Impacts of parameter drift and induced decoherence in entangled quantum link generation on RL-based policy performance

Radu Ionuţ Ciobanu[1]
Supervisor(s): Gayane Vardoyan[1]**, Bethany Davies**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Radu Ionut Ciobanu
Final project course: CSE3000 Research Project
Thesis committee: Gayane Vardoyan, Bethany Davies, Rihan Hai

Abstract

A key function of a quantum internet is the generation of entangled links between devices. The quality of these links decays over time due to interaction with the outside world. Various protocols exist for generating these links. The main trade-off to be considered when choosing a protocol is between the chance of producing the link and the quality of the generated link. Selecting the optimal protocol is a complex and computationally intensive task, especially as the scale of the problem increases. While analytical solutions are feasible for small-scale systems, they become impractical for larger networks due to their computational demands. In addition, the rate at which the generated links decay is usually assumed to be static, when in reality it typically fluctuates over time in a process called parameter drift. In this paper, we evaluate the effectiveness of reinforcement learning approaches to optimizing protocol selection for different models of this problem. Models are provided for a static rate of decay, for a drifting rate of decay, and for the phenomenon of induced decoherence. Different reinforcement learning agents are trained on all of these models, and the results are compared. The criterion for effectiveness is the speed at which the requested number of entangled links can be generated. A negligible effect on performance is detected, showing that models are able to adapt to the different sources of instability studied. We also provide methods to model this problem and identify promising directions for future research.

1 Introduction

Quantum networks represent the next frontier in communication technologies, promising unparalleled capabilities in secure communication and distributed quantum computing. Central to the functioning of these networks is the establishment of reliable quantum links between nodes, enabling the exchange of quantum information over long distances. However, achieving reliable quantum communication poses significant challenges due to the inherently fragile nature of quantum states and the influence of environmental noise.

A quantum network is a network of multiple quantum processors that are able to send quantum information, in the form of qubits, to each other. The type of quantum information that is of interest to this paper is that of the entangled pair. An entangled pair of qubits is a quantum state in which the qubits are entangled with one another. Particularly, Einstein-Padolsky-Rosen pairs [5] represent pairs of two maximally entangled qubits. This means that once the state of one qubit is determined, the state of the other will be correlated above what classical statistics would predict. Many quantum algorithms make use of this kind of pair.

One critical aspect in the development of quantum networks is the selection of appropriate protocols for entangled link generation. A major trade-off that is considered in these protocols is that between link fidelity and the chance that the protocol succeeds in generating the link. Link fidelity is a measure of how similar the actual quantum state of two qubits is to the desired maximally entangled state [14]. This is particularly important because the fidelity of the link decays over time while it's stored in quantum memory. This imposes a timing restraint on the problem: all links have to be generated before any of them decays into an unusable state.

Previous research has been carried out on achieving sequences of entanglement generation under the assumption that the entanglement generation parameters are identical (i.e., same probability to successfully generate a link) [4]. This paper builds upon this research, seeking to find the optimal policy for deciding between multiple protocols to generate the requested

links. An additional assumption is the constant link decay rate in the model. In reality link decay is a probabilistic process, and the decay rate varies over time.

Two sources of decay rate inconsistency are modelled and investigated in this paper. Parameter drift [3], [15] refers to the tendency of the decay rate to change from the ideal modeled value over time. Induced decoherence [16] refers to the increased instability of stored links caused by having links stored in quantum memory, increasing the decay rate proportionally to the number of stored links.

Previous work in the field has achieved promising results by modeling a similar problem as a Markov Decision Process [11]. This model is inherently well suited to the use of machine learning methods, particularly reinforcement learning.

Two different machine learning approaches are investigated, namely deep Q-learning [2], [9] and proximal policy optimisation [17]. Both of these approaches are reinforcement learning based, and are chosen due to both of them being state of the art models that are well suited to continuous value environments [10].

The aim of this paper is to study how effective a reinforcement learning approach is at solving this problem, and how changing decay rate impacts it. The following research sub-questions can be formulated:

1. How to model entangled link generation as a reinforcement learning problem?

2. How to model decay rate drift as a reinforcement learning environment?

3. How to model induced decoherence as a reinforcement learning environment?

4. How do the agents trained on the models constructed in questions 1-3 compare to one another in performance and training time?

In this regard two models are presented for the constant decay rate case at different levels of abstraction and two different models are presented for the variable decay rate: one varies over time, modelling parameter drift, while another varies with the number of links in memory, modelling induced decoherence. The performances of reinforcement learning models trained on these environments are compared with one another.

2 Methodology and System Model

The objective of this study is to develop and evaluate optimal policies for choosing entangled link generation protocols in quantum networks, focusing on the impact of variable link decay rates. To achieve this, four distinct environments are created, and two reinforcement learning agents are trained to navigate and solve the problem. All of the work is done with the Tensorflow library [1] and its interfaces, the agents' implementations are taken from the TF agents library.

