Introduction

Let’s consider a problem in which a Reinforcement Learning (RL) agent has to learn a regulation (e.g. a real traffic regulation) so that it can optimally move in a non-deterministic environment. In such environment the complexity is clearly dependent on the level of non-determinism and to how complicated is the underlying set of rules.
On one side there is a set of rules to be learnt, governed by non-monotonic logic possibly involving several chains of exceptions to rules/exceptions (although in real regulations such chain’s length is very unlikely to exceed 3 or 4).
On the other side there is a RL agent learning optimal policies from Markovian sequences of interactions, evaluated by means of rewards, imposing transitions in the state of an observable underlying environment.
This RL agent learns from experience that is a sequence of state transitions in the form of tuples containing the state s_t of the environment at step t, the chosen action a_t, the consequent reward r_t and the resulting new state s_{t+1}.
Therefore there is a clear dichotomy between the case-based mechanism a RL agent uses to infer knowledge and the rule-based approach that is adopted for encoding realistic regulations.
How can we effectively explain such a regulation to a standard RL agent? There might be several naive approaches:

1. We encode the whole regulation, feeding it to the agent as part of the environmental observation, at every step.
2. At every state transition, we feed to the agent an encoding of the rules responsible for the given reward (we might expect a negative reward when a rule is violated, a positive otherwise), as part of the environmental observation.

The first strategy can cause excessive overhead, not guaranteeing that agent is going to really “understand” the regulation, and imposing a re-training every time the regulation is even slightly changed, because the semantic of the regulation is explicitly provided as input thus shaping the state space.
The second strategy suffers the same issues of the first, but a smaller overhead and a few more possibilities that the agent is going to exploit more effectively the additional information about the rules.
With both the strategies we might have to encode rules’ explanations

into the state space, thus increasing the amount of resources required for learning an optimal policy (due to the bigger size of the state space) and requiring the agent to re-train every time the regulation changes.
How can we provide to a RL agent possibly complex and fairly articulated explanations without changing the state space of the environment?
The only way to do so is to convert explanations from a rule-based format to a case-based format, in which an explanation instead of being represented as a textual rule is represented as a collection of state-transitions. Such collection of state-transitions (or cluster of experience) shall contain the smallest amount of information to fully describe an explanation in a way that is relevant for the agent to reach its objectives.
But what is an explanation? Actually there are many different interpretations in philosophy of what an explanation is, among them we choose the one coming from Ordinary Language Philosophy, for its simplicity, for framing an explanation as an useful and meaningful answer to one or more questions. Where the shape of both “useful” and “meaningful” may change across different agents.
Then we have that many different types of explanation may exist for the same process/environment, e.g. those answering to “why” questions, those answering to “how” questions, “what-if”, “when”, “where”, “what”, etc..
Therefore we may univocally assign to every explanation at least one question, and to every question a task, hence a rule to be learnt (in the case of “why” explanations), or information about how well the agent is acting in a particular moment, etc..
As result, providing different explanations to the agent is about properly feeding sequences of state-transitions representing different tasks.
Interestingly, in any standard RL setting we would always have a reward function that is a rule-base deciding which rule to apply so to properly assign a reward. Therefore in such settings it is always possible to tag every reward with those relevant rules used for deciding it.
Hence we have that we can actually do cluster experience, feeding it to the agent so that it would learn abstract tasks according to their relevance to its objectives. In other, more technical, words what we propose is to encode explanations in the order transitions are fed to the agent, through prioritised experience replay on a graph of several

experience clusters (or buffers), where each cluster stands for an abstract enough explanation/task.
We show that this strategy is effective not only on off-policy RL algorithms (e.g. DQN, DDPG) but seemingly also on on-policy algorithms (e.g. APPO — that is PPO with v-tracing) for which there is no known mainstream experience prioritisation technique, differently from the off-policy ones.
What we hypothesise is that the more complex is the environment (thus the harder is the rule-base), the more useful eXplanation-Awareness (XA) would be.
In order to verify this hypothesis we designed two different and seemingly realistic environments depicting two different autonomous driving scenarios:

- The first environment is GridDrive, a discrete environment with a discrete action space: a grid of road segments having different properties according to which the agent has to modulate speed and steering angle.
- The second environment is GraphDrive, a continuous environment with a continuous action space, where a planar graph of roads having different properties and lengths has to be navigated by the agent according to its properties and following a given set of rules.

In both the environments the reward function is regulated by an argumentation framework with a culture of 3 different complexities: easy, medium, hard. The hard culture is the most realistic, involving more properties and several levels of exceptions among rules.

Technical Aspects

What we are proposing is a novel agent-centred explanatory process for RL agents, inspired by user-centred interactive explanatory processes designed for humans [1,2]. The problem with agent-centred explanatory processes is that they are computationally irreducible (https://en.wikipedia.org/wiki/Computational_irreducibility), thus it is not possible to know in advance what's the optimal explanation for the agent, also considering its knowledge might unpredictably change in time. In

order to tackle this problem, in [1,2] what is proposed is to design the agent-centred explanatory process as a process of searching an optimal path inside a so-called Explanatory Space (ES), that has to be structured in a proper way, allowing the agent to efficiently extract the simplest relevant explanations out of it. Otherwise, without a proper structure, for reasonably complex explananda, the ES would be too big (footnote: with an argumentation framework having more than 270 arguments, the set of all possible different explanations might be greater than the estimated number of atoms in the universe), thus impractical to be explored effectively and efficiently in a relevant way.
By definition an ES is a graph containing all the possible explanations about a given explanandum (e.g. an environment, the tasks) that any possible agent might encounter. For a RL agent, an explanation is an ordered minimal set of state-transitions properly representing an abstract task (e.g. a rule to be learnt), thus in this case the ES would be a graph of experience buffers, a graph of graphs. In [1,2] arguments are provided to show that, with humans, an ES can be effectively structured by following the ARS heuristics:

- Abstraction: for clustering the ES nodes into different buffers.
- Relevance: for organising the information internal to buffers, prioritising the most relevant ones to the agent's objectives.
- Simplicity: for filtering the information internal to buffers and selecting the viable ES edges, prioritising buffers (tasks) in a curricular fashion (that is from the simplest to the hardest).

