Jet grooming through reinforcement learning

Stefano Carrazza
TIF Lab, Dipartimento di Fisica, Università degli Studi di Milano and INFN Milan, Via Celoria 16, 20133 Milano, Italy

Frédéric A. Dreyer
Rudolf Peierls Centre for Theoretical Physics, University of Oxford, Clarendon Laboratory, Parks Road, Oxford OX1 3PU, United Kingdom

(Received 29 March 2019; published 15 July 2019)

We introduce a novel implementation of a reinforcement learning (RL) algorithm which is designed to find an optimal jet grooming strategy, a critical tool for collider experiments. The RL agent is trained with a reward function constructed to optimize the resulting jet properties, using both signal and background samples in a simultaneous multilevel training. We show that the grooming algorithm derived from the deep RL agent can match state-of-the-art techniques used at the Large Hadron Collider, resulting in improved mass resolution for boosted objects. Given a suitable reward function, the agent learns how to train a policy which optimally removes soft wide-angle radiation, allowing for a modular grooming technique that can be applied in a wide range of contexts. These results are accessible through the corresponding GroomRL framework.

DOI: 10.1103/PhysRevD.100.014014

I. INTRODUCTION

Jets are one of the most common objects appearing in proton-proton colliders such as the Large Hadron Collider (LHC) at CERN. They are defined as collimated bunches of high-energy particles, which emerge from the interactions of quarks and gluons, the fundamental constituents of the proton [1,2]. In modern analyses, final-state particle momenta are mapped to jet momenta using a sequential recombination algorithm with a single free parameter, the jet radius R, which defines up to which angle particles can get recombined into a given jet [3–5].

An example of an LHC collision resulting in two jets is shown in Fig. 1, where the towers correspond to energy deposits in the calorimeter. The right-hand side gives a schematic visualization of two different representations of jets, either as an image where the pixel intensity encodes the energy flow in that phase-space region [6] or as a tree defined by the recombination sequence of the jet algorithm.

Due to the very high energies of its collisions, the LHC is routinely producing heavy particles, such as top quarks and vector bosons, with transverse momenta far greater than their rest mass. When these objects are sufficiently energetic (or boosted), they can often generate very collimated decays, which are then reconstructed as a single fat jet. These fat jets originating from boosted objects can be distinguished from standard quark and gluon jets by studying differences in their radiation patterns. Since the advent of the LHC program, the physics of the substructure of jets has matured into a remarkably active field of research that has become notably conducive to applications of recent machine learning techniques [7–27].

A particularly useful set of tools for experimental analyses are jet grooming algorithms [28–33], defined as a postprocessing treatment of jets to remove soft wide-angle radiation which is not associated with the underlying hard substructure. Grooming techniques play a crucial role in Standard Model measurements [34,35] and in improving the boson- and top-tagging efficiencies at the LHC.

In this article we introduce a novel framework, which we call GroomRL, to train a grooming algorithm using reinforcement learning (RL) [36,37]. To this end, we decompose the problem of jet grooming into successive steps for which a reward function can be designed taking into account the physical features that characterize such a system. We then use a modified implementation of a Deep Q-Network (DQN) agent [36,38] and train a dense neural network (NN) to optimally remove radiation unassociated from the core of the jet. The trained model can then be applied on other datasets, showing improved resolution compared to state-of-the-art techniques as well as a strong resilience to nonperturbative effects.

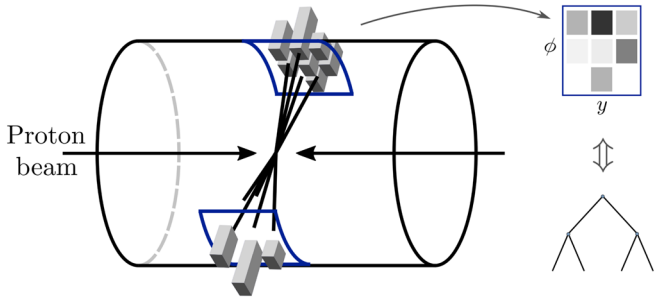


FIG. 1. Jets emerging from a proton-proton collision at the LHC and their representation as images in rapidity-azimuth (y, ϕ) space or as clustering trees.

II. JET REPRESENTATION

Let us start by introducing the representation we use for jets. We take the particle constituents of a jet, as defined by any modern algorithm, and recombine them using a Cambridge/Aachen (CA) sequential clustering algorithm [4,39]. The CA algorithm does a pairwise recombination, adding together the momenta of the two particles with the closest distance as defined by the measure

$$\Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2, \tag{1}$$

where y_i is the rapidity, a measure of relativistic velocity along the beam axis, and ϕ_i is the azimuthal angle of particle i around the same axis. This clustering sequence is then used to recast the jet as a full binary tree, where each of the nodes contains information about the kinematic properties of the two parent particles. For each node i of the tree we define an object $\mathcal{T}^{(i)}$ containing the current observable state s_t, as well as a pointer to the two children nodes and one to the parent node. The children nodes a and b are ordered in transverse momentum such that $p_{t,a} > p_{t,b}$, and we label a the "harder" child and b the "softer" one. The set of possible states is defined by a five-dimensional box, such that the state of the node is a tuple

$$s_t = \{z, \Delta_{ab}, \psi, m, k_t\}, \tag{2}$$

where $z = p_{t,b}/(p_{t,a} + p_{t,b})$ is the momentum fraction of the softer child b, $\psi = \tan^{-1}(\frac{y_b - y_a}{\phi_a - \phi_b})$ is the azimuthal angle around the i axis, m is the mass, and $k_t = p_{t,b}\Delta_{ab}$ is the transverse momentum of b relative to a.

A. Grooming algorithm

A grooming algorithm acting on a jet tree can be defined by a simple recursive procedure which follows each of the branches and uses a policy $\pi_g(s_t)$ to decide based on the values of the current tuple s_t whether to remove the softer of the two branches. This is shown in Algorithm 1, where the minus sign is understood to mean the update of the kinematics of a node after removal of a soft branch.

Algorithm 1. Grooming.