2.1 Link Decay

The main difficulty to be overcome when creating an optimal policy for entangled link generation is the unreliable nature of entangled links. The fidelity of the entangled links is assumed to decay over a time step according to the following formula:

$$F_{n+1} = \left(F_n - \frac{1}{4}\right) e^{-\Gamma} + \frac{1}{4} \tag{1}$$

where Γ is the decay rate of the links' fidelity and F_i is the fidelity at time step i

This decay is according to a commonly used noise model (called depolarising noise). One can check that after n time steps, the fidelity then decays to:

$$F_n = \left(F_1 - \frac{1}{4}\right) e^{-n\Gamma} + \frac{1}{4} \tag{2}$$

In order for the requesting algorithms to make use of the links their fidelity needs to be greater than some threshold F_{Thresh}. In order to generate these links one must choose from k protocols (p_i, F_i) $i \in 1, ..., k$ where p_i is the probability for protocol i to succeed and F_i is the fidelity of the link generated when it succeeds. The episode ends when there exist at least n_{links} links in memory with fidelity greater than F_{Thresh}.

2.2 Environments

Three environments are implemented in increasing order of complexity. This was done to ensure correctness as the model became more refined. All environments share the Γ, F_{Thresh}, n_{links}, and action space parameters, and the third and fourth environments also have a parameterized Gamma evolution function parameter. F_{Thresh} and n_{links} are constant numbers, the policies available for link generation are modelled as the action space, which is a list of k (p_i, F_i) pairs and Γ is a constant number in the first two environments and a function in the last two environments. Because the objective is to minimise the amount of time it takes to generate the required links, the reward is a constant -1 per action.

2.2.1 Discrete Static Decay Environment

The first environment we implement models the problem according to so-called "Fidelity Bins". A link is placed in fidelity bin i when it has i time steps left until it decays below F_{Thresh}. A formula for the fidelity bin in which a link with fidelity F is placed can be found starting with the fact that a fidelity of at least F_{Thresh} is desired:

$$F_N \geq F_{Thresh}. \tag{3}$$

Expanding F_N using (2) results in:

$$\left(F_1 - \frac{1}{4}\right) e^{-N\Gamma} + \frac{1}{4} \geq F_{Thresh}. \tag{4}$$

Rearranging the terms and taking the logarithms yields:

$$N \leq -\frac{1}{\Gamma} \ln\left(\frac{F_{Thresh} - \frac{1}{4}}{F_1 - \frac{1}{4}}\right). \tag{5}$$

and since N must be an integer, and is by definition the largest integer allowed, taking the floor gives the following expression for N:

$$N = \left\lfloor -\frac{1}{\Gamma} \ln\left(\frac{F_{Thresh} - \frac{1}{4}}{F - \frac{1}{4}}\right) \right\rfloor \tag{6}$$

At the end of every time step the environment decays all existing links, moving them one bin down. This discretization makes the state space smaller and allows for faster training due to lower calculation overhead. In order to further decrease the state space, these links are then sorted according to their fidelity bins. In addition, the number of links stored in memory

is also returned to the agent as an additional feature of the observation. This environment is used for the grid search performed and for initial testing. Due to its discretization it is faster to calculate actions performed on it, speeding up training.

2.2.2 Continuous Static Decay Environment

This environment is mostly the same as the discrete environment, except it does away with the fidelity bins. Instead, the state is a list of real numbers between 0 and 1 and the transition is modelled using (2). This environment serves as a control environment for the experiments that followed, however it was also useful as a benchmark to ensure the performance hit was not too strong and that the resulting policies were mostly unchanged in the transition between the discrete environment and this one.

2.2.3 Continuous Time-dependant Decay Environment (Parameter Drift)

This environment models the decay rate $\Gamma(t)$ as an arbitrary function of time, which is received as a parameter to the environment. This is used to test out the change in behaviour for various functions. It is otherwise unchanged from the static continuous environment.

2.2.4 Continuous Memory-dependant Decay Environment (Induced Decoherence)

Instead of modeling $\Gamma(m)$ as a function of time, this environment models it as an arbitrary function of the number of links stored in memory, received as a parameter to the environment. It is otherwise the same as the previous two environments.

2.3 Agents

Two models are trained on the described environments, namely Categorical Deep Q Learning (DQN) and Proximal Policy Optimization (PPO). These are chosen due to their general success in the field of reinforcement learning, especially in cases where the environment is continuous [6], [7], [12], [13], [18], [19].

Deep Q Learning seeks to approximate the Q function in any given state using a deep neural network. The Q function is defined as the expected reward if a given action is taken, and then the optimal policy is followed. Categorical Deep Q Learning estimates a distribution of Q function values for a given action instead of one value.

