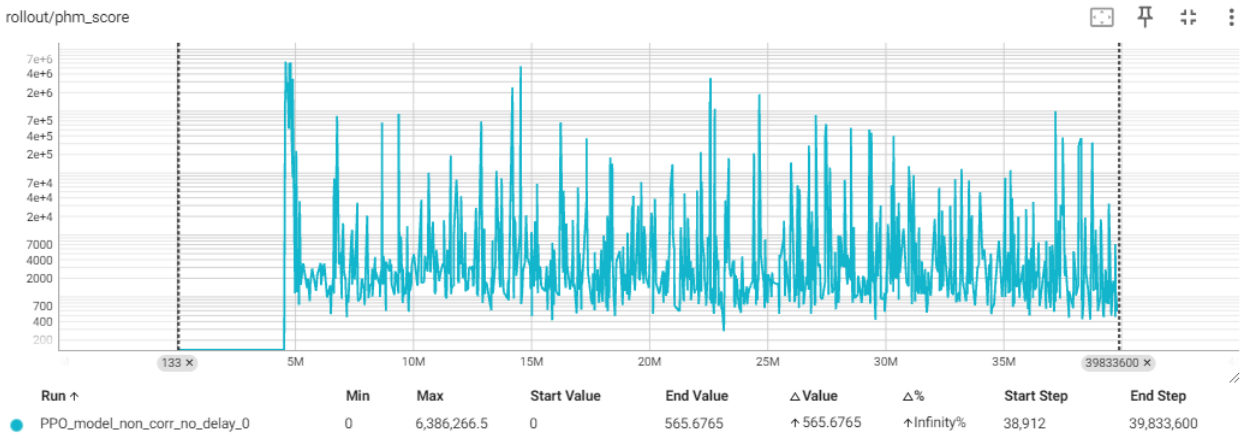


Figure 36 : PHM Score per Timestep of PPO Non-Corrective No Delay Model

After applying "Smoothing" at value 0.99 we can easily observe a downward trend:

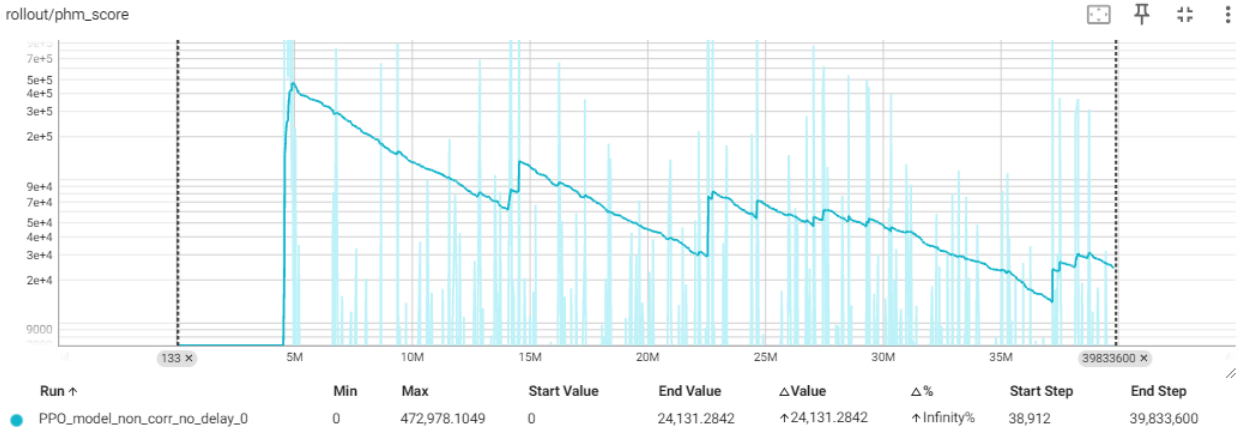


Figure 37 : "Smoothed" Version of PHM Score per Timestep of PPO Non-Corrective No Delay Model

The first graph shows the PHM score for the model with significant fluctuations throughout the training period, indicating instability in minimizing the metric. The second graph, with smoothing applied, reveals a clearer trend: the PHM score decreases gradually over time, despite initial high values and fluctuations. This trend suggests that the model is progressively improving and learning to minimize the PHM score, even though it is not immediately apparent in the unsmoothed data. The smoother graph demonstrates a general improvement and convergence towards lower PHM scores, indicating that the training process is heading in the right direction.

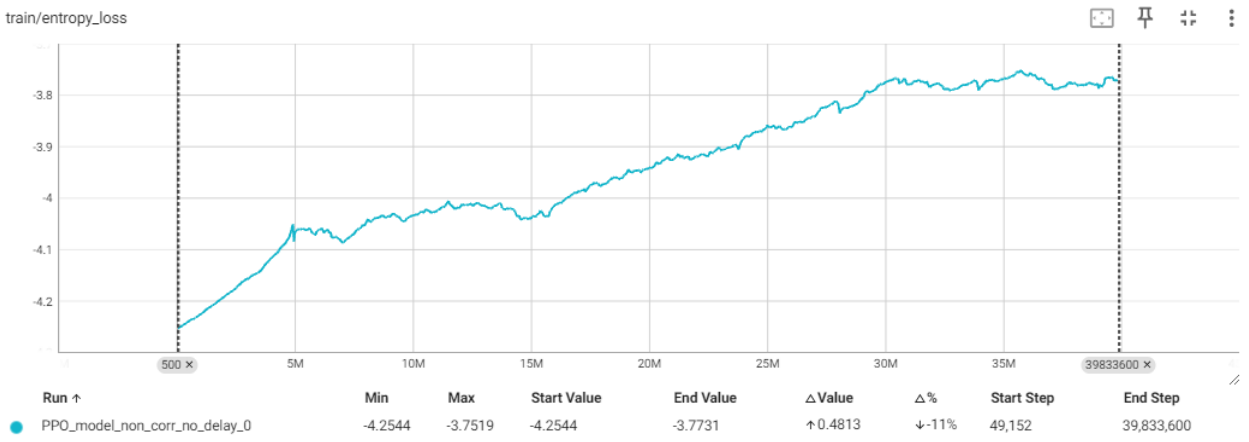


Figure 38 : Entropy Loss per Timestep of PPO Non-Corrective No Delay Model

Entropy loss measures the randomness in the policy's action selection. It starts at a lower value and gradually increases over time, that is an indication of good exploratory performance. Later, the increase of entropy loss is getting stabilized. This trend suggests that the model is converging and becoming more confident in its decisions. Despite some fluctuations, the overall increase in entropy loss reflects the model's learning process.

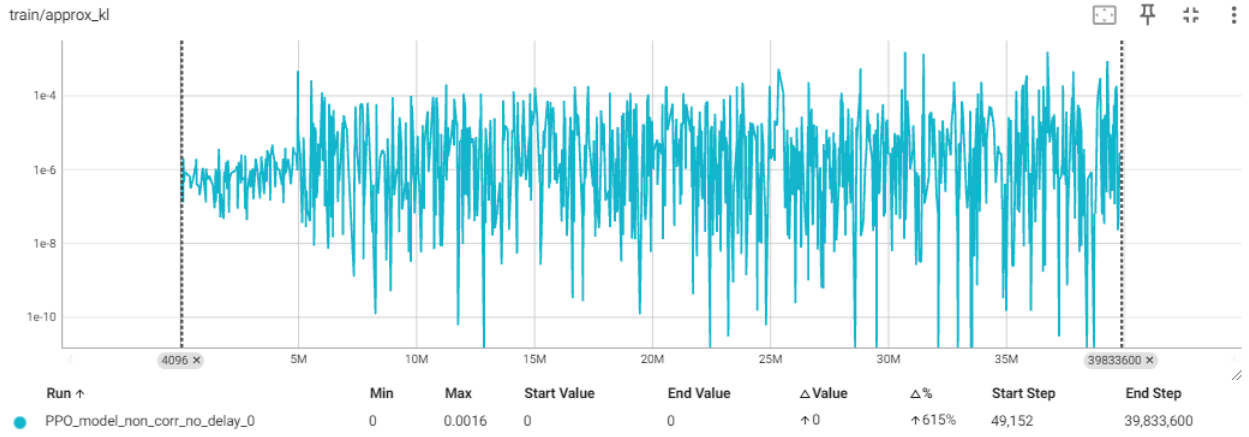


Figure 39 : "Approximate KL Divergence per Timestep of PPO Non-Corrective No Delay Model

KL divergence measures the difference between the updated policy and the old policy, indicating how much the policy is changing at each update. Especially in algorithms like PPO, KL divergence is often used as a regularization term to ensure that the updated policy remains close to the previous policy during training.

Throughout the training period, the KL divergence fluctuates significantly, with no clear trend towards stabilization. The fluctuation suggests that the policy updates are inconsistent, possibly due to the model continuously adjusting its actions in response to the environment. Ideally, KL divergence should show a decreasing trend or stabilize, indicating that the policy is converging. The current pattern indicates that the model may still be exploring and adjusting.

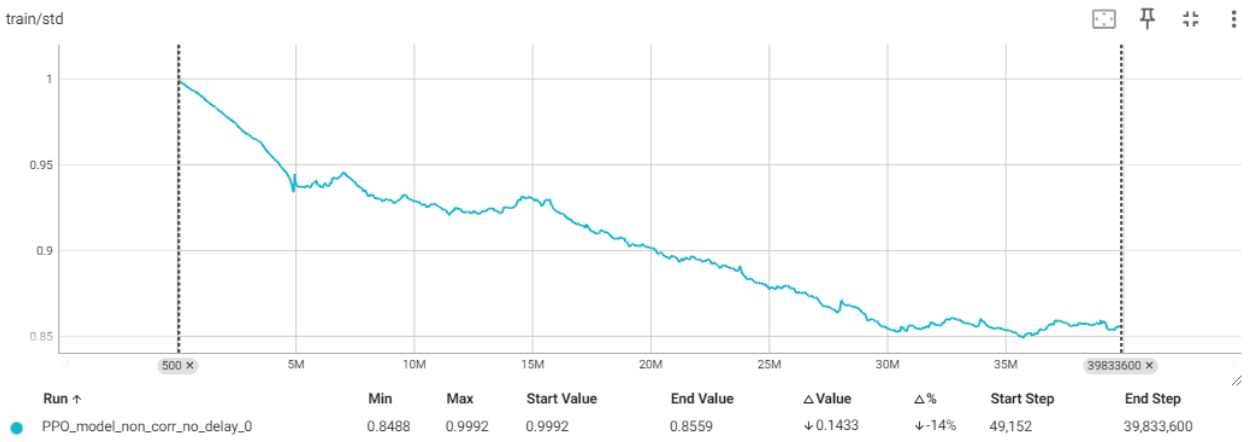


Figure 40 : Standard Deviation per Timestep of PPO Non-Corrective No Delay Model

Initially, the standard deviation starts at a higher value and gradually decreases, indicating that the actions selected by the policy are becoming more consistent and less exploratory over time. This decline suggests that the model is converging, and the policy is becoming more deterministic as it gains confidence in the optimal actions to take. The smooth, downward trend signifies a reduction in exploration, aligning with the expected behavior as the model learns and stabilizes. The decrease of about 14% from the start to the end of the training period matches the progressive stabilization and confidence in the policy's actions.

4.5.1.2. PPO With Delay

Figure 41 : Mean Episode Length per Timestep of PPO Non-Corrective With Delay Model

Starting at 53.33, the mean episode length increases rapidly and stabilizes at the max value of 314 cuts around 4 million steps. The model maintains consistent performance (in this metric) at this maximum episode length throughout the remainder of the training period, indicating efficient learning and stability.

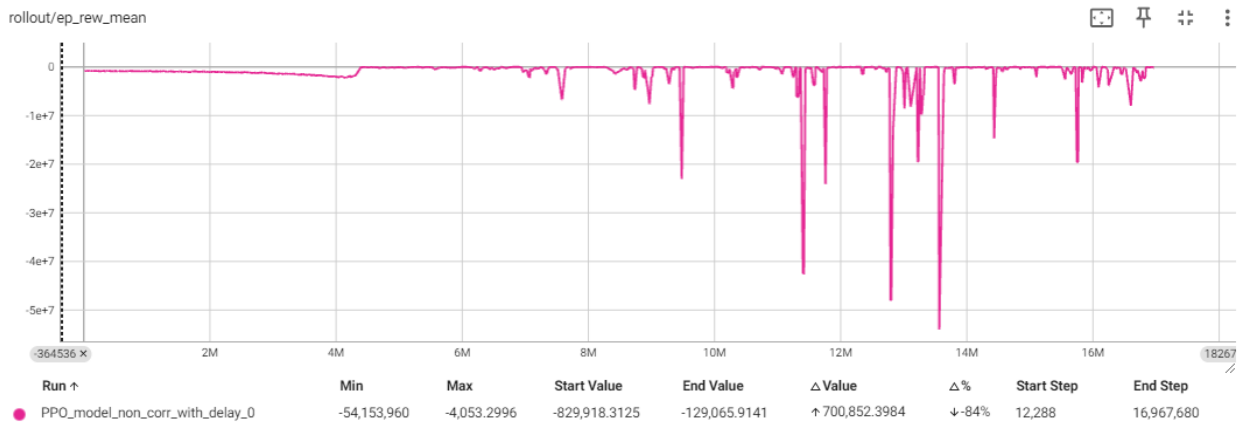


Figure 42 : Mean Episode Reward per Timestep of PPO Non-Corrective With Delay Model

Starting from a reward of -829,918, the model exhibits fluctuations with occasional significant drops, especially around the 11 million step mark. Despite these drops, the overall trend shows an improvement, ending at -129,065, reflecting an increase of 700,852.40 in the mean episode reward. This indicates that while the model faces some instability, it generally improves over time.

Figure 43 : PHM Score per Timestep of PPO Non-Corrective With Delay Model

A direct correlation with the previous two figures can be drawn here, as the first PHM score value appears around the 4 million step mark when the model reaches the maximum episode length. After reaching that, a significant decrease in the score is observed, which coincides with the corresponding significant improvement in the mean reward at similar timesteps.

Beyond the 4.5 to 5 million step mark, a lot of fluctuations can be observed, which indicate that while the model learns and improves, it faces challenges in maintaining consistent performance in minimizing the Phm score.

Figure 44 : Entropy Loss per Timestep of PPO Non-Corrective With Delay Model

This trend indicates that the policy becomes more explorative and then deterministic over time, with the fluctuations reflecting periods of exploration. The overall small increase of 0.1555 in entropy loss (a 4% change) suggests the model maintains a balance between exploration and exploitation throughout the training process.

PPO No Delay vs With Delay (Non-Corrective)

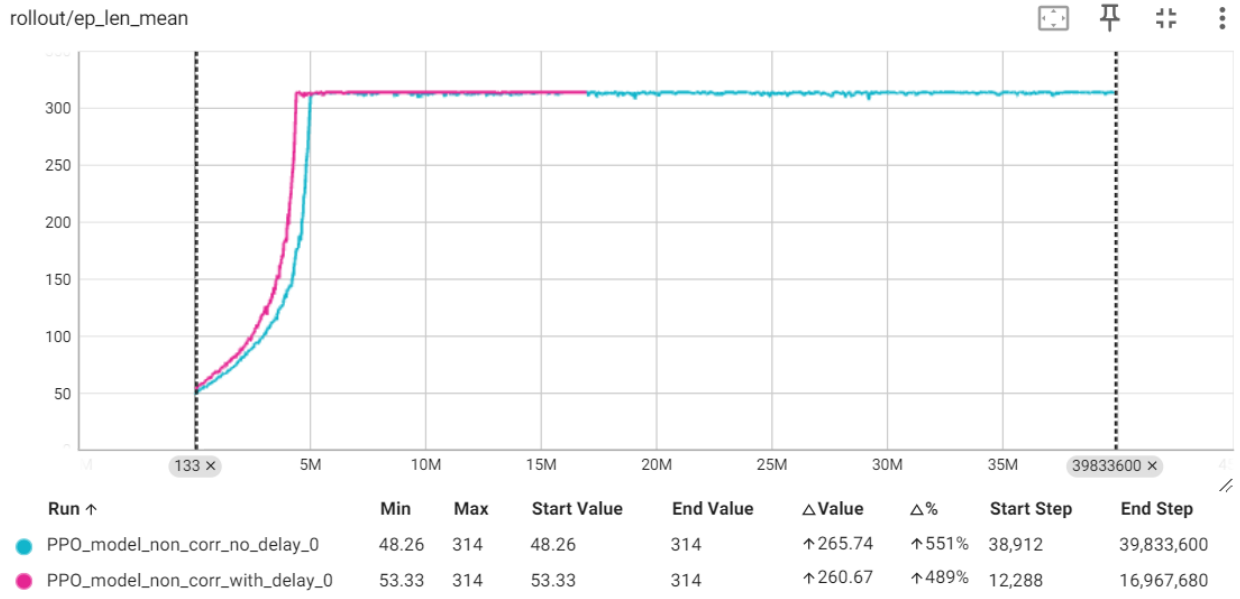
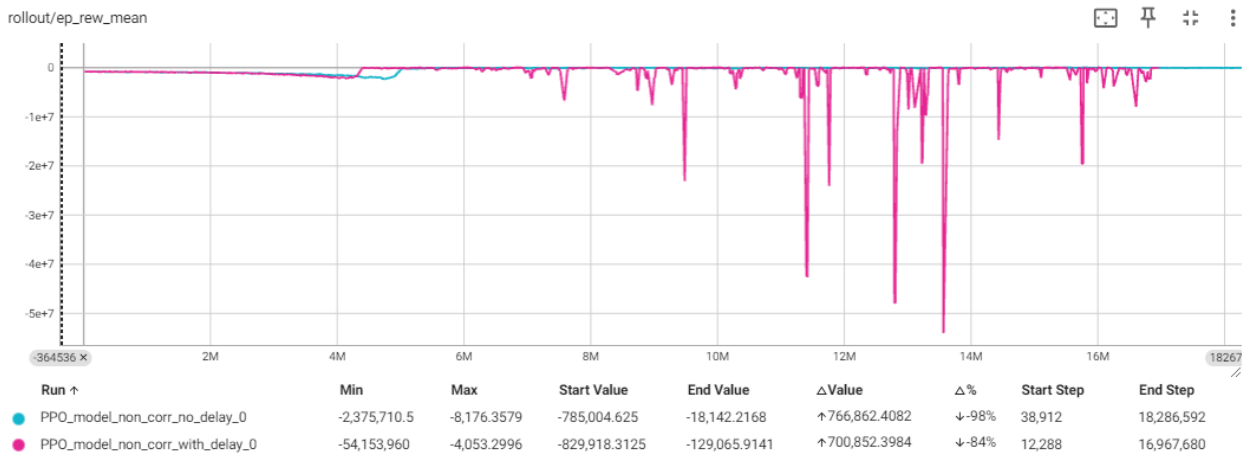


Figure 45 : Comparison of Mean Episode Length per Timestep of PPO Non-Corrective With Delay and No Delay Models

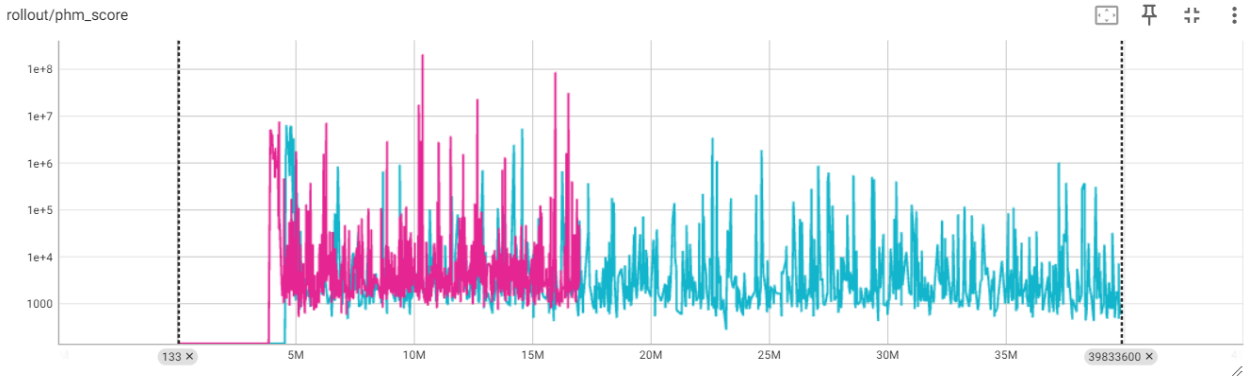
The model "with delay" reaches episode length of 314 faster than the "no delay" model, suggesting more efficient learning early on. Despite this, both models ultimately achieve the same final performance.



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
PPO_model_non_corr_no_delay_0	-2,375,710.5	-8,176.3579	-785,004.625	-18,142.2168	↑766,862.4082	↓-98%	38,912	18,286,592
PPO_model_non_corr_with_delay_0	-54,153,960	-4,053.2996	-829,918.3125	-129,065.9141	↑700,852.3984	↓-84%	12,288	16,967,680

Figure 46 : Comparison of Mean Episode Reward per Timestep of PPO Non-Corrective With Delay and No Delay Models

The model "with_delay" exhibits significantly larger fluctuations, including deep spikes. Over time, both models improve.

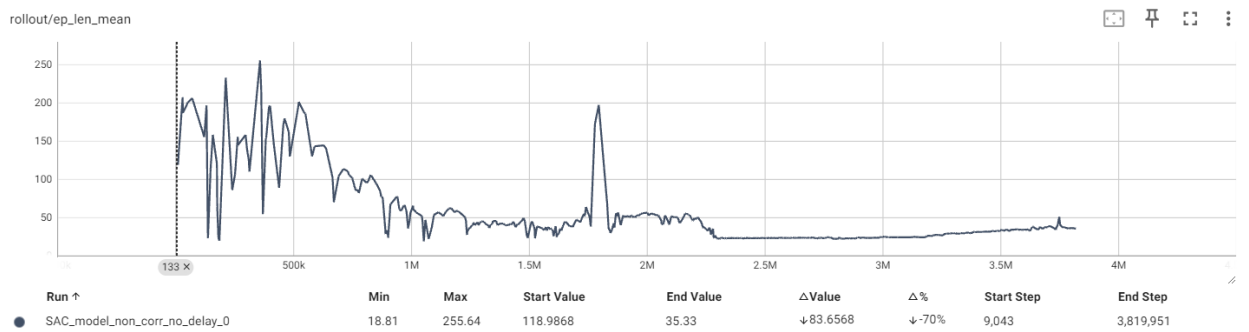


Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
PPO_model_non_corr_no_delay_0	0	6,386,266.5	0	565.6765	↑565.6765	↑Infinity%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	0	206,020,384	0	2,777.8508	↑2,777.8508	↑Infinity%	12,288	16,967,680

Figure 47 : Comparison of PHM Score per Timestep of PPO Non-Corrective With Delay and No Delay Models

The model "with delay" shows higher initial PHM scores and greater fluctuations compared to the model "no delay". Over time, both models reduce the PHM scores. The delay correction may accelerate learning initially, it results in higher variability and less optimal final performance in minimizing PHM scores.

4.5.1.3. SAC No Delay



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_non_corr_no_delay_0	18.81	255.64	118.9868	35.33	↓83.6568	↓-70%	9,043	3,819,951

Figure 48 : Mean Episode Length per Timestep of SAC Non-Corrective No Delay Model

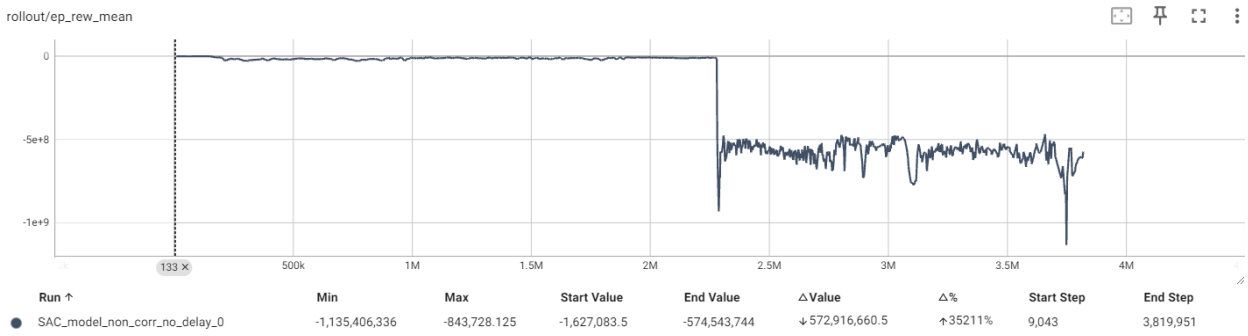


Figure 49 : Mean Episode Reward per Timestep of SAC Non-Corrective No Delay Model

The SAC Non-Corrective No Delay model shows a big downward trend in the mean episode length and the mean episode reward also falls down drastically. These difficulties in learning and maintaining good performance suggest bad training.

