Transferable control for quantum parameter estimation through reinforcement learning

Han Xu,1,2 Jinning Li,1 Liqiang Liu,3 Yu Wang,2 Haidong Yuan,3,* and Xin Wang3,†

1Department of Physics, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR, China, and City University of Hong Kong Shenzhen Research Institute, Shenzhen, Guangdong 518057, China
2School of Physics and Technology, Wuhan University, Wuhan 430072, China
3Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong SAR, China

Measurement and estimation of parameters are essential for science and engineering, where one of the main quests is to find systematic and robust schemes that can achieve high precision. While conventional schemes for quantum parameter estimation focus on the optimization of the probe states and measurements, it has been recently realized that control during the evolution can significantly improve the precision. The identification of optimal controls, however, is often computationally demanding, as typically the optimal controls depend on the value of the parameter which then needs to be re-calculated after the update of the estimation in each iteration. Here we show that reinforcement learning provides an efficient way to identify the controls that can be employed to improve the precision. We also demonstrate that reinforcement learning is highly transferable, namely the neural network trained under one particular value of the parameter can work for different values within a broad range. These desired features make reinforcement learning more efficient than conventional optimal quantum control methods.

I. INTRODUCTION

Metrology, which studies high precision measurement and estimation, has been one of the main driving forces in science and technology. Recently, quantum metrology, which uses quantum mechanical effects to improve the precision, has gained increasing attention for its potential applications in imaging and spectroscopy [1–6].

One of the main quests in quantum metrology is to identify the highest precision that can be achieved with given resources. Typically the unknown parameter is encoded in a dynamics. To achieve the highest precision, one needs to optimize the probe states, the controls during the evolution and the measurements on the output states. Previous studies have been mostly focused on the optimization of the probe states and measurements [6]. The control only starts to gain attention recently [7–16]. It has now been realized that properly designed controls can significantly improve the precision limits. The identification of optimal controls, however, is often highly complicated and time-consuming. This issue is particularly severe in quantum parameter estimation, as typically optimal controls depend on the value of the parameter, which can only be estimated from the measurement data. When more data are collected, the optimal controls also need to be updated, which is conventionally achieved by another run of the optimization algorithm. This creates a high demand for the identification of efficient algorithms to find the optimal controls in quantum parameter estimation.

Over the past few years, machine learning has demonstrated astonishing achievements in certain high-dimensional input-output problems, such as playing video games [17] and mastering the game of Go [18]. Machine learning techniques have been applied in physics covering many topics including experimental designs [19], finding optimal state transfer schemes in a spin chain [20] and discovering the quantum-error-correction strategies under noises [21]. Among the machine learning algorithms, reinforcement learning is one of the most actively researched [22]. Here we show that RL can be used to efficiently identify the optimal controls in quantum parameter estimation. A main advantage of RL, compared to conventional optimal quantum control algorithms, is that it is highly transferable, i.e., the agent trained through RL under one value of the parameter works for a broad range of the values. There is then no need for re-training after the update of the estimated value of the parameter from the accumulated measurement data. This makes RL more efficient and less time-consuming as compared to conventional gradient-based optimization algorithms.

II. RESULTS

A. Model and Method

We consider a generic control problem described by the Hamiltonian [23]:

\[ \hat{H}(t) = \hat{H}_0(\omega) + \sum_{k=1}^{p} u_k(t) \hat{H}_k, \]  

where \( \hat{H}_0 \) is the time-independent free evolution of the quantum state, \( \omega \) the parameter to be estimated, \( u_k(t) \)
the $k$th time-dependent control field, $p$ the dimensionality of the control field, and $H_k$ couples the control field to the state.

The density operator of a quantum state (pure or mixed) evolves according to the master equation [11],

$$\partial_t \hat{\rho}(t) = -i \left[ \hat{H}(t), \hat{\rho}(t) \right] + \Gamma [\dot{\rho}(t)],$$

where $\Gamma[\dot{\rho}(t)]$ indicates a noisy process, the detailed form of which depends on the specific noise mechanism and will be detailed later.

The key quantity in quantum parameter estimation is the QFI [24–27], defined by

$$F(t) = \text{Tr} \left[ \hat{\rho}(t) \hat{L}_s^2(t) \right],$$

where $\hat{L}_s(t)$ is the so-called symmetric logarithmic derivative that can be obtained by solving the equation

$$\partial_t \hat{\rho}(t) = \frac{1}{2} \left[ \hat{\rho}(t) \hat{L}_s(t) + \hat{L}_s(t) \hat{\rho}(t) \right],$$

According to the Cramér-Rao bound, the QFI provides a saturable lower bound on the estimation as $\delta \hat{\omega} \geq \frac{1}{\sqrt{n}F(t)}$, where $\delta \hat{\omega} = \sqrt{\mathbb{E}[\hat{\omega}^2]}$ is the standard deviation of an unbiased estimator $\hat{\omega}$, and $n$ is the number of times the procedure is repeated. Our goal is therefore to search for optimal control sequences $u_k(t)$ that maximize the QFI at time $t = T$ (typically the conclusion of the control), $F(T)$, respecting all constraints possibly imposed in specific problems. Practically, we consider piecewise constant controls so the total evolution time $T$ is discretized into $N$ steps with equal length $\Delta T$ labeled by $j$, and we use $u_k^{(j)}$ to denote the strength of the control field $u_k$ on the $j$th time step. Researches of such problem are frequently tackled by the Gradient Ascent Pulse Engineering (GRAPE) method [23], which searches for an optimal set of control fields by updating their values according to the gradient of a cost function encapsulating the goal of the optimal control. It has been found that GRAPE is successful in preparing optimal control pulse sequences that improve the precision limit of quantum parameter estimation in noisy processes [11, 12].

