MARLeME: A Multi-Agent Reinforcement Learning Model Extraction Library

Dmitry Kazhdan
Dept. of Computer Science and Technology
University of Cambridge
`dk525@cam.ac.uk`

Zohreh Shams
Dept. of Computer Science and Technology
University of Cambridge
`zohreh.shams@cl.cam.ac.uk`

Pietro Liò
Dept. of Computer Science and Technology
University of Cambridge
`pietro.lio@cl.cam.ac.uk`

Abstract

Multi-Agent Reinforcement Learning (MARL) encompasses a powerful class of methodologies that have been applied in a wide range of fields. An effective way to further empower these methodologies is to develop libraries and tools that could expand their interpretability and explainability. In this work, we introduce MARLeME: a MARL model extraction library, designed to improve explainability of MARL systems by approximating them with symbolic models. Symbolic models offer a high degree of interpretability, well-defined properties, and verifiable behaviour. Consequently, they can be used to inspect and better understand the underlying MARL system and corresponding MARL agents, as well as to replace all/some of the agents that are particularly safety and security critical.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) has achieved groundbreaking results in a wide range of fields, and is currently a highly active research area of Machine Learning (ML) [27, 1]. MARL deals with teams of agents that learn how to act optimally in stochastic environments through trial-and-error, and has been successfully applied to tasks requiring cooperative/competitive multi-agent behaviour, such as large-scale fleet management [21], swarm systems [15], and task allocation [24].

Unfortunately, the vast majority of existing MARL approaches represent decision-making policies using very complex models, such as Deep Neural Networks (DNNs) [20], making it impossible to directly understand an agent's action-selection strategy. A lack of interpretability of such systems leads to a lack of confidence in the correctness of their behaviour, which is crucial in safety-critical applications, such as self-driving cars or healthcare. Furthermore, this lack of interpretability implies that optimal strategies learned by the MARL agents cannot be used to improve our understanding of the corresponding domain. Approaches based on symbolic reasoning, on the other hand, offer interpretable, verifiable models with well-defined properties. As a result, there has recently been increasing interest in approaches capable of combining symbolic reasoning with ML [6, 8, 12, 29].

One technique that allows reaping the benefits of both ML-based and symbolic approaches is *model extraction* [2]. Model extraction refers to approaches that approximate a complex model (e.g. a DNN) with a simpler, interpretable one (e.g. a rule-based model), facilitating understanding of the complex model. Intuitively, statistical properties of the complex model should be reflected in the extracted model, provided approximation quality of the extracted model (referred to as *fidelity*) is high enough.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

This paper introduces MARLeME: a (M)ulti-(A)gent (R)einforcement (Le)arning (M)odel (E)xtraction library, designed to improve interpretability of MARL systems using model extraction. MARLeME is an open-source [1], easy-to-use, plug-and-play library, that can be seamlessly integrated with a wide range of existing MARL tasks, in order to improve explainability of the underlying MARL system. To the best of our knowledge, this is the first open-source MARL library focusing on interpretable model extraction from MARL systems.

The rest of this paper is structured as follows: Section 2 reviews work related to RL model extraction. Section 3 gives an overview of the MARLeME library. Section 4 and Section 5 present an evaluation of MARLeME using two RL benchmarks (Mountain Car and RoboCup Takeaway). Finally, Section 6 gives some concluding remarks.

2 Related Work

A range of recent work has focused on Single-Agent Reinforcement Learning (SARL) interpretable model extraction. Work in [33] generates interpretable and verifiable agent policies represented using a high-level, domain-specific programming language. Work in [13] uses model-based batch RL and genetic programming to learn policy equations. Finally, work in [5] uses boosted regression tree techniques to compute human-interpretable policies from a RL system. Unlike MARLeME, which is designed to work with teams of RL agents, all of the above approaches were designed for and evaluated using the single RL agent case only.

Work in [32] presents an approach to modelling teams of agents using Graph Neural Networks (GNNs), by training GNNs to predict agent actions, and then using their intermediate representations (e.g. the magnitudes of the edge attributes) to study the social dynamics of the MARL system (e.g. the degree of influence agents have on each other). This approach interprets the original GNN model directly, through a post-hoc analysis of its constituent parts, whereas MARLeME relies on approximating original models with simpler, interpretable ones. Crucially, GNN intepretation is still a relatively unexplored field, lacking well-established principles and best-practices. MARLeME on the other hand, makes use of well-studied and well-understood symbolic reasoning models when analysing MARL systems.