4.5.1.4. SAC With Delay

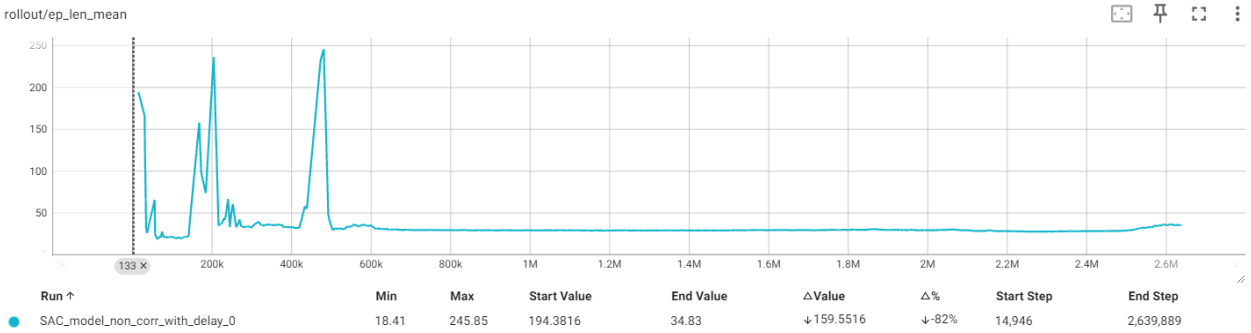


Figure 50 : Mean Episode Length per Timestep of SAC Non-Corrective With Delay Model

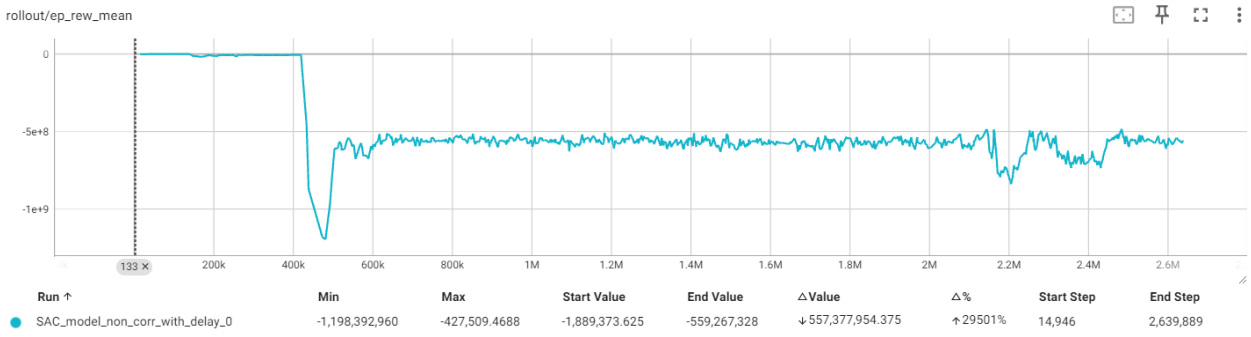
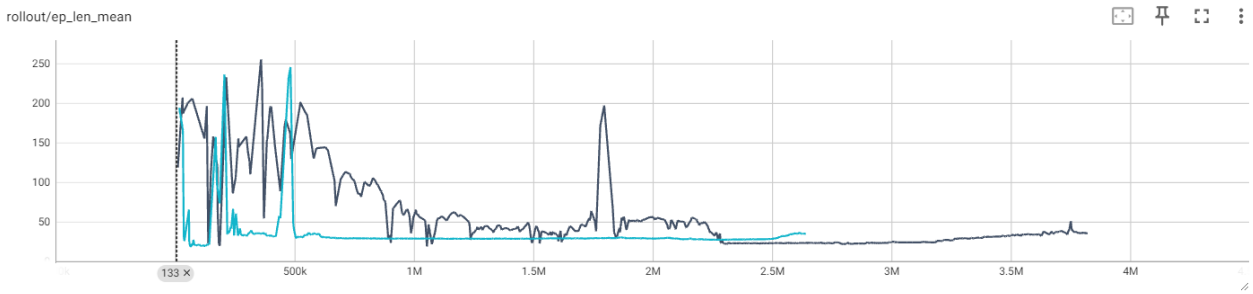


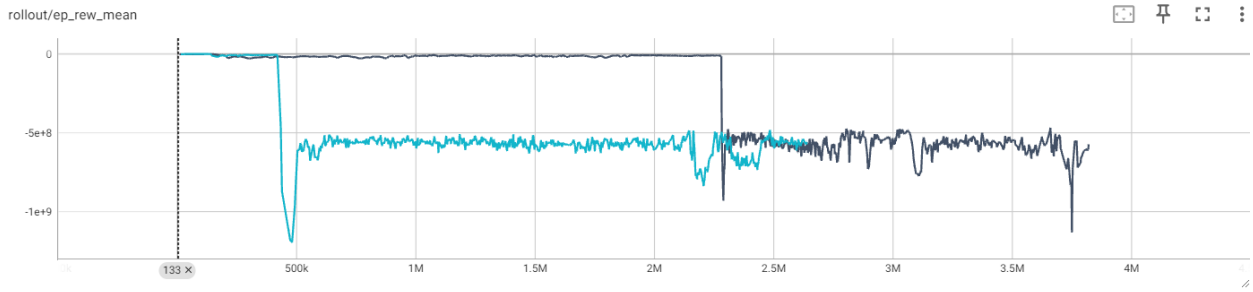
Figure 51 : Mean Episode Reward per Timestep of SAC Non-Corrective With Delay Model

The model initially explores different strategies, and it struggles to maintain longer episodes over time. The drop in reward is matched by the exploration attempt at the 400,000-step mark. Similar to the No Delay model, the learning process does not progress in the right direction and is stagnant.



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_non_corr_no_delay_0	18.81	255.64	118.9868	35.33	↓83.6568	↓-70%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	18.41	245.85	194.3816	34.83	↓159.5516	↓-82%	14,946	2,639,889

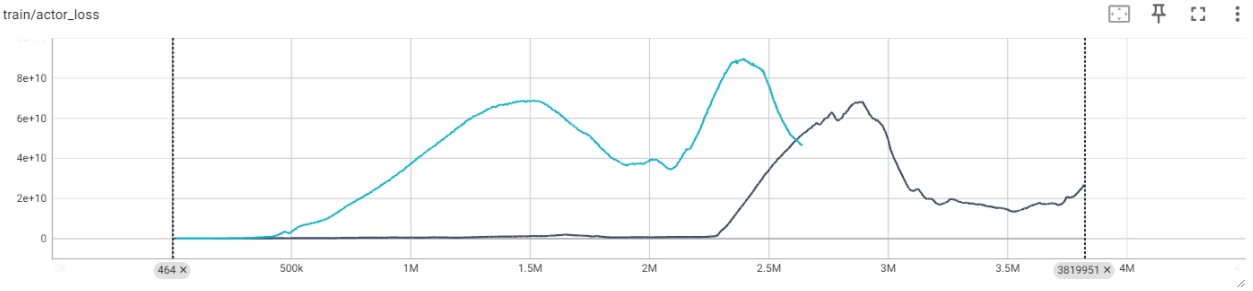
Figure 52 : Comparison of Mean Episode Length per Timestep of SAC Non-Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_non_corr_no_delay_0	-1,135,406,336	-843,728.125	-1,627,083.5	-574,543,744	↓572,916,660.5	↑35211%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	-1,198,392,960	-427,509.4688	-1,889,373.625	-559,267,328	↓557,377,954.375	↑29501%	14,946	2,639,889

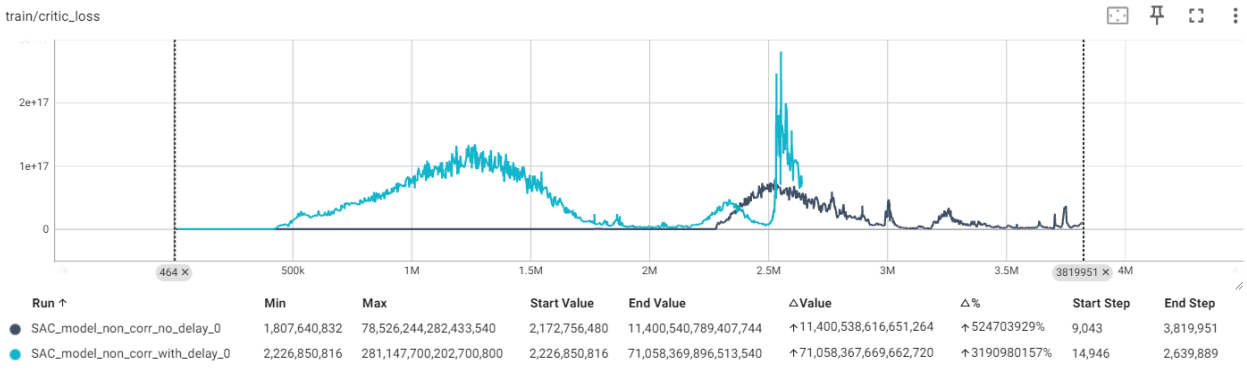
Figure 53 : Comparison of Mean Episode Reward per Timestep of SAC Non-Corrective With Delay and No Delay Models

None of the models reached the 314-cut mark so the PHM score cannot be examined



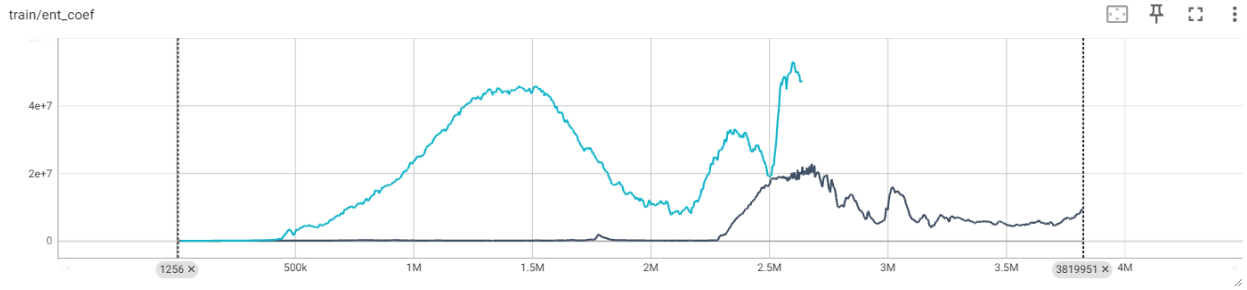
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_non_corr_no_delay_0	448,329.2813	68,223,971,328	448,329.2813	26,785,886,208	↑26,785,437,878.7188	↑5974501%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	403,940.5	89,741,172,736	694,773.6875	46,315,708,416	↑46,315,013,642.3125	↑6666201%	14,946	2,639,889

Figure 54 : Comparison of Actor Loss per Timestep of SAC Non-Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_non_corr_no_delay_0	1,807,640,832	78,526,244,282,433,540	2,172,756,480	11,400,540,789,407,744	↑11,400,538,616,651,264	↑524703929%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	2,226,850,816	281,147,700,202,700,800	2,226,850,816	71,058,369,896,513,540	↑71,058,367,669,662,720	↑3190980157%	14,946	2,639,889

Figure 55 : Comparison of Critic Loss per Timestep of SAC Non-Corrective With Delay and No Delay Models

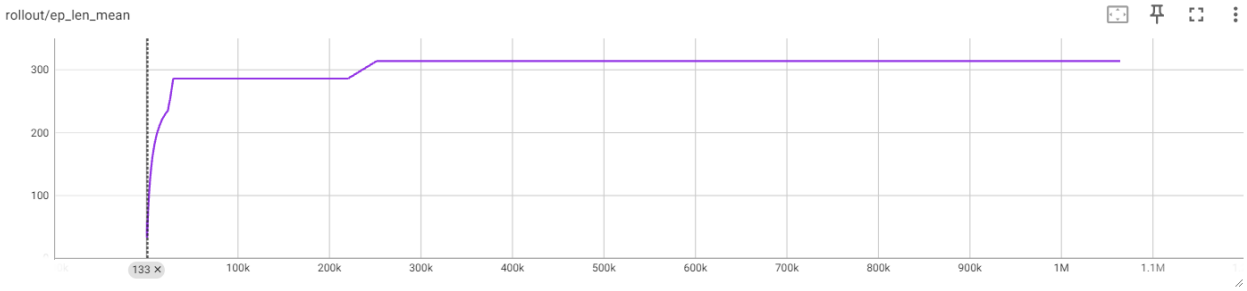


Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_non_corr_no_delay_0	15.0069	22,817,644	15.0069	9,480,854	↑9,480,838.9931	↑63176633%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	76.5747	52,936,676	76.5747	47,141,892	↑47,141,815.4253	↑61563199%	14,946	2,639,889

Figure 56 : Comparison of Entropy Coefficient per Timestep of SAC Non-Corrective With Delay and No Delay Models

The comparison of SAC non-corrective with delay and no delay models shows that both exhibit a downward trend in mean episode length and rewards and both models have not reached the max episode length of 314. The no delay model starts at 118.98 and ends at 35.33, while the with delay model starts at 194.38 and ends at 34.83, indicating that both struggle to maintain longer episodes. Both models have highly negative rewards, highlighting their poor performance. Actor loss and critic loss graphs for both models show significant variability, this results in instability in learning and challenges in value function estimation. The entropy coefficient graph indicates substantial changes in the exploration-exploitation balance for both models. Overall, despite initial exploration, both models fail to progress in the right direction, demonstrating significant difficulties in achieving good performance.

4.5.1.5 DDPG No Delay



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_no_delay_0	32.25	314	33.25	314	↑280.75	↑844%	133	1,064,325

Figure 57 : Mean Episode Length per Timestep of DDPG Non-Corrective No Delay Model

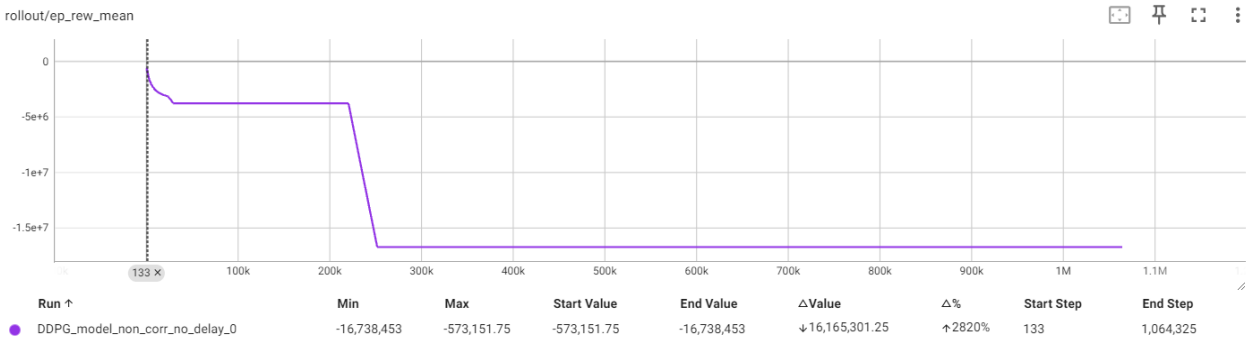


Figure 58 : Mean Episode Reward per Timestep of DDPG Non-Corrective No Delay Model

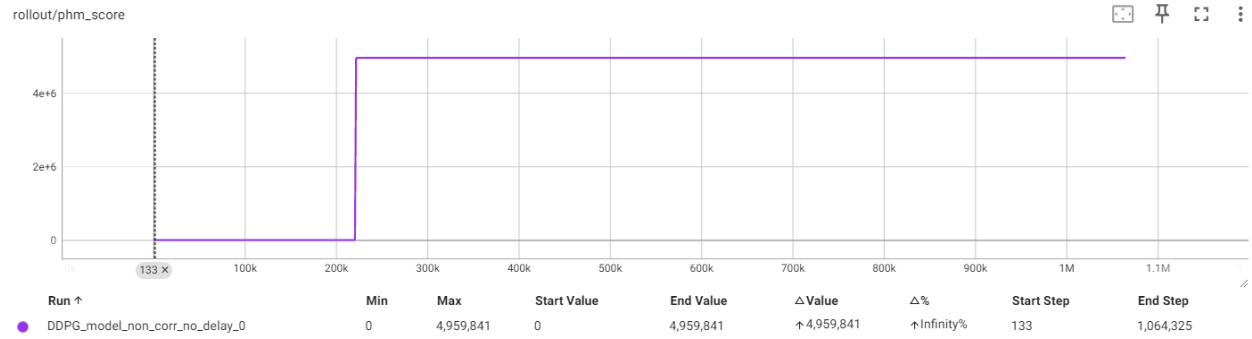


Figure 59 : PHM Score per Timestep of DDPG Non-Corrective No Delay Model

The DDPG Non Corrective No Delay model demonstrates an initial fast learning phase in terms of episode length but quickly converges to a suboptimal policy with poor rewards and high PHM scores. This suggests that while the model can achieve long episodes, it fails to optimize the rewards metrics and is stuck in a local minimum.

4.5.1.6. DDPG With Delay

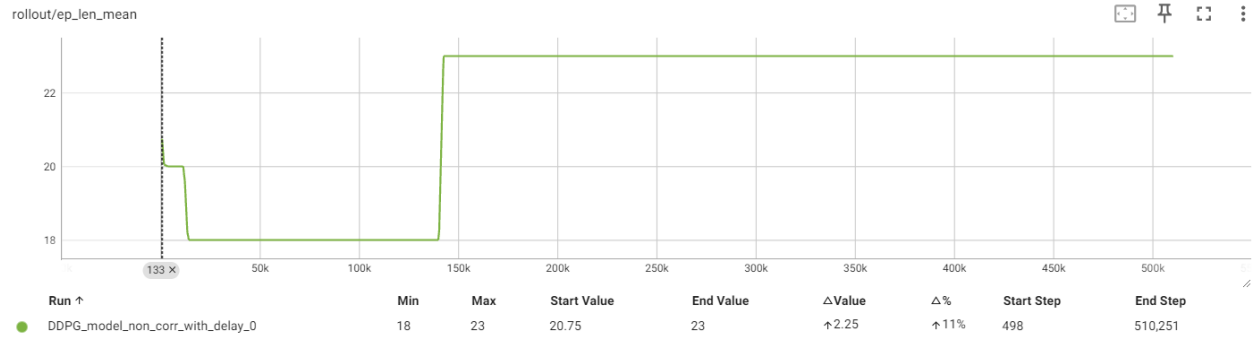
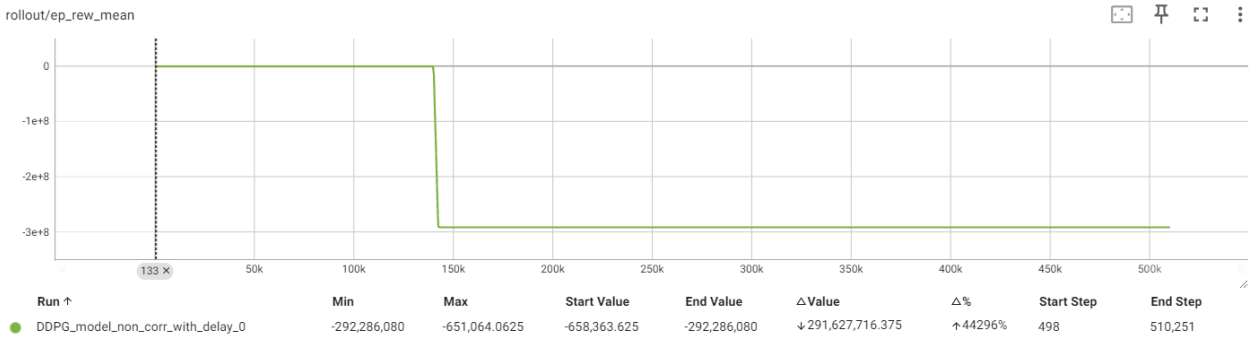


Figure 60 : Mean Episode Length per Timestep of DDPG Non-Corrective With Delay Model

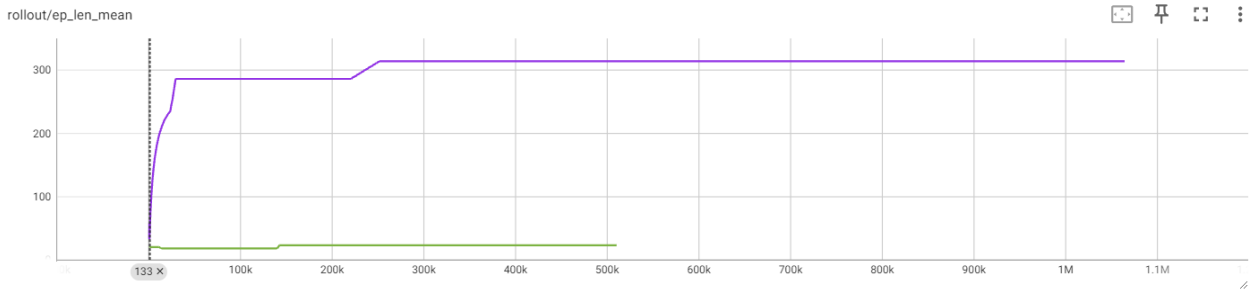


Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-658,363.625	-292,286,080	↓291,627,716.375	↑44296%	498	510,251

Figure 61 : Mean Episode Reward per Timestep of DDPG Non-Corrective With Delay Model

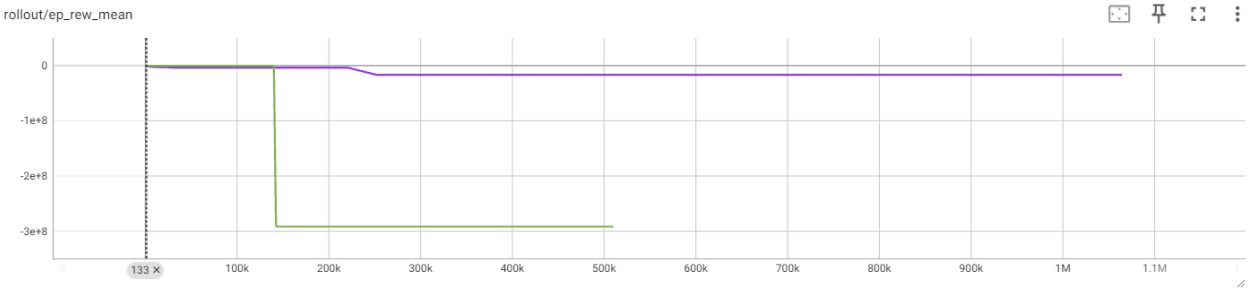
This model never reached the max cut 314 and therefore there is no PHM score graph.

The DDPG Non Corrective With Delay model is the worst performer in the training process, showing a minor increase in mean episode length and severe decline in the mean episode reward. This model failed to reach the maximum cut of 314, resulting in no PHM score record. Overall, the model has failed to train effectively, demonstrating significant limitations in its learning process and effectiveness.