Many alternative algorithms can tackle this optimization problem such as the stochastic gradient ascent (descent) method and microbial genetic algorithm [29], but the convergence to the optimal control fields becomes much slower when the dimensionality ($p$) of the control field or the discretization steps ($N$) increases. Other optimal quantum control algorithms, such as Krotov’s method [30, 30–34] and CRAB algorithm [35], typically depend on the value of the parameter, thus need to be run repeatedly along the update of the estimation, which is highly time-consuming. More efficient algorithms are thus highly desired.

In this work, we employ reinforcement learning to solve the problem. Fig. 1 shows schematics of the RL procedure used in this work. Fig. 1(a) shows the RL agent who takes an action as prescribed by a neural network. The action is essentially the control field which steers the qubit according to the master equation, Eq. (2), and the resulting state of the evolution determines the reward the agent receives. In practice, the reward encodes the QFI, i.e., higher reward will be obtained when greater QFI is given by the control.

In our problem, the action taken by the agent implies a time evolution of the quantum state according to Eq. (2) with the control field, $u_k(t)$. All possible actions therefore form a continuous set. We solve this problem using the Actor-Critic algorithm [22], as shown in Fig. 1(b). Such algorithm is particularly suitable to our problem as it can treat continuous actions. The key of the algorithm is that the neural network is not only updated using the reward, but also a state value, the latter of which greatly improves the efficiency of the training procedure. At certain time step, the neural network takes the quantum state as an input, and outputs both an action, and a state value which assesses how likely the state will lead to a larger QFI. The state is then evolved using the outputted action, obtaining the new state and QFI, which is then implemented into the reward function. This reward function, in conjunction with the state value, forms a loss function which is then used to update the neural network.

In order to improve the efficiency of computation, we used a parallel version of the Actor-Critic algorithm called Asynchronous Advantage Actor-Critic (A3C) algorithm [36]. The details of both the Actor-Critic and the A3C algorithm are described in the Supplementary Materials, as well as the pseudo-code describing the implementation of the algorithm.

Next we apply the algorithm to two commonly considered noisy processes: dephasing and spontaneous emis-
FIG. 2: Quantum parameter estimation under dephasing dynamics with \( \theta = \pi/4, \phi = 0 \) using square pulses. (a)-(c): results for \( \Delta T = 0.1, T = 5 \). (d)-(f): results for \( \Delta T = 1, T = 10 \). (a), (d) show the learning procedure, namely \( F(T)/T \) as functions of training epochs. (b) and (e) show \( F(t)/t \) for one of the best training results selected from (a) and (d) respectively. (c) and (f) show the pulse profiles corresponding to (b) and (e).

### B. Dephasing Dynamics

Under dephasing dynamics, the master equation, Eq. (2), takes the following form [11]:

\[
\partial_t \hat{\rho}(t) = -i \left[ \hat{H}(t), \hat{\rho}(t) \right] + \frac{\gamma}{2} \left[ \hat{\sigma}_n \hat{\rho}(t) \hat{\sigma}_n - \hat{\rho}(t) \right],
\]

where

\[
\hat{H}(t) = \frac{1}{2} \omega_0 \hat{\sigma}_3 + u(t) \cdot \sigma,
\]

the control field \( u(t) = (u_1, u_2, u_3) \) is a magnetic field that couples to \( \sigma = (\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3) \), and \( \gamma \) is the dephasing rate which is taken as 0.1 throughout the paper. We consider a dephasing along a general direction given by \( \mathbf{n} = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta) \), \( \hat{\sigma}_n = \mathbf{n} \cdot \sigma \). The parameter to be estimated is \( \omega_0 \) in Eq. (5), the true value of which is assumed to be 1, and we take \( \omega_0^{-1} = 1 \) as our time unit. We choose the probe state, i.e. the initial state of the evolution, as \( \mathbf{n} \cdot \sigma \).

In Fig. 2 we present our numerical results on QFI under dephasing dynamics with \( \theta = \pi/4, \phi = 0 \) using square pulses. Fig. 2(a)-(c) show the results for \( \Delta T = 0.1 \), Fig. 2(a) shows the training process in terms of \( F(T)/T \) as functions of the number of training epochs. The blue line shows results from the training using A3C algorithm. The value of \( F(T)/T \) corresponding to results from GRAPE and the case with no control are shown as the orange dotted line and grey dashed line, respectively. The red line shows results from “A3C+PPO”, an enhanced version of A3C which converges faster [37]. The details of this algorithm is explained in the Supplementary Materials. We can see that after sufficient training epochs, results from A3C exceed that for the case with no control, and approaches the optimal results found by GRAPE. On the other hand, “A3C+PPO” converges more quickly to essentially the same result of A3C.

We select one training outcome from those with best performances in Fig. 2(a) and show \( F(t)/t \) and the pulse profiles in Fig. 2(b), (c) respectively. As can be seen from Fig. 2(b), both GRAPE and A3C outperform the case with no control, while the results of A3C are comparable to those from GRAPE.