Our work is also partially related to imitation learning [14, 18]. Similarly to MARLeME, imitation learning approaches derive action-selection policies from input *trajectories* (an agent trajectory consists of a sequence of states and corresponding selected actions collected over a set of episodes). In particular, work described in [19] learns a coordination model, along with the individual policies, from trajectories of multiple coordinating agents, using imitation learning and unsupervised structure learning. Crucially, however, imitation learning, including the approach used in [19], focuses on the quality of the policy (i.e. its performance on the corresponding task), without paying attention to its interpretability. Achieving the latter while preserving performance is a unique focus of MARLeME.

3 MARLeME

A diagrammatic summary of how MARLeME can be applied to MARL systems is given in Figure 1, where interpretable models are extracted from MARL system agent data. The interprtable models approximate the behaviour of the MARL agents, and can be further analysed (e.g. through manual inspection, formal verification, or statistical analysis) to provide insight into the behaviour of the underlying MARL system at hand. Moreover, the extracted models can replace the original uninterpretable ones. The three main components of the MARLeME library are described below.

TrajectoryHandler: Agents' trajectories are essentially the agents' data MARLeME operates on. In practice, there is a range of ways in which trajectory data may be provided to MARLeME by the user (referred to as 'raw data' in Figure 1), including different data types (e.g. as text files, JSON objects, or in databases), data loading behaviour (e.g. from a server, or locally), and grouping (e.g. incremental, or batch). The TRAJECTORYHANDLER module is designed to handle these potential variations, before passing the formatted trajectory data to the MODELEXTRACTOR component.

ModelExtractor: The MODELEXTRACTOR component is designed to extract interpretable models from formatted trajectory data provided by the TRAJECTORYHANDLER. This component can utilise

[1]https://github.com/dmitrykazhdan/MARLeME

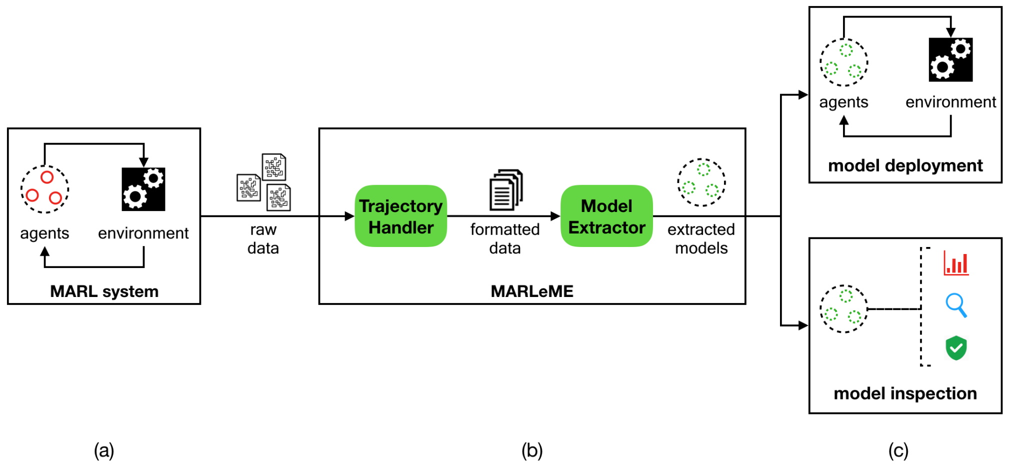


Figure 1: (a) A MARL system consists of a set of RL agents (red circles) interacting with each other and their environment. (b) MARLeME uses agent trajectory data obtained from the MARL system (raw data), in order to extract a set of interpretable models (green circles) from that data, which approximate the behaviour of the original agents. (c) The interpretable models can replace the original ones (model deployment) or be investigated to better understand the behaviour of the underlying MARL system (model inspection).

extraction algorithms for different types of interpretable models, and provide various underlying implementations for these algorithms (e.g. GPU-optimised). The resulting models are represented using TEAM and AGENT components, which are described below.

Team + Agent: Extracted agent models (shown by green circles in Figure 1) are represented using the TEAMMODEL and AGENTMODEL components. An AGENTMODEL represents a single agent, and a TEAMMODEL represents a group of agents (e.g. teams or sub-teams), giving the user an opportunity to encapsulate agent interactions (e.g. state information sharing). Together, these two classes capture the full spectrum of possible agent interactions: from fully centralised, with all agents represented by a single TEAMMODEL component, to fully decentralised, with all agents running independently (with every agent represented by an AGENTMODEL component). These extracted models can be inspected in order to extract new domain knowledge regarding the corresponding MARL task, or deployed instead their original models, providing more interpretable systems with verifiable properties.

