florl: A Framework for Federated Reinforcement Learning in Robotics

Hao Xiang Li
Department of Computer Science
University of Cambridge
hxl23@cam.ac.uk

Zejun Chen
Department of Computer Science
University of Cambridge
zc344@cam.ac.uk

Abstract

Distributed cloud robotics is a crucial learning paradigm with widespread applications. In connection, Federated Reinforcement Learning (FRL) is a promising research direction for learning under privacy-sensitive constraints. However, engineering complexities involving distributed heterogeneous systems hinder further progress. Our work addresses these challenges by bridging existing frameworks to construct a practical FRL pipeline for robotics applications, thus creating a platform for future research. We present `florl` (*floral*), an FRL library leveraging Federated Learning framework `flower`. In addition, we propose an architecture for FRL in robotics and demonstrate the successful coordination between `robot operating system 2`, `florl` and `flower`. Finally, we conduct experiments to validate the effectiveness of our framework in improving learning performance.

1 Introduction

Data privacy is becoming increasingly prevalent in the field of robotics, most recently triggering the European Commission's blocking of an acquisition deal between Amazon and iRobot worth US $1.4 billion [1]. Securely learning from private data remains challenging. Under the constraints of privacy preservation, Federated Reinforcement Learning (FRL) has shown promising potential as a distributed collective robot learning paradigm, attracting rising research interest in recent years. By sharing parameter weights instead of data samples with a central server, individual agents can reap performance generalisation gains with reduced loss of privacy [7, 10, 25]. However, from a software engineering and systems perspective, combining Cloud Robotics, Reinforcement Learning (RL) and Federated Learning (FL) is an innately complex task: practitioners must consider the influence of widespread system heterogeneity, massively distributed coordination and the technical sophistication needed to manage such systems. Our project makes the following contributions.

1. We propose `florl`, a bridge between FL and RL by interfacing with existing frameworks. `florl` simplifies the FRL boilerplate, enabling researchers to focus on the theory rather than engineering.
2. We develop a pipeline and architecture for FRL application within robotics as a proof-of-concept.
3. We experimentally validate the effectiveness of our platforms and examine various key aspects relevant to FRL. Our results demonstrate the difficulties and advantages of FRL both qualitatively and quantitatively.
4. All our code is released. We hope our work can serve as a starting platform for further research in this field.

University of Cambridge Federated Learning Course Report.

The remainder of the report is organised as follows. In section 2, we survey related work and present the necessary preliminaries. Next in section 3, we motivate and describe our design choices. In section 4, we formulate a set of experiments to evaluate both our framework, as well as general FRL. The results are described in section 5. Finally, we summarise our findings and propose future improvements in section 6.

2 Background

2.1 Federated Learning

Federated Learning (FL) is an approach for decentralised collaborative machine learning. In FL, participating nodes train locally on private data and communicate model parameters exclusively. For example, nodes using FL can learn using sensitive medical data [20] or when constrained by regulations [5]. Suppose there exist K nodes, each with local loss function $L_i(\boldsymbol{x}_j, \boldsymbol{y}_j; \boldsymbol{w})$ and a set of samples $|S_i| = n_k$. Then the training objective can be phrased as [16]

$$\min_{\boldsymbol{w}} \sum_{i=1}^{K} \sum_{j=1}^{n_k} L_i(\boldsymbol{x}_j, \boldsymbol{y}_j; \boldsymbol{w}) \tag{1}$$

One particularly commonplace FL paradigm is the synchronous server-client framework: a central *server* coordinates training across multiple *rounds*; in each round, a set of *clients* receives updated global parameters, apply a local update algorithm and return the updated parameters for centralised aggregation - distinguishing works in this field include `FedAvg` [16] (model averaging) and `FedProx` [12] (heterogeneous data).

Our work builds upon `flower` [2], a widely used and configurable federated learning framework.

2.2 Reinforcement Learning

Reinforcement Learning (RL is a machine learning paradigm uniquely familiar to how humans and animals learn: via the continuous interaction and feedback with the surrounding environment. Prior works have successfully applied RL in the context of Atari games [17], robotic manipulation [8] and robotic navigation [13].

In the context of RL, we denote the sets of states S - representing the sensor observation of robots - and the set of actions A - representing activities of available actuators. At each step, an *agent* first receives observation $\boldsymbol{s} \in S$ and then applies action from parameterized *policy* π_θ with probability $\pi_\theta(a|s)$. Next, the *environment* evolves according to transition probabilities P and returns some reward $r = R(s, a, s_1)$. Subscripts denote the time-step. The learning objective is to maximise the expected discounted reward for initial state distribution $\boldsymbol{s}'_0$ and discount factor $\gamma \in [0, 1]$

$$\max_{\theta} \mathbb{E}_{s_0 \sim \boldsymbol{s}'_0, \pi_\theta} \left[\sum_{i=0}^{\infty} \gamma^i \sum_{s \in S} p(s_i = s|s_0; \pi_\theta) \sum_{a \in A} \pi_\theta(a|s) \sum_{s' \in S} P(s'|s, a) R(s, a, s) \right] \tag{2}$$

gained by an agent within the environment. Nevertheless, many current methods [8, 19] for large-scale distributed training share transition data in the training process, which is unfeasible in a privacy-sensitive context. Others [4] focus on gaining the benefits of parallelised compute, rather than privacy within an enforced distributed setting. Thus, there exists a strong motivation for combining FL and RL pipelines.

Our project interfaces with `gymnasium` [22] and `kitten` [11] for RL. `gymnasium` is a standard environment interface, and `kitten` provides modular toolbox implementations of RL algorithms.

2.3 Cloud Robotics

Cloud robotics broadly refers to the architecture of robotics benefiting from networked operations in the form of computation, communication or data collection. Specifically, our project centres on the

problem of *collective robot learning*. In this paradigm, networked communications enable distributed collaboration on sharing control policies, metrics and outcomes [9]. However, existing learning techniques often rely on experience sharing - again, this is infeasible in the privacy-sensitive setting our project considers.

Robot Operating System 2 (`ros2`) [15] is the leading middleware layer for cloud robotics. The platform provides modular, reliable, and secure integration with existing components from hardware drivers to high-level path planners. Our project is a bridge between `ros2` and FRL, thus benefiting from the suitability of `ros2` in heterogeneous robotic architectures.