Fig. 2(d)-(f) show results with a larger time step, \( \Delta T = 1 \). From the training results shown in Fig. 2(d), we see that results from A3C occasionally exceed those from GRAPE, for example at training epoch approximately 1600 and 3000. \( F(t)/t \) and the pulse profile of one of the best performing results is again shown in Fig. 2(e) and (f), and we see from Fig. 2(e) that A3C indeed outperforms GRAPE in this case.

We have discussed dephasing dynamics along a particular axis pertaining to Fig. 2, and the results for several other dephasing axes are shown in the Supplementary Materials. We conclude from these results that in most
1. As \( \omega \) is largest at \( \omega = 1 \), but reduces as \( \omega \) deviates from 1. As \( \omega \) further varies, the QFI increases at some values of \( \omega \) which may be due to the geometric relationship of the phase that corresponding to those \( \omega \) values and the phase at \( \omega = 1 \). In any case, these QFI values are consistently lower than the value at \( \omega = 1 \). An obvious way to improve the QFI is to generate new optimal control sequences for each value of \( \omega \) from GRAPE, but this is costly as the computational complexity scales as \( O(N^3) \).

With A3C we have an efficient solution to this problem. We can train the neural network at \( \omega = 1 \), and use this particular network to generate control sequences for different \( \omega \) values. The neural network is only trained at \( \omega = 1 \). However, the trained neural network works for a broad range of parameter values. There is no need to re-train the neural network with the updated estimation of the parameter. The computational cost is thus simply \( O(N) \) so it is much more efficient than generating new sequences with GRAPE. These results from A3C are shown in the left column of Fig. 3 as blue solid lines which represents the best-performing sequence from 100 trials generated from the trained neural network. For \( \Delta T = 0.1 \) [Fig. 3(a)], although the QFI in the training \( \omega = 1 \) is slightly lower for A3C than that of GRAPE, A3C demonstrates higher transferability as the QFI decreases slowly when \( \omega \) deviates from 1. For \( \Delta T = 1 \) [Fig. 3(c)], the QFI of A3C is consistently higher than GRAPE except a narrow range of \( \omega_0 \) around 0.65.

To further reveal the transferability of different methods, we consider the measurement in an ensemble with \( \omega_0 \) uniformly distributed in \([1 - \Delta \omega, 1 + \Delta \omega]\). The performance of the quantum parameter estimation is therefore given by the average \( F(T)/T \),

\[
\langle F(T)/T \rangle = \frac{1}{2\Delta \omega} \int_{1-\Delta \omega}^{1+\Delta \omega} F(T)/T \, d\omega. \tag{6}
\]

These results are shown in the right column of Fig. 3, which are averages of the data in the corresponding panels in the left column. As seen from Fig. 3(b) \((\Delta T = 0.1)\), \( \langle F(T)/T \rangle \) for GRAPE is high at small \( \Delta \omega \) but drops quickly as \( \Delta \omega \) is increased. On the contrary, \( \langle F(T)/T \rangle \) for A3C is lower than that for GRAPE at small \( \Delta \omega \), but decays much more slowly. As a consequence, \( \langle F(T)/T \rangle / T \) for A3C exceeds that for GRAPE beyond \( \Delta \omega \gtrsim 0.22 \). This result indicates that for measurements involving a reasonably varying parameter, A3C demonstrates higher transferability than GRAPE. For \( \Delta T = 1 \), the results of A3C always exceed GRAPE as seen from Fig. 3(d). The result for A3C decays much more slowly than that for GRAPE, in consistency with the \( \Delta T = 0.1 \) case.

### C. Spontaneous Emission

A process involving the spontaneous emission is described by the Lindblad master equation [11]:

\[
\partial_t \rho(t) = -i \left[ \hat{H}(t), \rho(t) \right] + \gamma_+ \left[ \hat{\sigma}_- \rho(t) \hat{\sigma}_+ - \frac{1}{2} \{ \hat{\sigma}_- \hat{\sigma}_+, \rho(t) \} \right] + \gamma_- \left[ \hat{\sigma}_+ \rho(t) \hat{\sigma}_- - \frac{1}{2} \{ \hat{\sigma}_+ \hat{\sigma}_-, \rho(t) \} \right], \tag{7}
\]

where \( \hat{\sigma}_\pm = (\hat{\sigma}_1 \pm i \hat{\sigma}_2)/2 \) and \( \hat{H} \) is defined as Eq. (5). The dephasing rates are taken as \( \gamma_+ = 0.1, \gamma_- = 0 \) throughout our discussion.

Fig. 4 shows numerical results on QFI with spontaneous emission. Fig. 4(a)-(c) are for \( \Delta T = 0.1, T = 10 \), and Fig. 4(d)-(f) show calculations with a larger time
FIG. 4: Quantum parameter estimation under spontaneous emission using square pulses. (a)-(c): results for $\Delta T = 0.1$, $T = 10$. (d)-(f): results for $\Delta T = 1$, $T = 20$. (a), (d) show the learning procedure. (b) and (e) show $F(t)/t$ for one of the best training results selected from (a) and (d) respectively. (c) and (f) show the pulse profiles corresponding to (b) and (e).

step $\Delta T = 1$, $T = 20$. Fig. 4(a), (d) [left column] show the A3C training processes, in which the results from GRAPE are indicated as orange dotted line for reference. We see that “A3C+PPO” converges faster, and both A3C and “A3C+PPO” saturate to values slightly lower than GRAPE. Again, one of the best performing control is picked out and the corresponding $F(t)/t$ and pulse profiles are shown in the middle and right column respectively. From Fig. 4(b), (e) we see that for the best result from A3C, the QFI is lower than, but comparable to results from GRAPE.