Proximal Policy Optimization improves policies in reinforcement learning through a stable and efficient iterative process. It starts by collecting data from interactions with the environment using the current policy, then calculates the advantage function to estimate action benefits. PPO uses a surrogate objective function that includes a clipped probability ratio to limit the extent of policy updates, balancing progress and stability. Policy parameters are updated by maximizing this clipped objective, ensuring significant yet controlled improvements. This iterative process continues until the policy converges or a stopping criterion is met.

3 Experimental Setup and Results

The agents used are taken from tensorflow's [1] TFAgents [8] library, namely "CategoricalDQNAgent" and "ppo_clip_agent". Hyperparameter tuning is done using a grid search

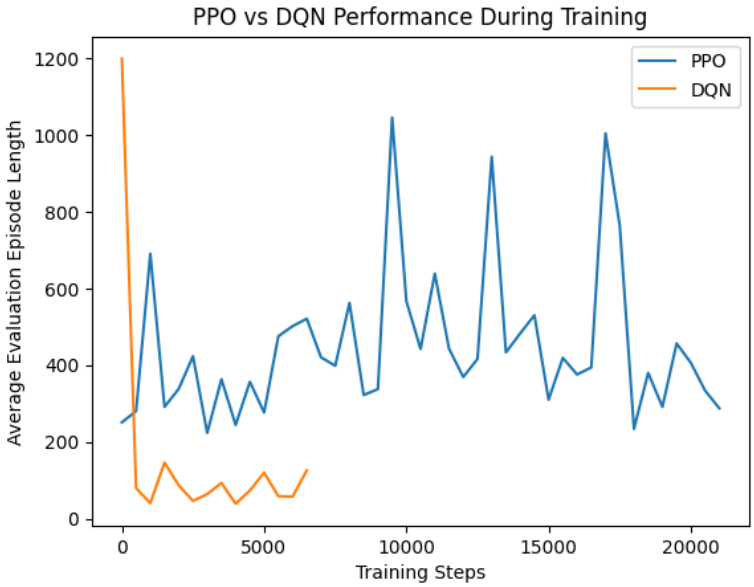


Figure 1: Average episode length of evaluations performed during model training for DQN and PPO. The PPO model was trained for four times longer than the DQN model and still failed to converge, while the DQN model rapidly converged in about 1000 steps.

over the search space in Appendix A.

3.1 Proximal Policy Optimisation

Preliminary results on the discrete environment are not promising for PPO (see Fig. 1). The training takes longer than DQN and is less stable. Besides this, the model also has a larger amount of hyperparameters, meaning that tuning would take longer. For these reasons, and due to limited computing resources and time constraints, PPO is considered a negative result, and further experimenting with it is not pursued.

3.2 Categorical Deep Q Network

In order to tune hyperparameters a grid search was done with the discrete environment configured as described in Table 1. The seven best performing configurations of the grid search were then chosen for further experimentation. A configuration refers to a permutation of the hyperparameters searched over. The performance metric used was the best time evaluated at any time during training. These seven configurations were trained for a longer period on both the continuous and the dynamic environments, with the same parameters as the grid search (see Appendix A for the set of parameters searched over and the sets chosen for the experiment).

Table 1: Parameters for the environment used during hyperparameter tuning

Number of links required	Threshold	Γ	Protocol 1		Protocol 2	
			Fidelity	Success Probability	Fidelity	Success Probability
4	0.5	0.2	1	0.2	0.6	0.4

Table 2: Environments used during the experiment and their respective Γ parameters. All other parameters used are the same as in Table 1. The function used for Γ evolution is a linear interpolation from 0.1 to 0.25. In the case of the time dependant environment this is done over 125 time steps. In the case of the memory dependant environment this is done over 4 links in memory (the required number of links).

Environment Name	Γ
Continuous (low constant Γ)	0.1
Continuous (high constant Γ)	0.25
Parameter Drift (time dependant Γ)	$\Gamma(t) = \min\left(0.25, 0.1 + \frac{0.15}{125}t\right)$
Induced Decoherence (memory dependant Γ)	$\Gamma(m) = 0.1 + \frac{0.15}{4}n$

3.2.1 Environment Parameters

Four different environments are used during the experiment. Two environments with a constant Γ value serve as a baseline for comparison. The Γ values used for these two environments also serve as endpoints for the linear interpolation functions used for the other two environments. The third environment models parameter drift, using a time dependant $\Gamma(t)$ function that interpolates the two endpoints over 125 time steps. The fourth environment models induced decoherence, using a memory dependant $\Gamma(m)$ function interpolating the two endpoints over the four memory slots required.

All environments share the same action space, threshold and required number of links. These are the same as in Table 1. The Γ parameters used can be found in Table 2. The values chosen for Γ to vary over were picked because they best reflect the dynamics of the system. see Fig. 2 for a plot of expected link lifetime as a function of Γ.

3.2.2 Training histories

During training every model is evaluated every 500 training steps for 100 episodes. This allows us to reconstruct the training history of every model and analyse them.