Input: policy π_g, binary tree node $\mathcal{T}^{(i)}$
$a_t = \pi_g(\mathcal{T}^{(i)} \to s_t)$
if $a_t = 1$ **then**
  $\mathcal{T}^{(j)} = \mathcal{T}^{(i)}$
  while $\mathcal{T}^{(j)} = (\mathcal{T}^{(j)} \to \text{parent})$ **do**
    $\mathcal{T}^{(j)} \to s_t = (\mathcal{T}^{(j)} \to s_t) - (\mathcal{T}^{(i)} \to b \to s_t)$
  end while
  $\mathcal{T}^{(i)} = (\mathcal{T}^{(i)} \to a)$
  Grooming$(\pi_g, \mathcal{T}^{(i)})$
else
  Grooming$(\pi_g, \mathcal{T}^{(i)} \to a)$
  Grooming$(\pi_g, \mathcal{T}^{(i)} \to b)$
end if

The grooming policy $\pi_g(s_t)$ returns an action $a_t \in \{0, 1\}$, with $a_t = 1$ corresponding to the removal of a branch and $a_t = 0$ leaving the node unchanged. The state s_t is used to evaluate the current action values $Q^*(s, a)$ for each possible action, which in turn are used to determine the best action at this step through a greedy policy.

An example of the action of a grooming algorithm on a tree is shown in Fig. 2, where the groomed branches are indicated in red. The tree nodes whose kinematics have been modified by the removal of a branch are indicated with a prime.

It is easy to translate modern grooming algorithms in this language. For example, recursive soft drop (RSD) [33] corresponds to a policy

$$\pi_{\text{RSD}}(s_t) = \begin{cases} 0 & \text{if } z > z_{\text{cut}}\left(\frac{\Delta_{ab}}{R_0}\right)^{\beta}, \\ 1 & \text{else}, \end{cases} \tag{3}$$

where z_{cut}, β and R_0 are the parameters of the algorithm and 1 corresponds as before to the action of removing the tree branch with smaller transverse momentum.

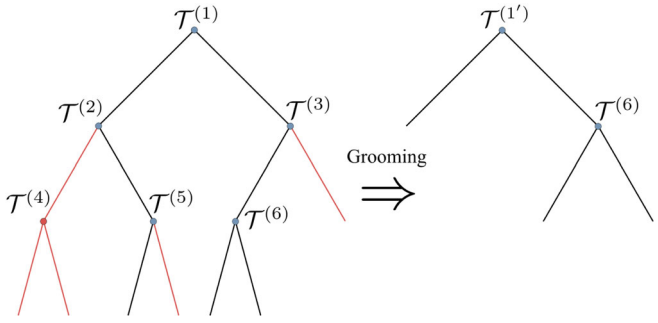


FIG. 2. Example of grooming on the binary tree representation of a jet with the resulting tree after applying Algorithm 1 shown on the right. Groomed branches are indicated in red, and the corresponding nodes have been removed on the right-hand side.

III. SETTING UP A GROOMING ENVIRONMENT

In order to find an optimal grooming policy π_g, we introduce an environment and a reward function, formulating the problem in a way that can be solved using a RL algorithm.

We initialize a list of all trees used for the training, from which a tree is randomly selected at the beginning of each episode. We then start by adding the root of the current tree to an empty priority queue, which orders the nodes it contains according to their Δ_{ab} value.[1]

Each step consists in removing the first node from the priority queue and taking an action on which of its branches to keep based on the state s_t of that node. Once a decision has been taken on the removal of the softer branch, and the parent nodes have been updated accordingly, the remaining children of the node are added to the priority queue. The reward function is then evaluated using the current state of the tree. The episode terminates once the priority queue is empty.

The framework described here deviates from usual RL implementations in that the range of possible states for any episode are fixed at the start. The transition probability between states $\mathcal{P}(s_{t+1}|s_t, a_t)$ therefore does not necessarily depend very strongly on the action, although a grooming action can result in the removal of some of the future states and will therefore still have an effect on the distribution.

For our implementation, we have relied on the GYM v0.12.1 [40] and KERAS-RL v0.4.2 [41] libraries for the reinforcement learning component, while the neural network is set up using KERAS v2.2.4 [42] with TENSORFLOW v1.13.1 [43] as the back end.

A. Finding optimal hyperparameters

The optimal choice of hyperparameters, both for the model architecture and for the grooming parameters, is determined using the distributed asynchronous hyperparameter optimization library HYPEROPT [44].

The performance of an agent is evaluated by defining a loss function, which is evaluated on a distinct validation set consisting of 50 000 signal and background jets. For each sample, we evaluate the jet mass after grooming of each jet and derive the corresponding distribution. To calculate the loss function $\mathcal{L}$, we start by determining a window $(w_{\min}, w_{\max})$ containing a fraction $f = 0.6$ of the final jet masses of the groomed signal distribution, defining w_{med} as the median value on that interval. The loss function is then defined as

$$\mathcal{L} = \frac{1}{5}|w_{\max} - w_{\min}| + |m_{\text{target}} - w_{\text{med}}| + 20 f_{\text{bkg}}, \quad (4)$$

where f_{bkg} is the fraction of the groomed background sample contained in the same interval and m_{target} is a reference value for the signal.

We scan hyperparameters using 1000 iterations and select the ones for which the loss $\mathcal{L}$ evaluated on the validation set is minimal. In practice we will do three different scans: to determine the best parameters of the reward function, to find an optimal grooming environment, and to determine the architecture of the DQN agent. The scan is performed by requiring HYPEROPT to use a uniform search space for continuous parameters, a log-uniform search space for the learning rate and a binary choice for all integer or boolean parameters. The optimization used in all the results presented in this work rely on the tree-structured Parzen estimator algorithm.

B. Defining a reward function

One of the key ingredients for the optimization of the grooming policy is the reward function used at each step during the training. We consider a reward with two components: a first piece evaluated on the full tree and another that considers only the kinematics of the current node.

The first component of the reward compares the mass of the current jet to a set target mass, typically the mass of the underlying boosted object. We implement this mass reward using a Cauchy distribution, which has two free parameters, the target mass m_{target} and a width Γ, so that

$$R_M(m) = \frac{\Gamma^2}{\pi(|m - m_{\text{target}}|^2 + \Gamma^2)}. \quad (5)$$

Separately, we calculate a reward on the current node which gives a positive reward for the removal of wide-angle soft radiation, as well as for leaving intact hard-collinear emissions. This provides a baseline behavior for the groomer. We label this reward component "soft drop" due to its similarity with the soft-drop condition [32] and implement it through exponential distributions

$$\begin{aligned} R_{\text{SD}}(a_t, \Delta, z) = {} & a_t \min(1, e^{-\alpha_1 \ln(1/\Delta) + \beta_1 \ln(z_1/z)}) \\ & + (1 - a_t)\max(0, 1 - e^{-\alpha_2 \ln(1/\Delta) + \beta_2 \ln(z_2/z)}), \end{aligned} \quad (6)$$

where $a_t = 0, 1$ is the action taken by the policy and α_i, β_i, and z_i are free parameters. The two terms determining R_{SD} are shown in the lower panel of Fig. 3, using parameter values determined through asynchronous hyperparameter optimization, shown in the upper row of the figure.