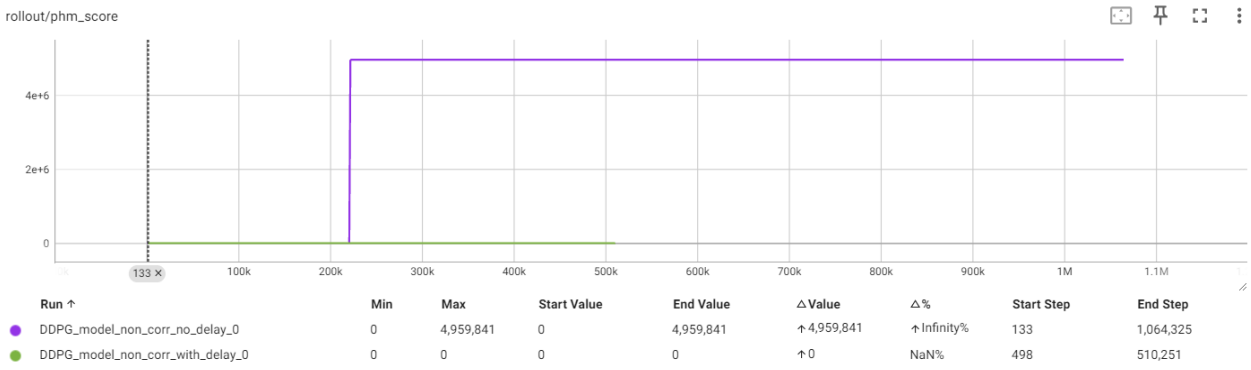
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_no_delay_0	32.25	314	33.25	314	↑280.75	↑844%	133	1,064,325
DDPG_model_non_corr_with_delay_0	18	23	20.75	23	↑2.25	↑11%	498	510,251

Figure 62 : Comparison of Mean Episode Length per Timestep of DDPG Non-Corrective With Delay and No Delay Models



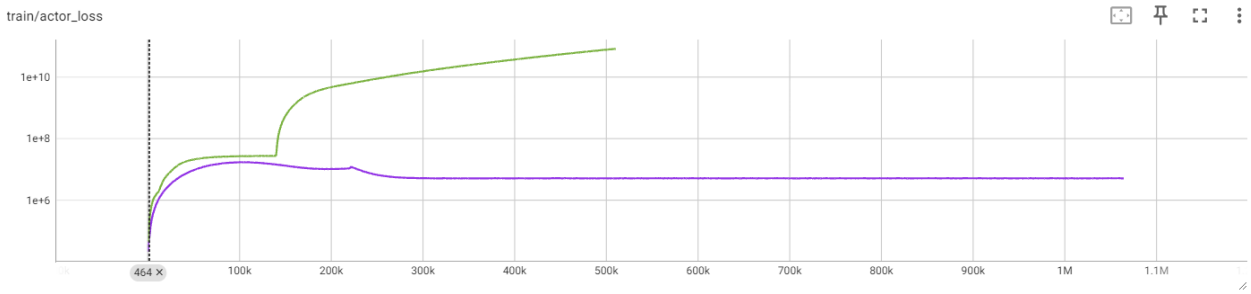
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_no_delay_0	-16,738,453	-573,151.75	-573,151.75	-16,738,453	↓16,165,301.25	↑2820%	133	1,064,325
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-658,363.625	-292,286,080	↓291,627,716.375	↑44296%	498	510,251

Figure 63 : Comparison of Mean Episode Reward per Timestep of DDPG Non-Corrective With Delay and No Delay Models



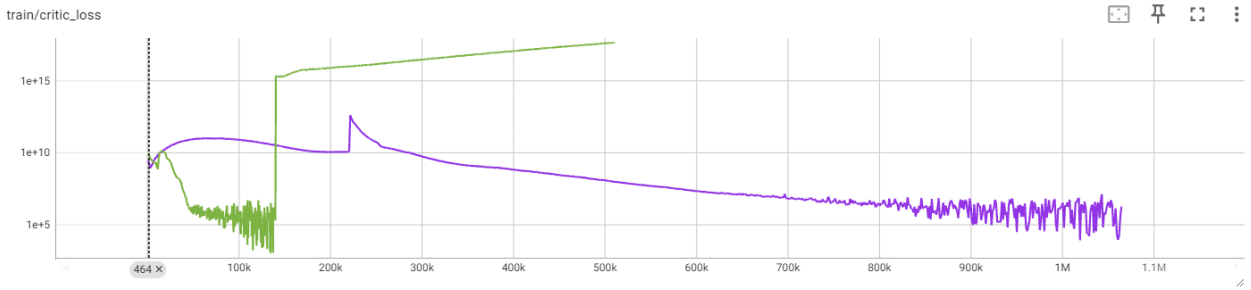
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_no_delay_0	0	4,959,841	0	4,959,841	↑4,959,841	↑Infinity%	133	1,064,325
DDPG_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	498	510,251

Figure 64 : Comparison of PHM Score per Timestep of DDPG Non-Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_no_delay_0	20,825.6172	16,853,906	20,825.6172	5,047,827	↑5,027,001.3828	↑24139%	464	1,064,325
DDPG_model_non_corr_with_delay_0	44,001.1367	82,904,326,144	44,001.1367	82,876,825,600	↑82,876,781,598.8633	↑188351456%	498	510,251

Figure 65 : Comparison of Actor Loss (Log Scale) per Timestep of DDPG Non-Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_non_corr_no_delay_0	7,942.6938	4,070,952,402,944	1,920,370,816	1,799,887.375	↓1,918,570,928.625	↓-100%	464	1,064,325
DDPG_model_non_corr_with_delay_0	976.3956	456,673,770,820,599,800	9,516,695,552	451,382,577,270,358,000	↑451,382,567,753,662,460	↑4743059871%	498	510,251

Figure 66 : Comparison of Critic Loss (Log Scale) per Timestep of DDPG Non-Corrective With Delay and No Delay Models

The comparison of DDPG Non-Corrective With Delay and No Delay models reveals significant differences in performance. The No delay model quickly achieves and maintains the maximum mean episode length of 314, while the With Delay model shows only a minor increase from 20.75 to 23. The actor loss graph indicates that the no delay model stabilizes, while the with delay model continues to increase, showing instability. Similarly, the critic loss graph shows that the no delay model stabilizes while the with delay model fluctuates significantly. Overall, the No-delay model is better, but it fails to optimize the reward metrics.

4.5.1.7. A2C No Delay

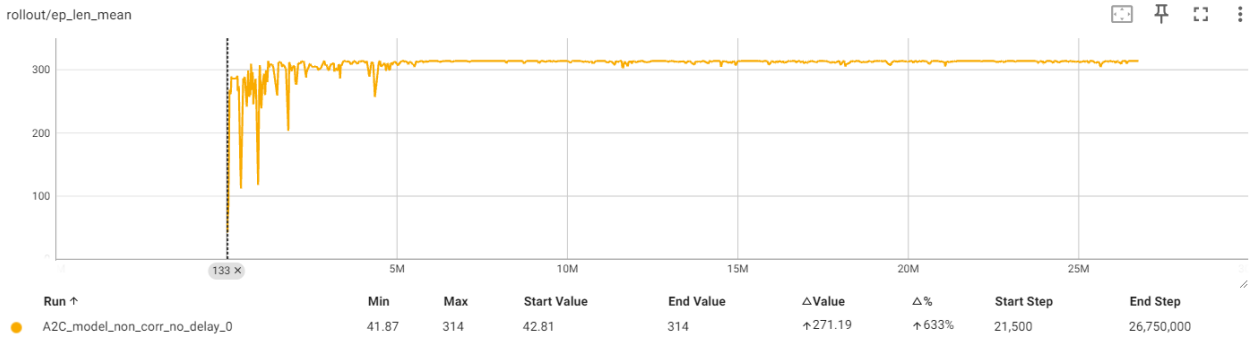


Figure 67 : Mean Episode Length per Timestep of A2C Non-Corrective No Delay Model

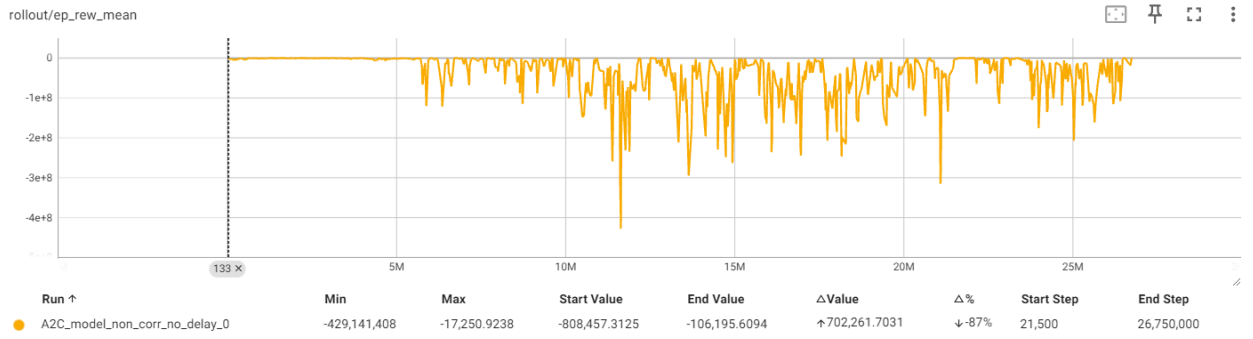


Figure 68 : Mean Episode Reward per Timestep of A2C Non-Corrective No Delay Model

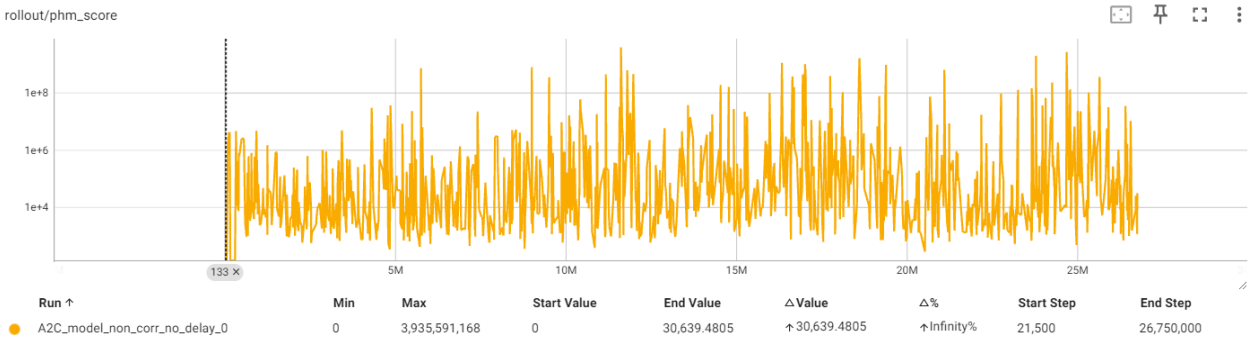


Figure 69 : PHM Score (Log Scale) per Timestep of A2C Non-Corrective No Delay Model

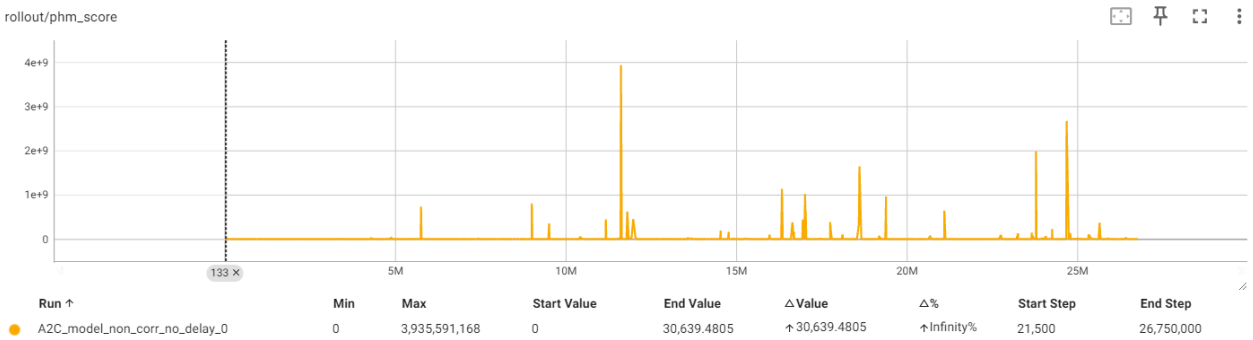


Figure 70 : PHM Score per Timestep of A2C Non-Corrective No Delay Model

The A2C Non-Corrective No delay model demonstrates rapid increases in mean episode length from 42.81 to the maximum value of 314, which is maintained throughout the rest of the training indicating quick learning and the ability to sustain long episodes. However, the mean episode reward shows a significant decrease with values dropping very low, suggesting convergence to a suboptimal policy with large negative rewards. Additionally, the PHM score graph fluctuates significantly. Overall, the model shows initial rapid learning in episode length but fails to optimize rewards and maintain stable health metrics, highlighting significant limitations in its learning process.

4.5.1.8. A2C With Delay

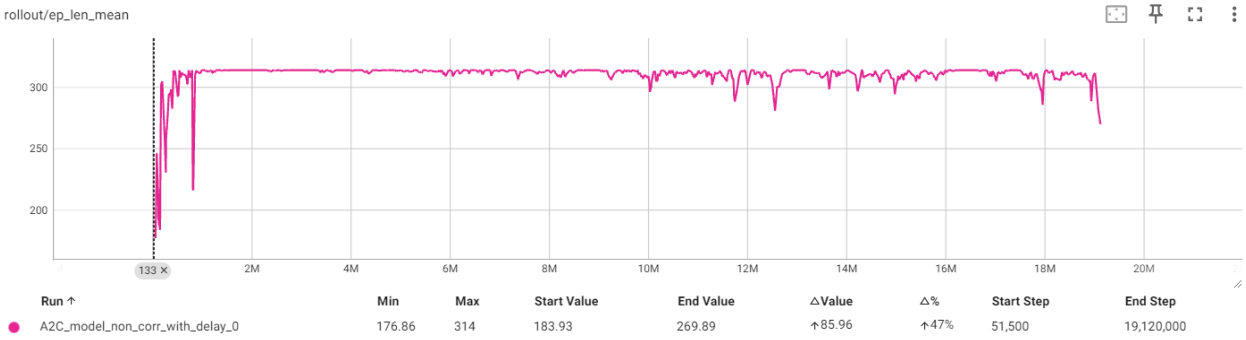


Figure 71 : Mean Episode Length per Timestep of A2C Non-Corrective With Delay Model

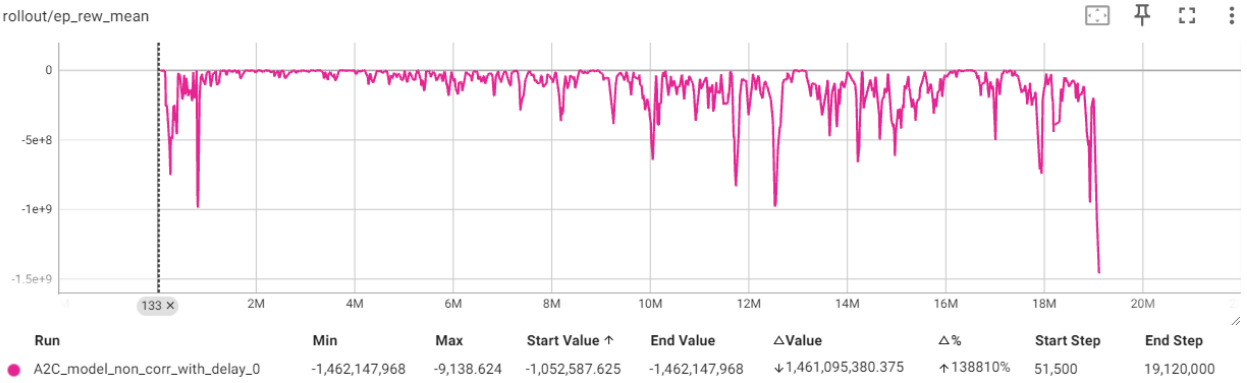
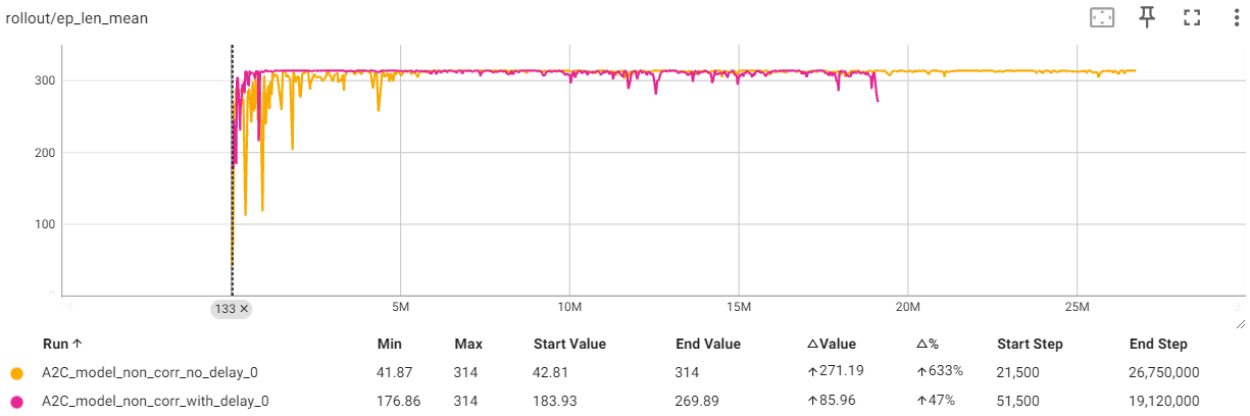


Figure 72 : Mean Episode Reward per Timestep of A2C Non-Corrective With Delay Model

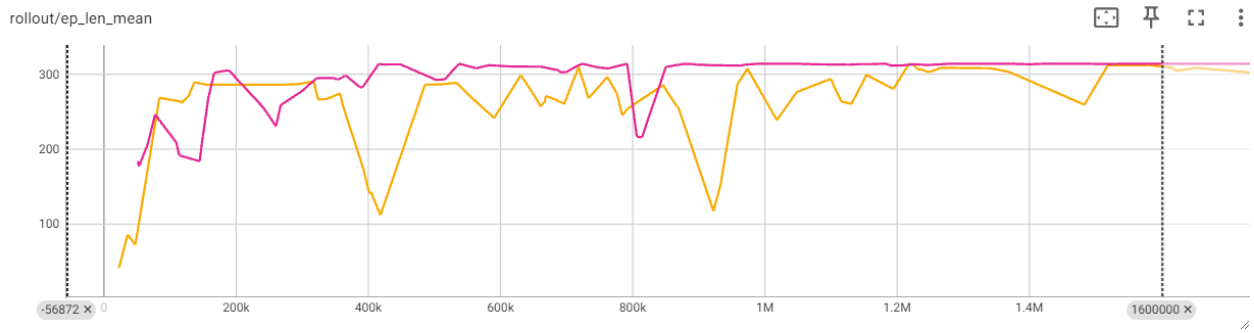
Figure 73 : Phm Score per Timestep of A2C Non-Corrective With Delay Model

The A2C Non-Corrective With Delay model reaches the max episode length quickly and then has slight fluctuations which are seen as big downwards spikes in the reward graph. During the severe declines in the mean episode reward, the PHM score graph fluctuates dramatically. Overall, even though the model maintains long episodes, it cannot improve its rewards and that showcases poor learning process.



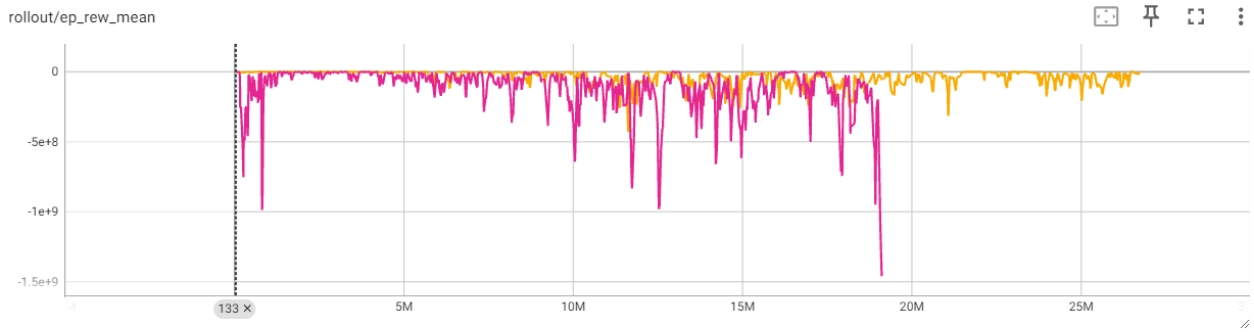
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	41.87	314	42.81	314	↑271.19	↑633%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	176.86	314	183.93	269.89	↑85.96	↑47%	51,500	19,120,000

Figure 74 : Comparison of Mean Episode Length per Timestep of A2C Non-Corrective With Delay and No Delay Models



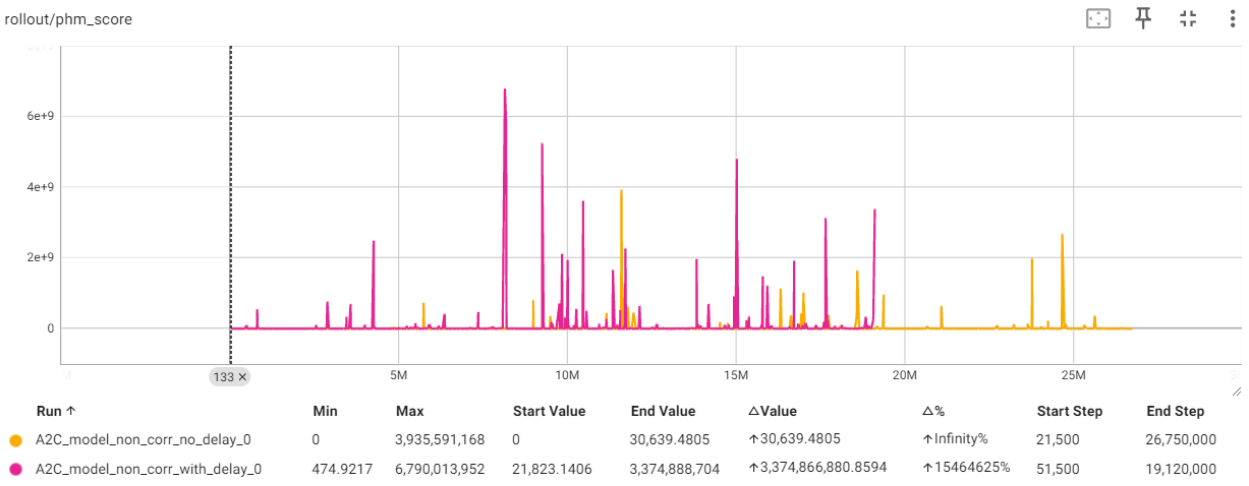
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	41.87	314	42.81	308.94	↑266.13	↑622%	21,500	1,615,500
A2C_model_non_corr_with_delay_0	176.86	314	183.93	314	↑130.07	↑71%	51,500	1,611,000

Figure 75 : Shorter Range Selection of the Comparison of Mean Episode Length per Timestep of A2C Non-Corrective With Delay and No Delay Models



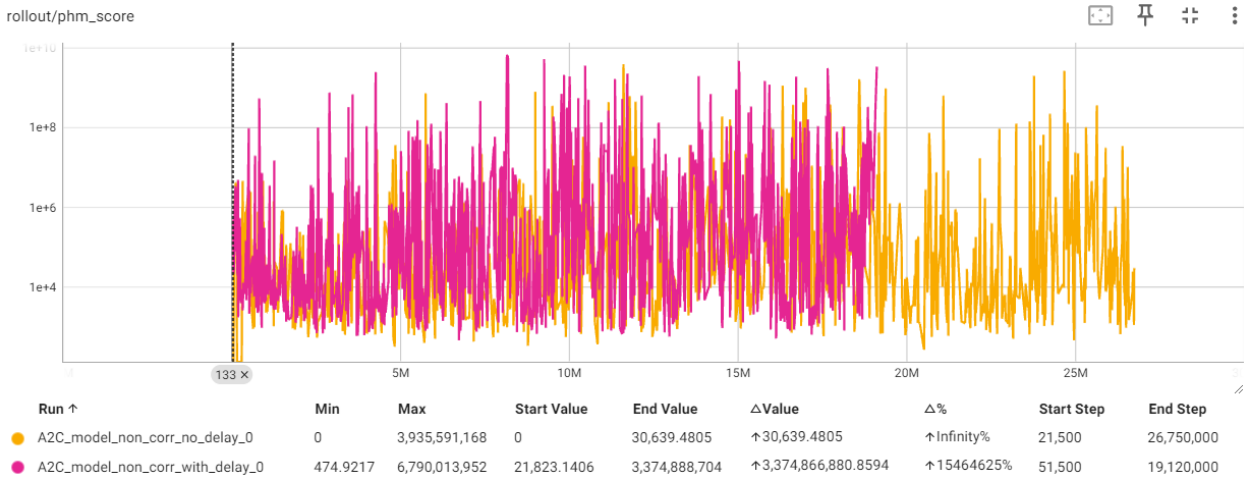
Run	Min	Max	Start Value ↑	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_with_delay_0	-1,462,147,968	-9,138.624	-1,052,587.625	-1,462,147,968	↓1,461,095,380.375	↑138810%	51,500	19,120,000
A2C_model_non_corr_no_delay_0	-429,141,408	-17,250.9238	-808,457.3125	-106,195.6094	↑702,261.7031	↓-87%	21,500	26,750,000

Figure 76 : Comparison of Phm Score per Timestep of A2C Non-Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	3,935,591,168	0	30,639.4805	↑30,639.4805	↑Infinity%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	474.9217	6,790,013,952	21,823.1406	3,374,888,704	↑3,374,866,880.8594	↑15464625%	51,500	19,120,000

Figure 77 : Comparison of PHM Score per Timestep of A2C Non-Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	3,935,591,168	0	30,639.4805	↑30,639.4805	↑Infinity%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	474.9217	6,790,013,952	21,823.1406	3,374,888,704	↑3,374,866,880.8594	↑15464625%	51,500	19,120,000

Figure 78 : Comparison of PHM Score (Log Scale) per Timestep of A2C Non-Corrective With Delay and No Delay Models

The A2C models are the second fastest Non-Corrective models to converge after only the DDPG No Delay. The no delay model initially shows significant variance in mean episode length but stabilizes at the maximum value of 314, indicating that it eventually learns to maximize episode duration although slower. Its mean episode reward graph, however, reveals large negative rewards, reflecting high variance and instability throughout the training process. Also, the corresponding PHM score graph shows substantial variability, indicating that the model often deviates from optimal behavior. On the contrary, the with delay model, while showing similar initial fluctuations, converges more steadily and faster to a high mean episode length and exhibits less reward instability, as seen in the smoother reward graph. This model's Phm score also indicates more consistent performance with fewer extreme values compared to the no delay model. Together, these observations suggest that incorporating delay improves the stability and performance of the A2C algorithm, although it still faces challenges in minimizing the Phm score.