It can be observed (see Fig. 3a) that the model regularly fails to learn the high constant Γ case. We believe that this is due to the fact that this particular case has stricter timing constraints: The first three links have to be generated with protocol 1 in order to last until the end of the episode.

Another interesting result observed is the effect of timeout on training time. During initial training of the variable Γ environments a timeout of 1200 steps was imposed, meaning that after 1200 steps passed without finishing the episode it would end and the environment would reset. With the large timeout the model converged slowly and the results were noisy. After decreasing the timeout to 200 steps the model converged faster and the variance of its output decreased. This might be caused by the fact that with a shorter timeout the environment is able to learn more episodes, increasing the speed at which it learns the time

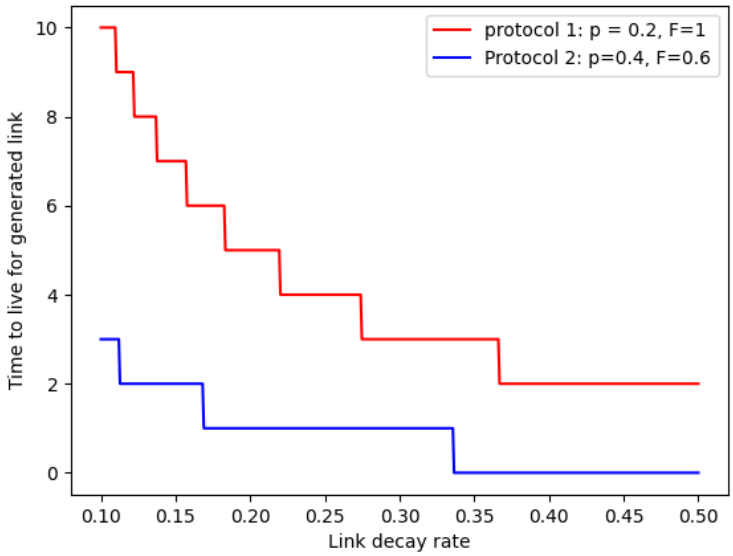


Figure 2: Dependence of the time to live of the links generated by the two actions the model had access to as a function of the decay rate Γ. Note that after about 0.27 it becomes impossible to generate 4 links, since links generated by protocol 1 would decay in 3 time steps.

evolution function.

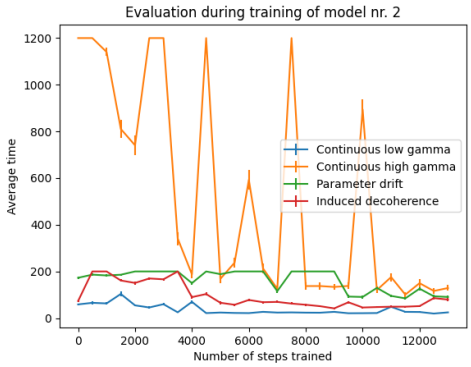
Another observation is that the low constant Γ model does not always converge faster than the dynamic environments, nor does it always finish episodes faster, despite it being a less complex environment. This might simply be caused by variance in the exploration of the models, or in the evaluation itself. It could also point to a limitation in the reinforcement learning approach: the constant reward function might not be optimal for training this particular case, since it doesn't immediately award the correct action choice.

3.2.3 Performance comparison

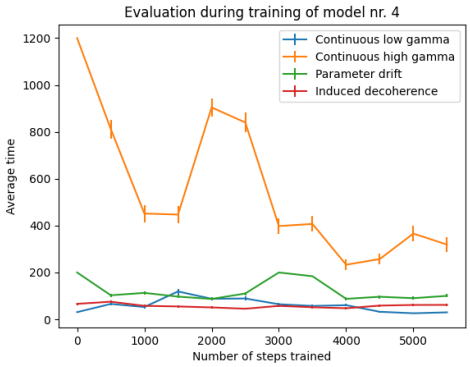
In order to compare the performance of models trained on each environment every model's policy is evaluated on every environment for 250 episodes. This cross evaluation is performed on every trained agent and the episode length is collected for every episode. Two such trials are of interest: those in Fig. 4 and in Fig. 5.

The case presented in Fig. 4 shows little difference in performance between the four models in all four environments. This could be caused by the small problem size, perhaps there does exist a difference but it is drowned out by the intrinsic noise of the system. In addition the models used for parameter drift and for induced decoherence assume linear evolution of Γ, and a noticeable performence difference might be observed in a more sophisticated model. Another possible explanation for this is the relatively low γ hyperparameter of the agent. This would cause it to choose its actions greedily, prioritising immediate reward.