The total reward function is then given by

$$R(m, a_t, \Delta, z) = R_M(m) + \frac{1}{N_{\text{SD}}} R_{\text{SD}}(a_t, \Delta, z). \quad (7)$$

[1]This is not strictly necessary for a fully recursive algorithm but allows for easier extensions to fixed depth algorithms such as the modified mass drop tagger [31] and soft drop [32].

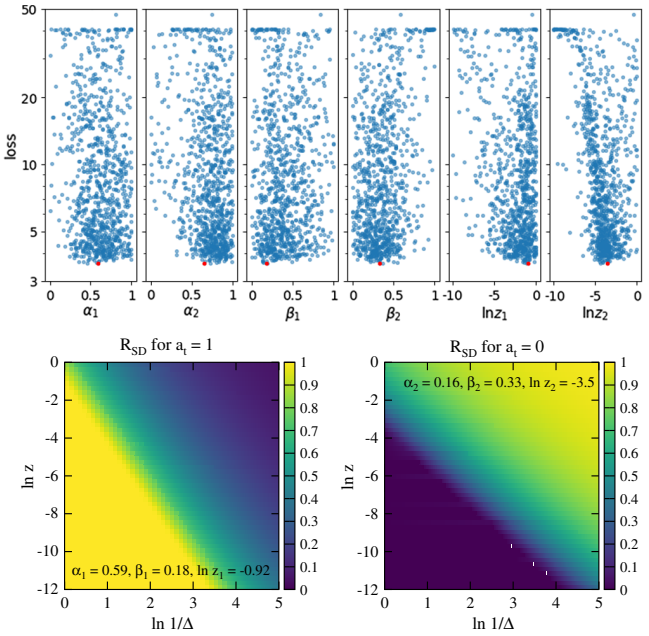


FIG. 3. Top: Loss as a function of the reward parameters, with the optimal parameters shown in red. Bottom: Value of the two terms in the soft-drop reward function given in Eq. (6) as a function of Δ and z.

Here N_{SD} is a normalization factor determining the weight given to the second component of the reward.

C. RL implementation and multilevel training

For the applications in this article, we have implemented a DQN agent that contains a groomer module, which is defined by the underlying NN model and the test policy used by the agent. The groomer can be extracted after the model has been trained, using a greedy policy to select the best action based on the Q values predicted by the NN. This allows for straightforward application of the resulting grooming strategy on new samples.

The training sample consists of 500 000 signal and background jets simulated using PYTHIA 8.223 [45]. We will construct two separate models by considering two signal samples, one with boosted W jets and one with boosted top jets, while the background always consists of QCD jets. We use the WW and $t\bar{t}$ processes, with hadronically decaying W and top, to create the signal samples, and the dijet process for the background. Jets are clustered using the anti-k_t algorithm [5,46] with radius $R = 1.0$ and are required to pass a selection cut, with transverse momentum $p_t > 500$ GeV and rapidity $|y| < 2.5$. The grooming environment is initialized by reading in the training data and creating an event array containing the corresponding jet trees.

To train the RL agent, we use a multilevel approach taking into account both signal and background samples. At the beginning of each episode, we select either a signal jet or a background jet, with probability $1 - p_{\mathrm{bkg}}$. For signal jets, the reward function uses a reference mass set to the W-boson mass, $m_{\mathrm{target}} = m_W$, or to the top mass, $m_{\mathrm{target}} = m_t$, depending on the choice of sample. In the case of the background the mass reward function in Eq. (7) is changed to

$$R_M^{\mathrm{bkg}}(m) = \frac{m}{\Gamma_{\mathrm{bkg}}} \exp\left(-\frac{m}{\Gamma_{\mathrm{bkg}}}\right). \tag{8}$$

The width parameters Γ and Γ_{bkg} are also set to different values for signal and background reward functions and are determined through a hyperparameter scan.

We found that while this multilevel training only marginally improves the performance, it noticeably reduces the variability of the model.

D. Determining the RL agent

The DQN agent uses an Adam optimizer [47], and the training is performed with a Boltzmann policy, which chooses an action according to weighted probabilities, with the current best action being the likeliest.

Let us now determine the remaining parameters of the DQN agent. To this end, we perform two independent scans, for the grooming environment and for the network architecture.

The grooming environment has several options, which are shown in Fig. 4. Here the distribution of loss values for discrete options are displayed using violin plots, showing both the probability density of the loss values as well as its quartiles. The first plot is the dimensionality of the state observed at each step, which can be a subset of the tuple given in Eq. (2). We can observe that as the dimension of the input state is increased, the NN is able to leverage this additional information, leading to a decrease of the loss function. The scan over the normalization parameters of the reward functions shows that it is preferable to use a small width Γ for the signal, with a large value Γ_{bkg} for the background, as well as a small value for the $1/N_{\mathrm{SD}}$ factor. One can also see that the multilevel training described in Sec. III C leads to a distribution of loss values concentrated at smaller values. We have also allowed for several functional forms of the signal mass reward function, although for our final model we will use a Cauchy distribution.

The parameters of the network architecture are shown in Fig. 5, with the first plot showing the mass window containing 60% of the signal distribution, with the median of that interval shown in blue. The scatter plot of the learning rate used for the Adam optimizer shows that a value slightly above 10^{-4} yields the best result. The scan shows a preference for a dense network with a large number of units and layers as well as a dropout layer as the

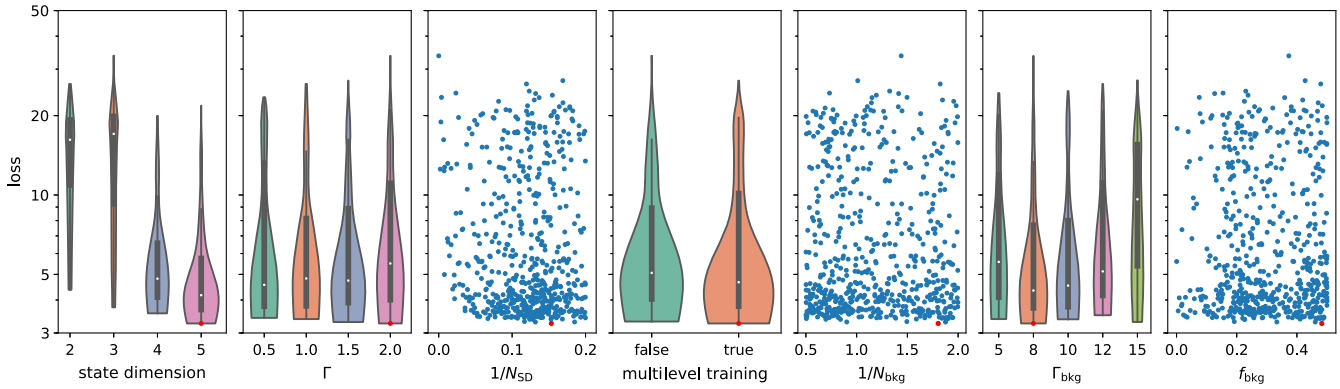


FIG. 4. Distribution of the loss value for different grooming parameters. The best performing model is indicated in red.