4.5.2. Corrective Environments

The Mean Episode Length Evaluation Metric consistently reaches the maximum value of 314 across all models, except for the A2C "no delay" corrective model. Therefore, displaying the graph of the mean

episode length for all other models is unnecessary, as it would simply show a straight line at the 314 value for every timestep. A complete graph of Mean Episode Length will be presented in the Comparative Analysis of Every Algorithm in the Corrective Environments section.

4.5.2.1 PPO No Delay

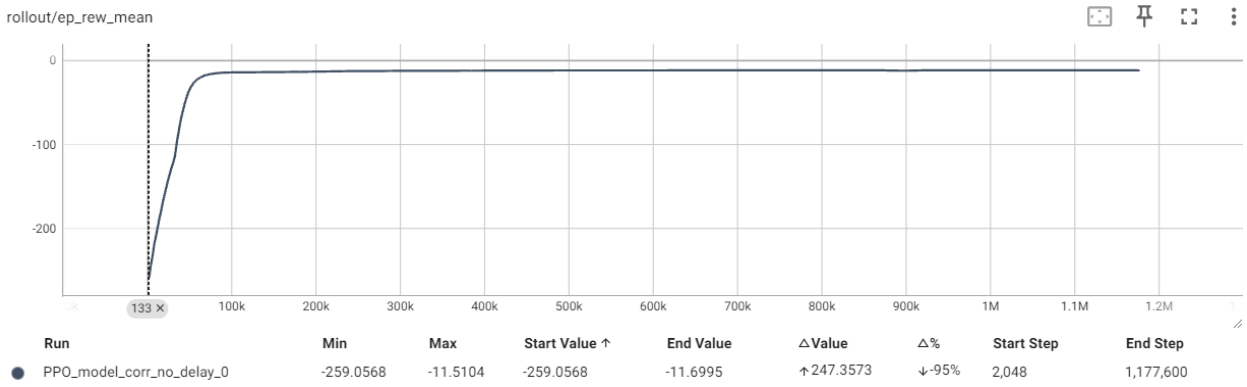


Figure 79 : Mean Episode Reward per Timestep of PPO Corrective No Delay Model

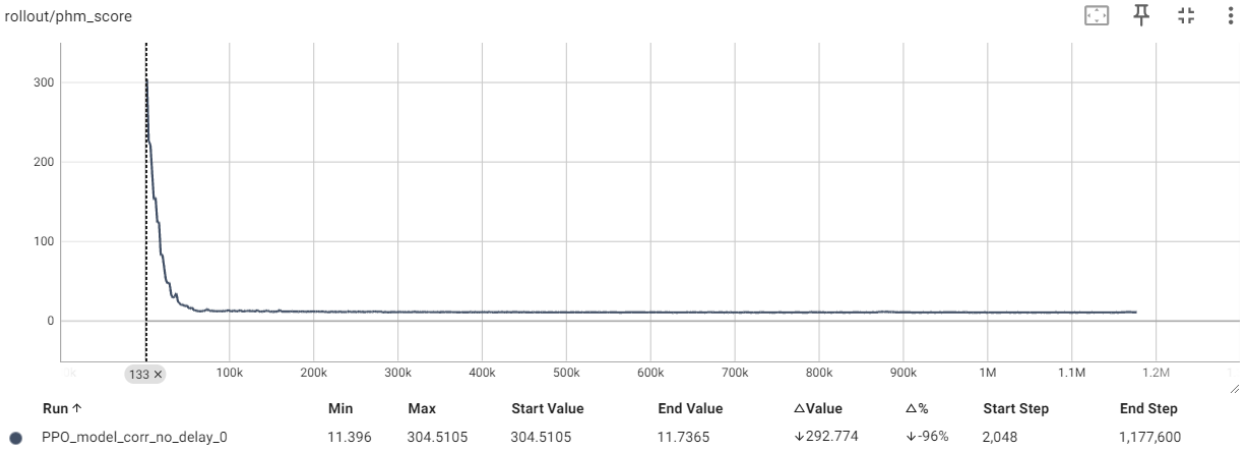


Figure 80 : PHM Score per Timestep of PPO Corrective No Delay Model

The model converges quickly into the max episode length as seen in the first graph, with a significant increase observed within the first 100,000 steps. The mean episode reward increases by 95% from its initial value, indicating successful training. The PHM Score also follows a similar trend, rapidly decreasing to a near-zero value within the same timeframe, which aligns with the reduction in variance and stabilization of the policy. This behavior highlights the model's efficiency in reaching optimal performance quickly and reliably.

4.5.2.2. PPO With Delay

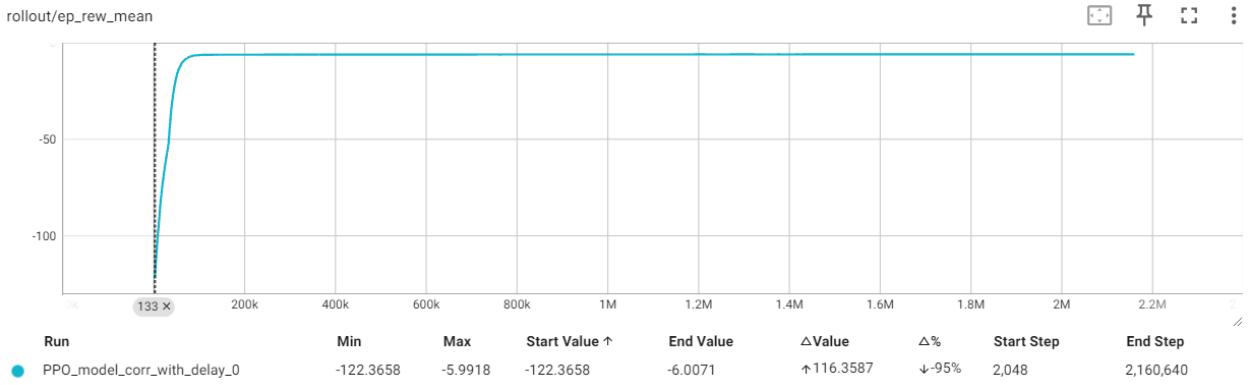


Figure 81 : Mean Episode Reward per Timestep of PPO Corrective With Delay Model

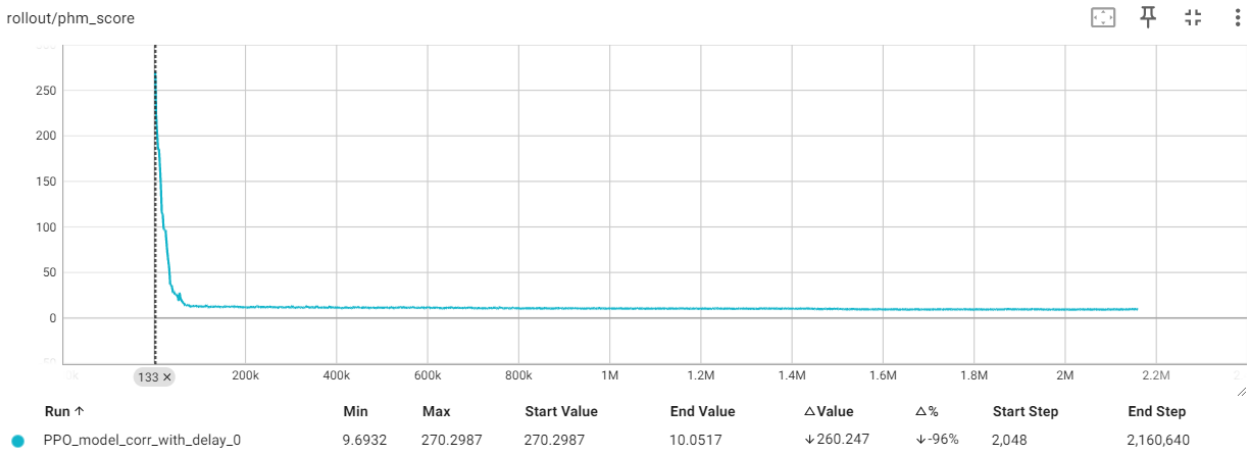


Figure 82 : PHM Score per Timestep of PPO Corrective With Delay Model

Similarly to the No Delay, the PPO Corrective With Delay model demonstrates a rapid increase in mean episode reward, reaching stability early and maintaining it throughout the training period. The reward improves from -122.3 to -6.0, representing a positive change of +116.3 (95%). The PHM score also shows a significant reduction from its peak, settling at around 10.05 with a decrease of 260 (96%).

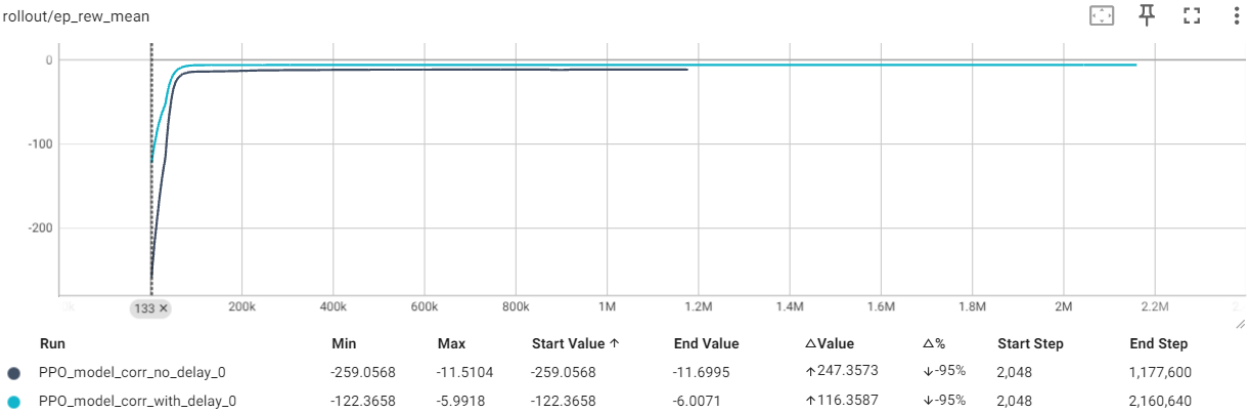
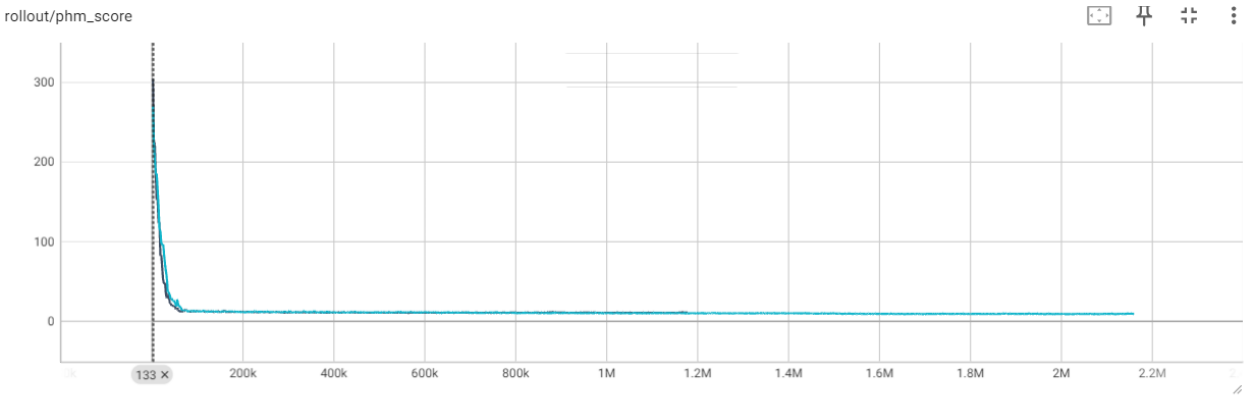
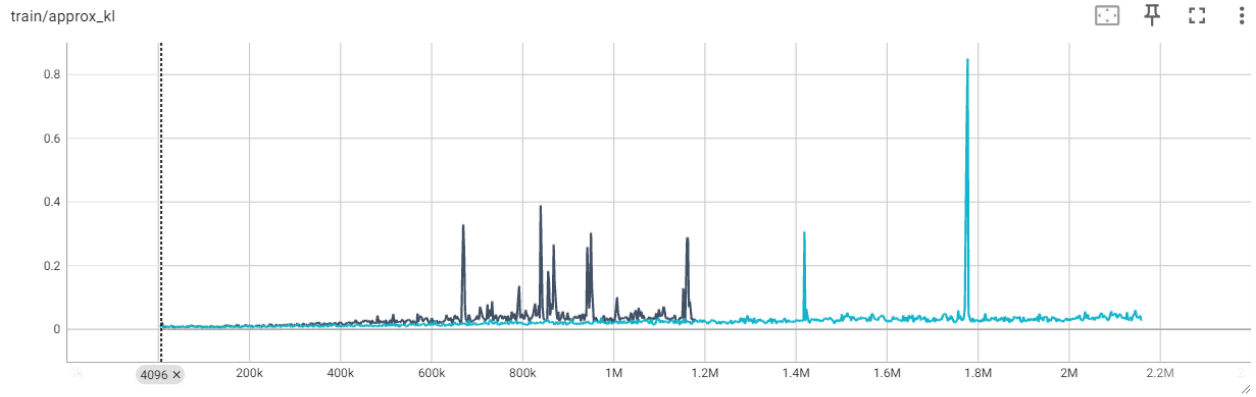


Figure 83 : Comparison of Mean Episode Reward per Timestep of PPO Corrective With Delay and No Delay Models



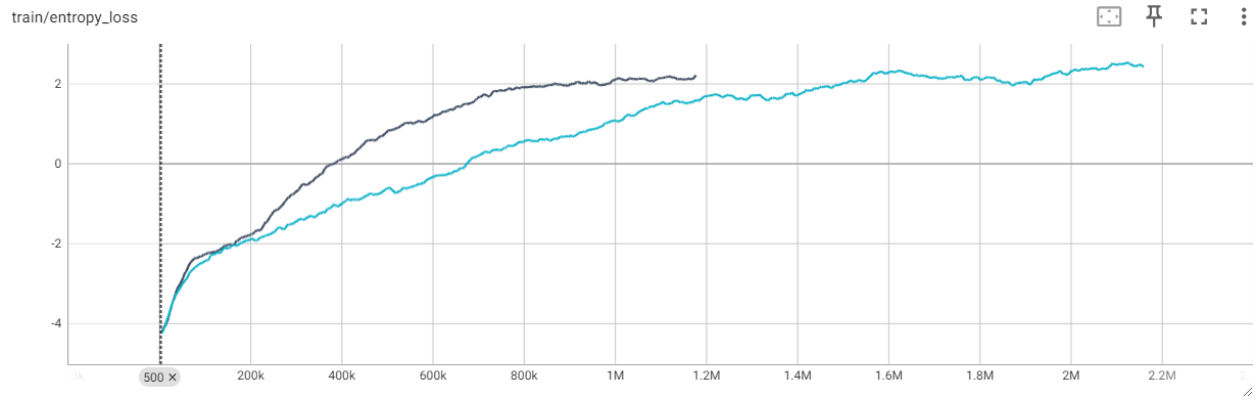
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
PPO_model_corr_no_delay_0	11.396	304.5105	304.5105	11.7365	↓292.774	↓-96%	2,048	1,177,600
PPO_model_corr_with_delay_0	9.6932	270.2987	270.2987	10.0517	↓260.247	↓-96%	2,048	2,160,640

Figure 84 : Comparison of PHM Score per Timestep of PPO Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
PPO_model_corr_no_delay_0	0.0072	0.3907	0.015	0.0358	↑0.0208	↑139%	4,096	1,177,600
PPO_model_corr_with_delay_0	0.0059	0.8503	0.0097	0.0322	↑0.0225	↑233%	4,096	2,160,640

Figure 85 : Comparison of Approximate KL Divergence per Timestep of PPO Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
PPO_model_corr_no_delay_0	-4.2255	2.2164	-4.2255	2.2164	↑6.4419	↓-152%	4,096	1,177,600
PPO_model_corr_with_delay_0	-4.2269	2.5303	-4.2269	2.4326	↑6.6595	↓-158%	4,096	2,160,640

Figure 86 : Comparison of Entropy Loss per Timestep of PPO Corrective With Delay and No Delay Models

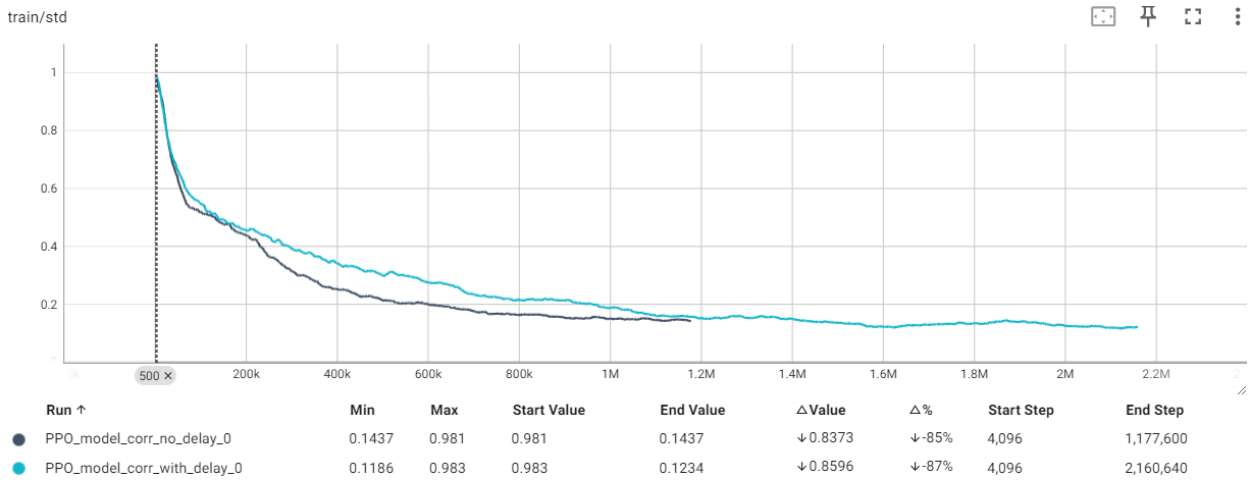


Figure 87 : Comparison of Standard Deviation per Timestep of PPO Corrective With Delay and No Delay Models

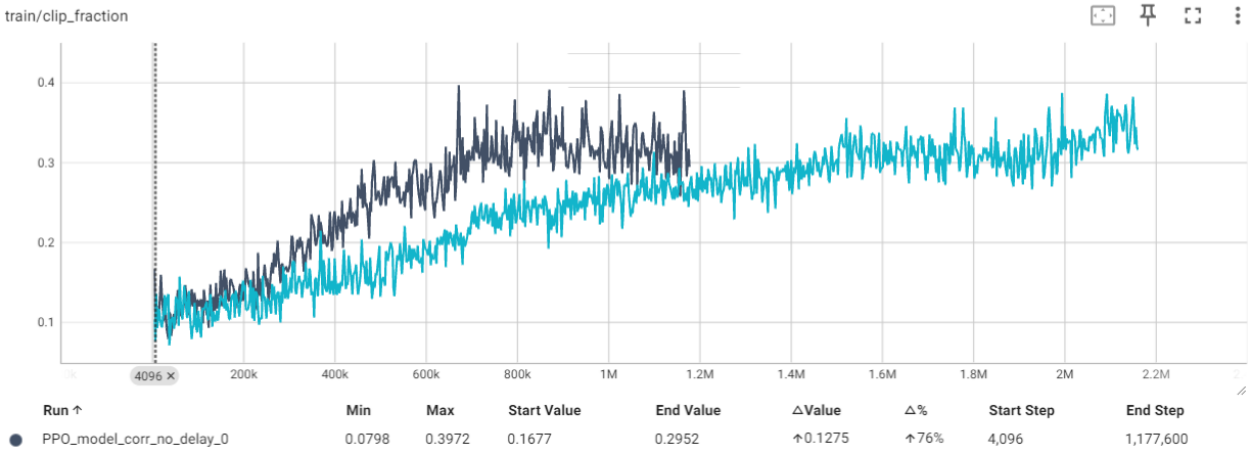


Figure 88 : Comparison of Clip Fraction per Timestep of PPO Corrective With Delay and No Delay Models

Both models exhibit efficient learning, quickly stabilizing their rewards and PHM scores, indicating successful convergence and effective training in the corrective environment. The only slight difference is that the with delay model is ever so slightly faster in converging.

4.5.2.3. SAC No Delay

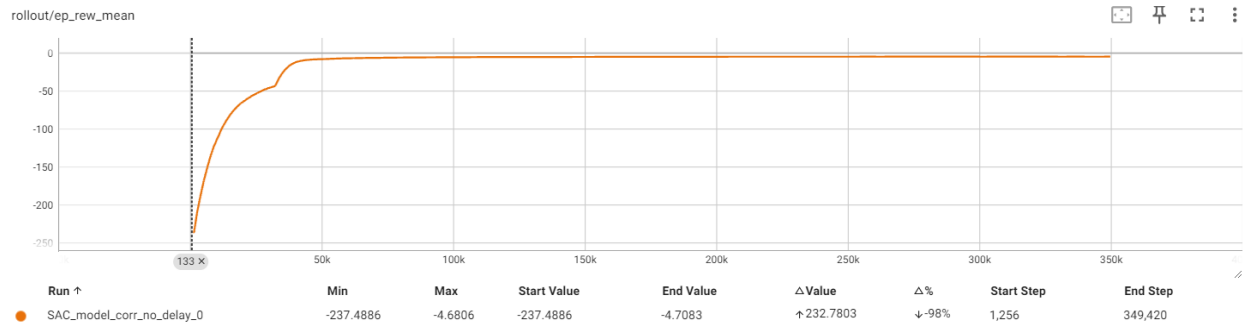


Figure 89 : Mean Episode Reward per Timestep of SAC Corrective No Delay Model

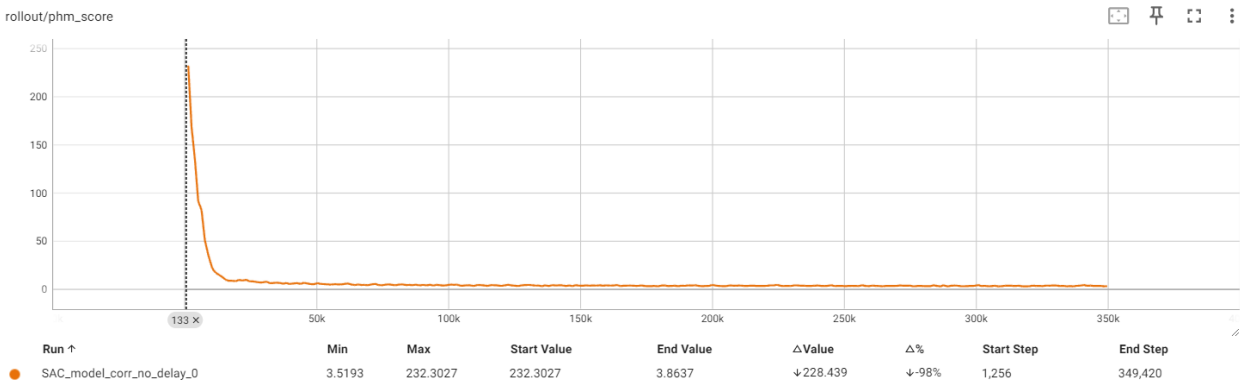


Figure 90 : PHM Score per Timestep of SAC Corrective No Delay Model

The model demonstrates a rapid convergence in terms of mean episode reward, with a significant increase up to around 237.4. The reward stabilizes quickly, indicating efficient learning. The PHM score also shows a rapid decrease. Overall, this model shows effective and stable performance throughout the training period.