MARLeME can be applied to a vast range of existing MARL systems, by simply logging trajectories of the corresponding MARL agents. Furthermore, MARLeME can be used to extract a wide variety of model types, such as decision trees, or fuzzy rule-based models [3]. This allows successful integration of RL approaches with the well-studied verification, knowledge extraction, and reasoning approaches associated with symbolic models.

3.1 Abstract Argumentation

Extracted models presented in this paper rely on Abstract Argumentation (AA) [7], due to the popularity of AA in autonomous agent and multi-agent decision making [11, 25, 28]. Argumentation allows reasoning in presence of incomplete and inconsistent knowledge, which is often the case in multi-agent settings, where the agents are unlikely to have complete knowledge of the environment and may have conflicting objectives. In addition, AA-based decision-making models are interpretable and thus highly suitable for interpretable model extraction tasks. Despite these advantages, to the best of our knowledge, this is the first time that AA-based models have been used for interpretable model extraction in RL.

In this work we make use of Value-based Argumentation Frameworks (VAFs) which are based on Abstract Argumentation Frameworks (AFs). Informally, an $AF = (Arg, Att)$, consists of a set of arguments (Arg), which are defeasible rules of inference, and a binary attack relation between arguments $Att \subseteq Arg \times Arg$ that represent conflicts between arguments. In a value-based argumentation framework $VAF = (Arg, Att, V, val, Valpref)$, there are values attached to arguments that represent their relative utility and hence dictate which argument is preferred to another in face of conflict: V is a set of possible argument values , $val : Arg \rightarrow V$ is a function that assigns values to arguments, and $Valpref$ is an ordering over these values. Naturally, argumentation frameworks can be represented as directed graphs, where nodes are arguments and an edge from node A to B represent an attack from the former to the latter.

When representing knowledge using a VAF, it is possible to determine which arguments in the VAF are 'winning' or optimal, by defining suitable *semantics* (sets of well-defined rules) that specify whether an argument should be *accepted* (i.e. treated as optimal). The set of accepted arguments of a VAF is referred to as its *extension*. In this work, we make use of the *grounded extension* (GE) semantics when computing acceptable arguments from a VAF, that will be described in the remainder of this section. Details regarding other extension types may be found in [7].

Given an AF (Arg, Att), suppose $S \subseteq Arg$, then:

- S attacks an argument A iff some member of S attacks A
- S is *conflict-free* iff S attacks none of its members
- S *defends* an argument A iff S attacks all arguments attacking A
- S is an *admissible extension* of an AF iff S is conflict-free and defends all its elements
- S is a *complete extension* of an AF iff it is an admissible extension and every argument defended by S belongs to S
- S is the *grounded extension* of an AF iff it is the smallest element (with respect to set inclusion) among the complete extensions of the AF.

Intuitively, a GE represents a 'sceptical' solution, consisting only of non-controversial arguments (those that do not attack each other).

3.2 Abstract Argumentation Agents

The interpretable agents presented in this work (referred to as *AA-based agents*) rely on VAFs for knowledge representation, and GE semantics for action derivation. The arguments used by these agents are *action arguments*, representing action-based heuristics. An action argument A has the form 'If *condition* holds then *agent should do action*'. In this definition, *condition* is a boolean function of agent state (specifying whether or not argument A is applicable in that state), *action* specifies the recommended action, and $agent$ refers to an agent in the team. In this work, all AA-based agents have Att defined as follows: argument A attacks argument B iff they recommend the same action to different agents, or different actions to the same agent. For a given agent, arguments in its argument set Arg that recommend actions to that agent are referred to as its *primary arguments* in the rest of this work. In our case, V is the integer set, $Valpref$ is the standard integer ordering, and val is a lookup table, mapping arguments to their integer values.

When deriving an action from a state, an AA-based agent computes all arguments in Arg that are applicable in that state (all arguments for which the argument's condition is true). Then, the agent constructs a VAF from these arguments (using attack construction rules Att, and argument values val). Finally, the agent uses GE semantics to compute the acceptable primary arguments of the VAF. By definition, all arguments acceptable under GE semantics are non-conflicting, thus, given the agent's definition of Att, all acceptable primary arguments must recommend the same action, which is the action executed by the agent.