On the other hand, Fig. 5 shows a case where the agents trained on dynamic environ-



(a) Training history of hyperparameter set 2



(b) Training history of hyperparameter set 4

Figure 3: Training histories of two different hyperparameter sets. There is some slight volatility in the models trained on the low constant Γ, parameter drift and induced decoherence environments. The model trained on the high constant Γ environment is very noisy. We believe this is caused by the tight timing constraint in the high Γ case.

ments greatly outperform those trained on a static Γ; the low Γ agent even times out for half of the trial environments. Another interesting observation in this case is that in the other two trial environments the low Γ agent seems to outperform the high Γ agent. This might be due to the low Γ agent having learned a policy that picks the high probability action more often than it should, thus being not viable in the cases where it times out, but faster in the other two cases.

Another result of interest is the absolute difference between the evaluation environments which can be seen in Fig. 6. The low constant Γ environment takes noticeably less than the other three. Of note is also that the lengths of the high Γ environment and the parameter drift environment are approximately the same, even for the parameter drift trained agent. This could be caused by the parameter drift agent being undertrained, or it being stuck in a local minimum of "waiting out" the drift before generating the required links. It could also be caused by the $\Gamma(t)$ function used interpolating too quickly between the two values, effectively reducing the environment to the high constant Γ environment.

4 Responsible Research

The research presented has no human-facing elements or personal data collection, as such there are no ethical risks involved with the research conducted. Taking reproducibility in consideration, the hyperparameters found using the grid search, and the search space over which it was performed, can be found in Appendix A. In addition, taking transparency into account, we acknowledge the negative result presented in section 3.1. All the code used during development can be found on github[1].

[1] https://github.com/ciobanuradu/research-project

5 Discussion

The results of the experiment are inconclusive, thus two hypotheses can be formed. The first hypothesis is that the models trained on the environment with a stable Γ are capable of adapting to the dynamic Γ cases, allowing future work to discard this aspect of modelling and simplifying future models. This would also mean that the models trained this way are robust, meaning that a model trained on one environment would be capable of handling many different environments.

The second option is that the environments used to train the agents are too simple. This could either be caused by the small scale of the environments utilized or by the simple model for Γ evolution. Limited resources and time constraints made investigating the first option not possible, and the second option falls out of the scope of this paper, requiring further theoretical development.

In spite of this, we feel that the reinforcement learning approach to the protocol selection problem presented is promising, however this paper only serves as a proof of concept. A larger scale case needs to be studied in order to fully demonstrate the power of our approach and to discover the limits of its robustness.

6 Conclusions and Future Work

In this paper we study the effectiveness of reinforcement learning in choosing an entangled link generation protocol. We also study the effects of parameter drift and induced decoherence on the robustness of this approach. In this regard we introduce multiple models for reinforcement learning environments, one of which serves as a control and the other two modelling the studied phenomena. Agents are trained on these environments using Deep Q-Learning. Results indicate that the effects of parameter drift and induced decoherence are negligible on policy optimality, however they are not conclusive. We recommend studying a larger scale implementation (higher generated link requirement) to confirm this result. Further theoretical development in modelling the evolution of the decay rate parameter would also help improve the certainty of our results.