As in Sec. II B we consider the transferability of different methods in a situation involving $\omega_0$ that distributes uniformly in a range. Again, we use GRAPE to obtain optimal control sequences for $\omega_0 = 1$ and apply that to other values. For A3C, we trained the neural network at $\omega_0 = 1$; the resulting sequence is then used to obtain an estimate of the true $\omega_0$ value. A new sequence is then generated using the neural network already trained at $\omega_0 = 1$ with the estimated $\omega_0$. The best-performing results out of 100 A3C outputs are shown as the blue solid lines in Fig. 5, while the results from GRAPE are shown as the orange dotted lines. The left column of Fig. 5 shows $F(T)/T$ as functions of $\omega_0$ for two $\Delta T$ values. In both cases, the GRAPE method outperforms A3C in a narrow neighborhood around $\omega_0 = 1$, but its QFI decreases substantially as $\omega_0$ further deviates. On the other hand, A3C exhibits great transferability: for $\Delta T = 0.1$ the QFI does not decrease until $\omega_0$ is reduced to $\omega_0 \lesssim 0.6$, while for $\Delta T = 1$ the QFI remains approximately the same for the entire range of $\omega_0$ considered. The average $F(T)/T$ in the range $[1 - \Delta \omega, 1 + \Delta \omega]$ are shown in the right column of Fig. 5. In Fig. 5(b), A3C outperforms GRAPE when $\Delta \omega \gtrsim 0.22$, while in Fig. 5(d), A3C outperforms GRAPE in an even larger range $\Delta \omega \gtrsim 0.07$.

Overall we conclude that in the case of spontaneous emission, the A3C algorithm provides comparable results to GRAPE, although it cannot give higher QFIs. Nevertheless, A3C has much greater transferability than GRAPE, as is consistent with Sec. II B.

D. Sequences with Gaussian Pulses

For all results shown above, the control sequences involve square pulses only. In practical experiments, shaped pulses are sometimes used. Therefore in this section we consider Gaussian pulses as an example. The total time $T$ is still divided into smaller pieces with $\Delta T$. However, at the $j$th piece the piecewise constant pulse is replaced by a Gaussian centering on that piece and truncated on the ends:

$$u^{(j)}(t) = A^{(j)} \exp \left\{- \frac{\left( t - t^{(j)} \right)^2}{\sigma^{(j)}_g} \right\},$$  \hspace{1cm} (8)

where $A^{(j)}$ indicates the amplitude and $\sigma^{(j)}_g$ the flatness of the pulse. We note that pulse sequences involving non-boxcar pulses are extremely difficult to treat using GRAPE, but we demonstrate here that with A3C method it is natural to accommodate different pulse shapes.
FIG. 5: Transferability of the control under spontaneous emission. (a), (c): $F(T)/T$ v.s. $\omega_0$ for three different methods. Note that the results from the GRAPE method are obtained using the pulses generated for $\omega_0 = 1$ only, while those from A3C are obtained using a neural network trained at $\omega_0 = 1$. (b), (d): average $F(T)/T$ in a range $[1 - \Delta \omega, 1 + \Delta \omega]$ corresponding to the results of (a) and (c) respectively. (a), (b): $\Delta T = 0.1$, $T = 10$; (c), (d): $\Delta T = 1$, $T = 20$.

In Fig. 6 we show A3C results using Gaussian pulses and compare them to GRAPE results using square pulses. Fig. 6(a)-(c) show results under dephasing dynamics with $\vartheta = \pi/4$, and Fig. 6(d)-(f) results under the spontaneous emission. In both cases $\Delta T = 1$, $T = 10$. For dephasing dynamics, our best results from A3C outperform GRAPE, as is also the case for square pulses generated by A3C. For spontaneous emission, our best performing result has a QFI value slightly lower than those from GRAPE with square pulses, but their values are very close. These results indicate that A3C method can naturally accommodate pulses other than square shape, which is also a key advantage over GRAPE.

### III. CONCLUSION

To summarize, RL, in particular the A3C algorithm, is capable of finding the control protocol that enhances QFI in a way comparable to the traditionally-used GRAPE method, and is in certain situations superior than GRAPE, e.g. for pulse sequences with larger time steps. Moreover, RL can accommodate non-boxcar pulse shapes, which would be otherwise difficult for GRAPE. Nevertheless, the key advantage afforded by RL is the transferability, namely the neural network trained for one estimated parameter value can efficiently generate pulse sequences that provide reasonably enhanced QFI for a broad range of parameter values, while in order to achieve the same level of QFI the GRAPE algorithm has to be applied in full each time with a new parameter estimation. Our results therefore suggest that RL-based methods can be powerful alternatives to commonly used gradient-based ones, capable to find control protocols that could be more efficient in practical quantum parameter estimation.

### Acknowledgements

This work is supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (Grant Nos. CityU 21300116, CityU 11303617, CityU 11304018, CUHK 14207717), the National Natural Science Foundation of China (Grant Nos. 11874312, 11604277, 11729402, 11574238), the Guangdong Innovative and Entrepreneurial Research Team Program (Grant No. 2016ZT06D348), and the Key R&D Program of Guangdong province (Grant No. 2018B030326001).

H.X. and J.L. contributed equally to this work.