Computing applicable arguments, VAF construction, and GE computation are all computationally cheap steps, implying that the AA-based agents successfully scale to complex tasks.

3.3 Abstract Argumentation Agent Model Extraction

Extraction of AA-based agents is performed by the algorithm described in Appendix A. This algorithm assumes both the agent argument set Arg and the attack construction rules Att are provided by the user, and uses these to derive the argument value ordering val from the input MARL trajectory data. The agent models are consequently generated using the user-specified Arg and Att, together with the derived val. The derived value ordering reflects the relative utility of the agent arguments Arg, and can thus serve as an indication of which arguments the agent primarily relies on during action selection, allowing interpretation of agent strategy (as will be shown in Section 5.1.3).

4 Experiments

We evaluate MARLeME on two well-known RL case studies: Mountain Car [23] and RoboCup Takeaway [16], using AA-based agents as the extracted models. The remainder of this section describes the two case studies, as well as the corresponding AA-based agent setups.

4.1 Mountain Car

Mountain Car was used to show how MARLeME can be applied to simpler tasks, including tasks consisting of a single RL agent. Furthermore, Mountain Car was used to demonstrate that MARLeME can be used in conjunction with other open-source RL tools, such as OpenAI Gym [4]. The Mountain Car RL task consists of a car that is on a one-dimensional track, positioned between two "mountains" [4]. The goal is to drive up the mountain to the right of the car; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. The RL agent here relies on the Deep Q-Network RL agent implementation [2], however alternative implementations [3] also exists.

Given the simplicity of the environment and the state and action spaces, the argument set Arg for the Mountain Car AA-based agent was generated using a brute-force approach defined as follows. The Mountain Car state space consists of two variables: *position* and *velocity*, with $position \in [-1.2, 0.6]$, and $velocity \in [-0.07, 0.07]$, as well as 3 possible actions: $actions = \{push\_left, no\_push, push\_right\}$, resembling which direction the car should drive towards. Value ranges for both state variables were split into sets of sub-ranges P and V, each consisting of 20 equally-sized sub-ranges, with $P = (p_0, ..., p_{19})$ and $V = (v_0, ..., v_{19})$, where $p_i \equiv [-1.2 + 0.09 * i, -1.2 + 0.09 * (i + 1)]$, and $v_i \equiv [-0.07 + 0.007 * i, -0.07 + 0.007 * (i + 1)]$. Using these ranges, the argument set Arg was defined as $Arg = \{$ If $velocity \in v$ and $position \in s$ then do $a \mid v \in V, s \in S, a \in actions\}$. Thus, given that $|P| = 20$, $|V| = 20$, and $|actions| = 3$, there were 1200 arguments generated in total.

4.2 RoboCup Takeaway

RoboCup Takeaway was used to evaluate MARLeME on a more challenging task, involving a large, continuous state space, multiple interacting agents, and long, variable delays in action effects. RoboCup Takeaway was proposed in [16] in order to facilitate RL research in the context of RoboCup Soccer [4], and focuses on two teams of simulated agents playing the Takeaway game in a two-dimensional virtual soccer stadium. In Takeaway, $N + 1$ hand-coded keepers are competing with N independent learning takers on a fixed-size field. Keepers attempt to keep possession of the ball, whereas takers attempt to win possession of the ball. The game consists of a series of episodes, and an episode ends when the ball goes off the field or any taker gets the ball. A new episode starts immediately with all the players reset. We focus here on the 4v3 Takeaway game, consisting of 4 keepers and 3 takers. The MARL taker team consisted of homogeneous, independent learning RL agents relying on the SARSA(λ) algorithm with tile-coding function approximation [31].

When defining arguments for AA-based taker agents, we made use of the work in [10, 9], which explored RL agent convergence, and relied on Takeaway as the case study. The mentioned works defined a set of arguments containing useful domain heuristics relevant to the takeaway game, which will be described below.