The explanatory process we propose can be summarised as follows. The agent gets the most relevant/important bits of explanation for itself thanks to the prioritisation strategy: experience is prioritised according to the objective of the agent (e.g. td-errors for DQN, gains for PPO), enforcing a curricular approach to problem solving. The order these bits of explanation are selected might depend on the complexity of the respective questions/tasks, thus it is defined by the different aspects/tasks to be learnt, which are organised according to some abstraction heuristics (e.g. the argumentation framework). The amount of bits of explanation to be fed and their freshness/age is defined by other heuristics meant to keep small-enough informative clusters and to extract small-enough informative train-batches standing for abstract-enough explanations.
The following sections show how the ARS heuristics are implemented and tested, in practice.

Abstraction

Abstraction is used to structure the outer graph of the ES: the graph of buffers.

- Instead of having a single experience buffer, we may have multiple buffers, one per explanation.
- To every batch of consecutive state transitions of the same episode we assign an explanation, according to many possible strategies.
- An explanation is a label attached to the batch, explaining something about that batch.
- We might have different types of explanations:
  - “Why” explanations: telling why the batch has the rewards it has. This type of explanations can be produced by an argumentation framework.
  - “How” explanations: telling how a batch is behaving at episode-level or locally. This type of explanations can be produced by studying the average behaviour of the agent, e.g. if an episode has a cumulative reward that is greater than the running mean, then the agent is behaving better than average on that episode.
  - “Why-How” explanations: a concatenation of a why and a how explanation.
- A clustering scheme is a scheme adopted to group experience into multiple (prioritised) buffers, according to their explanations.
- **Assumption:** Let an explanation be an answer to a question.
- The idea is that every cluster represents an explanation, every explanation is an answer to a question, every question represents a task, therefore every explanation represents a different task. Keeping track of all the possible useful (for the sake of learning faster) tasks would prevent the agent to catastrophically forget any of them and thus to generalise better. Furthermore it would be possible for the agent to automatically learn tasks in a curricular fashion if it samples clusters in a prioritised fashion, by using the sum of the priorities of a cluster as cluster’s priority (we will refer to this technique as cluster_prioritisation).
- We devised the following clustering strategies, but many others are possible:
  - reward_against_zero: “how” explanations containing information about the quality the batch; if the batch has a reward greater than 0, then it is said to be “good” otherwise “bad”. The maximum number of different clusters can be 2.

- reward_against_mean: “how” explanations containing information about the quality the batch; if the batch has a reward greater than the running mean of rewards seen so far with a sliding window of x, then it is said to be “better” otherwise “worse”. The maximum number of different clusters can be 2.
- multiple_types: “why” explanations containing explanations about the reasons behind the received rewards. Every reward can have multiple different explanations, where an explanation is an argument. If a state-transition has more than one explanation, then an explanation is selected randomly and the batch is inserted in the cluster associated to that bit of explanation. There is no need for duplicating the batch across multiple clusters, because it is a prioritised buffer, and clusters will be fairly represented, with minimum overhead in terms of memory and time. This allows a maximum number of clusters that is linear in the number of different arguments.
- multiple_types_with_reward_against_zero: “why-how” explanations; “multiple_types combined” with “reward_against_zero”.
- multiple_types_with_reward_against_mean: “why-how” explanations; “multiple_types combined” with “reward_against_mean”.

Relevance

Relevance is used for organising the information internal to buffers, prioritising the most relevant ones to the agent’s objectives. Prioritised experience replay is the most common heuristic for relevance, being extensively adopted for improving off-policy learning algorithms such as DQN (for discrete action-spaces) or DDPG (DQN for continuous action-spaces) [3,4]. Differently, in on-policy learning, experience replay is not common, because the agent is supposed to learn on the experience collected following its current policy. The current policy normally evolves frequently through time, thus forcing to drop all past experience. The family of on-policy algorithms based on PPO (for both continuous and discrete action-spaces) is the only one being adapted for effective experience replay. APPO is probably the state-of-the-art for on-policy algorithms, being usually faster and more precise, thanks to a technique called v-tracing (coming from IMPALA) that allows the agent to effectively exploit state-transitions that are not on-policy. This characteristic makes APPO the best

candidate for explanation-awareness within the family of on-policy algorithms.
Despite this, prioritised experience replay has not yet extensively explored in XAPPO, maybe also because the state-of-the-art prioritisation techniques [3] are based on the invariant that priorities are always greater than zero. In fact the priority/relevance of a batch is usually estimated by partially computing its loss with respect to the agent's objectives:

- In DQN/DDPG the absolute TD-error is usually adopted for estimating the relevance: the closer to 0, the lower the loss, the lower the relevance/priority. The idea is that batches with TD-error equal to zero are of no use for learning, because they represent an already solved challenge the agent has already learnt to tackle.
- In PPO, the gain (the advantage multiplied by the action importance ratio) could be used: the higher it is, the lower the loss, the higher the relevance/priority. PPO's objective function is the minimisation of clipped negative gains, but here, differently from TD-errors, we are not considering negative gains for the prioritisation, we are not considering exactly the loss. This is because of the semantics of the gain and because the agent is learning on-policy to find the actions that maximise gain. When a batch has gains lower than 0 it means that it contains actions pushing the agent towards less gain, thus it contains information about what action not to do, and in high action spaces this is of little or no use. Those batches with positive gains, on the other side, contain information about what actions may actually increase gain, thus they are of most use for the agent to reach its objective.

Thus, in PPO the relevance can also be negative, breaking a few invariants in the segment trees used for efficiently (in O[logN] where N is the batch's size) sampling experience in a prioritised fashion and also being not compatible with the standard formula [3] for re-weighting prioritised experience.
A simple trick to guarantee the required invariants in the sum segment trees is to subtract to runtime sums the minimum priority in the cluster, multiplied by the number of inserted elements, thus enforcing all priorities to be greater than or equal to zero. While, for beta-weighting, a different formula can be envisaged. The original formula computes the weight of a batch as (p_min / p_batch)^beta, where p_min is the minimum priority in the buffer and p_batch is the

priority of the batch, and it is supposed to always output values in (0,1]: the closer is p_batch to p_min, the lower is the weight. So, a similar formula admitting negative priorities could be the following: ((q_max - p_batch)/(q_max - p_min))^beta, where q_max = p_max*((1+eta) if p_max >= 0 else (1-eta)). This new formula ensures outputs always in (0,1] when eta > 0 (e.g. eta = 0.01). Eta is used to avoid importance weights equal to 0 when the sampled batch is the one with the highest priority in the buffer. The closer eta is to 0, the closer to 0 would be the weight of the batch with the highest-priority.