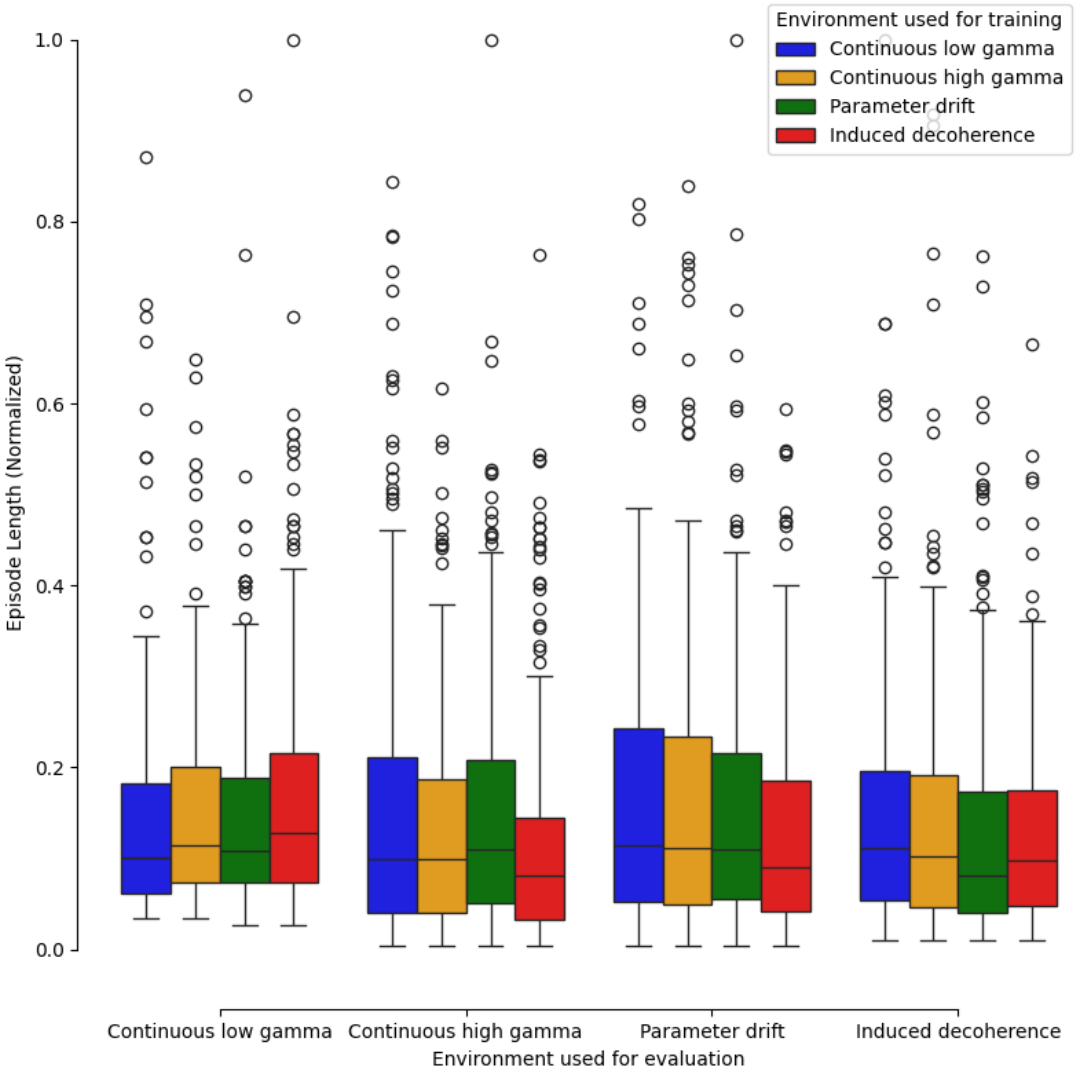


Figure 4: Normalized returns of hyperparameter set 3. Of interest is the relatively equivalent relative performance of every model. This implies that it is sufficient to train a model on an ideal environment and it would be capable of also handling the variable Γ cases. Of note is that this case has a lower γ hyperparameter than set 5 (see Fig 5) meaning that it prioritises future rewards less than that one.