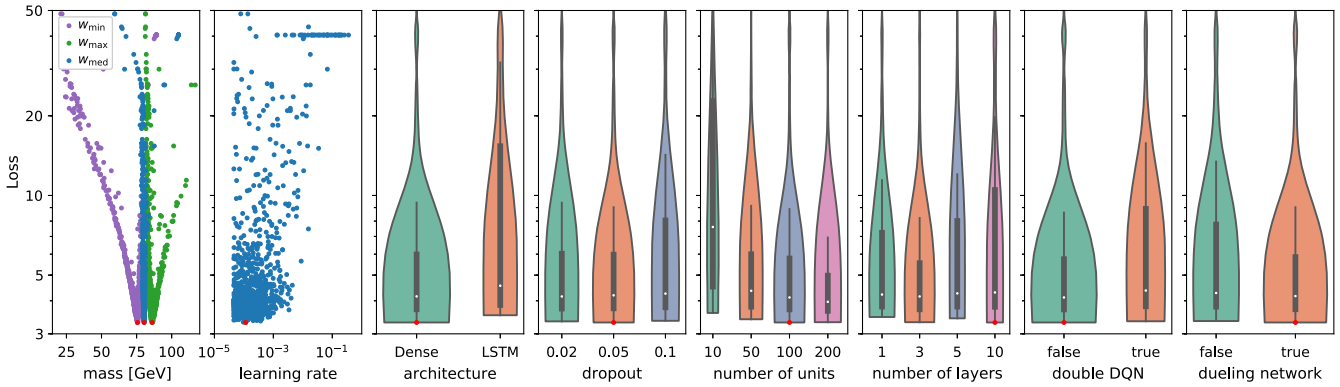


FIG. 5. Distribution of the loss value for different architecture configurations. The best performing model is indicated in red.

architecture of the NN. Finally, we see that using dueling networks [48] leads to a small improvement of the model, while double Q learning [49] does not.

E. Optimal GROOMRL model

The final GROOMRL model is trained using the full training sample with 500 000 signal or background jets for 1 000 000 epochs. The overall training time requires 4 hr of training using a single NVIDIA GTX 1080 Ti GPU with 12 GB of memory which includes all the training jet trees and the DQN parameters.

The parameters of the best GROOMRL model obtained following the strategy presented in the previous sections is listed in Table I. Here two values are given for the m_{target} parameter, which are used to train on either a sample consisting of W bosons or of top quarks. The resulting models are labeled GROOMRL-W and GROOMRL-Top, respectively.

In Fig. 6 we show the reward value during the training of the GROOMRL for W bosons and top quarks, after applying the locally weighted smoothing algorithm on the original curve. We observe an improvement of the reward function during the first 300 000 training epochs, with the reward becoming relatively stable after that point.

TABLE I. Final parameters for GROOMRL, with the two values of m_{target} corresponding to the W and top mass.

Parameters	Value
m_{target}	80.385 or 173.2 GeV
s_t dimension	5
Reward	Cauchy
Γ	2 GeV
$(\alpha_1, \beta_1, \ln z_1)$	$(0.59, 0.18, -0.92)$
$(\alpha_2, \beta_2, \ln z_2)$	$(0.65, 0.33, -3.53)$
$1/N_{\text{SD}}$	0.15
Multilevel training	Yes
Γ_{bkg}	8 GeV
$1/N_{\text{bkg}}$	1.8 or 1.0
p_{bkg}	0.48 or 0.2
Learning rate	10^{-4}
Dueling NN	Yes
Double DQN	No
Policy	Boltzmann
$N_{\text{epochs}}^{\text{max}}$	500 000
Architecture	Dense
Dropout	0.05
Layers	10
Nodes	100
Optimizer	Adam

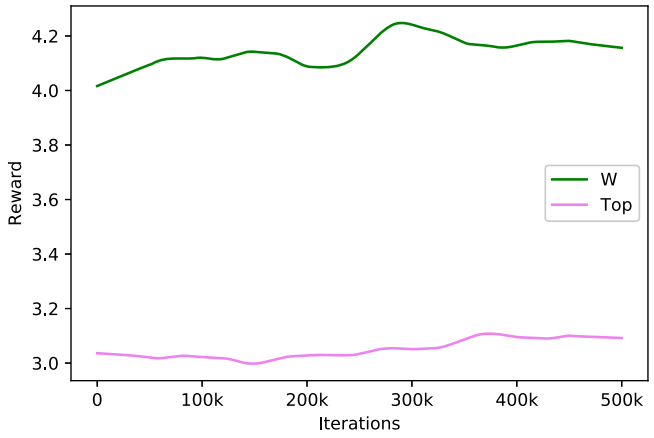


FIG. 6. Reward evolution during training of the GROOMRL on W and top data. A locally weighted smoothing is applied to the original curves.

F. Alternative approaches

In this section, we have introduced a novel implementation of RL to tackle the problem of tree pruning. A number of alternative methods could be studied to approach this problem, most notably Monte Carlo tree search (MCTS) algorithms [50,51] and binary classifiers. The heuristic search methods from MCTS explore the tree through random sampling, taking random actions to progress through the tree. Once an end point is reached, the result is used to weight the nodes and improve future decisions.

More recently, a NN-based MCTSnet implementation was proposed [52], which introduces a framework to learn how to search the tree, integrating simulation-based planning into a NN.

These techniques might provide an interesting basis to construct an efficient groomer. However due to the wide variability of the trees considered in our case study, where each new episode starts from a unique tree, this would require a substantial modification of the algorithm.