4.5.2.4. SAC With Delay

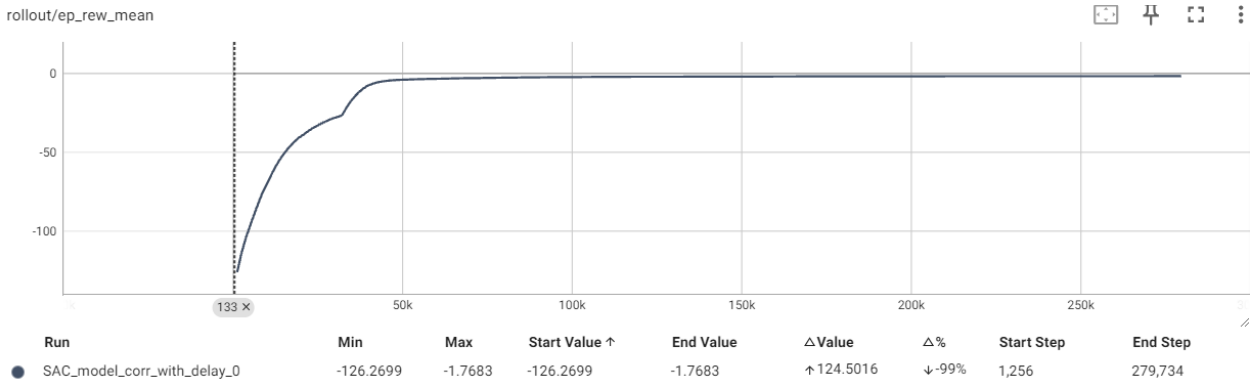


Figure 91 : Mean Episode Reward per Timestep of SAC Corrective With Delay Model

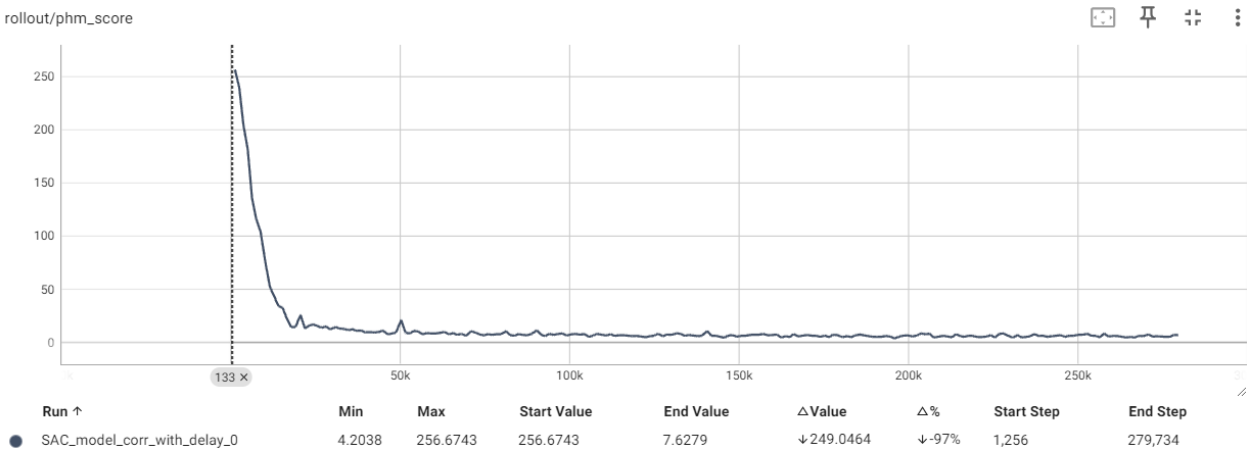
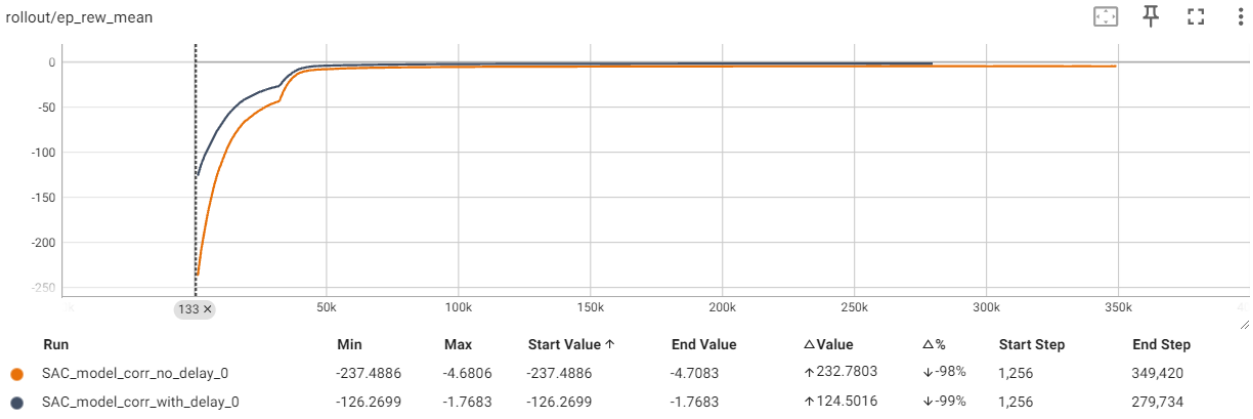


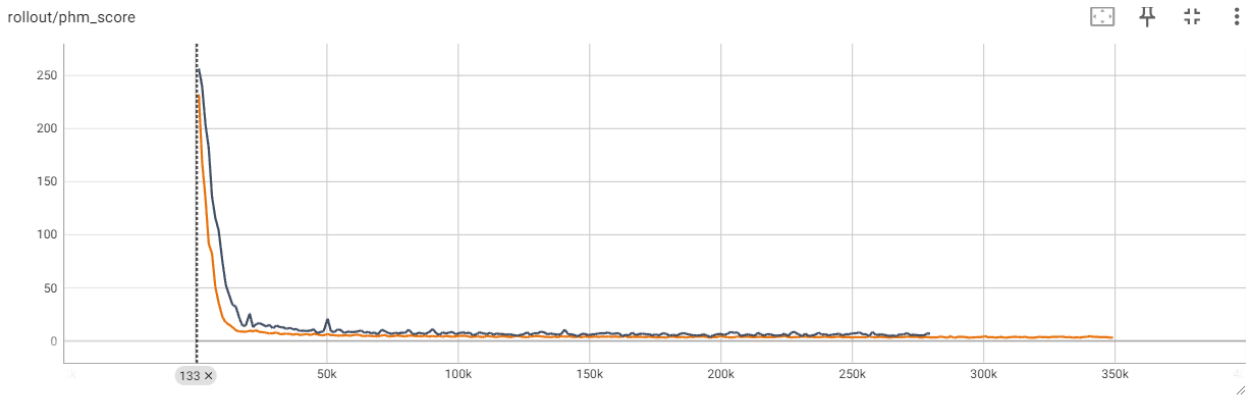
Figure 92 : PHM Score per Timestep of SAC Corrective With Delay Model

The SAC Corrective With Delay model, shows a faster increase in the mean episode length per timestep, but a slower and more gradual increase in mean episode reward, peaking at around -1.76 (perfect score). The learning process appears to be slowed by the delay, resulting in slightly less efficient learning and lower overall rewards compared to the no delay model. The PHM score similarly decreases but stabilizes at a higher level of uncertainty than the no delay model, indicating less efficient convergence.



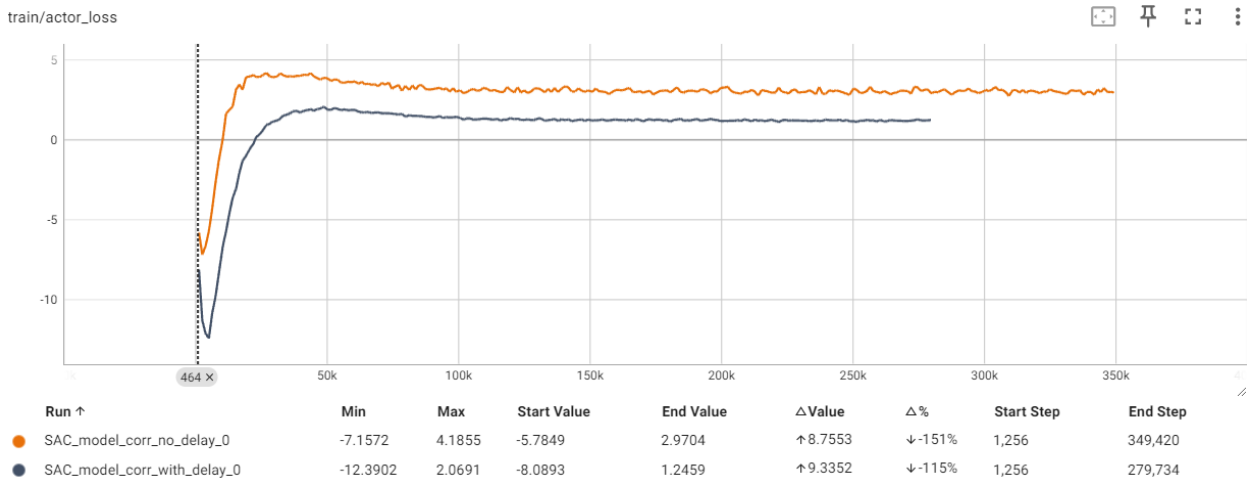
Run	Min	Max	Start Value ↑	End Value	△Value	△%	Start Step	End Step
SAC_model_corr_no_delay_0	-237.4886	-4.6806	-237.4886	-4.7083	↑232.7803	↓-98%	1,256	349,420
SAC_model_corr_with_delay_0	-126.2699	-1.7683	-126.2699	-1.7683	↑124.5016	↓-99%	1,256	279,734

Figure 93 : Comparison of Mean Episode Reward per Timestep of SAC Corrective With Delay and No Delay Models



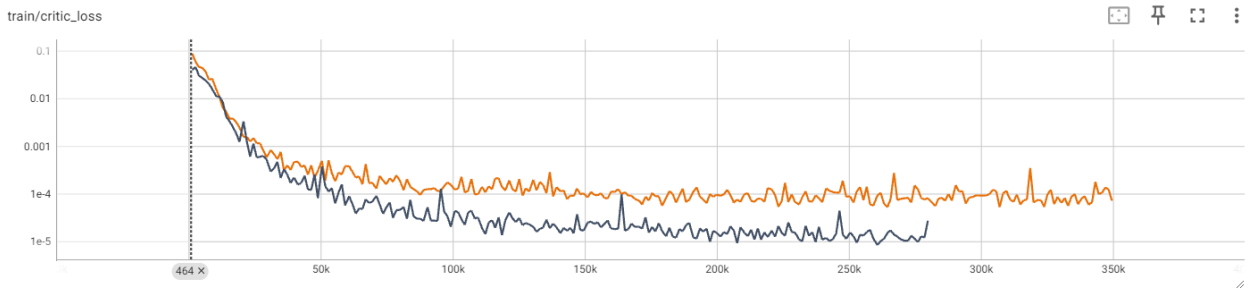
Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_corr_no_delay_0	3.5193	232.3027	232.3027	3.8637	↓228.439	↓-98%	1,256	349,420
SAC_model_corr_with_delay_0	4.2038	256.6743	256.6743	7.6279	↓249.0464	↓-97%	1,256	279,734

Figure 94 : Comparison of PHM Score per Timestep of SAC Corrective With Delay and No Delay Models



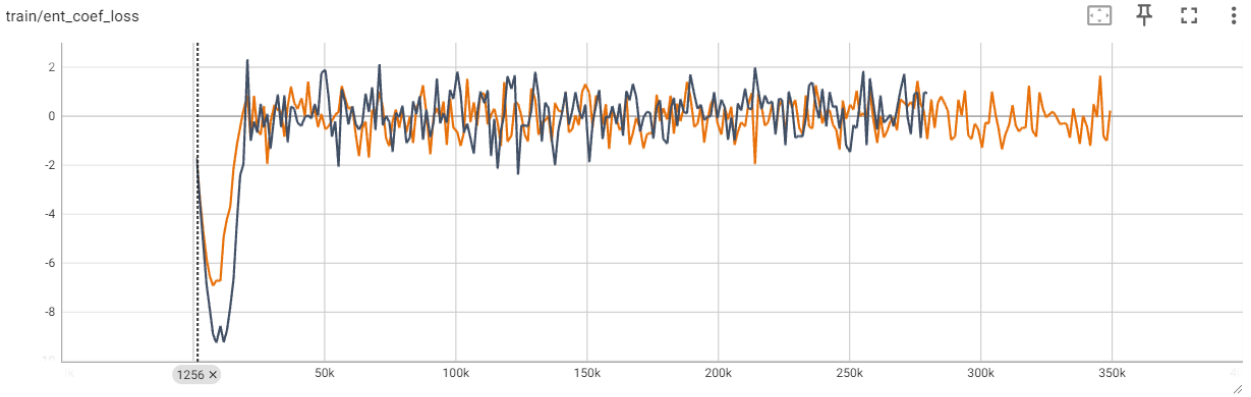
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_corr_no_delay_0	-7.1572	4.1855	-5.7849	2.9704	↑8.7553	↓-151%	1,256	349,420
SAC_model_corr_with_delay_0	-12.3902	2.0691	-8.0893	1.2459	↑9.3352	↓-115%	1,256	279,734

Figure 95 : Comparison of Actor Loss per Timestep of SAC Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_corr_no_delay_0	0.0001	0.0879	0.0879	0.0001	↓0.0879	↓-100%	1,256	349,420
SAC_model_corr_with_delay_0	0	0.0462	0.0402	0	↓0.0401	↓-100%	1,256	279,734

Figure 96 : Comparison of Critic Loss (Log Scale) per Timestep of SAC Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
SAC_model_corr_no_delay_0	-6.9027	1.6514	-1.6836	0.233	↑1.9166	↓-114%	1,256	349,420
SAC_model_corr_with_delay_0	-9.2221	2.3257	-1.7013	0.9541	↑2.6555	↓-156%	1,256	279,734

Figure 97 : Comparison of Entropy Coefficient Loss per Timestep of SAC Corrective With Delay and No Delay Models

Comparing the two models, it is evident that the No Delay model outperforms the With Delay model in terms of mean episode reward and PHM score. The No Delay model converges faster and maintains higher rewards. The With Delay model exhibits slower learning and higher uncertainty, highlighting the negative impact of delay on the SAC model's performance. In the Entropy Coefficient Loss graph both models exhibit fluctuation, which is due to the exploration-exploitation trade-off during learning. The No Delay model

initially shows a steeper increase in entropy coefficient loss but stabilizes faster, indicating a quicker adaptation to the optimal policy. In contrast, the With Delay model shows more prolonged fluctuations and a slower stabilization, suggesting that it takes longer to balance exploration and exploitation effectively. This further reinforces the observation that the delay impacts the efficiency and stability of the learning process in the SAC model.

4.5.2.5. DDPG No Delay

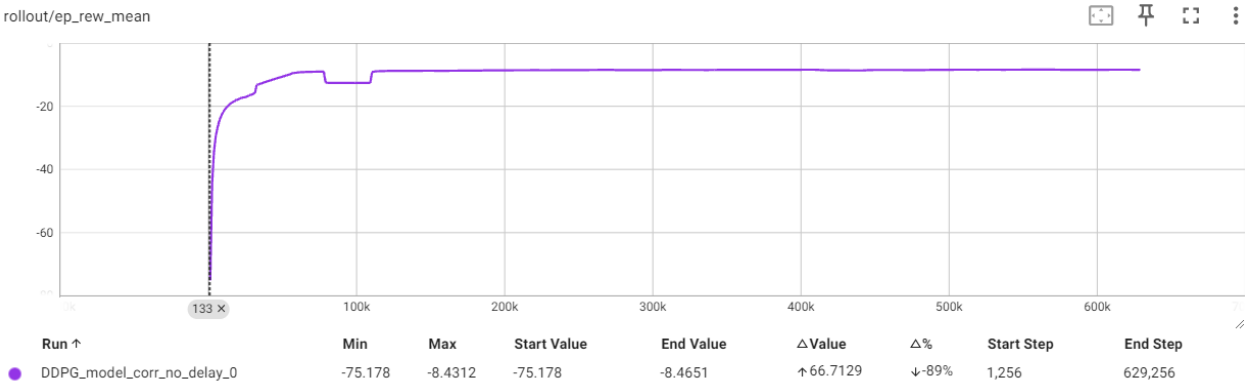


Figure 98 : Mean Episode Reward per Timestep of DDPG Corrective No Delay Model

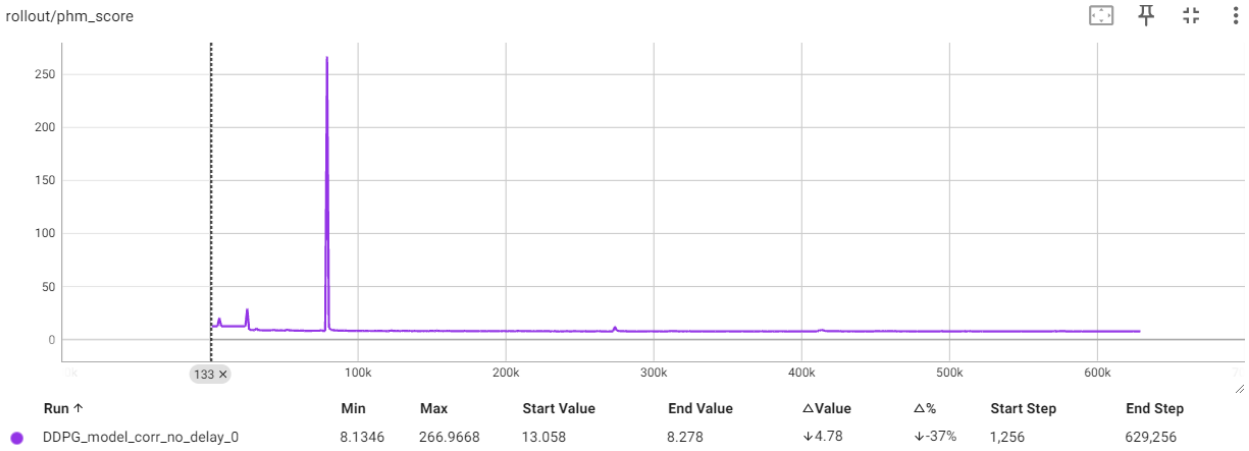


Figure 99 : PHM Score per Timestep of DDPG Corrective No Delay Model

The model exhibits a significant increase in mean episode reward, improving from -75.1 to -8.46, representing an 89% improvement. However, the PHM Score shows minimal improvement, with a change of -37%. This model manages to perform better over time but does not reach the optimal performance level of other models, as indicated by the relatively small decrease of PHM Score.

4.5.2.6. DDPG With Delay

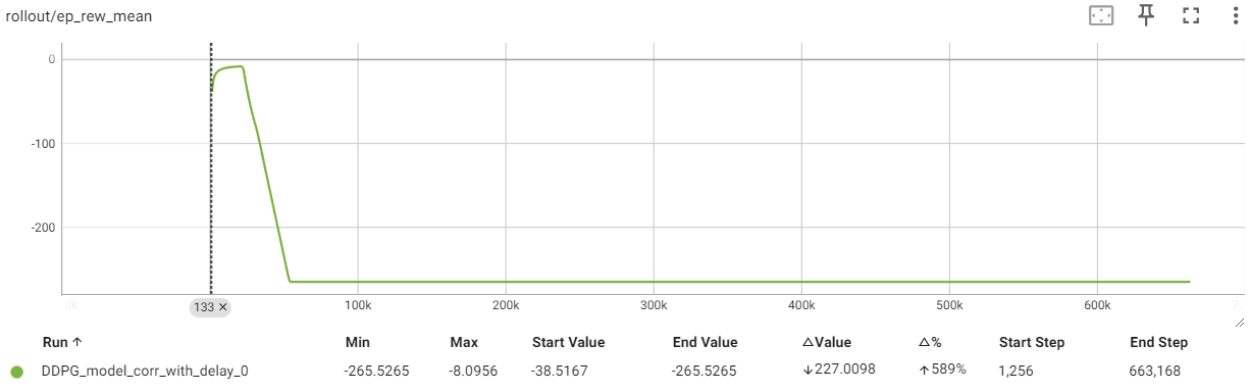


Figure 100 : Mean Episode Reward per Timestep of DDPG Corrective With Delay Model

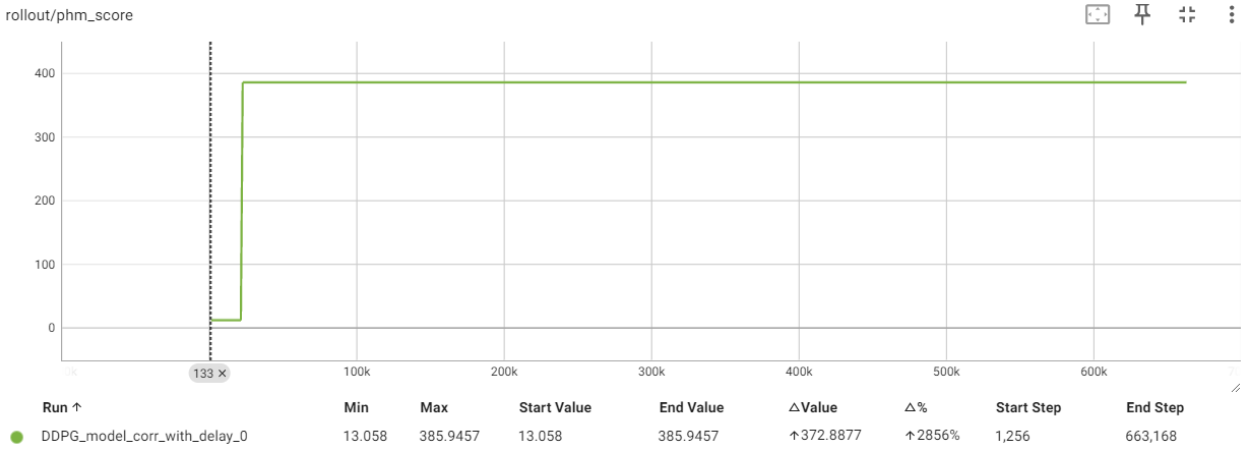


Figure 101 : PHM Score per Timestep of DDPG Corrective With Delay Model

The model demonstrates a severe drop in performance. The mean episode reward heavily decreases, and the PHM Score increases dramatically. This model appears to be stuck in a bad policy, leading to poor performance.

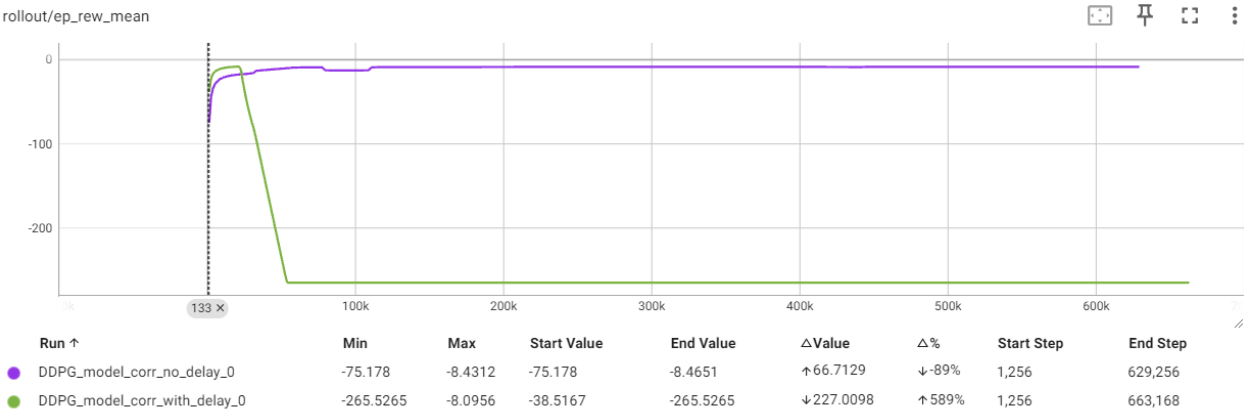
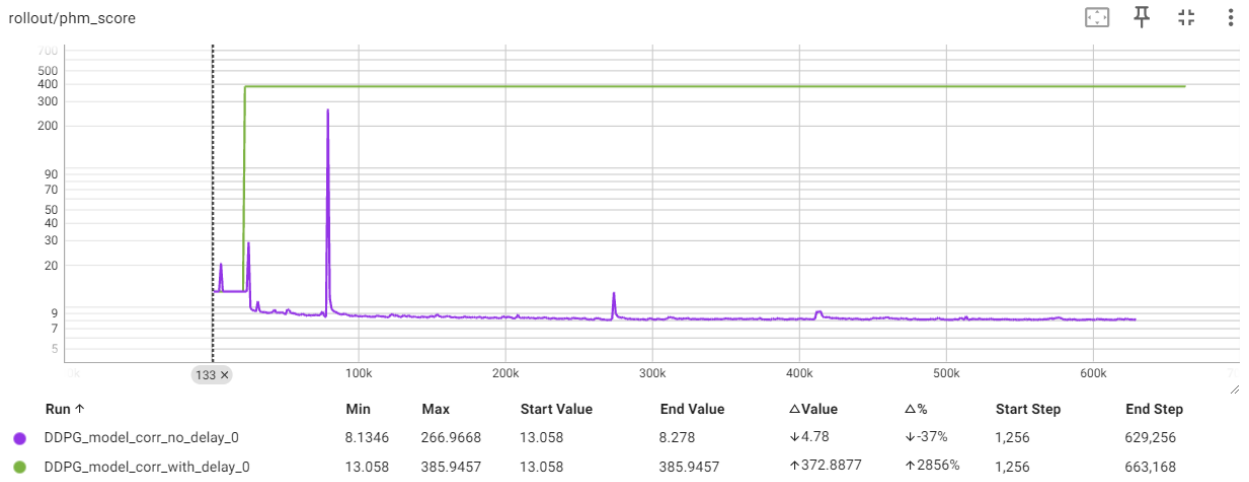
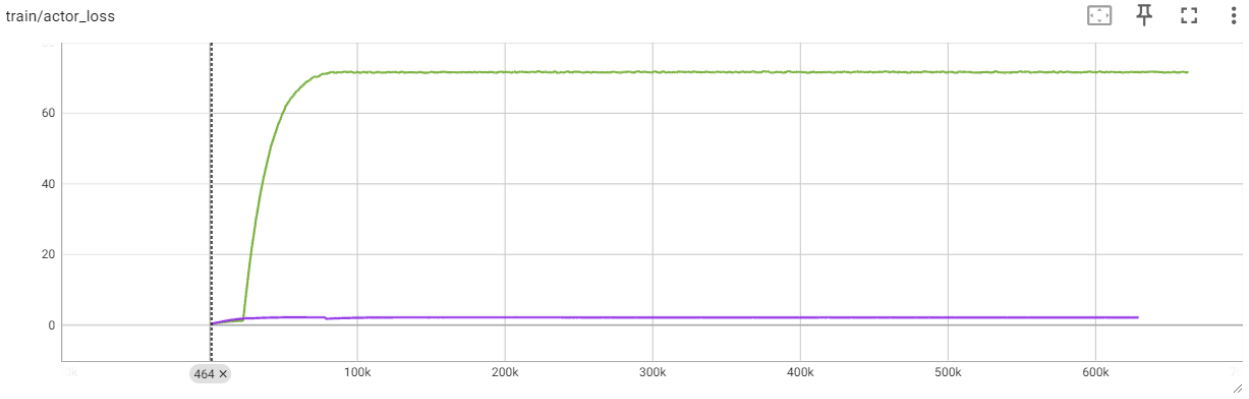