The relevance heuristic can be combined with the abstraction heuristic by sampling experience clusters in a prioritised fashion, and then performing prioritised sampling of enough batches, from the sampled cluster. A cluster's priority can be computed by summing the priorities of all the batches it contains. Given that the maximum number of batches that are stored in the whole experience buffer is fixed due to memory constraints, every time a new batch is collected and added to a full buffer another batch is dropped from the buffer from a cluster where it is possible to drop. The drop strategy we found useful is called PDrop: drop the batch with the lowest priority with probability p otherwise drop the oldest one (as in vanilla DQN). Another useful strategy might be the Global Distribution Matching proposed in [5]: To every batch assign a random number in [0,1] during insertion. Drop the batch with the lowest random number.

Simplicity

The simplicity heuristic is used for filtering the information internal to buffers and selecting the viable ES edges, prioritising buffers (tasks) in a curricular fashion (that is from the simplest to the hardest).
The point with clustered prioritised experience replay is that it changes the real distribution of tasks when they are correctly represented by clusters (as expected). Thus, instead of having the agent learning in a curricular fashion from the simplest to the hardest task we might have the agent trying to solve the hardest task first, even if that is not possible nor optimal, because it has the highest priority despite not being the most frequent task.
If all the clusters would have the same size, replaying the task's cluster with the highest priority might push the agent away from the most common task, thus preventing the agent from learning faster an optimal policy. The most exceptional tasks (the exceptions) shall be learnt after the most generic, common and basilar ones.

If the clusters' size would be proportional exactly to the real distribution of tasks, then it would be like not having any cluster. Having clusters helps over-sampling all those batches potentially related to under-represented tasks, learning to tackle them more efficiently.
Therefore we have that the clusters' minimum size shall be big enough for effective over-sampling, while having the maximum size being dependent on the real distribution of tasks, thus pushing the agent to tackle first the most frequent and relevant tasks.
For all cluster to have the same size Y, we have that Y = N/C, where N is the number of elements in all the clusters (the buffer) and C is the number of different clusters. If we want to guarantee that every cluster contains at least X=pY elements, then X is the minimum size of a cluster, where p is a number in (0,1). If we would also want to constrain the maximum size of a cluster, we would have to constrain with q the remaining (1-p)YC = (1-p)N elements so that (1-p)N = qX, thus enforcing that the size of a cluster is in [X, X+qX] when at least X different batches are inserted in the cluster. Therefore, we have that the minimum cluster's size has to be N/(C+q) and it can be easily controlled by changing q. This shall help having a buffer reflecting the real distribution of tasks (where each task is associated to a cluster). Thus, for a curricular prioritisation, if the cluster's priority is computed as the sum (and not the average) of its batch's priorities and q > 0 is not too big (e.g. q=1), the resulting cluster's priorities would reflect both the real distribution of tasks while smoothly over-sampling the most relevant tasks, thus avoiding over-estimation of task's priority. The degree of smoothness can be easily controlled with q.

Experimental Results

We run a few experiments on *Hard* (or *Medium*?) GridDrive and *Easy* GraphDrive. During the experiments we compared our proposed eXplanation-Aware (XA) agents against their respective baselines: DQN, DDPG and APPO. The adopted baselines are those coming with RLlib, an open-source library for RL that offers both high scalability and a unified API for a variety of applications including multi-agent RL. RLlib natively supports TensorFlow, TensorFlow Eager, and

PyTorch, but most of its internals are framework agnostic [6]. Our eXplanation-Aware versions are named respectively XADQN, XADDPG, XAPPO, and they have been implemented by extending RLlib's baselines, making them automatically compatible with the whole RLlib eco-system. As result we produced a modular python library called XARL that could be easily integrated in RLlib.

The neural network adopted for all the experiments is the same default one implemented in the respective baselines (although better ones can be certainly devised), and it is characterised by a fully connected layer of 256 units followed by the output layers for actors and/or critics, depending on the algorithm's architecture.

In most of the experiments we observe that a combination of multiple types of explanations (that is "why" and "how" explanations) is beneficial, also showing that it does not exist only one type of explanation agents needs, and that the variety of types can change across different tasks.

XAPPO

Compared to XADQN or XADDPG, the amount of new stuff added in XAPPO to the original APPO is a lot. This is due the fact that vanilla APPO it is not designed for prioritised experience replay. Thus, we had to test way more hyper-parameters with XAPPO, to show the effectiveness of the proposed eta-weighting mechanism and the effectiveness of prioritisation.
We compared XAPPO against a baseline APPO [7] with the following hyper-parameters:

- "rollout_fragment_length": 2**3, # Number of transitions per batch in the experience buffer
- "train_batch_size": 2**9, # Number of transitions per train-batch
- "replay_proportion": 4, # Set a p>0 to enable experience replay. Saved samples will be replayed with a p:1 proportion to new data samples.
- "replay_buffer_num_slots": 2**12, # Maximum number of batches stored in the experience buffer. Every batch has size 'rollout_fragment_length' (default is 50).