Alternatively, one could use a contextual bandit solver [53,54] to train a jet grooming policy. We would expect this method to yield similar results; however, this method does not allow for the modification of the future nodes by the current grooming decision and is not as easily extendable as our current framework.

Finally, one could attempt to build a jet grooming algorithm from a binary classifier, which uses an input state to determine which action to take next. The main drawback of this method is that one cannot straightforwardly impose as a loss function the mass resolution of the tree, as this depends on previous states of the current episode. As such, the problem we consider is particularly well adapted to a RL approach.

We leave a more thorough study of the application of these alternative tools to jet grooming for future work.

TABLE II. Size of the window containing 60% of the W mass spectrum and median value on that interval.

	$w_{\max} - w_{\min}$ [GeV]	w_{med} [GeV]
Plain	44.65	104.64
GROOMRL-W	10.70	80.09
GROOMRL-Top	13.88	80.46
RSD	16.96	80.46

IV. JET MASS SPECTRUM

Let us now apply the GROOMRL models defined in Sec. III E to new data samples. We consider three test sets of 50 000 elements each: one with QCD jets, one with W initiated jets and one with top jets. The size of the window containing 60% of the mass spectrum of the W sample, as well as the corresponding median value, is given in Table II for each different grooming strategy. As a benchmark, we compare to the RSD algorithm, using parameters $z_{\text{cut}} = 0.05$, $\beta = 1$ and $R_0 = 1$. One can notice a sizable reduction of the window size after grooming with the machine learning based algorithms, while all groomers are able to reconstruct the peak location to a value very close to the W mass.

The distribution of the jet mass after grooming for each of these samples is shown in Figs. 7 and 8. Each curve gives the differential cross section $d\sigma/dm_j$ normalized by the total cross section. Figure 7 shows results for the grooming algorithm trained on a W sample, while the results of the algorithm trained on top data are given in Fig. 8. As references, the ungroomed (or plain) jet mass and the jet mass after RSD grooming are also given, in blue and orange, respectively. As expected, one can observe that for the ungroomed case the resolution is very poor, with the QCD jets having large masses due to wide-angle radiation, while the W and top mass peaks are heavily distorted. In contrast, after applying RSD or GROOMRL, the jet mass is reconstructed much more accurately. One interesting feature of GROOMRL is that it is able to lower the jet mass for quark and gluon jets, further reducing the background contamination in windows close to a heavy particle mass.

For the W case, shown in Figs. 7(b) and 8(b), there is a sharp peak around the W mass m_W, with the GROOMRL method providing slightly better resolution. It is also particularly noteworthy that both the GROOMRL-W and the GROOMRL-Top algorithms have similar performance, despite the latter one having been trained on a completely different dataset. This demonstrates that the tools derived from our framework are robust and can be applied to datasets beyond their training range with good results.

In top jets, displayed in Figs. 7(c) and 8(c), the enhancements are even more noticeable. Here again, the performance of both algorithms is similar, despite the fact that the training of GROOMRL-W did not involve any top-related data.

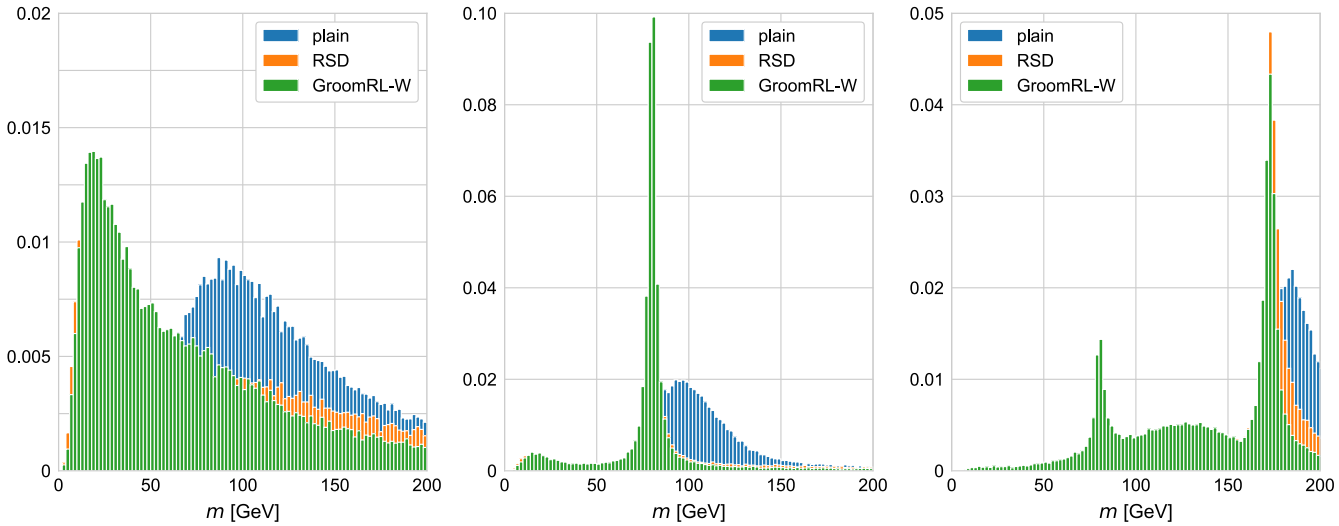


FIG. 7. Groomed jet mass spectrum for (a) QCD jets, (b) W jets, (c) top jets. The GroomRL-W curve is obtained from training on W data.

Finally, in Fig. 9, we show the primary Lund jet plane density as defined in [22] after grooming with GroomRL-W and GroomRL-Top, averaged over 50 000 jets. This gives a useful visualization of radiation patterns within a jet, providing a physical interpretation of the grooming behavior. The primary Lund jet plane is defined through the $(\ln 1/\Delta_{ab}, \ln k_t)$ coordinates of each of the states of the "primary" declustering sequence, i.e., traversing the jet tree by successively following the hardest branch $\mathcal{T}^{(i)} \to a$. The upper boundary of the triangle is due to the kinematic limit of emissions. In contrast, the lower edge corresponds to radiation that gets removed by the grooming algorithm, so that only sufficiently energetic or collinear partons remain in the groomed jet.

An interesting feature of Fig. 9 is that one can observe that despite producing similar jet mass spectra, the GroomRL-W and GroomRL-Top algorithms differ somewhat, with the former retaining more radiation at wide angles than the latter.