2.4 Federated Reinforcement Learning for Cloud Robotics

Federated Reinforcement Learning (FRL) is a relatively new technique, yet offers a promising path towards collective robot learning. [7] studies the theoretical convergence of FRL within heterogeneous environments. [26] considers differential privacy with merged Q networks. Although we run experiments validating the results of FRL, our project primarily focuses on architectural engineering. The most closely related work is [13], which applied FRL to the lifelong learning problem in robotic navigation. Compared to their method, our project considers learning in parallel rather than iterative transfer learning. Most importantly, there is limited FRL code openly available, greatly limiting experiment reproducibility and widespread application. [14] provides a framework for general FL, including two FRL examples, but is enterprise focused and comparably difficult to extend. We believe our proposal is the first open-source FRL platform applied to robotics.

3 Design / Methods

3.1 `florl`: Federated Learning over Reinforcement Learning

`florl` is a federated reinforcement learning library enabling the rapid development and application of FRL workloads. We approach the design of `florl` as an extension of the `flower` library, thus `flower` is the only major hard dependency of this project. To be specific, the distributed training and communication infrastructure is provided by `flower`, and `florl` modifies client-server endpoints to adapt to the idiosyncrasies of an FRL workload. In this section, we identify the traits of FRL and discuss the implementation of `florl`. (See appendix, Figure 11 for design overview).

First and foremost, RL pipelines often contain an ensemble of learnt modules, including policy networks, value estimation networks, target networks, world models, normalisation parameters etc. At the time of writing, `flower` communicates `Parameter` objects, each identifying a list of tensor objects. This representation model implicitly communicates all parameter weights in each round and existing `flower Strategy` implementations do not differentiate between different tensors. Consider the following two training scenarios

1. an RL pipeline consists of a RunningMean module for reward normalisation and an Actor network as a parameterised function from states to actions. It is desired to aggregate the Actor using `FedAvg`, but update RunningMean with the standard batch update definition.
2. The federated network has clients with heterogeneous training architectures. Clients have a single Actor, Critic networks or both. Our FL algorithm must aggregate mixed `Parameter` objects.

`florl` deals with this issue by introducing a `Knowledge` abstraction as a composition of `KnowledgeShard`. Any subset of Knowledge can be serialised and deserialised to `Parameter` for interfacing with `flower` abstractions and applies to the scenarios described above. It also allows a subset of parameters to be communicated from each client to the server to reduce communication overhead. It is in our opinion that `Knowledge` may be better implemented as a core section of `flower` as its usefulness extends beyond RL.

Furthermore, we provide abstractions for `Client` and `Server` that not only interface with Knowledge but also simplify the process of running RL experiments. As mentioned previously, `florl` has soft dependencies of `kitten` and `gymnasium` with specialised abstract classes, thus substantially reducing the amount of boilerplate code for defining configurations and allowing researchers to focus on implementing parameter updates. We also write utilities to enable stateful simulations of RL clients due to the ephemeral nature of `fl.simulation.run_simulation` conflicting with inherently stateful RL.

Finally, `florl` contains concrete example implementations of FRL algorithms with custom `Server` and `Client`. See section 4.

3.2 Robotics Learning Pipeline

We begin our efforts by demonstrating the successful transfer of a federated classifier workload to `ros2`, replacing virtual simulation clients with edge clients running as `ros2` nodes in the classifier training scenario. Our initial architecture assembles as two distinct modules - a `toy_fl_publisher` replacing the data-loader and `toy_client` acting as a bridge, inheriting both a `flower Client` and `ros2 node`. The `toy_client` buffers data received in a continuous stream (`ros` topic-subscription model) and performs local training. This example serves as a starting point for complicated use cases and would demonstrate the feasibility of a `flower-ros` pipeline.

Our work extended this to a realistic FRL pipeline that can be easily transferred to real-world robotic scenarios. This is done through the separation of responsibilities in an asynchronous data collection architecture (Figure 1). By utilising the ros2 communication system and modularisation to specialised nodes for replay buffers, policies, controller and the client, we can effectively leverage different tooling (e.g the replay buffer was built in C++ for efficiency) and compute parallelisation (data collection pipelines run in parallel with training and `flower` communication).

One specific design challenge involves the coordination between `flower` and `ros2`. Starting a `ros2 Node` as a `flower` client naively deadlocks due to the conflict between synchronous `flower` callbacks and the asynchronous `ros2` scheduler. To address this, we implement two different approaches: 1) starting a proxy `flower` client in a separate thread or 2) managing the communication between the client and server by overriding with `ros2` timer callbacks. In both methods, consistency can be ensured with standard Python concurrency locks.

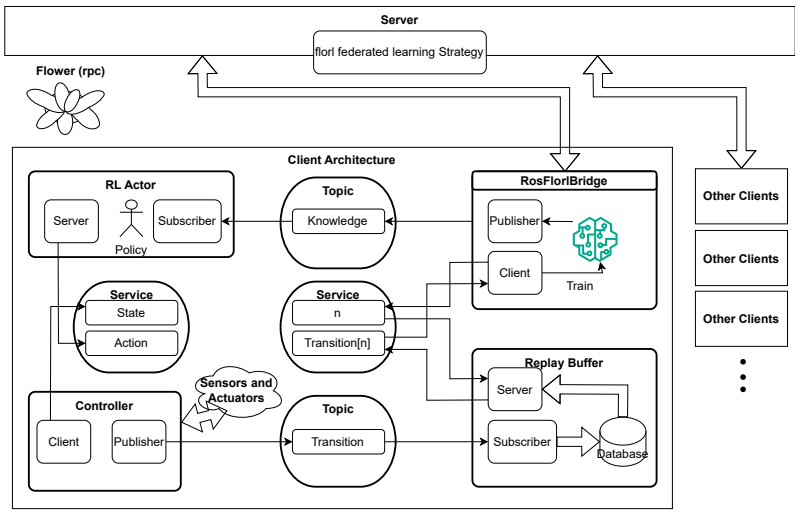


Figure 1: Architecture for federated reinforcement learning with `ros2` clients.

4 Experimental Setup and Methodology

On one hand, our experiments aim to evaluate the proposed solution's suitability as an FRL pipeline component. On the other hand, we also conduct preliminary exploration into the distinctive features of FRL to guide future research. We point the reader to .2 for notes on replicating results.