XAPPO's default hyper-parameters were the same of APPO plus the following:

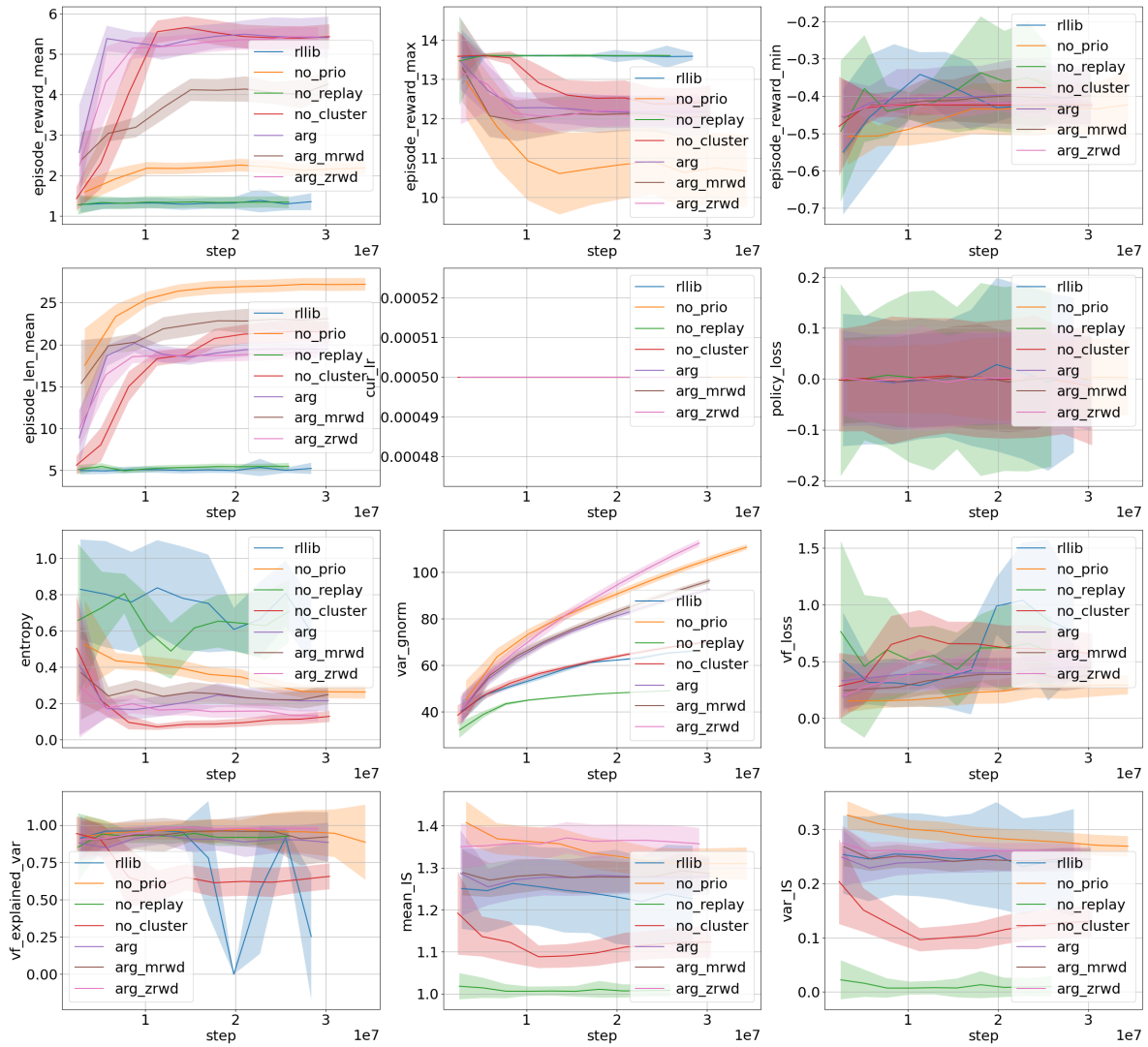
- "prioritized_replay": True, # Whether to replay batches with the highest priority/importance/relevance for the agent.

- "update_advantages_when_replaying": True, # Whether to recompute advantages when updating priorities.
- "learning_starts": 2**6, # How many batches to sample before learning starts. Every batch has size 'rollout_fragment_length' (default is 50).
- 'priority_id': "gains", # Which batch column to use for prioritisation. One of the following: gains, advantages, rewards, prev_rewards, action_logp.
- 'priority_aggregation_fn': 'np.mean', # A reduction function that takes as input a list of numbers and returns a number representing a batch priority.
- 'global_size': 2**12, # Maximum number of batches stored in all clusters (which number depends on the clustering scheme) of the experience buffer. Every batch has size 'rollout_fragment_length' (default is 50).
- 'min_cluster_size_proportion': 1, # Let X be the minimum cluster's size, and q be the min_cluster_size_proportion, then the cluster's size is guaranteed to be in [X, X+qX]. This shall help having a buffer reflecting the real distribution of tasks (where each task is associated to a cluster), thus avoiding over-estimation of task's priority.
- 'alpha': 0.5, # How much prioritization is used (0 - no prioritization, 1 - full prioritization).
- 'beta': 0.4, # To what degree to use importance weights (0 - no corrections, 1 - full correction).
- 'eta': 1e-2, # A value > 0 that enables eta-weighting, thus allowing for importance weighting with priorities lower than 0 if beta is > 0. Eta is used to avoid importance weights equal to 0 when the sampled batch is the one with the highest priority. The closer eta is to 0, the closer to 0 would be the importance weight of the highest-priority batch.
- 'epsilon': 1e-6, # Epsilon to add to a priority so that it is never equal to 0.
- 'prioritized_drop_probability': 0, # Probability of dropping the batch having the lowest priority in the buffer.
- "clustering_scheme": "multiple_types", # Which scheme to use for building clusters. One of the following: "none", "reward_against_zero", "reward_against_mean", "multiple_types_with_reward_against_mean", "multiple_types_with_reward_against_zero", "type_with_reward_against_mean", "multiple_types", "type".