Every AA-based taker uses the same argument set Arg, consisting of the following arguments (here i is the taker index, ranging from 1 to 3, and p is the keeper index, ranging from 1 to 4):

- $TackleBall_i$: If T_i is closest to the ball holder, T_i should tackle the ball
- $OpenKeeper_{i,p}$: If a keeper K_p is in a quite 'open' position, T_i should mark this keeper
- $FarKeeper_{i,p}$: If a keeper K_p is 'far' from all takers, T_i should mark this keeper
- $MinAngle_{i,p}$: If the angle between T_i and a keeper K_p, with vertex at the ball holder, is the smallest, T_i should mark this keeper
- $MinDist_{i,p}$: If T_i is closest to a keeper K_p, T_i should mark this keeper

[2]https://github.com/adibyte95/Mountain_car-OpenAI-GYM

[3]https://github.com/openai/gym/wiki/Leaderboard

[4]https://rcsoccersim.github.io/

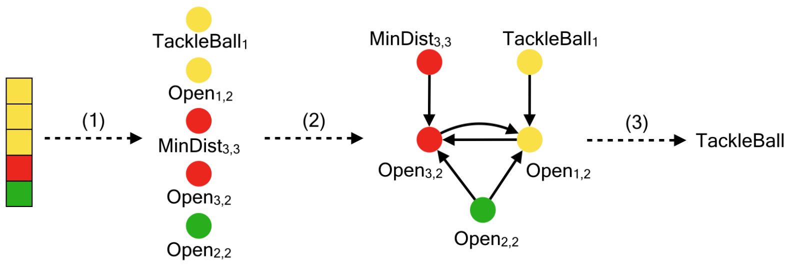


Figure 2: AA-based Taker Agent Model action selection sub-steps for taker T_1. (1) Determines the applicable agent arguments for all takers T_1, T_2, T_3 (shown by yellow, green, and red circles, respectively), using the input state attributes. (2) Constructs a VAF from the argument set by injecting attacks between arguments (using attack construction rules and the argument value ordering). (3) Derives an action from the VAF for T_1, using GE semantics.

5 Results

This section presents the results obtained by evaluating MARLeME on the chosen case studies using the previously described AA-based agent models. Evaluation is both qualitative, demonstrating how we can use MARLeME to inspect the extracted models and extract useful knowledge from them, and quantitative, demonstrating how well extracted models perform on their original tasks, and how closely they approximate their original models.

5.1 Model Inspection

Given that extracted models serve as approximations of original models, they may be used to identify and understand the high-level strategies of the original agents, as well as to explain individual action selection. This can be used to extract knowledge about the domain, the nature of agent interactions, and the types of roles agents can take in a team.

In this work, we rely on manual inspection of extracted models. However, MARLeME can be used with many other inspection approaches, depending on the task at hand. In case of larger extracted models that are difficult to analyse manually, inspection may be done by applying automated verification techniques to extracted models, formally proving their properties. Alternatively, extracted models can be analysed empirically, by relying on statistical libraries, such as scikit-learn [26], or Pandas [22].

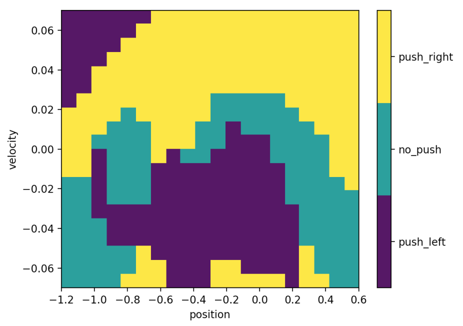
Our inspection of extracted models presented here is two-fold: firstly (Section 5.1.1), we inspect the extracted models at the level of action-selection, showing that extracted models select actions using an interpretable sequence of steps. Secondly (Section 5.1.2 and Section 5.1.3), we inspect the extracted models at the policy level.

5.1.1 Action Selection

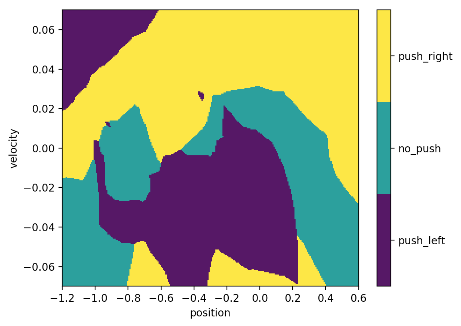
The AA-based agents proposed here make use of an interpretable action selection strategy when deriving an action from a state, compared to their original RL models (an example is shown in Figure 2). Any action selected by an AA-based agent in a given state can be easily traced back to the original set of defined heuristics (Arg), their interactions (Att), and their relative values (val). As described in Section 3.2, an agent's argument set contains arguments recommending actions to the agent itself, as well as arguments recommending actions to its teammates (as shown in Figure 2). Thus, action selection analysis can be used to study the motivations of an individual agent, as well as inter-agent interactions, that influence action selection.