4.1 Datasets and Environments

We use [22], which, among others, provides reference implementations to classic RL environments `Pendulum` and `Acrobot` [21]. At a high level, `Acrobot` is the task of swinging a linked chain upwards and `Pendulum` is the task of applying force to an inverted pendulum until it is still and upright. These two tasks were chosen as they have low dimensional observations, require minimal compute and are relatively simple to visualise. However, they differ in a key aspect - `Pendulum` has a continuous action space, while `Acrobot`'s is discrete. Clients seed their environments with a unique integer identifier value. A modification we investigate is *reset-heterogeneity* - fixing unique s_0 for each client. We believe this modification will help shed light on the impact of heterogeneity in FRL training data, despite not impacting the theoretically optimal policy between clients. For simplicity, we term the default environment as I.I.D (Independent, identical environments with identical $\boldsymbol{s}_0$ distribution) and our adaption as Fixed-Reset.

Our toy federated classifier with `ros2` clients uses [2] `flwr-datasets` to generate partitions of the [3] MNIST handwritten digit dataset with 10 classes. The toy `ros2` FRL task trains on `gymnasium` `CartPole`, a discrete task to balance a pole on a cart.

4.2 Federated Reinforcement Learning Algorithms

We generalise from methods proposed in [7, 10] to form a category of FRL algorithms. This consists of combining a client RL algorithm with a server federation algorithm (see algorithm 1).

Algorithm 1 Combining FL and RL. Given procedures `client.train(n:int)`, which runs standard RL; `server.aggregate(`$\{\boldsymbol{w}_t^c | c \in \boldsymbol{C}\}$`)`, which aggregates the weights from a set of fit results to obtain $\boldsymbol{w}_{t+1}$ for clients in $\boldsymbol{C}$

Require: $n \in \mathbb{Z}^+$ ▷ Client frame collection number per round
Require: $m \in \mathbb{Z}^+$ ▷ Number of clients sampled per round
  procedure SERVERUPDATE
    $\boldsymbol{w}_0 \leftarrow$ initialisation
    for $t \leftarrow 1, 2, \ldots$ **do**
      $\boldsymbol{C}_t \leftarrow$ sample set of m clients
      for all $c \in \boldsymbol{C}_t$ **do**
        $\boldsymbol{w}_{t+1}^c \leftarrow$ ClientUpdate($\boldsymbol{w}_t$)
      end for
      $\boldsymbol{w}_{t+1} \leftarrow$ aggregate($\{\boldsymbol{w}_{t+1}^c | c \in \boldsymbol{C}_t\}$)
    end for
  end procedure

  procedure CLIENTUPDATE($\boldsymbol{w}_t$, n)
    $\boldsymbol{w}^c = \boldsymbol{w}_t$ ▷ $\boldsymbol{w}^c$ is the client's own set of parameters
    train(n)
    return $\boldsymbol{w}^c$
  end procedure

For the client-side `train`, we consider the update rules described in TD3 [6], QTOpt [8] and DQN [17] . $\boldsymbol{w}$ is composed of the actor network (TD3) and the critic network (TD3, QTOpt, DQN), with their respective target networks [1].

[1] Delayed versions of the networks used to stabilise training

A critic network learns the expected value gained from taking action $a \in A$ on state $s \in S$.

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{s' \in S}\left(R(s,a,s')P(s'|s,a) + \gamma \mathbb{E}_{a' \sim \pi(s')}[Q(s',a')]\right)\right] \tag{3}$$

DQN define a policy by taking $\pi(s) = \text{argmax}_{a \in A} Q(s,a)$, and QTOpt replaces argmax with the cross-entropy sampling method for an extension on continuous action spaces. In contrast, TD3 learns an actor network to implicitly replace the argmax. The actor loss is minimising

$$L_\pi(s) = -Q_\pi(s, \pi(s)) \tag{4}$$

All clients explore using ϵ-greedy, where a random action is taken with probability ϵ.

On the FL side, we consider `FedAvg` [16] and `FedProx` [12]. We assume each client is weighted equally, as the problem of importance weighting is out of the scope of this investigation. Both aggregates by

$$\boldsymbol{w}_{t+1} = \frac{1}{|\boldsymbol{C}_t|} \sum_{c \in \boldsymbol{C}_t} \boldsymbol{w}_t^c \tag{5}$$

In addition, `FedProx` adds a penalty term to the client loss `train` of $\frac{\mu}{2}||\boldsymbol{w}_t^c - \boldsymbol{w}_{t+1}^c||^2$ to ameliorate training under heterogeneity. Finally, we also consider the impact of communicating a subset of networks using `Knowledge` API capabilities. See table 1 for a list of high level configurations.

Algorithm	**RL**	**FL**	**Notes**
DQN [17]	DQN	None	Baseline
QTOpt [8]	QTOpt	None	Baseline
DQNAvg [7]	DQN	FedAvg	
QTOptAvg	QTOpt	FedAvg	
QTOptProx	QTOpt	FedProx	
TD3Avg (variants)	TD3	FedAvg	Aggregating subsets

Table 1: FRL algorithms examined

Baseline refers to standard RL, where a client trains independently on their own dataset and is not influenced by other clients. We use `kitten` to provide implementations of RL and adapt `flower` baseline implementations with `florl` abstractions for FL. These are encapsulated as modules within `florl`.

4.3 Experiment Design

We first probe the impact of FRL in stabilising and accelerating training in I.I.D environments (results 5.1, hyperparameters .4.1). Despite not communicating raw transition experience, aggregating model parameters may share useful information from exploration between clients - there are overall more transitions experienced summed over clients, thus opportunities to learn useful information. This experiment is performed by comparison runs of FRL with 5 clients against baseline, where the total rounds of training are the same (i.e. federated methods will have collected more data in total due to parallelisation). Evaluation is centralised for FRL experiments after aggregation. In this (and other) experiments we report evaluation reward - the aggregated model is transformed into a purely exploitative policy and evaluation episodes are run. To avoid different initial states across evaluations as a source of error, we run each evaluation across the same set initial states. 5 initial states to avoid edge cases where a policy may perform well on a single starting state due to luck. We also anticipate observing effects of data heterogeneity, which is a significant challenge in FRL. We aim to examine the presence and impact of *label distribution skew* 5.2, where independent clients have transitions across mismatched (s,a) distributions. This is done for both the case of I.I.D environments, and also the Fixed-Reset environments (see 4.1). We briefly investigate empirically to see `FedProx` extends to FRL .5.4.