A. Robustness to nonperturbative effects

Let us now consider the impact of nonperturbative effects such as hadronization and underlying event on groomed jets. A key feature of grooming algorithms such

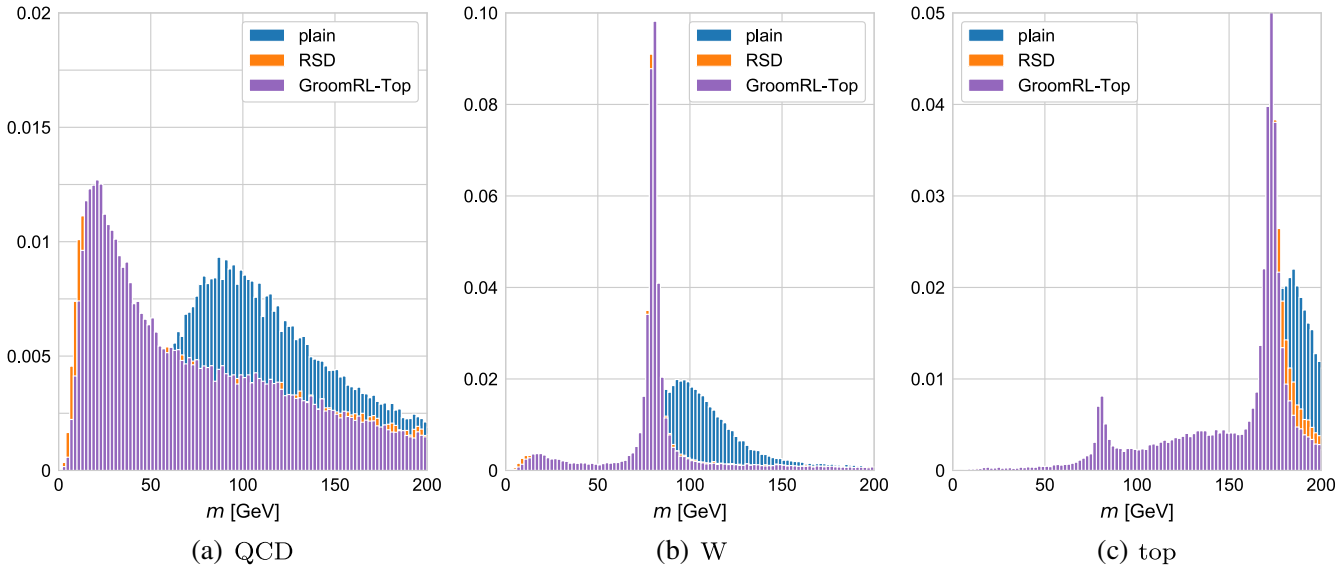
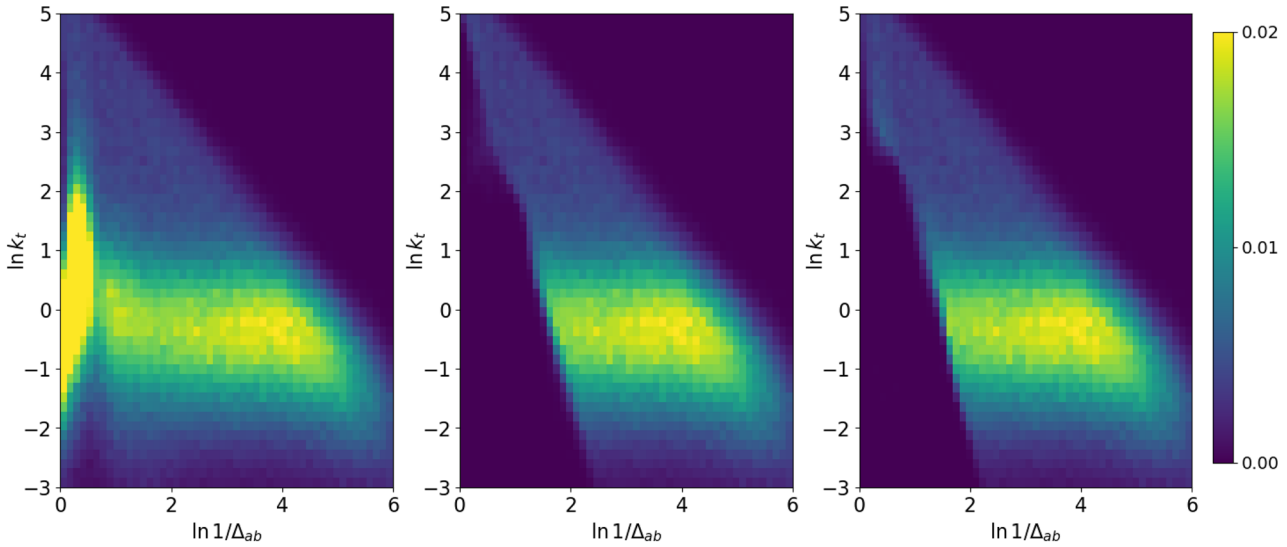
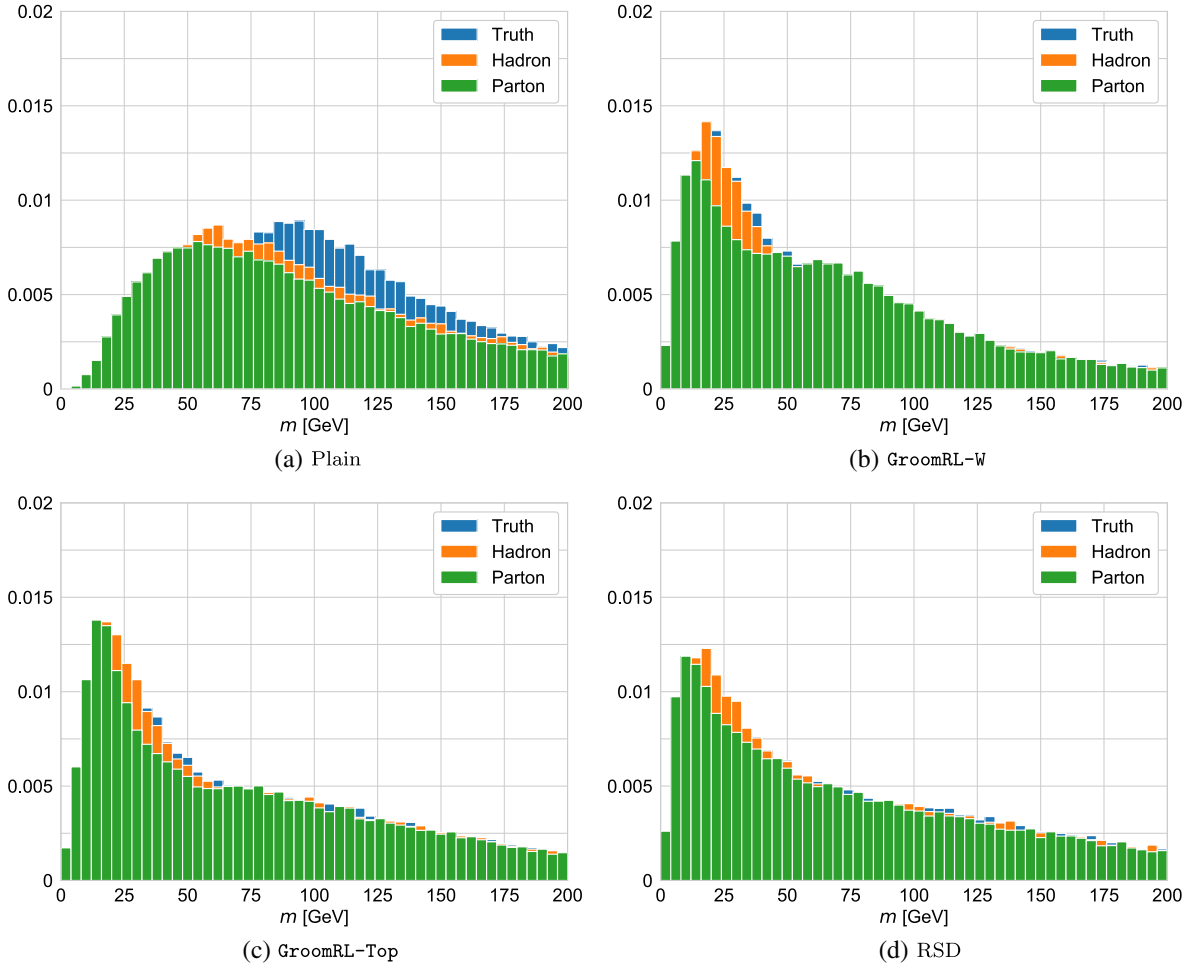