[1] M. I. Kolobov, Rev. Mod. Phys. 71, 1539 (1999).
[2] L. Lugliato, A. Gatti, and E. Brambilla, J. Opt. B-Quantum Semicl. Opt. 4, S176 (2002).
[3] P. A. Morris, R. S. Aspden, J. E. Bell, R. W. Boyd, and M. J. Padgett, Nat. Commun. 6, 5913 (2015).
[4] W. Roga and J. Jeffers, Phys. Rev. A 94, 032301 (2016).
[5] M. Tsang, R. Nair, and X.-M. Lu, Phys. Rev. X 6, 031033 (2016).
[6] V. Giovannetti, S. Lloyd, and L. Maccone, Nat. Photonics 5, 222 (2011).
[7] H. Yuan and C.-H. F. Fung, Phys. Rev. Lett. 115, 110401 (2015).
[8] H. Yuan, Phys. Rev. Lett. 117, 160801 (2016).
[9] S. Pang and A. N. Jordan, Nat. Commun. 8, 14695 (2017).
[10] S. Pang and T. A. Brun, Phys. Rev. A 90, 022117 (2014).
[11] J. Liu and H. Yuan, Phys. Rev. A 96, 012117 (2017).
[12] J. Liu and H. Yuan, Phys. Rev. A 96, 042114 (2017).
[13] J. Yang, S. Pang, and A. N. Jordan, Phys. Rev. A 96, 020301 (2017).
[14] M. Naghiloo, A. N. Jordan, and K. W. Murch, Phys. Rev. Lett. 119, 180801 (2017).
[15] P. Sekatski, M. Skotiniotis, J. Kolodynski, and W. Dur, Quantum 1, 27 (2017).
[16] J. M. E. Fra"isse and D. Braun,
FIG. 6: Quantum parameter estimation using Gaussian pulses as building blocks for A3C. (a)-(c): dephasing dynamics with \( \vartheta = \pi/4 \). (d)-(f): spontaneous emission. (a), (d) show the learning procedures. (b), (e) show \( F(t)/t \) for the best training results selected from each case. (c), (f) show the Gaussian pulse profiles, respectively. Note that the GRAPE results shown here use square pulses. Parameters: \( \Delta T = 1, T = 10 \).

Phys. Rev. A 95, 062342 (2017).

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Nature 518, 529 (2015).

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driesche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Nature 529, 484 (2016).

[19] A. A. Melnikov, H. P. Nautrup, M. Krenn, V. Dunjko, M. Tiersch, A. Zeilinger, and H. J. Briegel, Proc. Natl. Acad. Sci. U.S.A. 115, 1221 (2018).

[20] X.-M. Zhang, Z.-W. Cui, X. Wang, and M.-H. Yung, Phys. Rev. A 97, 052333 (2018).

[21] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, Phys. Rev. X 8, 031084 (2018).

[22] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction (MIT press, 2018).

[23] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, J. Magn. Reson. 172, 296 (2005).

[24] C. W. Helstrom, Quantum detection and estimation theory (Academic press, 1976).

[25] A. Holevo, Probabilistic and Quantum Aspects of Quantum Theory (North-Holland, Amsterdam, 1982).

[26] D. Petz and C. Ghinea, in Quantum Probability and Related Topics (2010) pp. 261–281.

[27] S. L. Braunstein and C. M. Caves, Phys. Rev. Lett. 72, 3439 (1994).

[28] S. L. Braunstein, C. M. Caves, and G. Milburn, Ann. Phys. 247, 135 (1996).

[29] I. Harvey, in Advances in Artificial Life. Darwin Meets von Neumann, edited by G. Kampis, I. Karsai, and E. Szathmáry (Springer Berlin Heidelberg, Berlin, Heidelberg, 2011) pp. 126–133.

[30] S. E. Sklarz and D. J. Tannor, Phys. Rev. A 66, 053619 (2002).

[31] J. P. Palao and R. Kosloff, Phys. Rev. A 68, 062308 (2003).

[32] S. Machnes, U. Sander, S. J. Glaser, P. de Fouquieres, A. Gruslys, S. Schirmer, and T. Schulte-Herbrüggen, Phys. Rev. A 84, 022305 (2011).

[33] D. M. Reich, M. Ndong, and C. P. Koch, J. Chem. Phys. 136, 104103 (2012).

[34] M. H. Goerz, K. B. Whaley, and C. P. Koch, EPJ Quantum Technol. 2, 21 (2015).

[35] P. Doria, T. Calarco, and S. Montangero, Phys. Rev. Lett. 106, 190501 (2011).

[36] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, arXiv preprint arXiv:1602.01783v2 (2016).

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, arXiv preprint arXiv:1707.06347v2 (2017).
S-I. REINFORCEMENT LEARNING

The Reinforcement Learning (RL) framework is schematically shown in Fig. 1(a). The key ingredients of
the RL process include a state space $\mathcal{S}$, an action space $\mathcal{A}$, and a reward $\mathcal{R}$ [22]. In the RL procedure, an
agent at state $s_t \in \mathcal{S}$ chooses an action $a_t \in \mathcal{A}$ according to a
probabilistic policy $\pi_\theta(a_t|s_t)$ where $\theta$ represents parameters
of the policy. For example, when using a neural network to represent the policy, $\theta$ represents the weights and
biases of the neural network. The action $a_t$ results in a new state $s_{t+1}$ according to which the agent receives
a numerical reward $r_{t+1} \in \mathcal{R}$. For a given optimization
problem, one encapsulates the goal of the problem into
the calculation of the rewards, as well as relevant con-
straints in the available states and actions. In practice,
the reward for a given state is not only related to its im-
mediate next step, but several steps in its future, so the
total discounted reward received at a given step.