5.1.2 Model Visualisation

In case of relatively simple tasks, such as Mountain Car, it is possible to visualise the action-selection policy of the extracted model directly, by plotting policy actions in all possible states, as shown in



(a) Extracted Model Policy



(b) Original Model Policy

Figure 3: Policy visualisations of the extracted and original Mountain Car models, showing policy outputs in all possible environment states.

Agent 1	**Agent 2**	**Agent 3**
$TackleBall_1$	$TackleBall_2$	$TackleBall_3$
$MinAngle_{1,3}$	$MinAngle_{2,3}$	$MinAngle_{3,4}$
$MinAngle_{1,2}$	$MinDist_{2,3}$	$MinDist_{3,4}$
$MinDist_{1,2}$	$OpenKeeper_{2,3}$	$OpenKeeper_{3,4}$
$MinDist_{1,3}$	$FarKeeper_{2,3}$	$FarKeeper_{3,4}$

Table 1: Highest-valued primary arguments of the AA-based Taker agents

Figure 3(a). This visualisation allows direct exploration of the extracted model, allowing users to derive useful information by observing model behaviour in various states.

Furthermore, visualisation allows directly comparing the extracted model policy to the original model policy (shown in Figure 3(b)), allowing a user to easily determine where and how the extracted model differs from the original one.

5.1.3 Strategy Analysis

In case of more challenging tasks, such as RoboCup Takeaway, the extracted models may be inspected by studying their constituent components. AA-based agent models offer an intuitive way of analysing agent strategies as a whole, exploring individual agent behaviour, and cooperative team behaviour, by using their argument value orderings val. As described in Section 3.2, an agent chooses an action recommended by its primary arguments. Given that arguments with higher values defeat arguments with lower values, high-valued primary arguments represent the main heuristics used by an agent during decision-making. Thus, analysing the highest-valued primary arguments can be used to explore individual agent strategies, and their behaviour. Table 1 gives the top 5 highest-valued primary arguments for every taker.

For all three agents, their *TackleBall* argument has the highest value, implying that tackling the ball when closest to the keeper is of paramount importance for every agent. For Agent 1, the four remaining arguments show that this agent focuses on tackling K_2, or K_3, if it is ‘closest’ to them (closest by angle, or distance). For Agents 2 and 3, their four remaining arguments all recommend marking K_3 and K_4 (respectively), implying that these agents focus on tackling these keepers throughout the game, preventing the ball-holder from passing to them.

The above analysis demonstrates the specialised roles taken by the different agents (e.g. different agents in the above example focus on tackling different keepers). As discussed previously, this knowledge may be used to study the Takeaway game in more detail, and attempt to synthesise winning agent strategies. For instance, the above example demonstrates that a possible strategy is for one taker to tackle the ball-holder, and the other two to ‘spread out’ and tackle distant keepers, in order to prevent them from receiving the ball.

	Original Models	**Extracted Models**
Mountain Car	55.3 +/- 11.3 ms	2.96 +/- 0.4 ms
Takeaway	9.55 +/- 3.3 s	10.71 +/- 3.8 s

Table 2: Task performance of the original models, and the extracted models

5.2 Model Deployment

The utility of extracted models was also evaluated quantitatively, by replacing the original RL agents with their corresponding AA-based agents, and comparing the task performance of the two approaches. Task performance was measured by recording average episode duration for 1000 episodes. In both case studies, a shorter episode duration signifies better performance. The above setup was run on a MacBook Pro computer with a 4-core 2.5GHz Intel Core i7 processor, and 16GB of main memory. The performances of the different approaches are shown in Table 2.

Increased interpretability of extracted models typically comes at a cost of their reduced flexibility, implying that they may perform sub-optimal decision making, compared to the original models, and therefore incur a reduction in performance, as is the case with AA-based taker agents.

On the other hand, extracted models are often simpler, and therefore more lightweight, compared to the original models. This implies that extracted models often take less time to compute actions from states. In case of simpler tasks such as Mountain Car, where the reduction in flexibility is not substantial, this speedup may have a large effect on task performance (a factor of 18 speedup in case of Mountain Car).

Overall, extracted models offer a substantial increase in interpretability, at the expense of reduced flexibility, which may percolate into reduced performance when considering more challenging tasks.