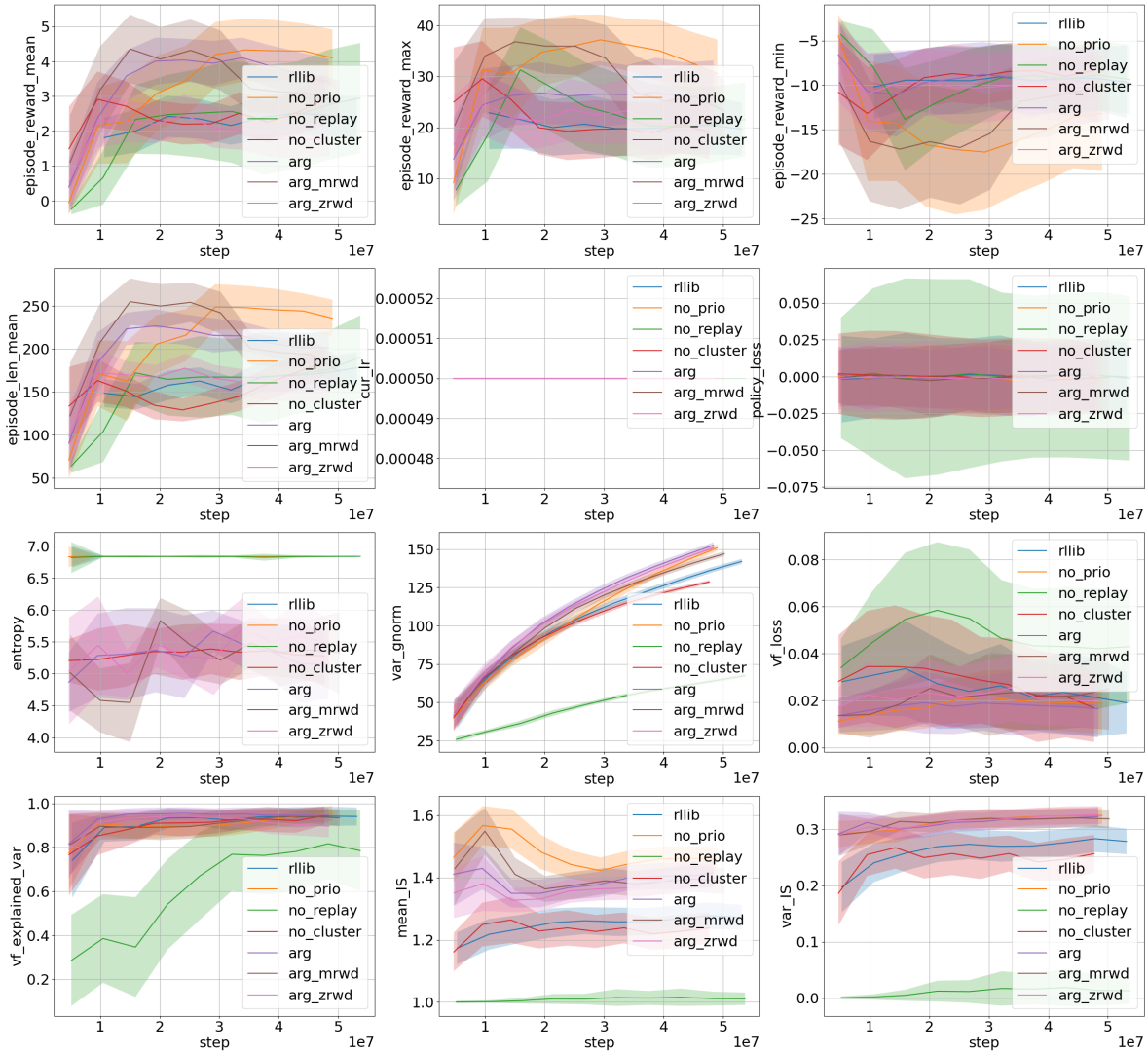
In the following plots we have that:

- “arg” stands for the default XA algorithm with clustering scheme “multiple_types”.
- “arg_mrwd” stands for the default XA algorithm with clustering scheme “multiple_types_with_reward_against_mean”.
- “arg_zrwd” stands for the default XA algorithm with clustering scheme “multiple_types_with_reward_against_zero”.
- “no_cluster” stands for the default XA algorithm with no clustering scheme.
- “no_prio” stands for the default XA algorithm with no prioritised replay.
- “no_replay” stands for the default XA algorithm with no experience replay at all.
- “rllib” stands for the main baseline.

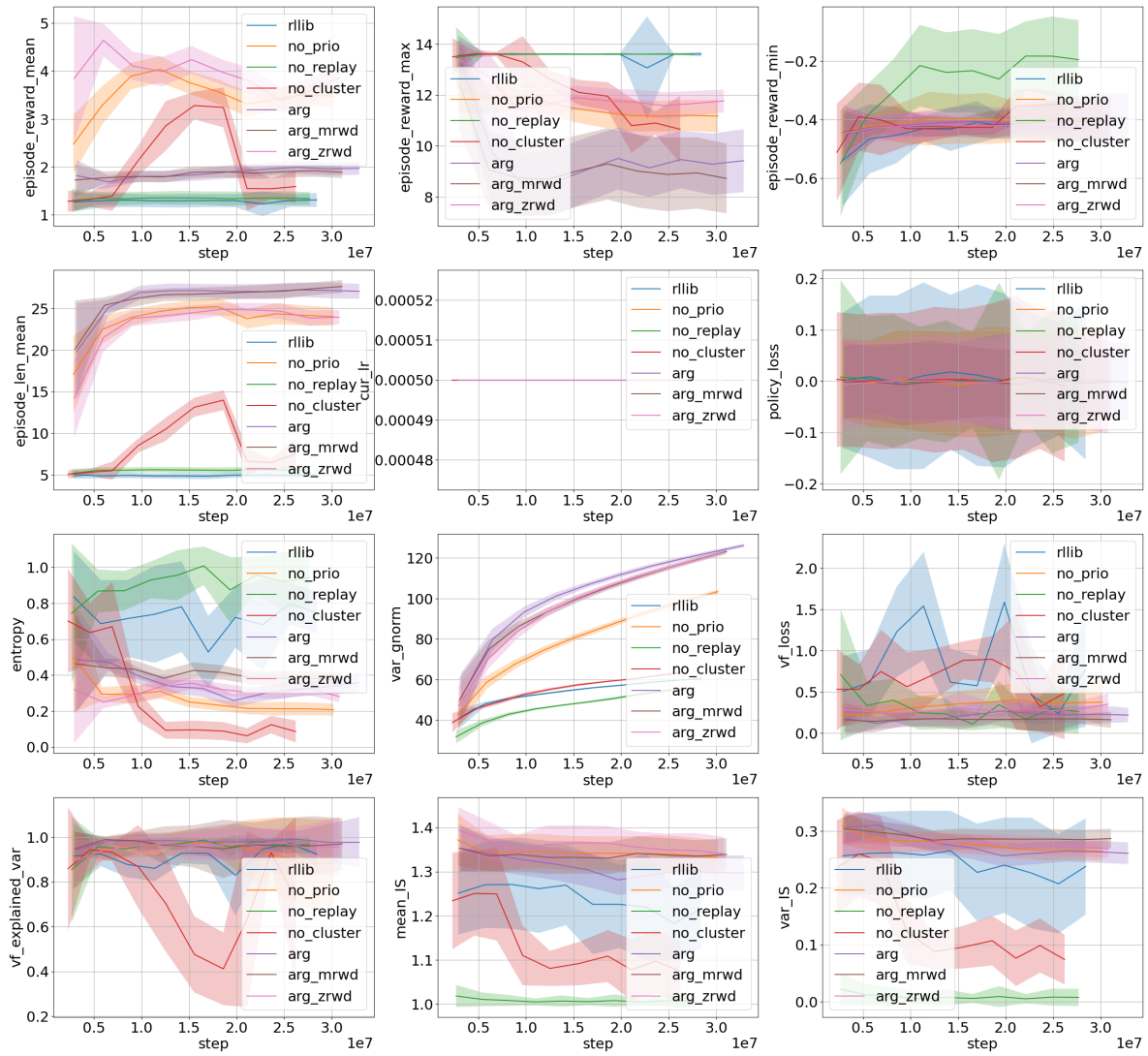
The results for Hard GridDrive are the following:



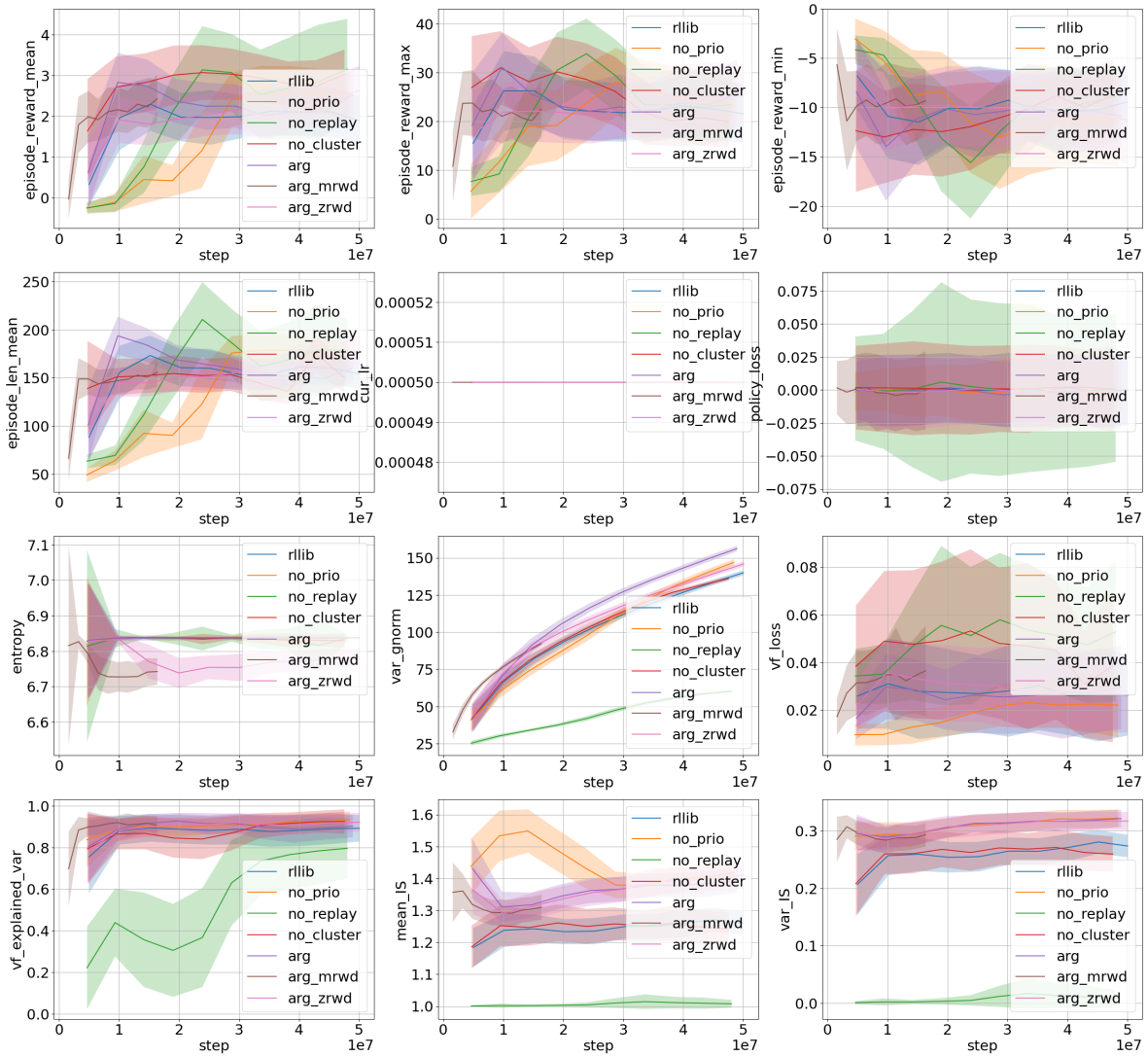
The results for Easy GraphDrive are the following:



The results for Hard GridDrive **without** eta-weighting are the following:



The results for Easy GraphDrive **without** eta-weighting are the following:



As we can see, eta-weighting is beneficial, allowing the agent to avoid catastrophic forgetting. Furthermore, eXplanation-Aware agents are in average always performing way better, faster, than the baselines (blue, green or red).

XADQN

We compared XADQN against a baseline DQN [8] with the following hyper-parameters:

- "dueling": True,
- "double_q": True,
- "prioritized_replay": True,
- "num_atoms": 21,
- "v_max": 2**5,
- "v_min": -1,
- "rollout_fragment_length": 1,
- "train_batch_size": 2**7,

- "num_envs_per_worker": 8, # Number of environments to evaluate vectorwise per worker. This enables model inference batching, which can improve performance for inference bottlenecked workloads.
- "grad_clip": None,
- "learning_starts": 1500,
- "buffer_size": 2**15, # Size of the experience buffer. Default 50000
- "batch_mode": "complete_episodes", # For some clustering schemes (e.g. extrinsic_reward, moving_best_extrinsic_reward, etc..) it has to be equal to 'complete_episodes', otherwise it can also be 'truncate_episodes'.