Figure 102 : Comparison of Mean Episode Reward per Timestep of A2C Corrective With Delay and No Delay Models



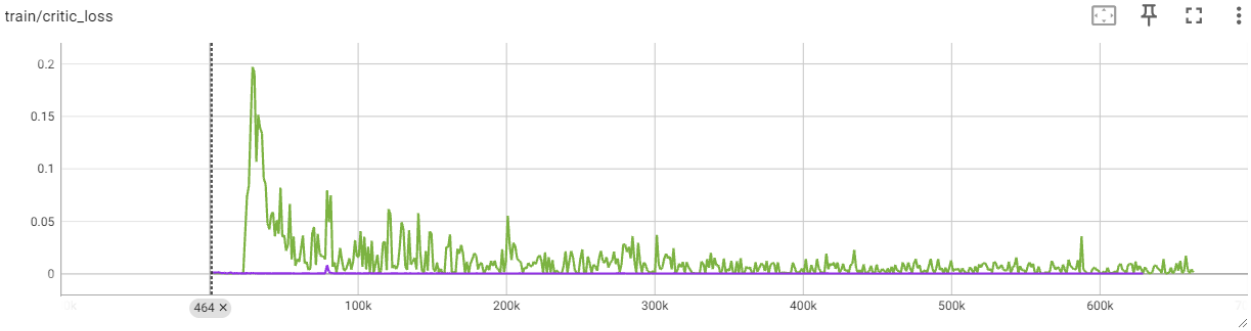
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_corr_no_delay_0	8.1346	266.9668	13.058	8.278	↓4.78	↓-37%	1,256	629,256
DDPG_model_corr_with_delay_0	13.058	385.9457	13.058	385.9457	↑372.8877	↑2856%	1,256	663,168

Figure 103 : Comparison of Phm Score per Timestep of A2C Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_corr_no_delay_0	0.4	2.3507	0.4	2.3151	↑1.9151	↑479%	1,256	629,256
DDPG_model_corr_with_delay_0	0.517	71.9847	0.517	71.5867	↑71.0697	↑13748%	1,256	663,168

Figure 104 : Comparison of Actor Loss per Timestep of A2C Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
DDPG_model_corr_no_delay_0	0	0.008	0.0007	0	↓0.0007	↓-99%	1,256	629,256
DDPG_model_corr_with_delay_0	0.0001	0.197	0.0008	0.0011	↑0.0003	↑38%	1,256	663,168

Figure 105 : Comparison of Critic Loss per Timestep of A2C Corrective With Delay and No Delay Models

Comparing the two DDPG corrective models, the No Delay model shows better performance with a higher mean episode reward improvement and a decent PHM Score. The With Delay model indicates poor performance overall with significant drops in mean episode rewards and increased actor loss variability.

The comparison highlights that the discounted rewards as the With Delay model is not good in a DDPG algorithm.

4.5.2.7. A2C No Delay

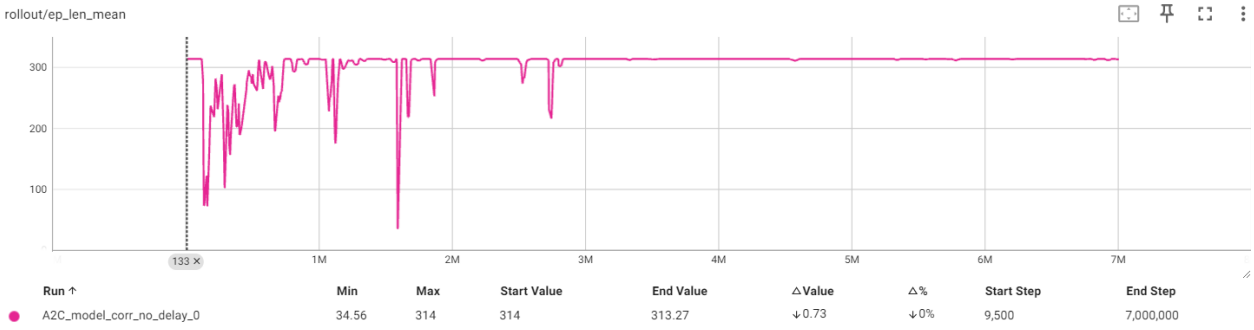


Figure 106 : Mean Episode Length per Timestep of A2C Corrective No Delay Model

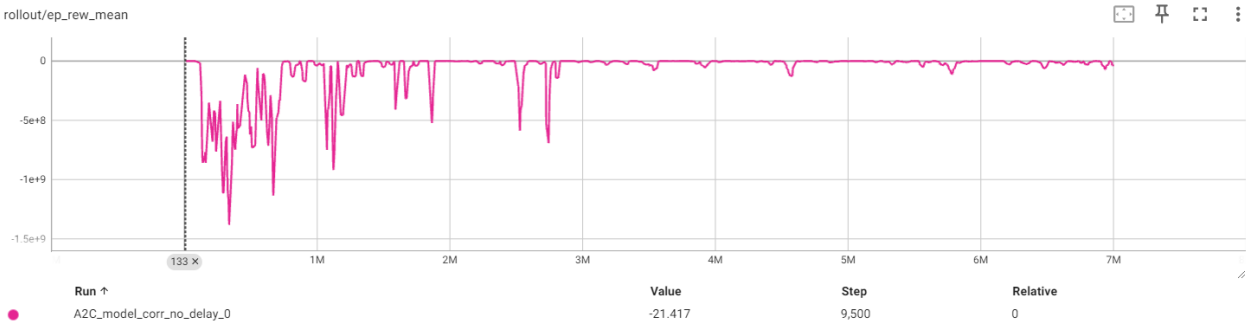


Figure 107 : Mean Episode Reward per Timestep of A2C Corrective No Delay Model

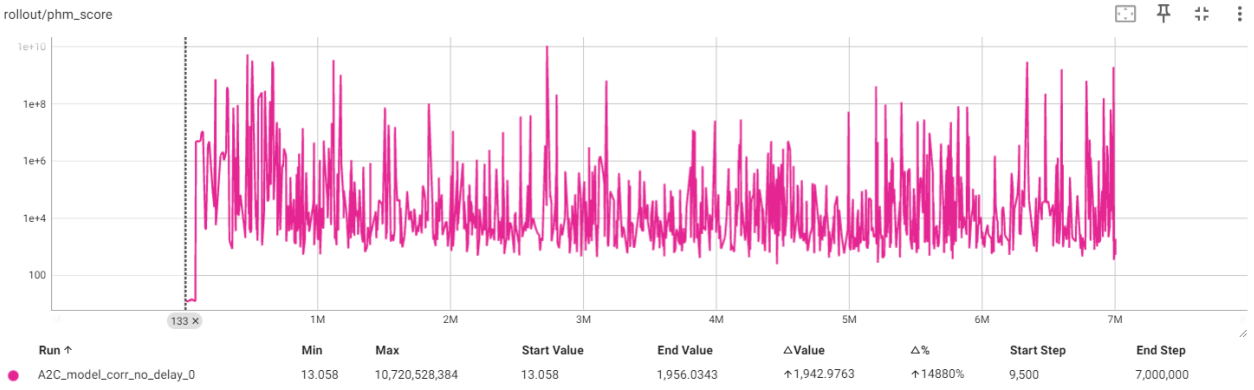


Figure 108 : PHM Score per Timestep of A2C Corrective No Delay Model

The A2C Corrective No Delay model is the only Corrective model that does not reach the max episode length on the first few timesteps. It shows significant fluctuations in the mean episode length initially, stabilizing after around three million timesteps. The mean episode reward graph indicates substantial variations in reward values and the Phm score also demonstrates considerable variability, reflecting a challenging learning environment.

4.5.2.8. A2C With Delay

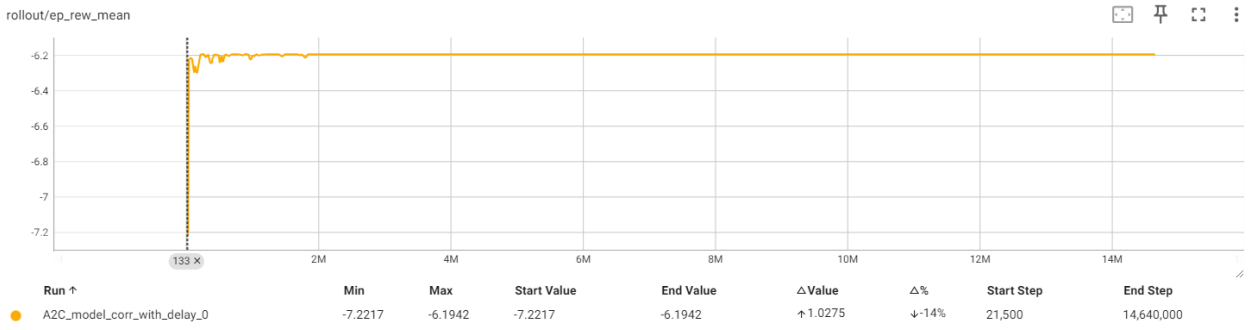


Figure 109 : Mean Episode Reward per Timestep of A2C Corrective With Delay Model

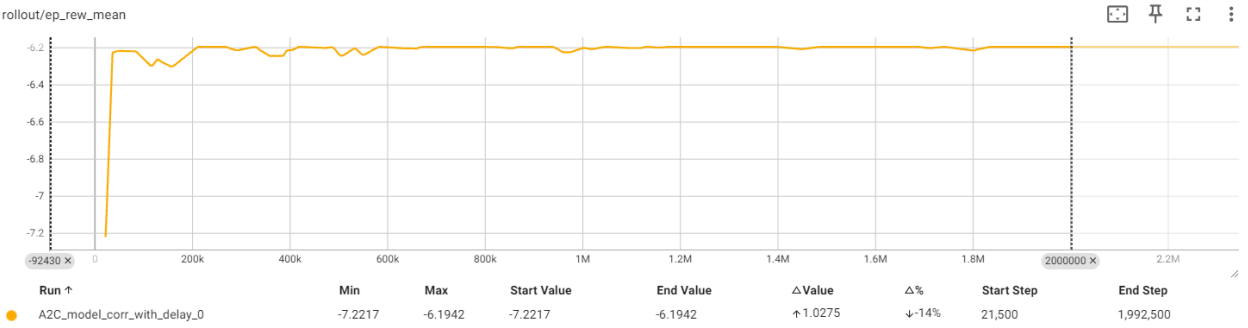


Figure 110 : Shorter Range Selection of Mean Episode Reward per Timestep of A2C Corrective With Delay Model

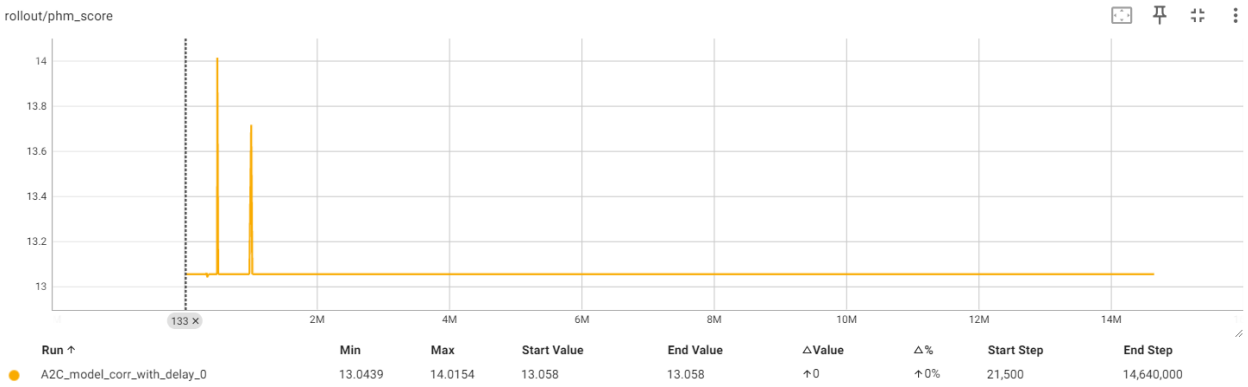
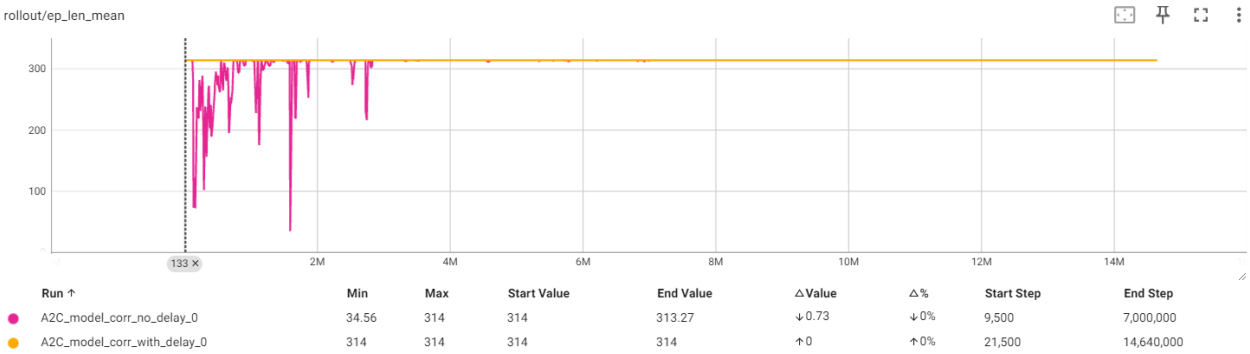


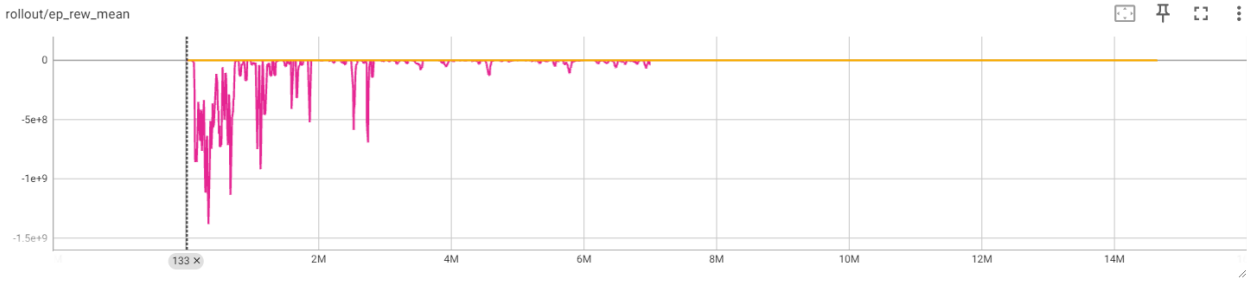
Figure 111 : PHM Score per Timestep of A2C Corrective With Delay Model

The A2C Corrective With Delay model shows a consistent mean episode length near the maximum limit, with minor early fluctuations. The mean episode reward graph indicates a stable performance with minor improvements over time. The PHM score graph remains flat, indicating that the model did not significantly improve in this metric, but it still has a good score.



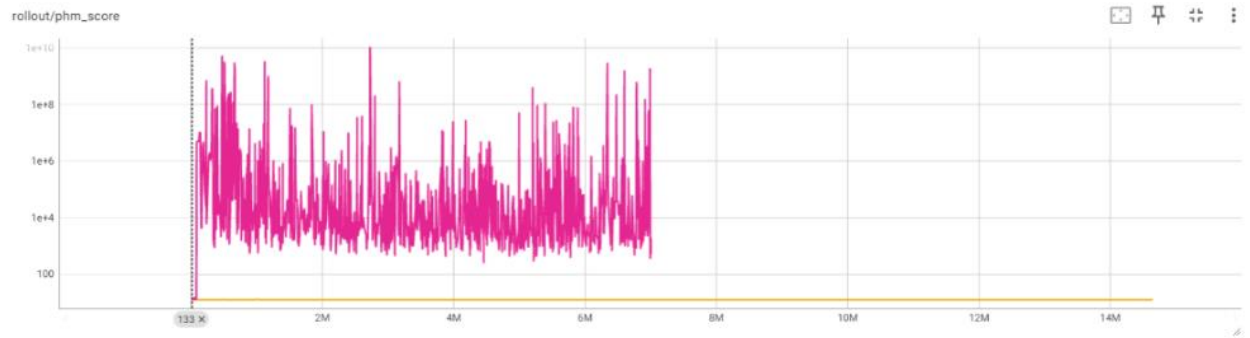
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_corr_no_delay_0	34.56	314	314	313.27	↓0.73	↓0%	9,500	7,000,000
A2C_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	21,500	14,640,000

Figure 112 : Comparison of Mean Episode Length per Timestep of A2C Corrective With Delay and No Delay Models



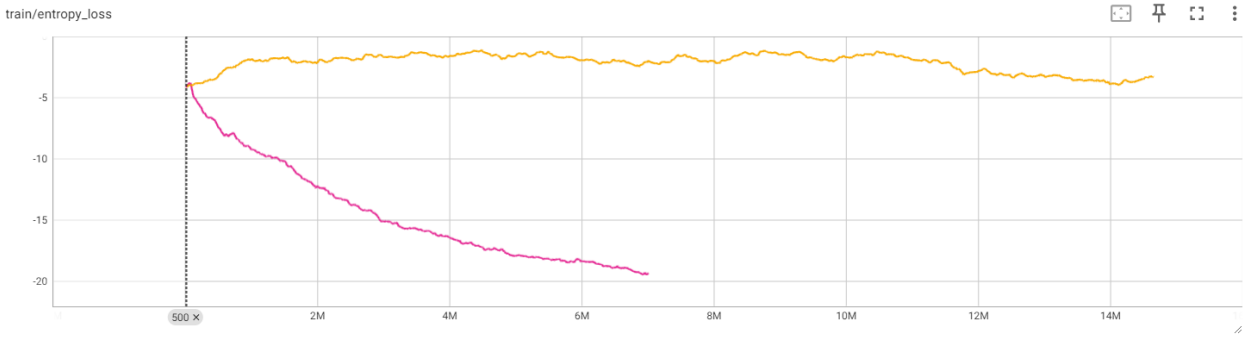
Run	Value	Step	Relative
A2C_model_corr_no_delay_0	-21.417	9,500	0
A2C_model_corr_with_delay_0	-7.2217	21,500	0

Figure 113 : Comparison of Mean Episode Reward per Timestep of A2C Corrective With Delay and No Delay Models



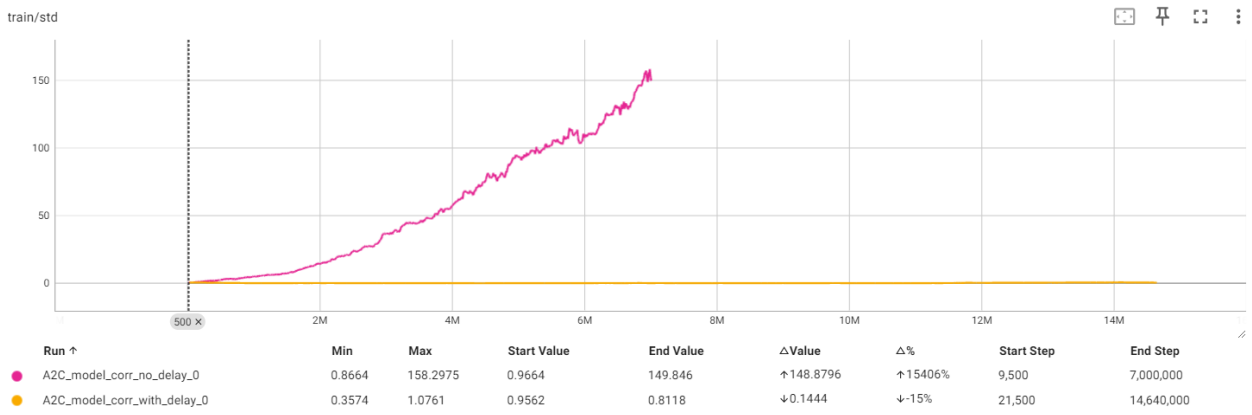
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_corr_no_delay_0	13.058	10,720,528,384	13.058	1,956.0343	↑1,942.9763	↑14880%	9,500	7,000,000
A2C_model_corr_with_delay_0	13.0439	14.0154	13.058	13.058	↑0	↑0%	21,500	14,640,000

Figure 114 : Comparison of Phm Score (Log Scale) per Timestep of A2C Corrective With Delay and No Delay Models



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_corr_no_delay_0	-19.4362	-3.8049	-4.1551	-19.2714	↓15.1163	↑364%	9,500	7,000,000
A2C_model_corr_with_delay_0	-4.1142	-1.1203	-4.1142	-3.2827	↑0.8315	↓-20%	21,500	14,640,000

Figure 115 : Comparison of Entropy Loss per Timestep of A2C Corrective With Delay and No Delay Models



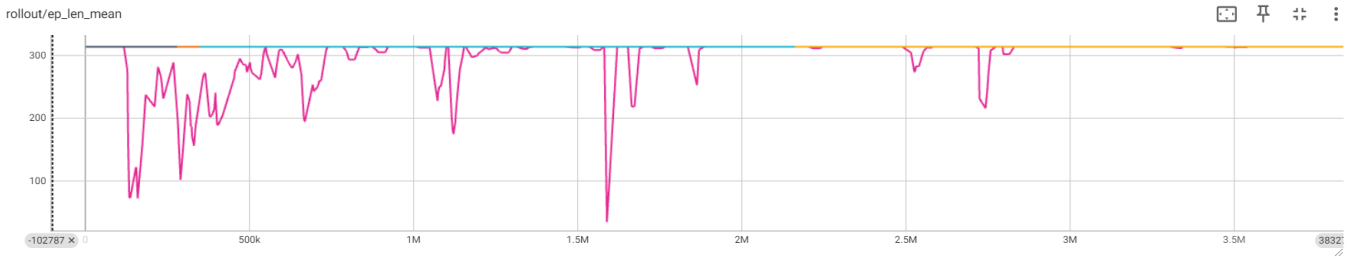
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_corr_no_delay_0	0.8664	158.2975	0.9664	149.846	↑148.8796	↑15406%	9,500	7,000,000
A2C_model_corr_with_delay_0	0.3574	1.0761	0.9562	0.8118	↓0.1444	↓-15%	21,500	14,640,000

Figure 116 : Comparison of Standard Deviation per Timestep of A2C Corrective With Delay and No Delay Models

Comparing the models above, we can see that the No Delay model demonstrates more significant fluctuations in both episode length and reward, suggesting a harder learning process. In contrast, the With Delay model shows stability but lacks significant improvement in reward values. The No Delay model has significantly higher Phm scores which rule it out as the best model since this score is the true metric of the implementation.