5.3 Model Fidelity

Finally, we evaluate fidelity of extracted models using the *0-1 loss* [30] computed from 1000 episodes of trajectory data, in order to assess the degree to which we can rely on the extracted models when studying the original ones. In case of Mountain Car, the extracted model achieved a high fidelity score of 0.93. In case of the more challenging Takeaway task, agent 1 and agent 3 achieved high fidelity scores of 0.86 and 0.84 (respectively), whilst agent 2 achieved a relatively lower fidelity score of 0.68. These results indicate that model extraction for agent 2 could be further improved by using a more flexible extracted model (e.g. an AA-based agent model with a larger argument set Arg), consequently making extracted model interpretation even more meaningful.

6 Conclusions

We introduce MARLeME, a Multi-Agent Reinforcement Learning Model Extraction library, designed to improve interpretability of MARL systems by approximating MARL agents with symbolic models, which can be used to study the behaviour of the underlying agents, as well as to replace all/some of them. MARLeME can be applied to a vast range of existing MARL systems, and can be used with a wide variety of symbolic model types. Thus, MARLeME allows successful integration of RL approaches with the well-studied verification, knowledge extraction, and reasoning approaches associated with symbolic models.

In this work, we focus on applying MARLeME to trained MARL agents, in order to extract useful domain knowledge by studying their behaviour. That said, given its ease-of-use, flexibility, and extendibility, MARLeME can be applied in a broader range of RL scenarios, depending on the task at hand. For instance, MARLeME can be used to periodically inspect RL models over time during agent training, seeing how MARL agent policies evolve, and exploring their convergence.

With the rapidly increasing interest in MARL systems and the development of associated tools (such as the recently released OpenAI MARL environments repository [5]), we believe MARLeME can play a fundamental role in enriching such MARL systems and tools with explainability and interpretability.

[5]https://openai.com/blog/emergent-tool-use/

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning, 2017.

[2] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via model extraction. *CoRR*, abs/1706.09773, 2017.

[3] J. M. Benitez, J. L. Castro, and I. Requena. Are artificial neural networks black boxes? *Trans. Neur. Netw.*, 8(5):1156–1164, September 1997.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Alexander Brown and Marek Petrik. Interpretable reinforcement learning with ensemble methods. *arXiv preprint arXiv:1809.06995*, 2018.

[6] Pedro Domingos and Daniel Lowd. Unifying logical and statistical ai with markov logic. *Commun. ACM*, 62(7):74–83, June 2019.

[7] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[8] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Int. Res.*, 61(1):1–64, January 2018.

[9] Yang Gao and F Toni. *Argumentation accelerated reinforcement learning*. PhD thesis, Imperial College London, 2014.

[10] Yang Gao and Francesca Toni. Argumentation accelerated reinforcement learning for cooperative multi-agent systems. In *ECAI*, pages 333–338, 2014.

[11] Yang Gao, Francesca Toni, Hao Wang, and Fanjiang Xu. Argumentation-based multi-agent decision making with privacy preserved. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1153–1161. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[12] Artur d'Avila Garcez, Marco Gori, Luis C Lamb, Luciano Serafini, Michael Spranger, and Son N Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *arXiv preprint arXiv:1905.06088*, 2019.

[13] Daniel Hein, Steffen Udluft, and Thomas A Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76: 158–169, 2018.

[14] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

[15] Maximilian Huttenrauch, Adrian Sosic, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*, 2017.

[16] Atil Iscen and Umut Erogul. A new perspective to the keepaway soccer: the takers. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1341–1344. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[17] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, November 1962.

[18] Jens Kober and Jan Peters. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, 17(2):55–62, 2010.

[19] Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1995–2003. JMLR.org, 2017.

[20] Yuxi Li. Deep reinforcement learning: An overview. *arXiv:1701.07274*, 2017.

[21] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783. ACM, 2018.

[22] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[23] Andrew Moore. Efficient memory-based learning for robot control. June 2002.

[24] Dhouha Ben Noureddine, Atef Gharbi, and Samir Ben Ahmed. Multi-agent deep reinforcement learning for task allocation in dynamic environment. In *ICSOFT*, pages 17–26, 2017.

[25] Santiago Ontanon and Enric Plaza. An argumentation framework for learning, information exchange, and joint-deliberation in multi-agent systems. *Multiagent and Grid Systems*, 7(2-3): 95–108, 2011.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Comput. Surv.*, 51(5):92:1–92:36, September 2018.

[28] Regis Riveret, Yang Gao, Guido Governatori, Antonino Rotolo, Jeremy Pitt, and Giovanni Sartor. A probabilistic argumentation framework for reinforcement learning agents. *Autonomous Agents and Multi-Agent Systems*, pages 1–59, 2019.