XADQN's default hyper-parameters were the same of DQN plus the following:

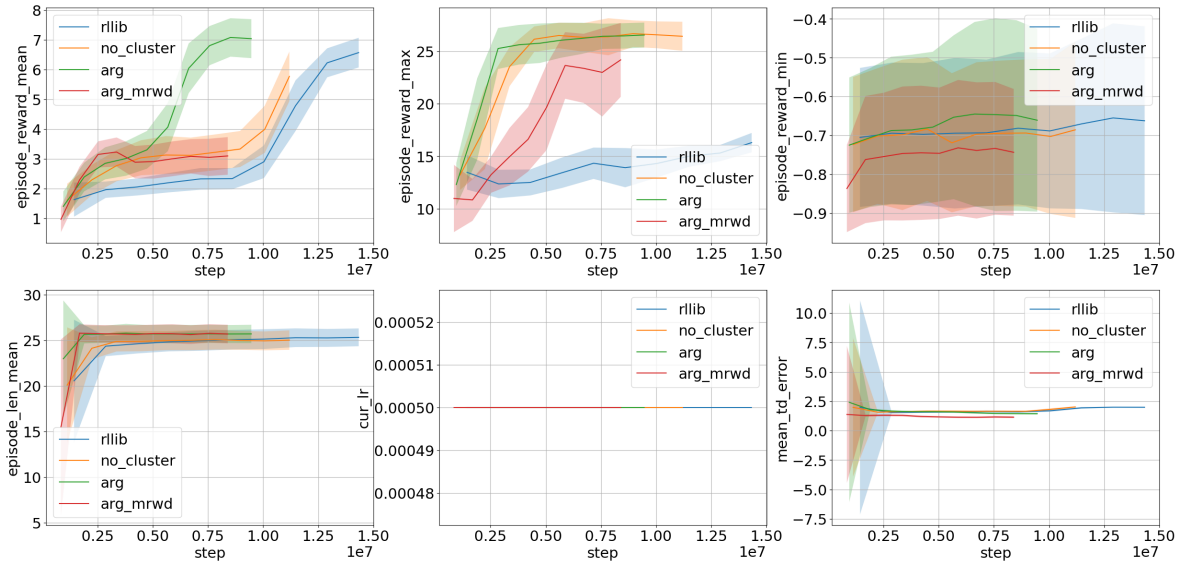
- "prioritized_replay": True, # Whether to replay batches with the highest priority/importance/relevance for the agent.
- 'priority_id': "td_errors", # Which batch column to use for prioritisation. One of the following: gains, advantages, rewards, prev_rewards, action_logp.
- 'priority_aggregation_fn': 'lambda x: np.mean(np.abs(x))', # A reduction function that takes as input a list of numbers and returns a number representing a batch priority.
- 'global_size': 2**15, # Maximum number of batches stored in all clusters (which number depends on the clustering scheme) of the experience buffer. Every batch has size 'rollout_fragment_length' (default is 50).
- 'min_cluster_size_proportion': 0, # Let X be the minimum cluster's size, and q be the min_cluster_size_proportion, then the cluster's size is guaranteed to be in [X, X+qX]. This shall help having a buffer reflecting the real distribution of tasks (where each task is associated to a cluster), thus avoiding over-estimation of task's priority.
- 'alpha': 0.6, # How much prioritization is used (0 - no prioritization, 1 - full prioritization).
- 'beta': 0.4, # To what degree to use importance weights (0 - no corrections, 1 - full correction).
- 'epsilon': 1e-6, # Epsilon to add to a priority so that it is never equal to 0.
- 'prioritized_drop_probability': 0, # Probability of dropping the batch having the lowest priority in the buffer.
- "clustering_scheme": "multiple_types", # Which scheme to use for

building clusters. One of the following: "none", "reward_against_zero", "reward_against_mean", "multiple_types_with_reward_against_mean", "multiple_types_with_reward_against_zero", "type_with_reward_against_mean", "multiple_types", "type".

In the following plots we have that:

- “arg” stands for the default XA algorithm with clustering scheme “multiple_types”.
- “arg_mrwd” stands for the default XA algorithm with clustering scheme “multiple_types_with_reward_against_mean”.
- “no_cluster” stands for the default XA algorithm with no clustering scheme.
- “rllib” stands for the main baseline.

The results for Hard GridDrive are the following:



XADDPG

We compared XADDPG against a baseline DDPG [9] with the following hyper-parameters:

- "rollout_fragment_length": 1,
- "train_batch_size": 2**7,
- "buffer_size": 50000, # Size of the experience buffer. Default 50000
- "batch_mode": "complete_episodes", # For some clustering schemes (e.g. extrinsic_reward, moving_best_extrinsic_reward, etc..) it has to be equal to 'complete_episodes', otherwise it can also be 'truncate_episodes'.

XADQN’s default hyper-parameters were the same of DQN plus the following:

- "prioritized_replay": True, # Whether to replay batches with the highest priority/importance/relevance for the agent.
- 'priority_id': "td_errors", # Which batch column to use for prioritisation. One of the following: gains, advantages, rewards, prev_rewards, action_logp.
- 'priority_aggregation_fn': 'lambda x: np.mean(np.abs(x))', # A reduction function that takes as input a list of numbers and returns a number representing a batch priority.
- 'global_size': 50000, # Maximum number of batches stored in all clusters (which number depends on the clustering scheme) of the experience buffer. Every batch has size 'rollout_fragment_length' (default is 50).
- 'min_cluster_size_proportion': 0, # Let X be the minimum cluster's size, and q be the min_cluster_size_proportion, then the cluster's size is guaranteed to be in [X, X+qX]. This shall help having a buffer reflecting the real distribution of tasks (where each task is associated to a cluster), thus avoiding over-estimation of task's priority.
- 'alpha': 0.6, # How much prioritization is used (0 - no prioritization, 1 - full prioritization).
- 'beta': 0.4, # To what degree to use importance weights (0 - no corrections, 1 - full correction).
- 'epsilon': 1e-6, # Epsilon to add to a priority so that it is never equal to 0.
- 'prioritized_drop_probability': 0.5, # Probability of dropping the batch having the lowest priority in the buffer.
- "clustering_scheme": "multiple_types", # Which scheme to use for building clusters. One of the following: "none", "reward_against_zero", "reward_against_mean", "multiple_types_with_reward_against_mean", "multiple_types_with_reward_against_zero", "type_with_reward_against_mean", "multiple_types", "type".