$$R_j = \sum_{k=1}^{\infty} \alpha^{k-1} r_{j+k},$$
(S-1)

where $\alpha \in (0, 1]$ is the reward decaying rate indicating
the relative weight between adjacent steps in calculating
the total discounted reward received at a given step.
When $\alpha = 1$ the rewards from all future steps contribute
equally, while when $\alpha \to 0$ only the immediate next step
provides the major contribution. Then, the probability
that the agent takes certain action is enhanced or sup-
pressed, according to the value of the total discounted
reward. After sufficient iterations of training, the agent
learns the optimal actions to take in order to maximize
the total discounted reward, thereby giving an optimal so-
lution to the desired problem.

In the RL procedure, the exploration of the agent
in the state and action spaces is summarized into a
sequence $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_k, a_k, r_{k+1}, \ldots$, called a
trajectory. To figure out what is the best action to take
at state $s$, we define the state-action value function,

$$Q^\pi(s, a) = \mathbb{E}[R_j | s_j = s, a_j = a],$$
(S-2)

where the expectation includes discounted rewards of all
the trajectories after taking the action $a$ at the state $s$ in
the $j$th step of the trajectory, provided that the policy $\pi$
is observed thereafter [22]. We also define the value of a
state to evaluate the likelihood that a given state would
lead to a higher reward,

$$V^\pi(s) = \mathbb{E}[R_j | s_j = s],$$
(S-3)

where the expectation includes discounted rewards of all
the trajectories starting from the state $s$ in the $j$th step,
provided that the policy $\pi$ is followed thereafter [22].

An RL policy $\pi$ is declared “optimal” when the actions
selected by the policy in each state are such that the
resulting expectation value of discounted rewards for all
states $s \in \mathcal{S}$ is no less than that from any other policy $\pi'$, i.e., $V^\pi(s) \geq V^{\pi'}(s)$ [22]. Corresponding to the optimal
policy $\pi_\theta(a|s)$, the optimal value functions are

$$Q^*(s, a; \theta_v^*) = \max_{\pi} Q^\pi(s, a; \theta_v),$$
(S-4)

$$V^*(s; \theta_v^*) = \max_{\pi} V^\pi(s; \theta_v),$$
(S-5)

where the notations $\theta_v$ and $\theta_v^*$ represent optimal choices
of the neural network parameters for the policy and value
functions. If the optimal value functions are known,
the RL agent simply chooses the action $a_j$ that has the
largest state-action value $Q^*(s_j, a_j)$ in state $s_j$. Alter-
natively, at state $s_j$ one may choose the next state $s_{j+1}$
that has the largest state value $V^*(s_{j+1})$. Thus, there
are two ways for an RL algorithm to solve an optimiza-
tion problem: the agent either learns the optimal policy,
or if the policy is otherwise specified, the optimal value
functions [S1]. The two methods are discussed below.

In the so-called value-based method, the RL agent
learns optimal value functions. The state value function
and state-action value function are solved iteratively us-
ing the Bellmann equations,

$$Q^*(s, a; \theta_v^*) = r + \alpha \max_{a'} Q^*(s', a'; \theta_v^*),$$
(S-6)

$$V^*(s; \theta_v^*) = \max_{a'} V^*(s'; \theta_v^*),$$
(S-7)

where $a'$ represents all possible actions in the next state $s'$ [22]. We define the loss functions as

$$L_Q = \left[ R_j^n + \alpha^n \max_{a'} Q^\pi(s_{j+n}, a) - Q^\pi(s_j, a) \right]^2,$$
(S-8)

$$L_V = \left[ R_j^n + \alpha^n V^\pi(s_{j+n}) - V^\pi(s_j) \right]^2,$$
(S-9)

where $R_j^n = \sum_{k=1}^{n} \alpha^{k-1} r_{j+k}$ is called the “n-step” return
[22]. We take the $\epsilon$-greedy policy commonly used in deep
Q-learning network [17, 20] as an example. Under this
policy, the neural network does either of the two things
at state $s_j$: with probability $1 - \epsilon$ the network takes the
action $a_j$ that maximizes $Q^\pi(s_j, a_j)$, or with probability
$\epsilon \in (0, 1]$ an action is randomly chosen. The latter me-
chanism encourages the agent to explore a wider range in
the search space to reach a globally optimal solution. In
practice, $Q^\pi(s, a)$ in the loss function Eq. (S-8) is the pre-
diction by the neural network and $r + \alpha \max_{a'} Q^\pi(s', a')$
is calculated from the trajectories of the RL agent. The
training procedure of the neural network is essentially
minimization of the loss function. As a result, the trained
neural network gives optimal state-action values.

We note that in the value-based algorithm, the policy
is fixed, and only the value functions are updated, which
may not be sufficient to find a globally optimal solution.
[S1]. More importantly, the way of storing the action space and trajectories have assumed that the actions are discrete, and it becomes far more complicated to treat problems with continuous actions, as is the case of control fields. As shall be discussed below, the policy-based algorithm is most suitable for our problem.