4.5.3. Comparative Analysis of All Algorithms

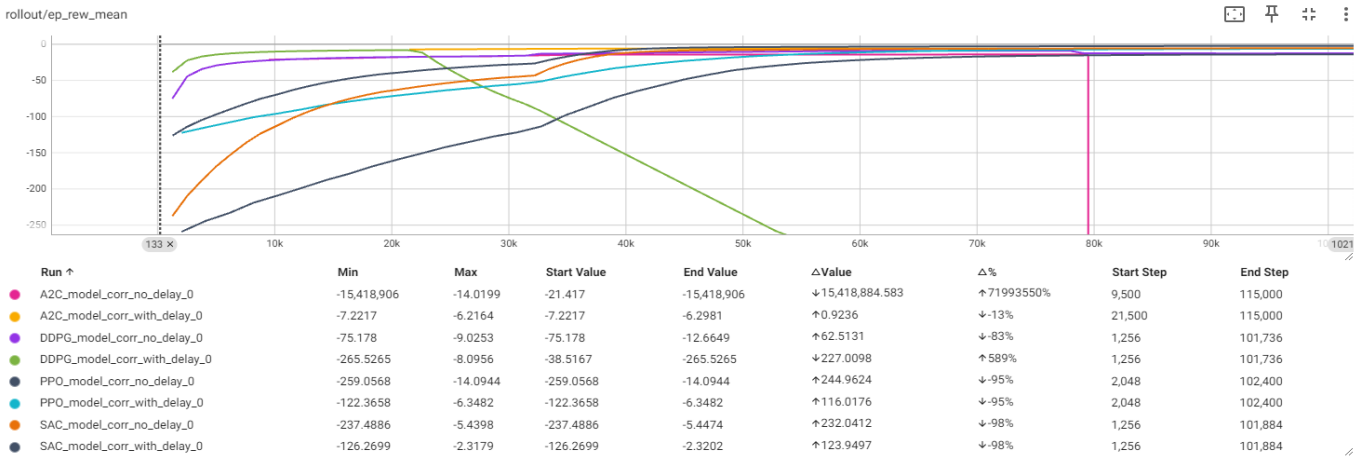
4.5.3.1 Corrective Environments



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_corr_no_delay_0	34.56	314	314	314	↑0	↑0%	9,500	3,831,000
A2C_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	21,500	3,822,500
DDPG_model_corr_no_delay_0	314	314	314	314	↑0	↑0%	1,256	629,256
DDPG_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	1,256	663,168
PPO_model_corr_no_delay_0	314	314	314	314	↑0	↑0%	2,048	1,177,600
PPO_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	2,048	2,160,640
SAC_model_corr_no_delay_0	314	314	314	314	↑0	↑0%	1,256	349,420
SAC_model_corr_with_delay_0	314	314	314	314	↑0	↑0%	1,256	279,734

Figure 117 : Mean Episode Length per Timestep for all Algorithms in the Corrective Environment With / No Delay

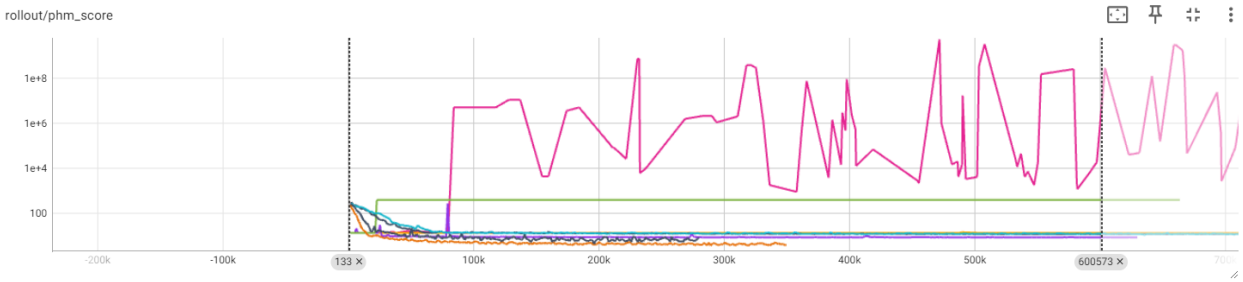
The graph shows the tendency of A2C “without delay” model to fluctuate significantly (Range Selection is around the 3.8 million timestep). Simultaneously, the diagram indicates that all other algorithms maintain a stable mean episode length at the maximum value (314), which is desirable as they have saturated.



Run	Min	Max	Start Value	End Value	Value	%	Start Step	End Step
A2C_model_corr_no_delay_0	-15,418,906	-14.0199	-21.417	-15,418,906	↓15,418,884.583	↑71993550%	9,500	115,000
A2C_model_corr_with_delay_0	-7.2217	-6.2164	-7.2217	-6.2981	↑0.9236	↓-13%	21,500	115,000
DDPG_model_corr_no_delay_0	-75.178	-9.0253	-75.178	-12.6649	↑62.5131	↓-83%	1,256	101,736
DDPG_model_corr_with_delay_0	-265.5265	-8.0956	-38.5167	-265.5265	↓227.0098	↑589%	1,256	101,736
PPO_model_corr_no_delay_0	-259.0568	-14.0944	-259.0568	-14.0944	↑244.9624	↓-95%	2,048	102,400
PPO_model_corr_with_delay_0	-122.3658	-6.3482	-122.3658	-6.3482	↑116.0176	↓-95%	2,048	102,400
SAC_model_corr_no_delay_0	-237.4886	-5.4398	-237.4886	-5.4474	↑232.0412	↓-98%	1,256	101,884
SAC_model_corr_with_delay_0	-126.2699	-2.3179	-126.2699	-2.3202	↑123.9497	↓-98%	1,256	101,884

Figure 118 : Mean Episode Reward per Timestep for all Algorithms in the Corrective Environment With / No Delay

The graph clearly displays the A2C “without delay” model attempt of big exploration. While most models eventually stabilize, they show varying levels of performance improvement. The A2C model “without delay” experiences significant negative rewards, reflecting its difficulty in learning initially. Over time, other models, particularly those with delay correction, show more consistent improvements in rewards.

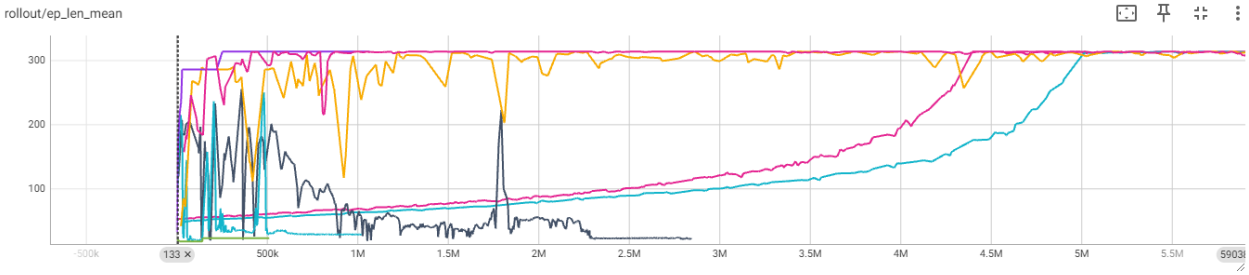


Run	Min	Max	Start Value	End Value	Value	%	Start Step	End Step
A2C_model_corr_no_delay_0	13.058	5,328,888,320	13.058	287,088,864	↑287,088,850.942	↑2198561071%	9,500	603,500
A2C_model_corr_with_delay_0	13.0439	14.0154	13.058	13.058	↑0	↑0%	21,500	590,500
DDPG_model_corr_no_delay_0	8.1346	266.9668	13.058	8.1915	↓4.8665	↓-37%	1,256	600,368
DDPG_model_corr_with_delay_0	13.058	385.9457	13.058	385.9457	↑372.8877	↑2856%	1,256	600,368
PPO_model_corr_no_delay_0	11.4676	304.5105	304.5105	11.6496	↓292.8609	↓-96%	2,048	600,064
PPO_model_corr_with_delay_0	11.3973	270.2987	270.2987	12.0185	↓258.2802	↓-96%	2,048	600,064
SAC_model_corr_no_delay_0	3.5193	232.3027	232.3027	3.8637	↓228.439	↓-98%	1,256	349,420
SAC_model_corr_with_delay_0	4.2038	256.6743	256.6743	7.6279	↓249.0464	↓-97%	1,256	279,734

Figure 119 : PHM Score per Timestep for all Algorithms in the Corrective Environment With / No Delay

Once again A2C “without delay” fluctuates greatly, while A2C “with delay” remains stable. The DDPG “no delay” model initially fluctuates but then remains steady, and DDPG with delay shows increased exploitation remaining almost constant after the initial exploration. Both PPO models consistently maintain low, stable scores, indicating strong performance. SAC models also perform well with stable, low scores. Overall, PPO and SAC are the most robust, with delay correction improving stability for A2C and DDPG.

4.5.3.2 Non-Corrective Environments



Run	Min	Max	Start Value	End Value	Δ Value	Δ %	Start Step	End Step
A2C_model_non_corr_no_delay_0	41.87	314	42.81	311.87	↑269.06	↑628%	21,500	5,880,000
A2C_model_non_corr_with_delay_0	157.28	314	157.28	308.91	↑151.63	↑96%	39,500	5,910,000
DDPG_model_non_corr_no_delay_0	32.25	314	33.25	314	↑280.75	↑844%	133	1,064,325
DDPG_model_non_corr_with_delay_0	18	23	20.75	23	↑2.25	↑11%	498	510,251
PPO_model_non_corr_no_delay_0	48.26	314	48.26	313.29	↑265.03	↑549%	38,912	5,883,904
PPO_model_non_corr_with_delay_0	53.33	314	53.33	314	↑260.67	↑489%	12,288	5,902,336
SAC_model_non_corr_no_delay_0	18.81	255.64	118.9868	22.51	↓96.4768	↓-81%	9,043	2,844,187
SAC_model_non_corr_with_delay_0	18.41	250.26	84.0833	28.69	↓55.3933	↓-66%	2,018	1,019,921

Figure 120 : Mean Episode Length per Timestep for all Algorithms in the Non-Corrective Environment With / No Delay.

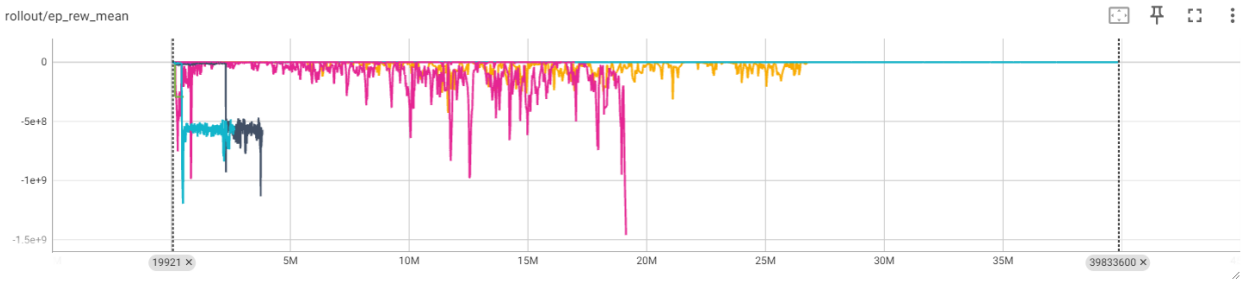
The graph above provides a comparison of the performance of the reinforcement learning algorithms (A2C, DDPG, PPO, SAC) with and without delay correction in terms of mean episode length over training steps. PPO (Proximal Policy Optimization) stands out for its consistent and strong performance. Both PPO models, with and without delay correction, demonstrate significant improvements and stable learning trajectories, stabilizing around the maximum episode length of 314. This indicates that PPO is highly effective for the task at hand, with or without the additional reward scaling introduced by the delay correction.

For A2C, the models show notable improvements in episode length, with the version without delay correction achieving a slightly better episode length of 311.87 compared to the version with delay correction (308.91). The delay correction helps the A2C model stabilize more quickly, suggesting that while it aids in the learning process, the final performance is slightly better without it.

DDPG models exhibit more alternation in performance. The DDPG model without delay correction reaches the max episode length of 314, indicating effective learning, though it has minimal fluctuations during training. In contrast, the DDPG model with delay correction struggles, showing minimal improvement and achieving only a slight increase from 20.75 to 23, suggesting that delay correction does not benefit DDPG in this context.

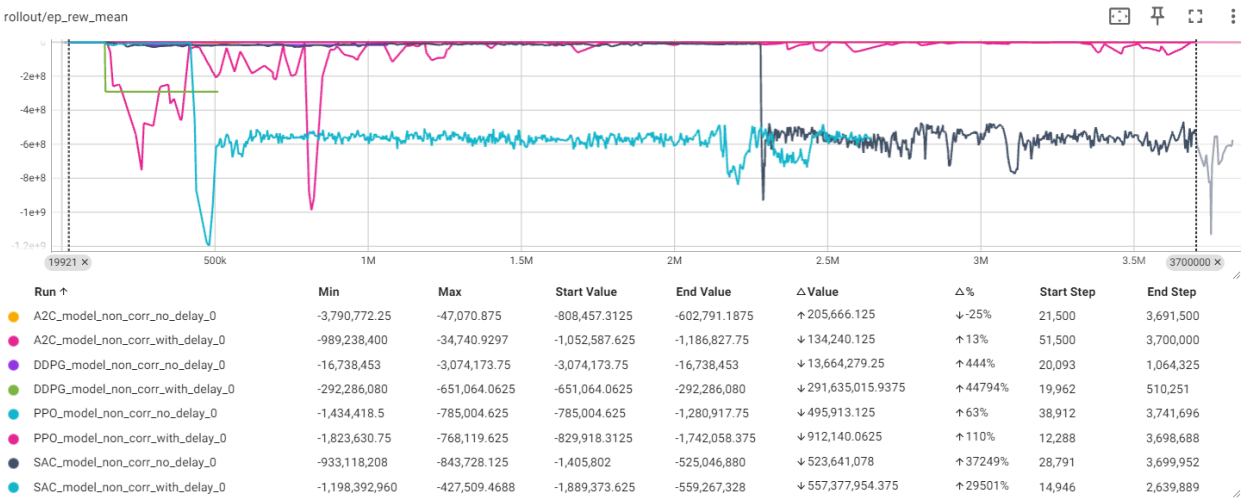
SAC models perform poorly compared to the other algorithms. Both SAC models, with and without delay correction, show a significant decline in episode length over time. The SAC model without delay correction starts at 118.98 and drops to 22.51, while the SAC model with delay correction starts at 84.08 and drops to 28.69. This suggests that SAC struggles with this particular task and the delay correction aids in the decline in performance.

In summary, PPO is the most robust and reliable algorithm for this task, showing excellent performance with minimal sensitivity to the delay correction. A2C also benefits from delay correction for quicker stabilization but ultimately performs slightly better without it. DDPG shows promise without delay correction but suffers significantly with it. SAC underperforms in both configurations, indicating it may not be well-suited for this specific application. The delay correction technique has mixed effects across different algorithms, helping some (like A2C) to stabilize faster while hindering others (like DDPG and SAC).



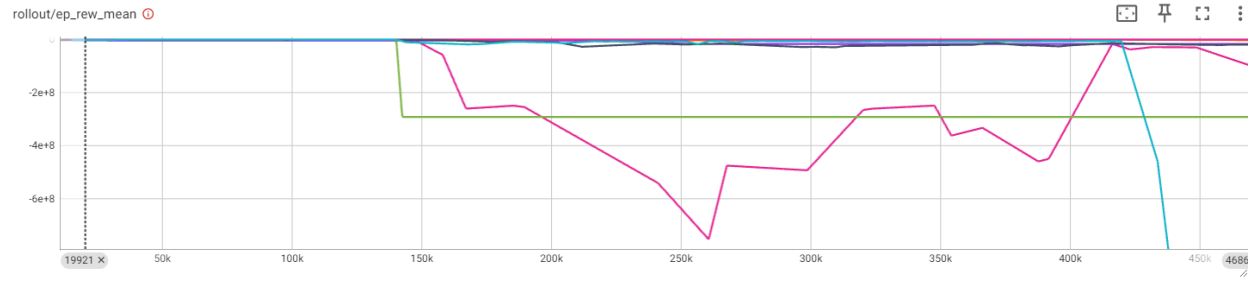
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	-429,141,408	-17,250.9238	-808,457.3125	-106,195.6094	↑702,261.7031	↓-87%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	-1,462,147,968	-9,138.624	-1,052,587.625	-1,462,147,968	↓1,461,095,380.375	↑138810%	51,500	19,120,000
DDPG_model_non_corr_no_delay_0	-16,738,453	-3,074,173.75	-3,074,173.75	-16,738,453	↓13,664,279.25	↑444%	20,093	1,064,325
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-651,064.0625	-292,286,080	↓291,635,015.9375	↑44794%	19,962	510,251
PPO_model_non_corr_no_delay_0	-2,375,710.5	-3,986.1587	-785,004.625	-15,653.6504	↑769,350.9746	↓-98%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	-54,153,960	-4,053.2996	-829,918.3125	-129,065.9141	↑700,852.3984	↓-84%	12,288	16,967,680
SAC_model_non_corr_no_delay_0	-1,135,406,336	-843,728.125	-1,405,802	-574,543,744	↓573,137,942	↑40769%	28,791	3,819,951
SAC_model_non_corr_with_delay_0	-1,198,392,960	-427,509.4688	-1,889,373.625	-559,267,328	↓557,377,954.375	↑29501%	14,946	2,639,889

Figure 121 : Mean Episode Reward per Timestep for all Algorithms in the Non-Corrective Environment With / No Delay



Run	Min	Max	Start Value	End Value	Value	%	Start Step	End Step
A2C_model_non_corr_no_delay_0	-3,790,772.25	-47,070.875	-808,457.3125	-602,791.1875	↑205,666.125	↓-25%	21,500	3,691,500
A2C_model_non_corr_with_delay_0	-989,238,400	-34,740.9297	-1,052,587.625	-1,186,827.75	↓134,240.125	↑13%	51,500	3,700,000
DDPG_model_non_corr_no_delay_0	-16,738,453	-3,074,173.75	-3,074,173.75	-16,738,453	↓13,664,279.25	↑444%	20,093	1,064,325
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-651,064.0625	-292,286,080	↓291,635,015.9375	↑44794%	19,962	510,251
PPO_model_non_corr_no_delay_0	-1,434,418.5	-785,004.625	-785,004.625	-1,280,917.75	↓495,913.125	↑63%	38,912	3,741,696
PPO_model_non_corr_with_delay_0	-1,823,630.75	-768,119.625	-829,918.3125	-1,742,058.375	↓912,140.0625	↑110%	12,288	3,698,688
SAC_model_non_corr_no_delay_0	-933,118,208	-843,728.125	-1,405,802	-525,046,880	↓523,641,078	↑37249%	28,791	3,699,952
SAC_model_non_corr_with_delay_0	-1,198,392,960	-427,509.4688	-1,889,373.625	-559,267,328	↓557,377,954.375	↑29501%	14,946	2,639,889

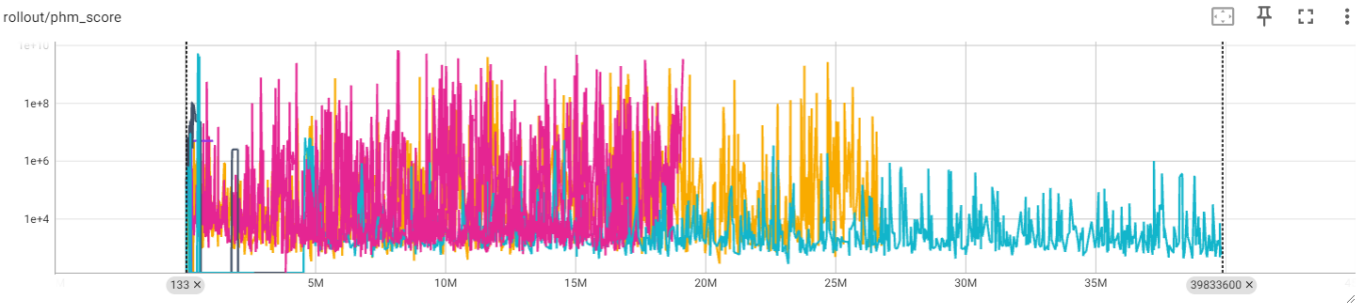
Figure 122 : Shorter Range Selection of Mean Episode Reward per Timestep for all Algorithms in the Non-Corrective Environment With / No Delay



Run	Min	Max	Start Value	End Value	Value	%	Start Step	End Step
A2C_model_non_corr_no_delay_0	-3,790,772.25	-656,930.4375	-808,457.3125	-3,772,500.25	↓2,964,042.9375	↑367%	21,500	486,000
A2C_model_non_corr_with_delay_0	-754,371,200	-1,052,587.625	-1,052,587.625	-30,058,470	↓29,005,882.375	↑2756%	51,500	448,500
DDPG_model_non_corr_no_delay_0	-16,738,453	-3,074,173.75	-3,074,173.75	-16,738,453	↓13,664,279.25	↑444%	20,093	468,981
DDPG_model_non_corr_with_delay_0	-292,286,080	-651,064.0625	-651,064.0625	-292,286,080	↓291,635,015.9375	↑44794%	19,962	468,759
PPO_model_non_corr_no_delay_0	-882,022.1875	-785,004.625	-785,004.625	-882,022.1875	↓97,017.5625	↑12%	38,912	471,040
PPO_model_non_corr_with_delay_0	-926,713.1875	-768,119.625	-829,918.3125	-926,713.1875	↓96,794.875	↑12%	12,288	471,040
SAC_model_non_corr_no_delay_0	-29,557,516	-843,728.125	-1,405,802	-19,539,142	↓18,133,340	↑1290%	28,791	461,640
SAC_model_non_corr_with_delay_0	-1,182,167,680	-427,509.4688	-1,889,373.625	-1,182,167,680	↓1,180,278,306.375	↑62469%	14,946	473,045

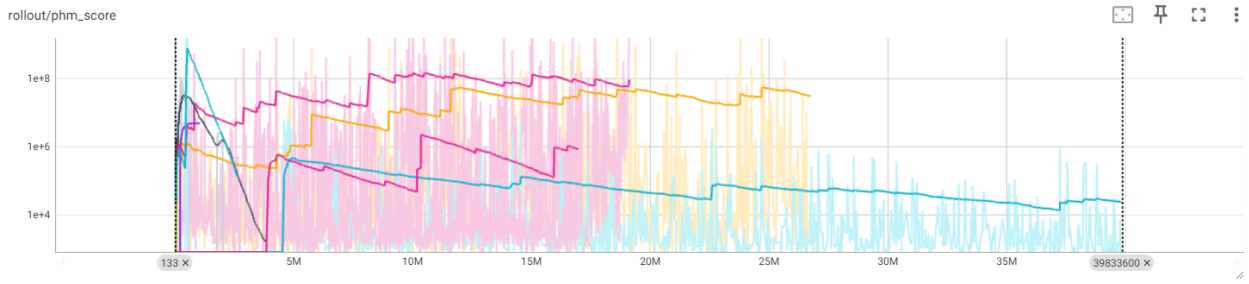
Figure 123 : Even Shorter-Range Selection of Mean Episode Reward per Timestep for all Algorithms in the Non-Corrective Environment With / No Delay

All of the three images depict the Mean Episode Reward per Timestep. The first image shows the full range of timesteps, highlighting the performance of the eight models over a wide scale, with a lot of variation in the scale of the results. The second image zooms into a shorter range, providing a more detailed view of the initial performance and stabilization phases for the models, with A2C and SAC showing substantial variability. The third image of the models, revealing that models like A2C and DDPG experience drastic reward drops probably due to matching exploration as it was observed in the previous figures, while PPO and SAC are relatively stable but still show notable variance.