`Knowledge` interface presents an option to partially share parameters. We investigate federating subsets of parameters for `TD3Avg` 5.3. Not only could this reduce communication costs, but we also we anticipate that practitioners would benefit from considering the role of different RL modules in deciding when and how to aggregate. Due to compute constraints, we reduce evaluation episodes to 1 at each round. `ros2` FRL pipelines will be tested and evaluated 5.4. We demonstrate convergence in both cases, thus successfully showing proof-of-concept, and scrutinise how FRL would adapt to realistic scenarios in the context of our proposed framework.

We also clarify that standard deviations reported necessarily do not accurately reflect true reward confidence bounds, due to the practically low number $(10 - 20)$ of seeding repeats used in RL experiments from the high compute cost. In particular, many rewards have upper and lower bounds that the std interval passes. Furthermore, we acknowledge a significant limitation of our experiments is the low number of clients considered - this is briefly discussed in section 5.4.

5 Results and Discussion

5.1 Centralised v Federated

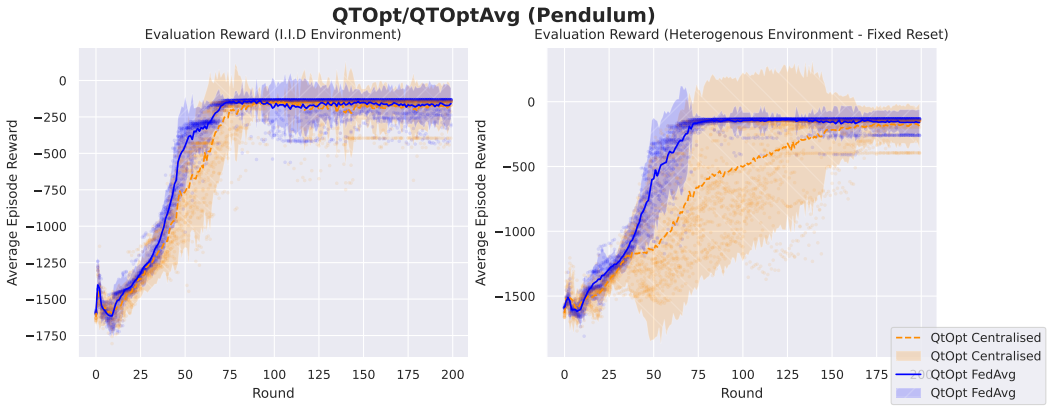
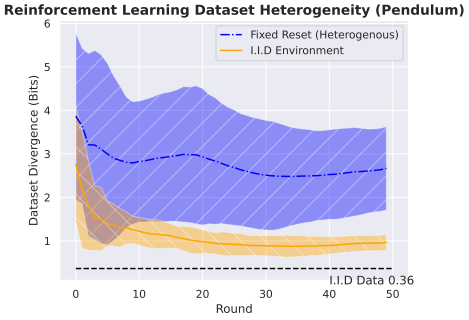


Figure 2: Federating with `QTOptAvg` (Pendulum) across 20 seeds, 5 clients, with centralised evaluation across constant 5 seeds per round. The line represents mean and shaded 1.96 standard error.

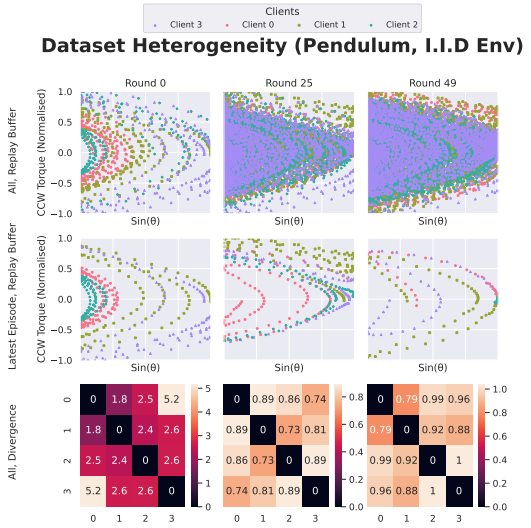
We first test the impact of federating an RL workload with `QTOptAvg` and `DQNAvg` against their respective baselines. Results of `DQNAvg` are discussed in section .5.2, however, they are generally similar to those of `QTOpt`, with significant impact on improving stability. As shown by Figure 2, FRL can take advantage of parallelised experience collection and model averaging to quickly converge to Pendulum's near-optimal solution with fewer rounds than baseline. The effect is particularly pronounced with heterogeneous Fixed-Resets, demonstrating FRL's ability to improve *generalisation*. One interesting result was the reduced variance in performance, which suggests FRL improves training stability. We conjecture this is due to two reasons. On one hand, aggregation reduces the occurrence of catastrophic loss, as the impact of a bad training round for any single client is reduced by other clients. On the other hand, federated gradients stem from different replay buffers with reduced correlation, similar to [18] and the motivation behind experience replay [17]. There is an unexpected result where evaluation performance actually has a higher variance past a certain point as a result of drops in ability. This could be due to the replay buffer aggregating too many similar states of upright balances, and is a sign of the neural network over-fitting - this effect leads us to report best reward in section 5.3, analogous to early stopping.

5.2 Dataset Heterogeneity

To further investigate the differences in datasets, we visualise the differences in the training buffer at various rounds. (Figure 3) Measured by divergence .4.3, label distribution skew is present in both



(a) Dataset Heterogeneity (Divergence / Round). Expected divergence with 95% confidence interval. Lower bound is approximated divergence between sampled uniform distribution.



(b) Visualisation of single training run (5 clients). Matrix (bottom row) indicates divergence between clients. Scatter plot (top 2 rows) shows the distribution of the replay buffer in a representation .4.2 of the state space for `Pendulum`.

Figure 3: Presence of dataset heterogeneity in FRL: larger divergence .4.3 indicates label distribution skew between clients. 5 clients, reduced 128 transitions of initial random collection.

scenarios considered. Moreover, as expected, it is more prevalent in the fixed-reset case, as episodes tend to cluster around similar trajectories given fixed starting states. High levels of variance in divergence are seen for the fixed-reset case. This may be due to the randomly chosen seeds changing the similarity of start states. We see that although divergence decreases as the policy converges and the replay buffer fills up, it nevertheless remains above the lower bound (true I.I.D samples) even for I.I.D environments in practice, indicating that some levels of data heterogeneity is innate to the RL learning process. To address this observed heterogeneity, we briefly experiment with adding a proximal loss term in appendix .5.4.