The policy-based algorithm directly updates the policy parameters \( \theta \) without the need of storing a large amount of RL trajectories. A typical form of the loss function is defined as \[ S-10 \]

\[
L = - \sum_j \log (\pi_\theta(a_j|s_j)) A_j, \quad (S-10)
\]

where \( A_j \) is the advantage function,

\[
A_j = R_j - b(s_j), \quad (S-11)
\]

which evaluates the advantage of the chosen trajectory, with the baseline function \( b(s_j) \), normally being the estimated state value function, that reduces the variance and speeds up the learning process [22, 36]. When the value of \( A_j \) is large for an action \( a_j \), minimizing \( L \) increases \( \pi_\theta(a_j|s_j) \), implying that the probability to choose the action \( a_j \) in state \( s_j \) is increased.

In our problem, a quantum state is completely described by the density matrix \( \hat{\rho}^{(j)} \) for each time step \( j \). Therefore our state in the RL procedure is defined using elements of the density matrix as

\[
s_j = (\text{Re}(\hat{\rho}_{00}^{(j)}), \text{Im}(\hat{\rho}_{00}^{(j)}), \text{Re}(\hat{\rho}_{10}^{(j)}), \text{Im}(\hat{\rho}_{10}^{(j)}), \text{Re}(\hat{\rho}_{01}^{(j)}), \text{Im}(\hat{\rho}_{01}^{(j)}), \text{Re}(\hat{\rho}_{11}^{(j)}), \text{Im}(\hat{\rho}_{11}^{(j)})). \quad (S-12)
\]

Our action space is formed by a set of control fields \((u_1^{(j)}, u_2^{(j)}, ..., u_p^{(j)}) \equiv a_j \in A\), which steers our quantum state \( s_j \) to \( s_{j+1} \) according to the master equation Eq. (2). Evaluation of the new state \( s_{j+1} \) and the agent obtains the single step reward:

\[
r_{j+1} = \begin{cases} 
F(j+1) - \eta F_0(j+1) & j + 1 < N,
F(j+1) - \eta F_0(j+1) - C & j + 1 = N,
\end{cases} \quad (S-13)
\]

where \( F \) and \( F_0 \) are the corresponding QFI from Eq. (3) with and without control, respectively. \( \eta \geq 1 \) and \( C \geq 1 \) are constant parameters used in the training process. \( \eta \) ensures a non-zero reward to the agent in case the RL agent would apply \( u_{1,2,3}(t) = 0 \), while \( C \) gives an extra significance to the last evolution step. After an episode of training, the action sequence in each trajectory constitutes the control field. We also note that our choice of the reward function is not unique.

S-II. ACTOR-CRITIC ALGORITHM

The Actor-Critic algorithm combines the advantages of policy-based and value-based methods. Fig. 1(b) illustrates the basic procedure of the Actor-Critic algorithm. Two neural networks are involved: the actor network governing the policy that chooses actions, and the critic network managing the value functions, which in turn changes the baseline function used in further policy-making [22]. More specifically, the state value \( V^\pi(s) \) generated by the critic network is plugged into Eq. (S-11),

\[
A_j = R^n_j + \alpha^\pi V^\pi(s_{j+n}) - V^\pi(s_j). \quad (S-16)
\]

Note that the “n-step” return is used instead of \( R_j \) so that only the \( n \) future steps are involved. This is the key distinction from the policy-based method [22]. In the training process, the actor and the critic networks minimize the loss function simultaneously. We update the critic network through Eq. (S-9) while the actor network is trained through Eq. (S-10) using the advantage function defined by Eq. (S-16).

In order to improve the efficiency of the learning process, a parallelized version of Actor-Critic algorithm called A3C, short for Asynchronous Advantage Actor-Critic [36], is implemented in our calculation.

S-III. ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC ALGORITHM

The key structure of Asynchronous Advantage Actor-Critic (A3C) is sketched in Fig. S1. The desired policy and value functions are generated by the neural network (left column in Fig. S1), called the “global” network. The
neural network is composed of the state value network $V^\pi(s)$ (orange color), the policy network $\pi(a|s)$ (green color) and the fully-connected linear layers (blue color). At the beginning of the training process, we made $N_{\text{env}}$ copies of the global network, called “local” networks. Then, each of the local networks is allowed to run in independent RL environments, in which the RL agents, called the “local” agents, optimize policies and value functions via gradients with respect to the loss functions. At the end of a training episode for each parallel RL procedure, the local agent uploads the accumulated gradient to update the global network. Then, the updated global network is downloaded back to the local environment, starting a new episode with the environment properly reset. Note that in the entire process, all local agents act independently, which is why the algorithm is asynchronous \cite{36, S4}.

We now give details of our implementation of the A3C algorithm. The RL states are first fed through 4 hidden layers, each composing 200 ReLU units \cite{S5}. The resulting outputs are then passed to both the value and policy networks. The value network is constructed by one hidden layer with 200 ReLU units and one fully-connected linear layer outputting a real number as the state value. The policy network has one hidden layer with 200 ReLU units and two fully-connected linear layers as output layers. The outputs are six real numbers $\mu_k, \sigma_k^Q$, $k = 1, 2, 3$ forming three normal distributions $N(\mu_k, \sigma_k^Q)$. Here, $\mu_k$ is modified by the SoftShrink(\lambda) activation function with $\lambda = 0.25$ and $\sigma_k^Q$ is modified by the SoftPlus activation function \cite{S5}. The continuous actions $u_k$ are randomly sampled from those normal distributions.
We use the differentiation of the normal distribution as the entropy regularization term, \(-\frac{1}{2}(\log(2\pi\sigma^2)+1)\), to encourage the agent to explore the entire search space. We use the RMSProp optimizers with shared parameters that are updated asynchronously among parallel environments [36]. We keep the choice of hyper-parameters which are listed in the left column of Table S-I similar to those used in [36]. The pseudocode for A3C can be found in [36]. Next we will discuss an optimized version of the code, i.e. with Proximal Policy Optimization (PPO) algorithm [37, 86].