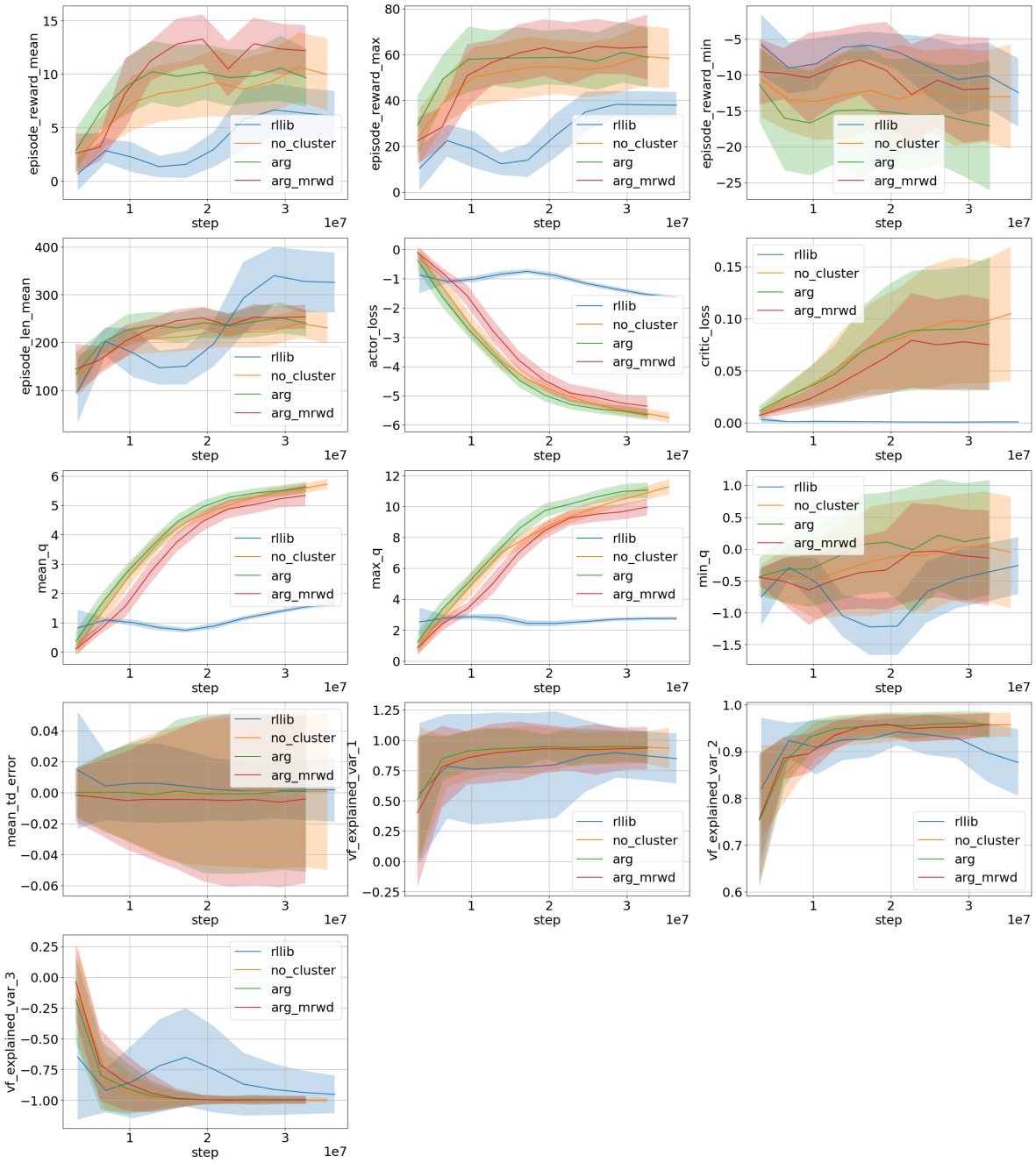
In the following plots we have that:

- “arg” stands for the default XA algorithm with clustering scheme “multiple_types”.
- “arg_mrwd” stands for the default XA algorithm with clustering scheme “multiple_types_with_reward_against_mean”.
- “no_cluster” stands for the default XA algorithm with no clustering

scheme.

- “rllib” stands for the main baseline.

The results for Easy GraphDrive are the following:



Bibliography

1. Sovrano, Francesco; Vitali, Fabio; Palmirani, Monica, *Modelling GDPR-Compliant Explanations for Trustworthy AI*, in: Electronic Government and the Information Systems Perspective, Cham, Springer, «LECTURE NOTES IN ARTIFICIAL INTELLIGENCE»,

2020, 12394, pp. 219 - 233 (atti di: International Conference on Electronic Government and the Information Systems Perspective. EGOVIS 2020, Bratislava, September 14 - 17, 2020)
2. Sovrano, Francesco; Vitali, Fabio, *From Philosophy to Interfaces: an Explanatory Method and a Tool Based on Achinstein's Theory of Explanation*, in: Intelligent User Interfaces (IUI) 2021
3. Schaul, Tom, et al. "Prioritized experience replay." arXiv preprint arXiv:1511.05952 (2015).
4. Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. No. 1. 2018.
5. *Isele, David, and Akansel Cosgun. "Selective experience replay for lifelong learning." Thirty-second AAAI conference on artificial intelligence. 2018.*
6. *https://docs.ray.io/en/master/rllib.html*
7. *https://docs.ray.io/en/master/rllib-algorithms.html#asynchronous-proximal-policy-optimization-appo*
8. *https://docs.ray.io/en/master/rllib-algorithms.html#deep-q-networks-dqn-rainbow-parametric-dqn*
9. *https://docs.ray.io/en/master/rllib-algorithms.html#deep-deterministic-policy-gradients-ddpg-td3*