5.3 Partial Knowledge Aggregation

By leveraging `florl Knowledge` abstraction's flexibility, we are able to test the impact of federating a subset of parameters (See Table 7, and Figure 7 for convergence details).

Name	A	A'	C_1	C_1'	C_2	C_2'	**Best Reward**
`TD3Avg`	☑	☑	☑	☑	☑	☑	$\mathit{136.48 \pm 2.0}$
$\texttt{TD3Avg}_{Actor}$	☑	☐	☐	☐	☐	☐	$\mathbf{-136.016 \pm 0.943}$
$\texttt{TD3Avg}_{Critic}$	☐	☐	☑	☑	☑	☑	-946 ± 281
$\texttt{TD3Avg}_{Critic'}$	☐	☐	☑	☐	☑	☐	-652 ± 531
$\texttt{TD3Avg}_{Target}$	☐	☑	☐	☑	☐	☑	-238 ± 261
$\texttt{TD3Avg}_{Target'}$	☑	☐	☑	☐	☑	☐	-137.30 ± 3.43

Table 2: Partially Aggregating `TD3Avg` - Pendulum across 5 clients, 10 seeds, 150 rounds with 50 frames per round. A is the actor network, C_i is the i critic network, and $'$ represents the respective target network. A check mark indicates that the server will aggregate this module. We use a single evaluation episode, initial state fixed across all runs, to reduce compute.

At a high level, TD3 trains the actor to maximise the value of the critic, and the critic to minimise error in predicting transitions assisted by target networks. We refer to [6] for a more detailed overview. In

the configurations we tested, 3 successfully converge near the optimal policy: aggregating everything, only the actor, and everything except target networks. There was no statistically significant difference in training these 3 configurations. We note that, in particular for $\texttt{TD3Avg}_{Actor}$, there exists strategies that maintain the FRL training performance with vastly reduced communication overheads.

This could be due to the following postulation. In some sense, a critic network can be viewed as a compressed representation of the replay buffer data. Assuming the replay network stays stable and accurate to the locally collected transitions, this provides a general gradient direction for the actor network - while FL has already shown success in aggregating varying gradients. Furthermore, updating to the global actor network helps each individual client collect similar trajectories, thus reducing the data heterogeneity present. If we update the critic or target networks and not the actor, this just serves to shift a moving training target for the actor, increasing policy instability - thus these methods perform worse.

5.4 `ros2-flower-bridge`

We were able to successfully complete implementation of both `ros2` federated workloads, which serves the core purpose of demonstrating technical feasibility. Both pipelines showed convergence and training, see appendix .5.5 for results.

One system challenge for FRL is the computational requirements of clients. In our `ros2` pipelines, this could be due to the overhead of spinning other working threads such as the data collection nodes. It is also a fundamental issue with the need for environment simulation in RL. While our architecture is valid in a scenario where one processor maps to a single set of nodes on many devices, simulating large scale FRL tasks with minimal resource usage brings a set of engineering complexities that remains unsolved.

6 Conclusion

We have successfully designed and implemented a framework for Federated Reinforcement Learning (FRL) on top of Federated Learning (FL) library `Flower`. This framework allows us to rapidly iterate on FRL experiments and lays the foundation for our other efforts. We construct FRL baselines from a general method (`QTOptAvg, DQNAvg [7], TD3Avg`). We empirically show both convergence, as well as speed, stability and generalisability improvements from training parallelisation, all without sharing privacy-compromising transition samples. We explore heterogeneity - label distribution skew - in the FRL setting, and discover that it is intrinsically present even with I.I.D environments. Having different initial state distributions exacerbates this effect. We explore improvements to `TD3Avg` by reducing communication overhead via transmitting only the latest actor network. We integrate our framework with `Robot Operating System 2` and present an architecture for real-world FRL.

Our work is limited by time and compute - we would like to have had the possibility to explore additional insights across diverse and difficult environments, as well as to run larger-scale experiments in our `ros2-florl` pipeline. One interesting experiment would be examining the impact of asynchronous data collection to training rounds.

Furthermore, we believe for software design purposes `florl` would be better integrated as a part of core `flower` rather than an extension. This would allow, for example, direct integration of `Knowledge` abstraction across gRPC. We hope our framework will enable additional efficient research in FRL, and believe the development of a set of related benchmarks is key to progress. Finally, there are currently many areas of FRL that deserve further attention, from federated learning of generalisable representations across heterogeneous tasks, actors and environments, to considering asynchronous and decentralised forms of aggregation across different RL modules, to adversarial attacks and security.

References

[1] E. Ackerman. Amazon's acquisition of irobot falls through. *IEEE Spectrum*, 01 2024. URL https://spectrum.ieee.org/irobot-amazon.

[2] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, H. L. Kwing, T. Parcollet, P. P. d. Gusmão, and N. D. Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[3] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[4] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. 02 2018.

[5] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council, 05 2016. URL https://data.europa.eu/eli/reg/2016/679/oj.

[6] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[7] H. Jin, Y. Peng, W. Yang, S. Wang, and Z. Zhang. Federated reinforcement learning with environment heterogeneity. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 18–37. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/jin22a.html.

[8] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 29–31 Oct 2018. URL https://proceedings.mlr.press/v87/kalashnikov18a.html.

[9] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015. doi: 10.1109/TASE.2014.2376492.

[10] S. Lee and D.-H. Choi. Federated reinforcement learning for energy management of multiple smart homes with distributed energy resources. *IEEE Transactions on Industrial Informatics*, 18:488–497, 2022. URL https://api.semanticscholar.org/CorpusID:229227268.

[11] H. X. Li. Kitten, 2024. URL https://github.com/MarkHaoxiang/kitten.

[12] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[13] B. Liu, L. Wang, M. Liu, and C. Xu. Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. *CoRR*, abs/1901.06455, 2019. URL http://arxiv.org/abs/1901.06455.

[14] H. Ludwig, N. Baracaldo, G. Thomas, Y. Zhou, A. Anwar, S. Rajamoni, Y. Ong, J. Radhakrishnan, A. Verma, M. Sinn, et al. Ibm federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987*, 2020.

[15] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074. URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mniha16.html.

[19] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. Massively parallel methods for deep reinforcement learning, 2015.

[20] N. Rieke, J. Hancox, W. Li, F. Milletarì, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, S. Ourselin, M. Sheller, R. M. Summers, A. Trask, D. Xu, M. Baust, and M. J. Cardoso. The future of digital health with federated learning. *npj Digital Medicine*, 3(1):119, Sep 2020. ISSN 2398-6352. doi: 10.1038/s41746-020-00323-1. URL https://doi.org/10.1038/s41746-020-00323-1.