FIG. 8. Groomed jet mass spectrum for (a) QCD jets, (b) W jets, and (c) top jets. The GroomRL-Top curve is obtained from training on top data.



(a) Plain (b) GroomRL-W (c) GroomRL-Top

FIG. 9. Primary Lund jet plane density for QCD jets before (a) and after grooming with GroomRL trained on W (b) or top (c) samples.



(a) Plain (b) GroomRL-W

(c) GroomRL-Top (d) RSD

FIG. 10. Jet mass spectrum for QCD jets at parton level (green) and hadron level (orange) and including an underlying event (blue). Distributions are shown for ungroomed jets (a), as well as after grooming with GroomRL trained on W data (b), on top data (c) or with RSD (d).

as mass drop tagger and soft drop is that they reduce the sensitivity of observables to nonperturbative effects, allowing for precise comparisons between theoretical predictions and experimental measurements.

To study the robustness of GroomRL to these contributions, we consider three different QCD jet samples generated through Pythia's dijet process. The first one, which we denote as "truth level" and used already in the previous sections, includes all nonperturbative effects. A "hadron-level" sample is obtained by removing multiple parton interactions from the simulation, and finally a "parton-level" sample is generated by further turning off the hadronization step in Pythia.

The jet mass spectrum for each sample is shown in Fig. 10, with results for ungroomed jets as well as after grooming with GroomRL-W, GroomRL-Top and RSD. One can see immediately that the ungroomed jet mass spectrum is strongly affected by nonperturbative effects, while groomed jets become much more robust to these contributions. For masses $m > 50$ GeV, both GroomRL models become very robust, showing a resilience to hadronization and underlying event similar to that of RSD. In the low mass range, GroomRL remains robust to multiple parton interactions but starts to show some dependence on hadronization effects.

We note that no parton-level or hadron-level data were used in the training, such that one would not *a priori* expect the derived algorithm to be particularly resilient to these effects. Although GroomRL already performs surprisingly well, one could easily further improve the robustness of the model by including some of these data with a suitable modification of the reward function in the training of the DQN agent.

V. CONCLUSIONS

We have shown a promising application of RL to the issue of jet grooming. Using a carefully designed reward function, we have constructed a groomer from a dense NN trained with a DQN agent.

This grooming algorithm was then applied to a range of data samples, showing excellent results for the mass resolution of boosted heavy particles. In particular, while the training of the NN is performed on samples consisting of W (or top) jets, the groomer yields noticeable gains in the top (or W) case as well, on data outside of the training range.

The improvements in resolution and background reduction compared to alternative state-of-the-art methods provide an encouraging demonstration of the relevance of machine learning for jet grooming. In particular, we showed that it is possible for a RL agent to extract the underlying physics of jet grooming and distill this knowledge into an efficient algorithm.

Due to its simplicity, the model we developed also retains most of the calculability of other existing methods such as soft drop. Accurate numerical computations of groomed jet observables are therefore achievable, allowing for the possibility of direct comparisons with data. Furthermore, given an appropriate sample, one could also attempt to train the grooming strategy on real data, bypassing some of the limitations due to the use of parton shower programs.

The GroomRL framework, is generic and can easily be extended to higher-dimensional inputs, for example to consider multiple emissions per step or additional kinematic information. While the method presented in this article was applied to a specific problem in particle physics, we expect that with a suitable choice of reward function, this framework is in principle also applicable to a range of problems where a tree requires pruning.

The framework and data used in this paper are available as open-source and published material in [55–57].

ACKNOWLEDGMENTS

We are grateful to Jia-Jie Zhu and Gavin Salam for comments on the manuscript and to Jesse Thaler for useful discussions. We also acknowledge the NVIDIA Corporation for the donation of a Titan Xp GPU used for this research. F. A. D. is supported by the Science and Technology Facilities Council (STFC) under Grant No. ST/P000770/1. S. C. is supported by the European Research Council under the European Union's Horizon 2020 research and innovation Program (Grant Agreement No. 740006).

[1] G. F. Sterman and S. Weinberg, Phys. Rev. Lett. **39**, 1436 (1977).
[2] G. P. Salam, Eur. Phys. J. C **67**, 637 (2010).
[3] S. D. Ellis and D. E. Soper, Phys. Rev. D **48**, 3160 (1993).
[4] Y. L. Dokshitzer, G. D. Leder, S. Moretti, and B. R. Webber, J. High Energy Phys. 08 (1997) 001.
[5] M. Cacciari, G. P. Salam, and G. Soyez, J. High Energy Phys. 04 (2008) 063.
[6] J. Cogan, M. Kagan, E. Strauss, and A. Schwartzman, J. High Energy Phys. 02 (2015) 118.
[7] L. G. Almeida, M. Backovi, M. Cliche, S. J. Lee, and M. Perelstein, J. High Energy Phys. 07 (2015) 086.