[29] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pages 3788–3800, 2017.

[30] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[31] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[32] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

[33] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. *arXiv preprint arXiv:1804.02477*, 2018.

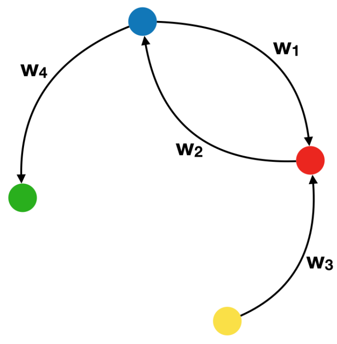


Figure 4: Simple example of an APG consisting of four arguments (represented by coloured circles), and weighted preference edges with weights $w_1, ..., w_4$.

A AA-based Model Extraction

The algorithm introduced in this work for extracting AA-based agent models from MARL agent trajectories is given in Algorithm 1. Algorithm 1 assumes that the set of candidate arguments (*Arguments*) is fixed, and is passed as an input parameter. The algorithm focuses on deriving an argument ordering (*extractedOrdering*) using the input trajectories (*Trajectories*) and a default input ordering (*DefaultOrdering*).

Algorithm 1 extractOrdering($Trajectories, Arguments, DefaultOrdering$)

```
1:  $preferenceGraph = newArgPreferenceGraph()$
2:  **for** $(state, action) \in Trajectories$ **do**
3:      $applicableArguments = getApplicableArguments(Arguments, state)$
4:      $relevantArguments = \{arg \in applicableArgs \mid arg.Conclusion == action\}$
5:      $irrelevantArguments = applicableArguments - relevantArguments$
6:      **for** $relevantArg \in relevantArguments$ **do**
7:          **for** $irrelevantArg \in irrelevantArguments$ **do**
8:              $preferenceGraph.incrementEdge(relevantArg, irrelevantArg)$
9:          **end for**
10:     **end for**
11: **end for**
12: $DAG = convertToAcyclic(preferenceGraph)$
13: $extractedOrdering = topologicalSort(DAG.nodes, DAG.edges, DefaultOrdering)$
14: **return** $extractedOrdering$
```

Algorithm 1 relies on topological sorting and operates on a novel Argument Preference Graph (APG) structure (also introduced in this work). The fundamental idea behind the algorithm is that for a given pair of arguments (A, B) and a RL agent trajectory, the argument that is 'in closer agreement' with the agent contains relatively more useful information, and should thus have a relatively higher value. These pair-wise argument preferences are stored in an APG. An APG is a weighted directed graph (with non-negative weights) in which the nodes represent arguments, and weighted edges represent preferences between arguments. Figure 4 shows an example of an APG constructed by the *extractOrdering* method.

The relative argument utility is computed by iterating over all $(state, action)$ pairs of input agent trajectories. For every such $(state, action)$ pair, the APG is updated by incrementing the weights of directed edges between all pairs of arguments (A, B), where both A and B are applicable in $state$, A was in agreement with the agent (recommended the same action), and B was not (recommended a different action). Thus, a directed edge with weight w from argument A to argument B in an APG implies that A was applicable and in agreement with the agent, whilst argument B was applicable was not in agreement with the agent, for w different trajectory states. A high value of weight w of a directed edge (A, B) thus signifies that argument A frequently suggested more relevant information than argument B, implying that argument A should have a higher value.

Once the APG is constructed, an argument ordering is extracted from it by first converting it into a Directed Acyclic Graph (DAG) (by calling the *convertToAcyclic* method), and then topologically sorting the DAG (by calling the *topologicalSort* method). Graph cycle removal is achieved by relying on a pruning heuristic, which removes all edges of weight less than a given pruning value p from the graph. Topological sorting is achieved using a slight variation of Kahn's algorithm [17], where the node extraction order relies on the default argument ordering given by the user, instead of relying on stack or queue structures.

Overall, the ordering extraction algorithm relies on both knowledge derived from the trajectory data (in the form of an APG), and on heuristics injected by the user (the default ordering). The utilised pruning approach is advantageous since it provides a straightforward way of controlling the tradeoff between the importance of prior knowledge and extracted knowledge, by using the pruning parameter p. A higher p value removes more edges and yields a sparser APG with more possible argument orderings, leading to greater reliance on the default ordering. A lower p value has the reverse effect.