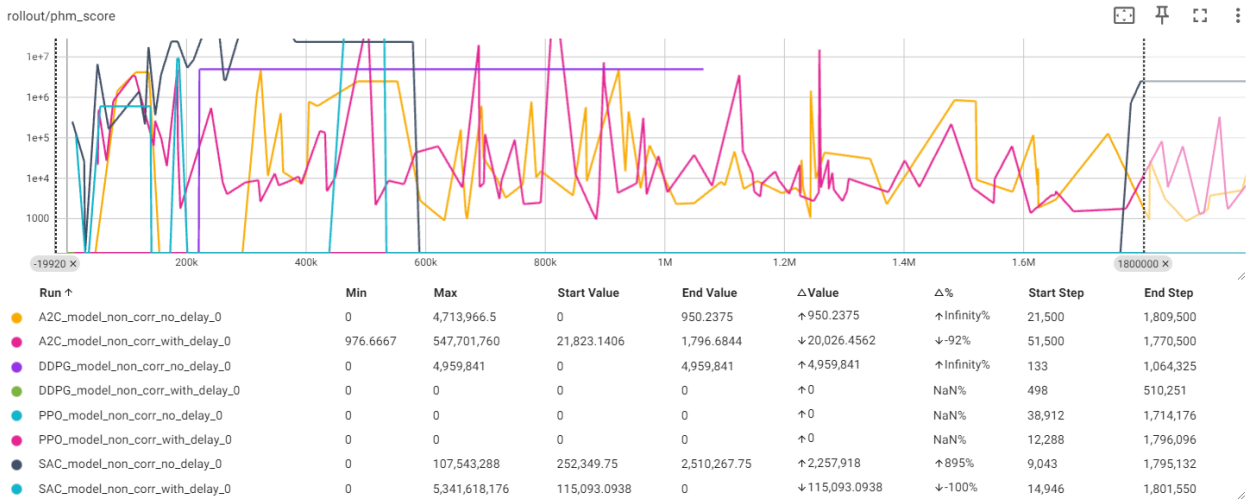
Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	3,935,591,168	0	30,639.4805	↑30,639.4805	↑Infinity%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	474.9217	6,790,013,952	21,823.1406	3,374,888,704	↑3,374,866,880.8594	↑15464625%	51,500	19,120,000
DDPG_model_non_corr_no_delay_0	0	4,959,841	0	4,959,841	↑4,959,841	↑Infinity%	133	1,064,325
DDPG_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	498	510,251
PPO_model_non_corr_no_delay_0	0	6,386,266.5	0	565.6765	↑565.6765	↑Infinity%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	0	206,020,384	0	2,777.8508	↑2,777.8508	↑Infinity%	12,288	16,967,680
SAC_model_non_corr_no_delay_0	0	107,543,288	252,349.75	0	↓252,349.75	↓-100%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	0	5,341,618,176	115,093.0938	0	↓115,093.0938	↓-100%	14,946	2,639,889

Figure 124 : Phm Score per Timestep for all Algorithms in the Non-Corrective Environment with / no delay



Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	55,360,212.9076	0	30,636,905.3972	↑30,636,905.3972	↑Infinity%	21,500	26,750,000
A2C_model_non_corr_with_delay_0	21,823.1406	147,271,019.9556	21,823.1406	93,594,384.123	↑93,572,560.9823	↑428777%	51,500	19,120,000
DDPG_model_non_corr_no_delay_0	0	4,954,853.8202	0	4,954,853.8202	↑4,954,853.8202	↑Infinity%	133	1,064,325
DDPG_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	498	510,251
PPO_model_non_corr_no_delay_0	0	472,978.1049	0	24,131.2842	↑24,131.2842	↑Infinity%	38,912	39,833,600
PPO_model_non_corr_with_delay_0	0	2,295,272.6857	0	845,879.8488	↑845,879.8488	↑Infinity%	12,288	16,967,680
SAC_model_non_corr_no_delay_0	1,689.2414	32,404,227.9729	252,349.75	1,689.2414	↓250,660.5086	↓-99%	9,043	3,819,951
SAC_model_non_corr_with_delay_0	22,558.2696	784,422,693.061	115,093.0937	133,538.6964	↑18,445.6027	↑16%	14,946	2,639,889

Figure 125 : "Smoothed" Graph of Phm Score per Timestep for all Algorithms in the Non-Corrective Environment With / No Delay



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
A2C_model_non_corr_no_delay_0	0	4,713,966.5	0	950.2375	↑950.2375	↑Infinity%	21,500	1,809,500
A2C_model_non_corr_with_delay_0	976.6667	547,701,760	21,823.1406	1,796.6844	↓20,026.4562	↓-92%	51,500	1,770,500
DDPG_model_non_corr_no_delay_0	0	4,959,841	0	4,959,841	↑4,959,841	↑Infinity%	133	1,064,325
DDPG_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	498	510,251
PPO_model_non_corr_no_delay_0	0	0	0	0	↑0	NaN%	38,912	1,714,176
PPO_model_non_corr_with_delay_0	0	0	0	0	↑0	NaN%	12,288	1,796,096
SAC_model_non_corr_no_delay_0	0	107,543,288	252,349.75	2,510,267.75	↑2,257,918	↑895%	9,043	1,795,132
SAC_model_non_corr_with_delay_0	0	5,341,618,176	115,093.0938	0	↓115,093.0938	↓-100%	14,946	1,801,550

Figure 126 : Shorter Range Selection of Phm Score per Timestep for all Algorithms in the Non-Corrective Environment With / No Delay

The goal is to minimize the Phm Score but due to the initialization of the phm_score variable as zero, the true minimum value is not displayed here. However, it is easy to spot graphically in the smoothed graph which models behave the best.

The worst model is by far the DDPG "with delay" model which never managed to reach episode length of 314 and never got a Phm score. All other models have reached the max episode length over the training process. DDPG "no delay" has a constant high Phm score which is undesirable. A2C, SAC and PPO all display a lot of variability and high frequency of low scores followed by high spikes. Even though there is a continual downward improvement in all cases, the big instability suggests that the models are still imperfect to some extent.

4.5.3.3. General Commentary on the Training of the Models

Advantages & Disadvantages

PPO

PPO displayed a stable learning process and robust performance across all environments. The algorithm compared to the other alternatives is relatively straightforward to implement and does not require extensive hyperparameter tuning. Also, it is the least computationally demanding algorithm resulting in the fastest training time per timestep. Consistently achieves high and stable rewards, making it a reliable choice for reinforcement learning tasks.

As an on-policy algorithm, PPO requires a large number of samples, which can be computationally expensive. While effective, handling very high-dimensional continuous action spaces (the action space is a 3-dimensional vector $a_t = [action_1, action_2, \ldots, action_n]$) it can still be challenging.

SAC

SAC uses entropy maximization, encouraging exploration and often leading to better policy discovery in complex environments such as the one examined. As an off-policy algorithm, SAC can reuse samples,

making it more sample-efficient compared to on-policy methods like PPO. It is particularly effective in environments with continuous action spaces and the performance of the algorithm reflects that.

The implementation of SAC is more complex due to the need to maintain multiple networks (policy, Q-function, and Value function). This results in higher training times as SAC can be computationally intensive, requiring significant resources for training.

SAC is the algorithm with the observed fastest time to converge. At the same time, it required the least timesteps to converge.

A2C

A2C benefits from using multiple parallel environments, which can speed up the training process and stabilize learning. The algorithm is simpler compared to other actor-critic methods like SAC, making it easier to understand and implement.

At the same time, A2C can be less stable than PPO, particularly in environments with high variance like the one examined. As an on-policy method, it also requires fresh samples for each update (which is not the case in this framework), leading to higher sample inefficiency.

DDPG

DDPG is specifically designed for environments with continuous action spaces, performing well in such settings, however this is not the case in the dataset examined. DDPG is the worst of the four algorithms examined in the thesis. It stays constant after some exploration and constantly exploits a local minimum. The deterministic policy has led to insufficient exploration, requiring additional mechanisms like noise injection for effective exploration. The need to manage both actor and critic networks, as well as the replay buffer and target networks, adds to the complexity of DDPG.

In summary, DDPG is a highly computationally intensive algorithm, producing bad resulting scores, making it a bad algorithm for this use case scenario.

Best Training Models

As far as training is concerned, PPO is the best algorithm due to its fast-training time per timestep and overall efficiency. It is straightforward to implement, requires less computational power, and consistently achieves high and stable rewards, making it the most reliable choice for the specific reinforcement learning problem (this happens during training, the results in the test datasets in the section 4.5 will not follow the same pattern). SAC is the second-best algorithm, showing excellent performance with the fastest time to converge and the least number of timesteps required for convergence. However, it is more computationally intensive and complex to implement. On the other hand, DDPG is undoubtedly the worst algorithm for this use case scenario. It suffers from insufficient exploration, leading to poor results and higher computational demands, making it unsuitable for the given dataset and degradation problem.

4.6 Results

Evaluation of the Trained Models

To evaluate the trained models, a custom evaluation function needs to be created. The library in use, Stable Baselines 3, provides a built-in evaluation function called **`evaluate_policy()`**. This function checks the policy of the selected algorithm for a specified number of evaluation episodes and returns the average reward. Although this is useful for typical RL problems, the grading criteria for this problem are unique, making the built-in **`evaluate_policy()`** function unsuitable.

The evaluation requires comparing the environment's or algorithm's score over the same episode length, particularly up to the maximum episode length of cut 314. Therefore, a custom evaluation function has been created to register the generated reward and PHM score only up to cut 314, as earlier scores should not be considered. Additionally, for scenarios like the DDPG Non-Corrective With Delay where the model does not reach the cut 314, the custom function also registers the average reward generated during the evaluation episodes. The pseudocode below displays the above in practice:

```
def Custom_Evaluate_Policy(algo, env, n_eval_epsodes):

   while not stop :

     i = latest_model()
     load_model_i(algo, env)

     for episode in range(n_eval_episodes):
        obs = env.reset()

        while not done:

        action, episode = model.predict(obs,
deterministic=True)
        obs, reward, terminated, truncated, info =
env.step(action)

        episode_reward += reward
        current_step, phm_score,  = info.get('cut', 0)

        if (current_step > MAX_STEPS - 2) :
           phm_score_log.append([phm_score, i])
           reward_314_log.append([episode_reward, i])
           done = True

        if (terminated or truncated) :
           done = True

      total_reward += episode_reward
      reward = total_reward / n_eval_episodes
      reward_timestep_log.append([reward, i])
```

```
        i = i-1

        if(i<=1):
           stop = True

        return reward_timestep_log, phm_score_log,
 reward_314_log
```

The function **`Custom_Evaluate_Policy`** has the following parameters:

- `algo`: The RL algorithm being evaluated.
- `env`: The environment (MDP Model) in which the policy is being evaluated.
- `n_eval_episodes`: The number of episodes to run for evaluation.

The main loop loads every model for the given algorithm and environment starting from the latest. The episode loop is the inner loop and for every model it evaluates it, iterating for the number of evaluation episodes provided. This number is the same for every algorithm and it is 10 episodes.

For each episode the model predicts the next action based on the current observation, then it takes action returning the new observation, the reward received, whether the episode has terminated, whether the episode was truncated and additional information like the current_step (the current cut) and the Phm score.

The reward accumulates overtime for each episode. If the episode reaches the max cut, then it logs the tuples of (reward, model number) and (Phm score, model number) and ends the episode loop. For special cases like DDPG Non-Corrective With Delay an episode reward summary is being generated and logged. After finishing the episode loop the outer loop checks the next model. When all the models have been checked the function returns the logs generated

When an episode reaches the maximum cut, it logs the tuples of (reward, model number) and (Phm score, model number) and ends the episode loop. For special cases, such as DDPG Non-Corrective With Delay, an episode reward summary is generated and logged. After finishing the episode loop, the outer loop proceeds to the next model of the same environment and algorithm (the previous one chronologically). Once all models have been evaluated, the function returns the generated logs.

Each category of model created is divided into a number of submodels. These submodels are generated after training the RL algorithm in a specific environment. As mentioned in Section 4.4, during training, the latest submodel is loaded and a new one is produced approximately every 10,000 steps. For example, the PPO Corrective No Delay model has a total of 1,117,000 timesteps, resulting in nearly 112 submodels.

For each of the 16 models created, every submodel is evaluated using the **`Custom_Evaluate_Policy`** function. From the returned log files, the best-performing submodel is selected. The tables below showcase the results of this procedure.

Corrective Environment | RL Algorithm Performance							
	Environment (Delay) & Dataset (Cutters)		Max Reward		Min Phm Score		Training Time & Total Timesteps
			Value	Timestep	Value	Timestep	
PPO	No	c4	-17.21	40,000	16.99	40,000	3 hours 1,117,000 steps
		c6	-16.21	70,000	18.00	70,000	
	With	c4	-8.98	20,000	15.73	20,000	2 hours 2,160,000 steps
		c6	- 8.23	1,460,000	17.85	670,000	
SAC	No	c4	-37.37	90,000	45.76	90,000	6.2 hours 349,420 steps
		c6	-14.80	30,000	16.36	30,000	
	With	c4	-10.94	20,000	23,07	20,000	5.3 hours 279,374 steps
		c6	-8,21	110,000	16,22	110,000	
DDPG	No	c4	-18.84	20,000	19.32	20,000	12.67 hours 629,256 steps
		c6	-17.04	250,000	18.55	520,000	
	With	c4	-10.75	20,000	19.36	20,000	11.5 hours 452,160 steps
		c6	-8.83	20,000	18.12	20,000	
A2C	No	c4	-17.84	700,000	18.33	700,000	13.7 hours 19,120,000 steps
		c6	-16.33	7,000,000	18.12	7,000,000	
	With	c4	-10.75	14,640,000	19.36	14,640,000	17.2 hours 14,640,000 steps
		c6	-8.83	14,640,000	18.12	14,640,000	

Table 8 : Performance of RL Algorithms in Corrective Environments

The best results are observed mostly at specific timesteps rather than the end of training. This could be due to the models initially finding a good policy and then overfitting as training continues. This results in the model learning the training data too well, including noise and anomalies, which do not generalize to new data like the cutter 4 and cutter 6 datasets.

PPO performs efficiently, finding optimal results early in training without delay but requiring more timesteps with delay, particularly on dataset c6. The improved performance with delay and higher rewards suggest PPO can adapt its policy to long-term rewards more effectively. The need for extensive training with delay indicates it can handle complex environments but may be susceptible to overfitting if not properly managed.

SAC shows significant improvement with delay, finding optimal results at different relative timesteps, indicating a mismatch of the different test dataset's perfomances. SAC's ability to optimize quickly aligns with its entropy-based exploration, making it less prone to overfitting and more adaptable to delayed rewards.

DDPG achieves its best results quickly and requires long training times, suggesting it might be working suboptimally. The stable performance across different conditions indicates robustness, though knowing the training process as it was displayed above the model is not improving overtime and it is stuck at a local maximum.

A2C has long training times and high timesteps indicate inefficiency in convergence but consistent performance. The improvement with delay suggests it can optimize long-term rewards but requires significant computational resources. The consistency across conditions indicates robustness, also it is the only model where in all test cases the training process is amplifying the performance. There are no cases of the best result being in early timesteps, meaning that the model is learning in the right direction.

Non-Corrective Environment | RL Algorithm Performance

	Environment & Dataset		Max Reward		Min Phm Score		Saturated (Yes / No)	Training Time & Total Timesteps
			Value	Timestep	Value	Timestep		
PPO	No	c4	-25,255,715	38,730,000	35,497,029	38,730,000	Yes	17 hours 39,833,600 steps
		c6	-163,762,286	20,000	233,645,404	20,000		
	With	c4	-544,135,308	20,000	726,430,945	20,000	Yes	19,2 hours 16,967,680 steps
		c6	-583,253,362	20,000	876,304,975	20,000		

SAC	No	c4	-643	400,000	752	390,000	No	43.3 hours 3,819,951 steps
		c6	-5,679	410,000	3140	460,000		
	With	c4	-23,455	200,000	11,418	200,000	No	32.1 hours 2,639,889 steps
		c6	-15,048	470,000	7,712	190,000		
DDPG	No	c4	-563,092,188	1,050,000	735,266,738	1,050,000	Yes	19.5 hours 1,064,325 steps
		c6	-637,838,860	1,050,000	910,064,743	1,050,000		
	With	c4	DNF	-	DNF	-	No	13.5 hours 510,251 steps
		c6	DNF	-	DNF	-		
A2C	No	c4	-50,834,200	30,000	3,469,607	30,000	Yes	26.1 hours 26,750,000 steps
		c6	-28,469,419	30,000	39,536,971	30,000		
	With	c4	-550,755,354	19,110,000	735,266,790	19,110,000	Yes	22.8 hours 19,120,000 steps
		c6	-606,030,553	19,110,000	910,064,743	19,110,000		

Table 9 : Performance of RL Algorithms in Non-Corrective Environments

For Non-Corrective environments, SAC shows relatively better performance without delay, while PPO and A2C struggle significantly. DDPG fails to converge effectively, particularly with delay.

As far as PPO is concerned, the training times and the total timesteps are substantial. Also, the best results occur early. The performance of the model especially in the “with delay” environment suggests overfitting and inefficiency. The high saturation indicates that PPO quickly reaches a point where no further improvement is observed, likely due to a poor adaptation to this type of environment.

SAC is by the far the best model and it is outperforming every algorithm. It shows better adaptability due to its entropy-based exploration, which helps in managing delayed rewards and avoiding overfitting to some extent. Like PPO the best performance is attributed to the earlier stages of the training process.

DDPG struggles with convergence in the Non-Corrective environment, especially with delay. It is the only model that did not manage to finish and reach the cut 314, as it was truncated before reaching the final cut due to extremely low reward. As shown in the Figure 127 and Figure 128 the model is stuck, the episode length and the reward are not changing over time. The high negative rewards and Phm scores indicate poor performance, possibly due to inadequate exploration.

A2C exhibits inefficiency with prolonged training times and large negative rewards, suggesting that its synchronous updates are not well-suited for the non-corrective environment, leading to overfitting and poor generalization**.**

Overall**,** after comparing every algorithm in all the different environments, it is clear that the **SAC** algorithm **performs best in the Non-Corrective Environment,** while **PPO excels in the Corrective Environment.** In terms of delay, the "with delay" environment achieves good results more quickly during training. However, in the testing phase, the performance of the "with delay" environment is usually surpassed by the "no delay" environment, which consistently achieves superior results.

5. Conclusions and future work

The objective of this thesis was to employ various reinforcement learning (RL) algorithms to address a predictive maintenance (PdM) problem, specifically the wear prediction of the flutes in cutters of a high-speed CNC milling machine. This prediction was based on data from dynamometer, accelerometer, and acoustic emission sensors. All algorithms were trained and evaluated using real-world data, and a broad analysis was conducted to examine their effectiveness.

The results demonstrated the effectiveness of this approach, highlighting that RL techniques can be sample-efficient and produce reliable results without requiring a model of the examined machine.

To further improve the performance of the RL PdM Agent, we can either increase the amount of data used for training, which will become more feasible as the number of IoT devices grows and the industry continues to adopt new techniques using more IoT technology. Another approach is to attempt to model the CNC environment by collaborating with material and mechanical engineers to create a model that closely approximates reality. While this approach is not guaranteed to succeed, it would require less data.

Overall, if sufficient data is available, the model-free approach is preferable due to it being universal and the lack of need for detailed knowledge of the specific machine's use and interactions.

Future Work

Future work could involve comparing different machine learning approaches with the RL approach and possibly combining them. Additionally, obtaining the full dataset from the 2010 PHM Data Challenge would allow for further improvement of the models' performance.

Another interesting path for future research is to address a different PdM objective for the same machine using the same dataset. Specifically, predicting the Remaining Useful Life (RUL) of the flutes, which was the objective of the 2010 PHM competition, could be a valuable extension of this work by incorporating RL techniques to solve this problem.

References

Achiam , J. (2018). *Simplified PPO-Clip Objective*. From https://drive.google.com/file/d/1PDzn9RPvaXjJFZkGeapMHbHGiWWW20Ey/view

Aggarwal, C., Hineeburg, A., & Keim, D. (2002). *On the Surprising Behavior of Distance Metric in High-Dimensional Space* . Research Gate.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour.* Princeton University Press.

Ben-Daya, M., Dufuaa, S., & Raouf, A. (2012). Maintenance, modeling and optimization. *Springer*.

Beyer, K., & Goldstein, J. (1999). When Is “Nearest Neighbor” Meaningful? *Database Theory ICDT'99* (pp. 217-235). Springer.

Bottou, L. (1991). Stochastic Gradient Ascent Learning in Neural Networks. *AT&T Bell Laboratories*.

Bousdekis, A., Apostolou, D., & Mentzas, G. (2020). *Predictive Maintenance in the 4th Industrial Revolution: Benefits, Business Opportunities, and Managerial Implications.* IEEE.

Boyd, S., & Vandenberghe, L. (2009). *Convex Optimization.* Cambridge University Press.

Burke , R., Hartigan, M., & Sniderman , B. (2017). *The smart factory*. From Deloitte Insights: https://www2.deloitte.com/us/en/insights/focus/industry-4-0/smart-factory-connected-manufacturing.html

Ding, F., He, Z., & Zi , Y. (2008). Application of support vector machine for equipment reliability forecasting. *2008 6th IEEE international conference on industrial informatics* (pp. 526-530). IEEE.

Eke, S., Aka-Ngnui, T., & Glerc, G. (2017). Characterization of the operating periods of a power transformer by clustering the dissolved gas data. *: 2017 IEEE 11th International symposium on diagnostics for electrical machines, power electronics and drives* (pp. 298-303). SDEMPED.

Frangopol, D., Lin, K., & Estes, A. (1997). Life-cycle cost design of deteriorating structures. *ASCE Library*.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. *arXiv*.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv*.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., . . . Levine, S. (2019). Soft Actor-Critic Algorithms and Applications. *arXiv*.

Kabir , F., Foggo, B., & Yu, N. (2018). Data Driven predictive maintenance of distribution transformers. *2018 China international conference on electricity distribution (CICED)*, (pp. 312-316).

Kakade, S. (2001). A Natural Policy Gradient. *Gatsby Computational Neuroscience Unit*.

Laape , S., Dollar, B., & Cotteleer , M. (2020). *Implementing the smart factory*. From Deloitte Insights: https://www2.deloitte.com/us/en/insights/topics/digital-transformation/smart-factory-2-0-technology-initiatives.html

Lewis , F., Vrabie, D., & Vamvoudakis, K. (2012). Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE*.

Li, X., Sim, B., Zhou, J., Huang, S., Phua, S., Shaw, K., & Er, M. (2009). Fuzzy Neural Network Modelling for Tool Wear Estimation in Dry Milling Operation. *Vol. 1 No. 1 (2009): Proceedings of the Annual Conference of the PHM Society 2009* . PHM.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wiestra, D. (2016). Continuous Control with Deep Reinforcement Learning. *arXiv - ICLR 2016* .

Ma, Z., Guo, J., & Mao, S. (2020). An interpretability research of the XGBoost algorithm in remaining. *2020 International conference on big data & artifcial intelligence & software engineering* (pp. 433-438). ICBASE.

MathWorks. (2024). *What is Reinforcement Learning.* The MathWorks, Inc.

Mnih, V., Badia , A., Mirza, M., Graves, A., Lilicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *arXiv*.

Open AI. (2024). *Kind of RL Algorithms*. From OpenAI Spinning Up: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

Open AI. (2024). *Proximal Policy Optimization*. From Open AI Spinning Up: https://spinningup.openai.com/en/latest/algorithms/ppo.html#id3

Open AI. (2024). *Vanilla Policy Gradient*. From Open AI Spinning Up: https://spinningup.openai.com/en/latest/algorithms/vpg.html

OpenAI. (2024). *Deep Deterministic Policy Gradient*. From OpenAI Spinning Up: https://spinningup.openai.com/en/latest/algorithms/ddpg.html

OpenAI. (2024). *Soft Actor Critic*. From OpenAI Spinning Up: https://spinningup.openai.com/en/latest/algorithms/sac.html

OpenAI. (2024). *Twin Delayed DDPG*. From OpenAI Spinning Up: https://spinningup.openai.com/en/latest/algorithms/td3.html

Pestov, V. (2000). On the geometry of similarity search: Dimensionality curse and concentration of measure . *arXiv*.

Russel, S., & Norvig, P. (2021). *Artificial Intelligence, A Modern Approach* . Pearson Education, Inc.

Sateesh Babu, G., Zhao, P., & Li, X. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. *International conference on database systems for advanced applications*, (pp. 214-228).

Saydam, D., & Frangopol, D. (2015). Risk-based maintenance optimization of deteriorating bridges. *ASCE Library*.

Sayyad , S., Kumar, S., & Bongale, A. (2022). Tool wear prediction using long short-term memory variant and hybrid feature selection techniques . *The Internation Journal of Advanced Manufacturing Technology*.

Schulman, J. (2016). *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs.* University of California, Berkeley.

Schulman, J., Levine, S., Moritz, P., Jordan, M., & Abbeel, P. (2015). Trust Region Policy Optimization. *University of California, Berkeley, Department of Electrical Engineering and Computer Sciences*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv*.

scikit Learn. (2023). *Stochastic Gradient Ascent*. From scikit-learn: https://scikit-learn.org/stable/modules/sgd.html

Silver , D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. *31st International Conference on Machine Learning.* JMLR: W&CP volume 32.

Siraskar, R., Kumar, S., Patil, S., Bongale, A., & Kotecha, K. (2023). Reinforcement learning for predictive maintenance: a systematic technical review. *Springer*.

Susto , G., Schirru, A., & Pampuri, S. (2013). A predictive maintenance system for integral type faults based on support vector machines: an application to ion implantation. *2013 IEEE international conference on automation science and engineering (CASE)* (pp. 195-200). IEEE.

Susto, G., Wan, J., & Pampuri, S. (2014). An adaptive machine learning decision system for fexible predictive maintenance. *2014 IEEE international conference on automation science and engineering* (pp. 806-811). CASE.

Susto, G., Wan, J., & Pampuri, S. (2014). An adaptive machine learning decision system for flexible predictive maintenance . *IEEE International conferene on automation science and engineering* (pp. 806-811). CASE .

Sutton , R., & Barto, A. (2018). *Reinforcement learning: an introduction.* MIT, Cambridge.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *AT&T Labs*.

Tensorflow. (2024). *Trust Region Policy Optimization* . From https://spinningup.openai.com/en/latest/algorithms/trpo.html: https://spinningup.openai.com/en/latest/algorithms/trpo.html

Torres-Garcia, A., Mendoza, O., Reyes-Garcia, C., & Villasenor-Pineda, L. (2022). *Biosignal Processing and Classification Using Computational Learning and Intelligence.* Science Direct.

Uhlenbeck, George, E., Ornstein, & Leonard, S. (1930). On the theory of the brownian motion.

Vailshery, L. S. (2024). *Number of IoT connections worldwide 2022-2033, with forecasts to 2030.* Statista.

Zheng , S., Ristovski, K., & Farahat, A. (2017). Long short-term memory network for remaining useful life. *2017 IEEE international conference on prognostics and health management (ICPHM)* (pp. 88-95). IEEE.