[21] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[22] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, Mar. 2023. URL https://zenodo.org/record/8127025.

[23] Q. Wang, S. Kulkarni, and S. Verdú. Divergence estimation for multidimensional densities via -nearest-neighbor distances. *Information Theory, IEEE Transactions on*, 55:2392 – 2405, 06 2009. doi: 10.1109/TIT.2009.2016060.

[24] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[25] L. Zhang, H. Yin, Z. Zhou, S. Roy, and S. Yaping. Enhancing wifi multiple access performance with federated deep reinforcement learning. pages 1–6, 11 2020. doi: 10.1109/VTC2020-Fall49728.2020.9348485.

[26] H. H. Zhuo, W. Feng, Y. Lin, Q. Xu, and Q. Yang. Federated deep reinforcement learning, 2020.

Appendices

.1 Division of Work

https://github.com/MarkHaoxiang/florl/blob/main/report/divison_of_work.md

.2 Reproducibility

Experiments are seeded and conducted over multiple runs. The experimental hyperparameters are available on .4. Versioning / compute details can be found in .3. We publicly release our code on two repositories here: https://github.com/MarkHaoxiang/florl, https://github.com/MarkHaoxiang/ros2_flower_rl.

.3 Versioning and Compute

Development and experiments were performed on 3 Arch Linux machines running kernel 6.7 with `ROS2 Humble`. `flower:da7125a` and `kitten:2408cdc` were built from source. We had access to a Nvidia 3090 GPU, but most experiments were completed in reasonable time on CPU only machines.

Versions for core dependencies are listed below:

Python	3.11.8
Flower	a2da8f0 (Source)
Kitten	2408cdc (Source)
RobotOperatingSystem	Humble
PyTorch	2.2.1
Numpy	1.26.4
Gymnasium	0.29.1
Scikit-Learn	1.4.1
Matplotlib	3.8.3
Seaborn	0.13.2

Table 3: Dependency Versions

.4 Experiment Setup Details

.4.1 Architectures and Hyper-parameters

Our training uses the setup detailed as follows. Critic network is a multi-layer perceptrons (MLP) with 1 hidden layer and 64 hidden size. The network has LeakyRELU [24] activations. Actor networks were the same with adjusted input and output sizes, and $\tanh$ applied to the output layer, then scaled to the size of the action space. The list of hyperparameters can be found in table 4.

.4.2 Post-processing (Pendulum)

During analysis, we transform the state space of `Pendulum` by

- $\theta = \arctan_2(x, y)$
- Divide by a scale of 8 for torque

This simplifies the visualisation to $2D$.

.4.3 Divergence between datasets

To calculate the heterogeneity of the training target distributions between two clients, we use an approximation of Jensen-Shannon divergence based on the consistent and unbiased k-nearest neighbor Kullback–Leibler divergence estimator proposed in [23]. Take $k = 5$. We calculate this based on the state samples of the transitions in each client's replay buffer. To avoid division by 0 error, we lower bound the distance between samples by $\epsilon_{KL} = 0.01$.

Table 4: Hyperparameters Overview

Name	Description	Value
γ	Discount Factor	0.99
τ	Target network update factor per step	0.005
Learning Rate		0.001
Optimiser		Adam
Update / Collection Ratio		1
Clip Grad norm		1
Batch Size		64
n_{cem}	Samples per iteration for cross-entropy-method (QTOpt)	64
m_{cem}	Maximised over per iteration for cross-entropy-method (QTOpt)	6
cem iterations		2
Target Noise	Extra noise applied to training targets (TD3)	0.1
Target Noise Clip	Extra noise applied to training targets (TD3)	0.2
Critic Update Frequency	TD3	1
Policy Update Frequency:	TD3	2
Early Start	Number of collections using random policy before training	128 (Data Heterogeneity), 1024 (Otherwise)
Memory Capacity	Maximum number of transitions before old transitions are forgotten	TOTAL_ROUNDS $\times$ FRAMES_PER_ROUND
ϵ	Random exploration percentage	0.1

.5 Additional Experimental Results

.5.1 `QTOpt` / `QTOptAvg` Loss

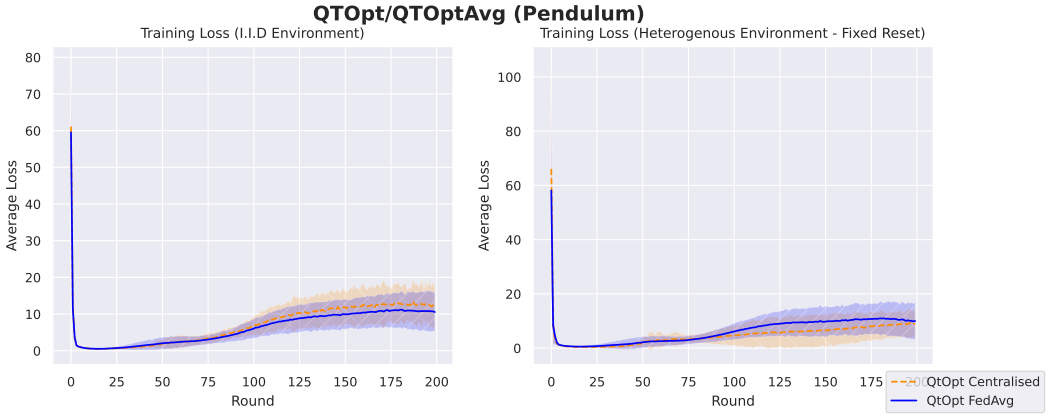


Figure 4: Federating with `QTOptAvg` across 20 seeds. Loss is distributed across clients. The line represents mean and shaded 1.96 standard error.

Due to the shifting dataset distribution, loss is not necessarily representative of learning progress in reinforcement learning. However, we note that it does indicate successful convergence. It is interesting that there does not appear to be a significant difference between the loss curves of a federated pipeline and the baseline, while we may expect it to be worse - perhaps as the data collection process is influenced strongly by aggregation.