Generally, optimization with the logarithm of the policy gradient leads to large policy updates which, in some cases, makes the learning process unstable. The Proximal Policy Optimization (PPO) algorithm replaces the logarithm in Eq. (S-10) with the probability ratio between the old and the new policy:

\[
\nu_j(\theta) = \frac{\pi_\theta(a_j|s_j)}{\pi_{\theta,old}(a_j|s_j)}, \quad (S-17)
\]

and the loss function is also truncated at certain values of the probability ratio [37]. Algorithm 1 shows the pseudocode for the A3C algorithm utilizing the PPO strategy. In this algorithm, we replace the global RMSProp optimizer with the thread-specified Adam optimizers [S5]. The right column of Table S-I lists the hyper-parameters in the A3C algorithm with PPO strategy.

We have used PyTorch [S5] to implement the algorithms and QuTip [S7, S8] to obtain numerical solutions of Eqs. (2)-(3). We also note that practically, when \(\Delta T = 1\), we have to set smaller learning rates, gradient norm, entropy weight and \(N_{\text{max}}^{\text{ppo}}\).

### S-IV. ADDITIONAL RESULTS ON DEPHASING DYNAMICS

In the main text, we have provided results of quantum parameter estimation under dephasing dynamics along a chosen axis in Fig. 2, i.e. \(\vartheta = \pi/4\). Here, we present results along two other axes: parallel dephasing (\(\vartheta = 0\)) and transverse dephasing (\(\vartheta = \pi/2\)). In Fig. S2, the training process is shown in the upper row, \(F(T)/T\) v.s. \(\omega_0\) the middle row and the average \(F(T)/T\) in [1 − \(\Delta\omega, 1 + \Delta\omega\)] in the bottom row. For parallel dephasing, our results are very similar to \(\vartheta = \pi/4\) results shown in the main text, namely \(F(T)/T\) calculated from A3C is lower than that from GRAPE only in a narrow range of \(\Delta\omega\). For \(\Delta T = 0.1\), A3C outperforms GRAPE when \(\Delta\omega \gtrsim 0.15\), while for \(\Delta T = 1\), A3C is better than GRAPE in a wider range, \(\Delta\omega \gtrsim 0.05\). For transverse dephasing, the situation is slightly more complicated (note that analytical solutions [11] are provided as references). When \(\Delta T = 1\), results from GRAPE has very low \(F(T)/T\), thus A3C always outperforms GRAPE. However, for \(\Delta T = 0.1\), A3C does not possess considerable advantages. For \(0 \leq \Delta\omega \leq 0.4\), the A3C results have lower \(F(T)/T\) than GRAPE, albeit being very close. For \(\Delta\omega \geq 0.4\), the A3C results is only slightly higher than GRAPE. These calculations therefore suggest that the transferrability of our method is superior as compared to GRAPE in most situations, in particular for cases with larger time step (\(\Delta T\)). Nevertheless, in some situations, usually associated with smaller \(\Delta T\), our method would not provide considerable improvement. One therefore has to be judicious in choosing appropriate methods for a specific problem. For example, if transferrability is not desired, GRAPE may be more appropriate for pulse sequences with smaller time steps. On the other hand, if pulse sequences have larger time steps, or non-boxcar pulse shape is desired, or transferrability becomes important in the problem, the A3C method is desired.

References:

[S1] V. R. Konda and J. N. Tsitsiklis, SIAM J. Control Optim. 42, 1143 (2003).
[S2] R. J. Williams, Machine learning 8, 229 (1992).
[S3] D. Seita, “Actor-Critic methods: A3C and A2C” https://danieltakeshi.github.io/2018/06/28/a2c-a3c/, accessed April 19, 2019.
[S4] We note that “asynchronism” is not a necessary condition since one may train the agents synchronously using batched experiments in a parallel fashion [S9].
[S5] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, in NIPS-W (2017).
[S6] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, arXiv preprint arXiv:1707.02286v2 (2017).
[S7] J. R. Johansson, P. D. Nation, and F. Nori, Comput. Phys. Commun. 183, 1760 (2012).
[S8] J. R. Johansson, P. D. Nation, and F. Nori, Comput. Phys. Commun. 184, 1234 (2013).
[S9] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, arXiv preprint arXiv:1708.05144v2 (2017).
Supplementary Figure S2: Quantum parameter estimation under dephasing dynamics along two different axes using square pulses. First column: parallel dephasing ($\theta = 0$) and $\Delta T = 0.1$. Second column: parallel dephasing with $\Delta T = 1$. Third column: transverse dephasing ($\theta = \pi/2$) and $\Delta T = 0.1$. Fourth column: transverse dephasing with $\Delta T = 1$. The upper row shows the learning procedure, i.e. $F(T)/T$ as functions of training epochs. The middle row compares $F(T)/T$ v.s. $\omega_0$ for different methods. The bottom row shows average $F(T)/T$ in a range $[1 - \Delta \omega, 1 + \Delta \omega]$. The total times $T$ are indicated.