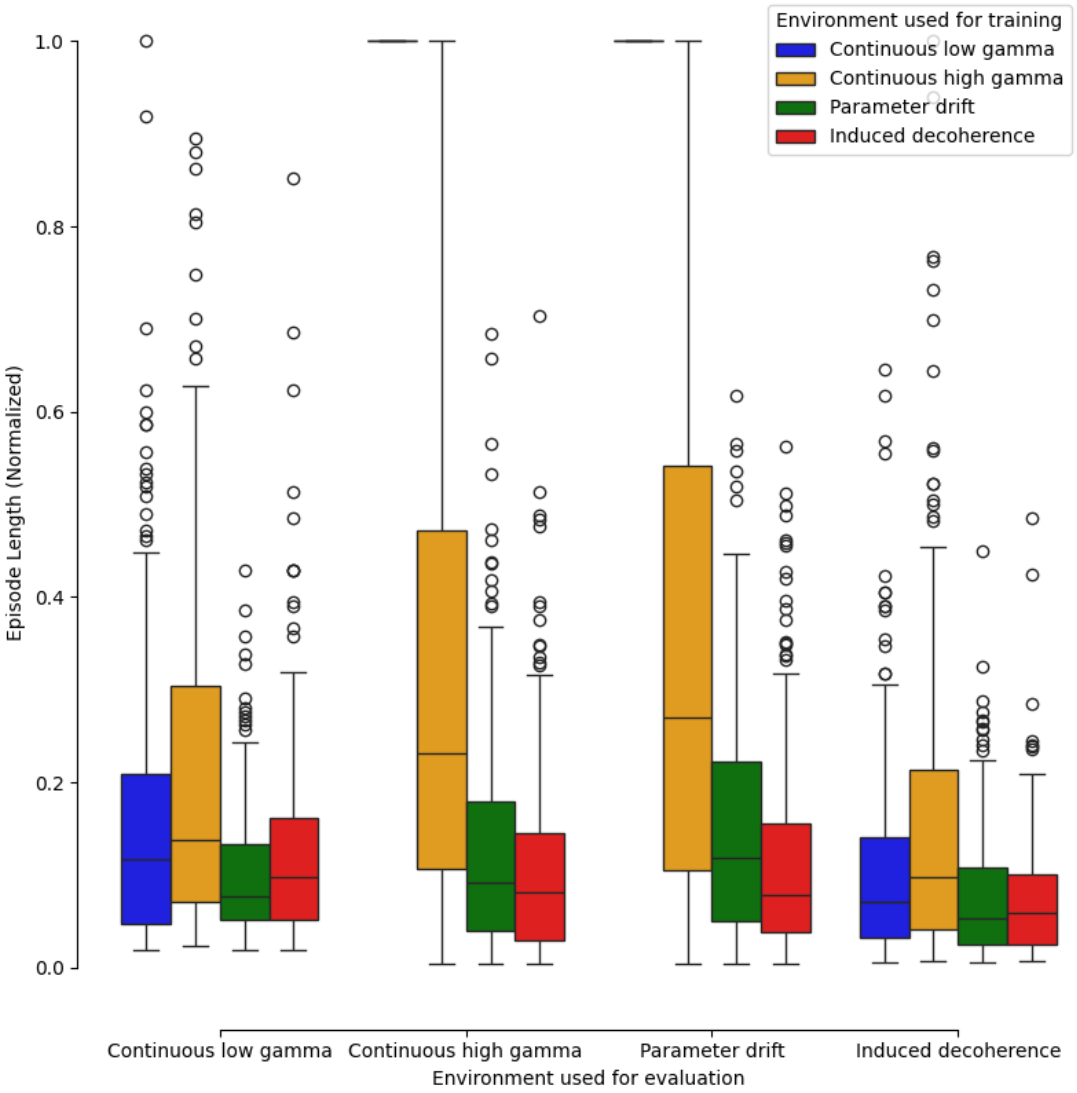


Figure 5: Normalized returns of hyperparameter set 5. This set shows a case where the agents trained on the static Γ cases are incapable of handling the parameter drift environment. Of note is that this case has a lower γ hyperparameter than set 3 (see Fig 4) meaning that it prioritises future rewards less than that one.