.5.2 DQN / DQNAvg

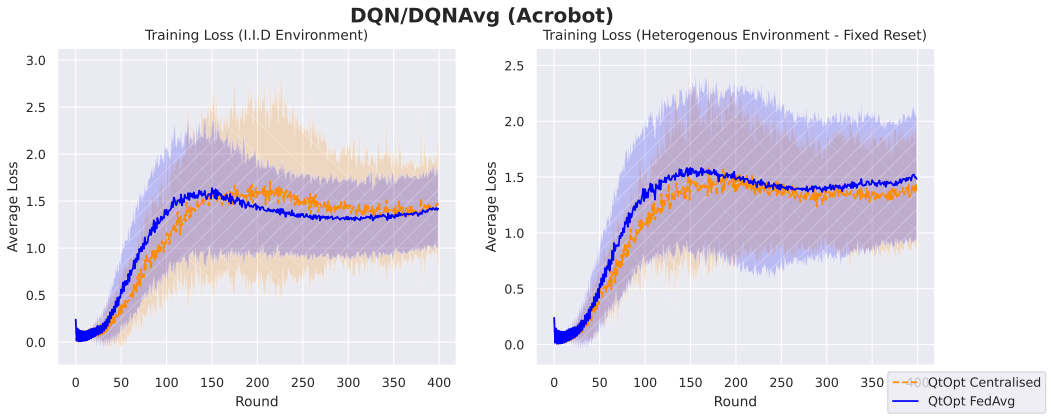


Figure 5: Federating with DQN across 20 seeds, 5 clients, with centralised evaluation across a constant 5 seeds per round. The line represents mean and shaded 1.96 standard error. Distributed loss

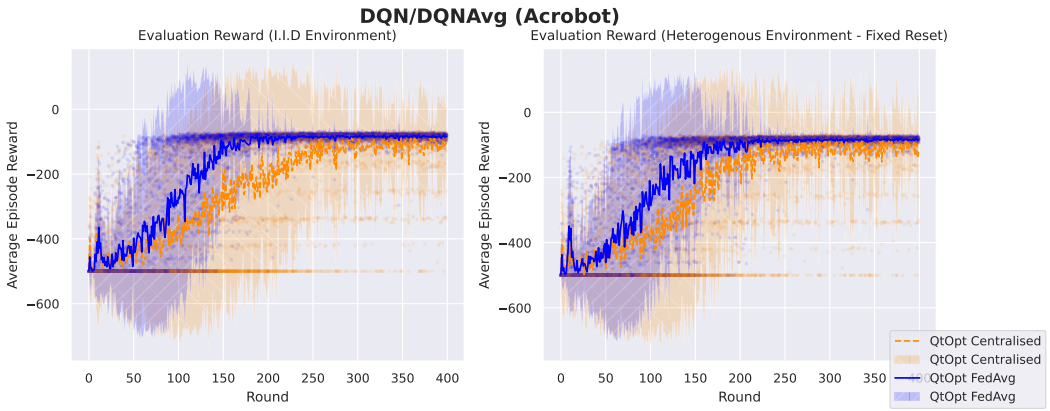


Figure 6: Federating with DQN across 20 seeds, 5 clients, with centralised evaluation across constant 5 seeds per round. The line represents mean and shaded 1.96 standard error.

The results of DQN roughly follow in line with those of QTOptAvg. Federating successfully transfers knowledge from extra transitions, leading to faster learning in both I.I.D and heterogeneous environments. We also see stability and generalisability improvements, leading expected return of DQNAvg to converge to a higher value in heterogeneous environments. Loss is increasing since kitten DQN doesn't have bias reduction techniques.

.5.3 TD3Avg (Subset)

Due to compute constraints, we were restricted to 10 seeds, hence highly unstable variance. The primary impact of this graph 7 is to show that convergence rate is roughly the same for the top 3 methods.

.5.4 QTOptProx

As shown by Figure 8, there is minimal difference created by adding the proximal term. We do observe stability improvements close to the end of training (clients are less encouraged to have smaller differences), and slightly slower training during the middle of training (slower network change). More work is needed to classify the different types of heterogeneity in FRL, and whether FedProx could be successfully applied to any, but this is out of the scope for this report.

.5.5 Federating ros2 with flower and florl

See Figure 9 and 10.

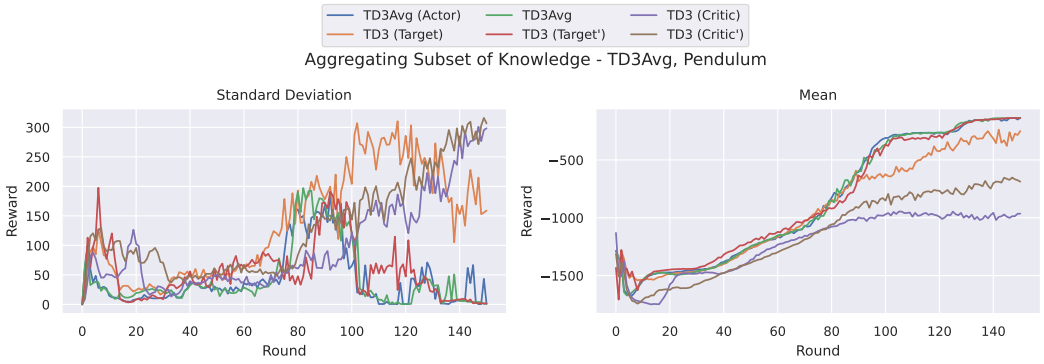


Figure 7: Impact of aggregating different modules for `TD3Avg`

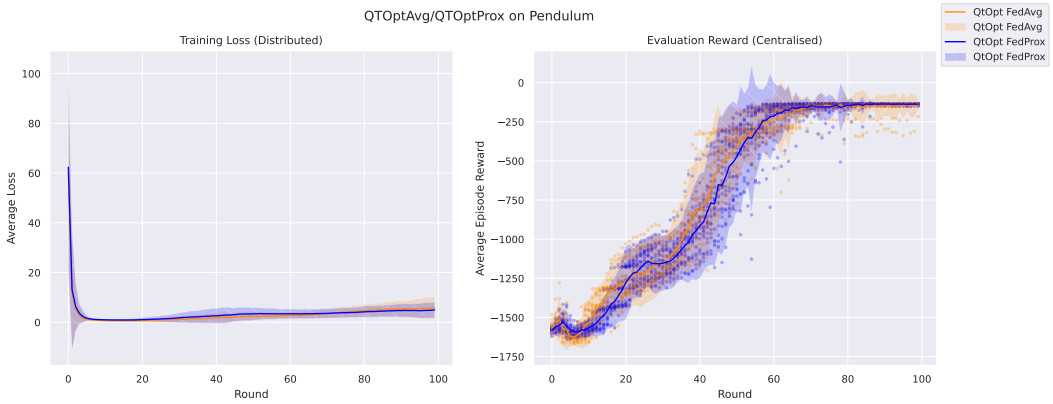


Figure 8: Adding a proximal term ($\mu = 1.0$), 10 clients, 50 frames per round on Pendulum.