[8] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman, and A. Schwartzman, J. High Energy Phys. 07 (2016) 069.
[9] P. Baldi, K. Bauer, C. Eng, P. Sadowski, and D. Whiteson, Phys. Rev. D **93**, 094034 (2016).
[10] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban, and D. Whiteson, Phys. Rev. D **94**, 112002 (2016).
[11] G. Kasieczka, T. Plehn, M. Russell, and T. Schell, J. High Energy Phys. 05 (2017) 006.
[12] G. Louppe, K. Cho, C. Becot, and K. Cranmer, J. High Energy Phys. 01 (2019) 057.
[13] L. de Oliveira, M. Paganini, and B. Nachman, Comput. Softw. Big Sci. **1**, 4 (2017).
[14] C. Shimmin, P. Sadowski, P. Baldi, E. Weik, D. Whiteson, E. Goul, and A. Sgaard, Phys. Rev. D **96**, 074034 (2017).
[15] K. Datta and A. Larkoski, J. High Energy Phys. 06 (2017) 073.
[16] A. J. Larkoski, I. Moult, and B. Nachman, arXiv:1709.04464.
[17] J. Pearkes, W. Fedorko, A. Lister, and C. Gay, arXiv:1704.02124.
[18] A. Butter, G. Kasieczka, T. Plehn, and M. Russell, SciPost Phys. **5**, 028 (2018).
[19] P. T. Komiske, E. M. Metodiev, B. Nachman, and M. D. Schwartz, J. High Energy Phys. 12 (2017) 051.
[20] A. Andreassen, I. Feige, C. Frye, and M. D. Schwartz, Eur. Phys. J. C **79**, 102 (2019).
[21] E. M. Metodiev and J. Thaler, Phys. Rev. Lett. **120**, 241602 (2018).
[22] F. A. Dreyer, G. P. Salam, and G. Soyez, J. High Energy Phys. 12 (2018) 064.
[23] P. T. Komiske, E. M. Metodiev, and J. Thaler, J. High Energy Phys. 01 (2019) 121.
[24] J. Arjona Martnez, O. Cerri, M. Pierini, M. Spiropulu, and J.-R. Vlimant, arXiv:1810.07988.
[25] K. Datta, A. Larkoski, and B. Nachman, arXiv:1902.07180.
[26] A. Butter *et al.*, arXiv:1902.09914.
[27] H. Qu and L. Gouskos, arXiv:1902.08570.
[28] J. M. Butterworth, A. R. Davison, M. Rubin, and G. P. Salam, Phys. Rev. Lett. **100**, 242001 (2008).
[29] S. D. Ellis, C. K. Vermilion, and J. R. Walsh, Phys. Rev. D **81**, 094023 (2010).
[30] D. Krohn, J. Thaler, and L.-T. Wang, J. High Energy Phys. 02 (2010) 084.
[31] M. Dasgupta, A. Fregoso, S. Marzani, and G. P. Salam, J. High Energy Phys. 09 (2013) 029.
[32] A. J. Larkoski, S. Marzani, G. Soyez, and J. Thaler, J. High Energy Phys. 05 (2014) 146.
[33] F. A. Dreyer, L. Necib, G. Soyez, and J. Thaler, J. High Energy Phys. 06 (2018) 093.
[34] M. Aaboud *et al.* (ATLAS Collaboration), Phys. Rev. Lett. **121**, 092001 (2018).
[35] A. M. Sirunyan *et al.* (CMS Collaboration), J. High Energy Phys. 11 (2018) 113.
[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, arXiv:1312.5602.
[37] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, Nature (London) **550**, 354 (2017).
[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, Nature (London) **518**, 529 (2015).
[39] M. Wobisch and T. Wengler, in *Monte Carlo Generators for HERA Physics. Proceedings, Workshop, Hamburg, Germany, 1998-1999* (1998), pp. 270–279, https://arxiv.org/abs/hep-ph/9907280.
[40] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, arXiv:1606.01540.
[41] M. Plappert, KERAS-RL, https://github.com/keras-rl/keras-rl (2016).
[42] F. Chollet *et al.*, KERAS, https://keras.io (2015).
[43] M. Abadi *et al.*, TENSORFLOW: Large-Scale Machine Learning on Heterogeneous Systems (2015), software available from https://www.tensorflow.org/.
[44] J. Bergstra and D. Yamins, J. Mach. Learn. Res. **13**, 281 (2012).
[45] T. Sjstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, Comput. Phys. Commun. **191**, 159 (2015).
[46] M. Cacciari, G. P. Salam, and G. Soyez, Eur. Phys. J. C **72**, 1896 (2012).
[47] D. P. Kingma and J. Ba, arXiv:1412.6980.
[48] Z. Wang, N. de Freitas, and M. Lanctot, arXiv:1511.06581.
[49] H. van Hasselt, A. Guez, and D. Silver, arXiv:1509.06461.
[50] R. Coulom, in *Proceedings of the 5th International Conference on Computers and Games* (Springer-Verlag, Berlin, 2007), pp. 72–83.
[51] L. Kocsis and C. Szepesvári, in *Machine Learning: ECML 2006* (Springer, New York, 2006), pp. 282–293.
[52] A. Guez, T. Weber, I. Antonoglou, K. Simonyan, O. Vinyals, D. Wierstra, R. Munos, and D. Silver, in *International Conference on Machine Learning* (2018), pp. 1822–1831.
[53] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, SIAM J. Comput. **32**, 48 (2002).
[54] A. Agarwal, D. J. Hsu, S. Kale, J. Langford, L. Li, and R. E. Schapire, arXiv:1402.0555.
[55] S. Carrazza and F. A. Dreyer, JetsGame/data v1.0.0 (2019), this repository is git-lfs, https://doi.org/10.5281/zenodo.2602514.
[56] S. Carrazza and F. A. Dreyer, JetsGame/GroomRL v1.0.0 (2019), https://doi.org/10.5281/zenodo.3265836.
[57] S. Carrazza and F. A. Dreyer, JetsGame/libGroomRL v1.0.0 (2019), https://doi.org/10.5281/zenodo.3265836.