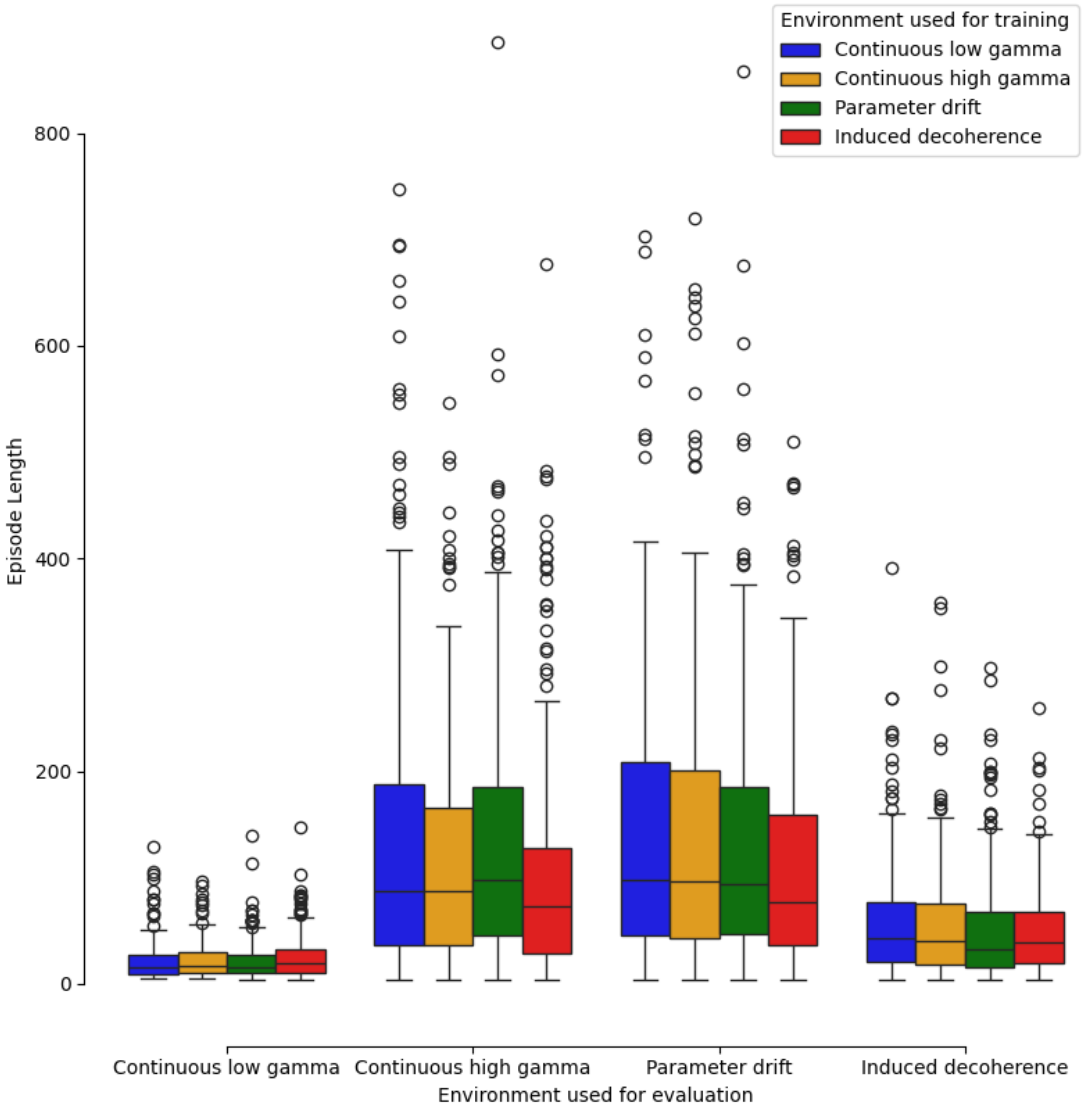


Figure 6: A non-normalized version of Fig. 4, showing the absolute difference between the different evaluation environments. This shows that there is a difference between the environments on which every model was trained and that the phenomena modelled do affect the speed of entangled link generation.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Marc G. Bellemare, Will Dabney, and RÃ©mi Munos. A distributional perspective on reinforcement learning, 2017.

[3] Arshag Danageozian. Recovery with incomplete knowledge: Fundamental bounds on real-time quantum memories. *Quantum*, 7:1195, December 2023.

[4] Bethany Davies, Thomas Beauchamp, Gayane Vardoyan, and Stephanie Wehner. Tools for the analysis of quantum protocols requiring state generation within a time window, 2023.

[5] Albert Einstein, Boris Podolsky, and Nathan Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical Review*, 47, 1935.

[6] Qiongxiao Fu, Enchang Sun, Kang Meng, Meng Li, and Yanhua Zhang. Deep q-learning for routing schemes in sdn-based data center networks. *IEEE Access*, 8:103491–103499, 2020.

[7] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR.

[8] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow, 2018. [Online; accessed 19-June-2024].

[9] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

[10] Qingyan Huang. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221, 2020.

[11] Álvaro G. Iñesta, Gayane Vardoyan, Lara Scavuzzo, and Stephanie Wehner. Optimal entanglement distribution policies in homogeneous repeater chains with cutoffs. *npj Quantum Information*, 9(1), May 2023.

[12] Stephen James and Edward Johns. 3d simulation for robot arm control with deep q-learning, 2016.

[13] Luckeciano C. Melo and Marcos R. O. A. Maximo. Learning humanoid robot running skills through proximal policy optimization, 2019.

[14] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*, chapter 9.2. Cambridge University Press, 2010.

[15] Timothy Proctor, Melissa Revelle, Erik Nielsen, Kenneth Rudinger, Daniel Lobser, Peter Maunz, Robin Blume-Kohout, and Kevin Young. Detecting and tracking drift in quantum information processors. *Nature Communications*, 11(1), October 2020.

[16] Maximilian Ruf, Noel H. Wan, Hyeongrak Choi, Dirk Englund, and Ronald Hanson. Quantum networks based on color centers in diamond. *Journal of Applied Physics*, 130(7):070901, 08 2021.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[18] Cristian J. Vaca-Rubio, Carles Navarro Manchón, Ramoni Adeogun, and Petar Popovski. Proximal policy optimization for integrated sensing and communication in mmwave systems, 2023.

[19] Wanli Wen, Jiping Yan, Yulu Zhang, Zhen Huang, Liang Liang, and Yunjian Jia. Adaptive cooperative streaming of holographic video over wireless networks: A proximal policy optimization solution, 2024.

A Grid Search Parameters

All permutations of the following values for their respective parameters are searched over:

Hyperparameter	Values
Number of atoms	51
Minimum Q value	-1200
Maximum Q value	0
ϵ_{max}	1.0, 0.8, 0.5
ϵ_{min}	0.01, 0.05, 0.1
ϵ_{steps}	500, 1000, 2500, 5000
γ	1.035, 0.99, 0.9, 0.8
Learning rate	10^{-4}, 10^{-5}

Table 3: Parameter space searched during DQN hyperparameter training.

ϵ decreases linearly from ϵ_{max} to ϵ_{min} over ϵ_{steps} time steps. Training lasts for $5 \cdot \epsilon_{steps}$, stopping early if there is no improvement in loss over the last 1000 steps.

The neural network utilised is a "categorical_q_network" from the TF agents library [8] with three layers as follows: (128, 16, 16).

Out of the permutations searched the following seven were selected to be trained on all the environment models (values that don't change across different permutations are left out):

Index	ϵ_{max}	ϵ_{min}	ϵ_{steps}	γ	Learning rate
0	1.0	0.1	1000	0.99	10^{-4}
1	1.0	0.05	1000	0.99	10^{-4}
2	0.5	0.05	1000	0.9	10^{-4}
3	0.5	0.1	1000	0.8	10^{-4}
4	0.5	0.1	1000	0.99	10^{-4}
5	0.5	0.1	1000	0.99	10^{-4}
6	0.8	0.1	1000	1.035	10^{-4}

Table 4: Hyperparameters of the DQN models used during the experiment

Due to there being little empirical benefit in the slower learning rate models, and due to time constraints on the project, they are manually removed from the set of models used during the experiment.