Note that primary difference between the FRL algorithm used here and algorithm 1 is due to asynchronous data collection. While the number of update epochs remain consistent across rounds, there are no guarantees on the number of frames collected, and newly collected frames may not be consistent with the latest trained policy (weights not yet propagated to `Actor` node).

.6 `florl` overview

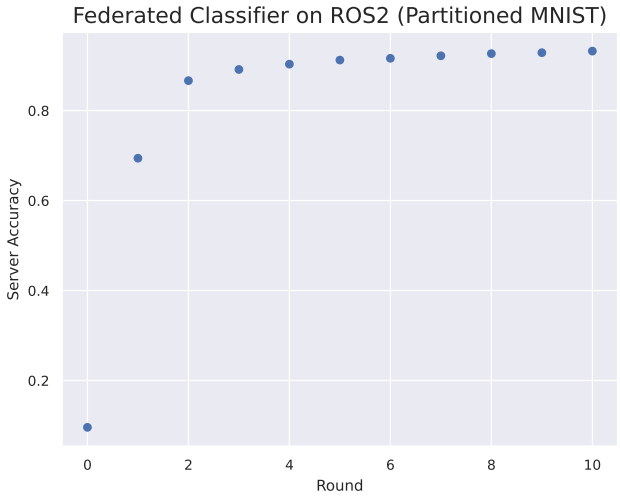


Figure 9: Successful training of a classifier with our `ros2` pipeline. 10 clients with partitioned data from MNIST, with a CNN followed by MLP.

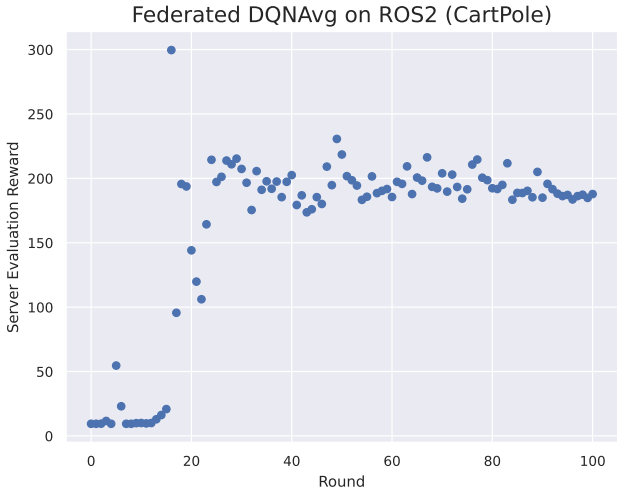


Figure 10: Successful training on CartPole with our `ros2` pipeline. 5 clients with asynchronous data collection and individual replay buffers

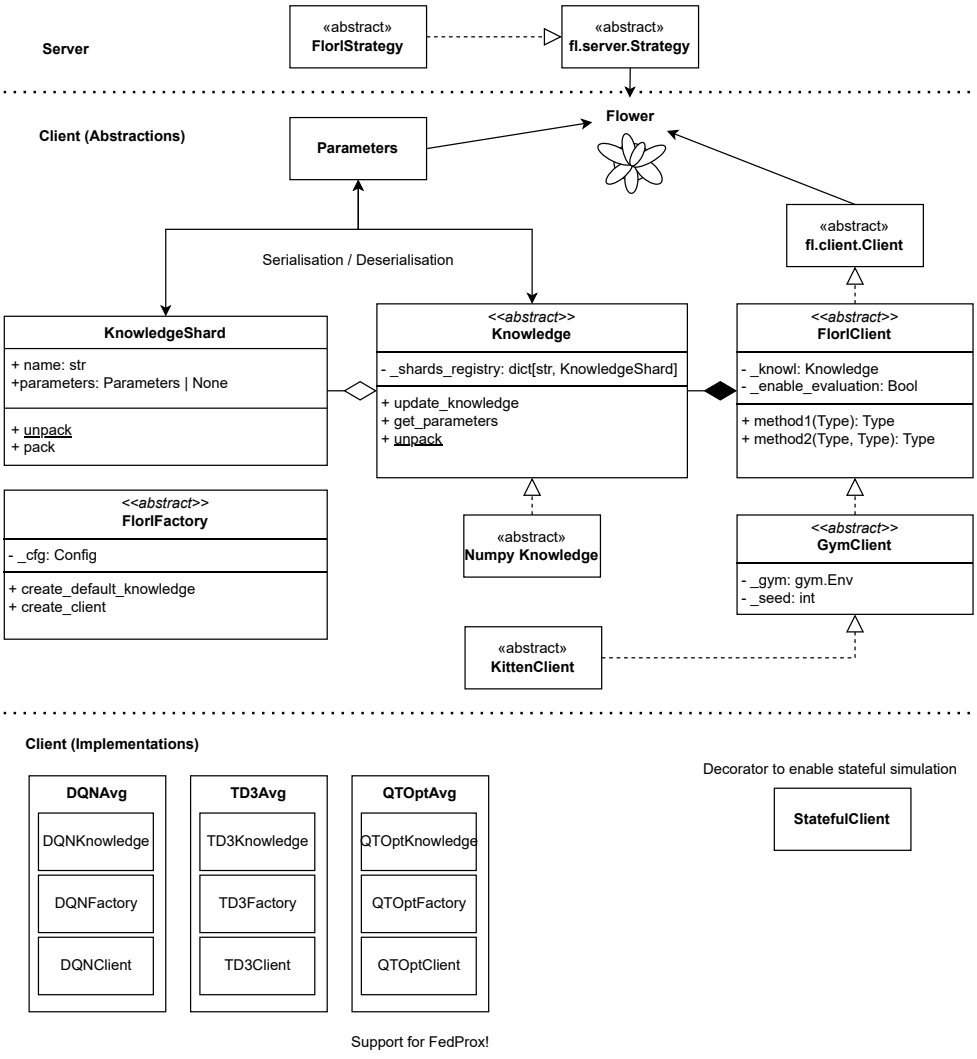


Figure 11: Informal `florl` design diagram with subset of implemented components