Lorentz Equivariant Neural Networks for Jet Classification

Farouk Mokhtar[1] and Aditya Mishra[2]
[1]*Department of Physics, University of California at San Diego, La Jolla, CA 92093*
[2]*Department of Mechanical Engineering, University of California San Diego, La Jolla, CA 92093*

Neural networks which are Lorentz group equivariant are extremely desirable for high energy physics data. In this work, we attempt to build such a model and train it to the task of jet-classification. The model reaches state-of-the-art benchmark performance, in addition to passing a series of physics equivariance tests; suggesting a new realm of experimentation with physics equivariant models.

I. INTRODUCTION

The success of machine learning (ML) as a data analysis tool cannot be overstated. ML techniques, such as deep neural networks (DNNs), have been of extensive use for recent years due to their wide range of applicability in different fields. The reason being that many tasks can be casted in the form of an optimization problem.

Perhaps the earliest work on DNNs can be traced back to the 1970s when the first working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Lapa [1]. A natural question to ask is why is it that DNNs became a popular tool only in the last decade or so. The answer to this question is two-fold. For one, the training of these DNNs require large enough datasets, for better generalization capability, which were not always available back in the days. The other reason why DNNs became popular is the better computational resources available through the development of better graphical processing units (GPUs) by big companies like Nvidia.

Meanwhile, the amounts of data gathered by high energy physics (HEP) detectors, specifically the Large Hadron Collider (LHC) at CERN, is huge. The LHC experiments produce about 90 petabytes of data per year [2]. These amounts of data necessarily motivate the implementation of ML algorithms such as DNNs to perform data analysis for a series of HEP tasks, like: particle reconstruction, charged particle tracking, event classification, and others. In a nutshell, any LHC task is probably better off casted as a ML problem due to the large amounts of data gathered.

Moreover, there are different types of DNNs, each was first proposed to better suit different tasks or datasets. A natural question to ask is, can we build a ML model that is best suited for HEP data. To answer this question, let's first look at a particular class of DNNs that have built-in equivariance to certain transformations. For example, convolutional neural networks (CNNs) are also known as shift-invariant networks because they perform operations on the input that involve parameter sharing which in turn keeps the network robust to translations in the input. This is why CNNs have been regarded as the industry standard in computer vision. Another example is graph neural networks (GNNs) which are permutation invariant because they perform operations that are immune to node ordering in the inputs. These are two examples of what we call *equivariant networks*, designed to respect the symmetries of their respective data. This motivates the exploration of other equivariant networks which might, instead of being translation invariant, be invariant with respect to physics symmetries, and hence better suited for physics data. If such network architectures are developed, they will be more easily interpretable within the physics community, and perhaps more successful, as they are better suited for the data.

The *Lorentz group Equivariant Network* (LGN) is a recently proposed architecture of a DNN which is equivariant with respect to Lorentz group symmetries. This means that the LGN is immune to Lorentz boosts (or rotations) of the inputs. From a physicists point of view, this seems like a natural imposition on a ML model. We want our model (in our case a DNN classifier) to base it's result (in our case a label that classifies a physics phenomenon) independently from the frame where the phenomenon was observed.

The LGN model was proposed in 2020, and is the earliest major attempt to tackle ML-type problems in HEP using a Fourier space equivariant architecture that preserves Lorentz group symmetries [3]. It is instructive to study the structure of this LGN model and understand how it works, because many problems in HEP are already addressed using DNNs and may be further optimized if such networks were Lorentz equivariant.

In this project, we attempt to replicate the results of this paper [3] by building an LGN model and training it to the task of jet-classification on HEP data. We can then evaluate the performance of the model with respect to (a) ML metrics and (b) physics equivariance.

The structure of the report is as follows: in section II and III we give a background introduction on HEP and DNNs respectively, in section IV we explore the architecture of the LGN model, in section V we discuss the training of the model in terms of computational cots and efficiency, and in section VI we evaluate the model with respect to ML metrics and physics equivariance tests.

II. HEP PRELIMINARIES

HEP, also known as Particle Physics, is a branch of physics that studies the nature of particles that constitute matter and radiation. It is a fundamental science in the sense that it tries to answer questions regarding the fundamental building blocks of the universe and their interactions. At the moment, the dominant theory explaining these fundamental particles and fields, along with their dynamics, is called the Standard Model (SM).

Figure 1 presents a picture of all known elementary particles, alongside the Higgs boson, which was recently discovered at the LHC in 2012. The Higgs boson discovery represents a major stepping-stone to what the LHC detector is capable of discovering. Similar ongoing attempts are happening at the LHC, trying to uncover new particle discoveries and finding answers to SM limitations, such as dark matter or the matter-antimatter asymmetry problem.

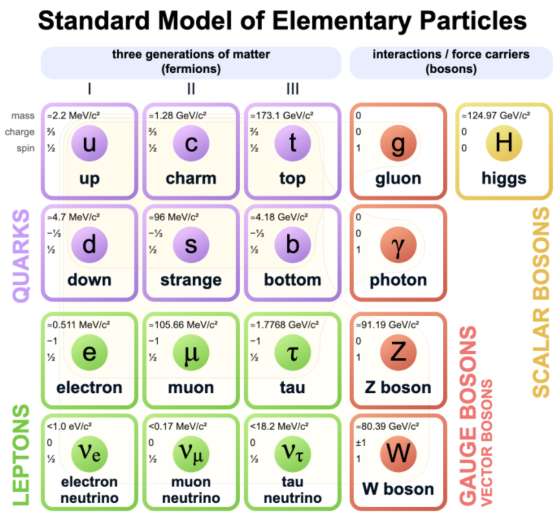
Standard Model of Elementary Particles
three generations of matter (fermions)
interactions / force carriers (bosons)
I
II
III
mass
charge
spin
≈2.2 MeV/c² ⅔ ½ u up
≈1.28 GeV/c² ⅔ ½ c charm
≈173.1 GeV/c² ⅔ ½ t top
0 0 1 g gluon
≈124.97 GeV/c² 0 0 H higgs
≈4.7 MeV/c² −⅓ ½ d down
≈96 MeV/c² −⅓ ½ s strange
≈4.18 GeV/c² −⅓ ½ b bottom
0 0 1 γ photon
≈0.511 MeV/c² −1 ½ e electron
≈105.66 MeV/c² −1 ½ μ muon
≈1.7768 GeV/c² −1 ½ τ tau
≈91.19 GeV/c² 0 1 Z Z boson
<1.0 eV/c² 0 ½ νe electron neutrino
<0.17 MeV/c² 0 ½ νμ muon neutrino
<18.2 MeV/c² 0 ½ ντ tau neutrino
≈80.39 GeV/c² ±1 1 W W boson
QUARKS
LEPTONS
GAUGE BOSONS
VECTOR BOSONS
SCALAR BOSONS

FIG. 1: A picture of all known elementary particles according to the Standard Model.

The LHC [4] [5] is a 27km ring of superconducting 7.7T dipole magnets designed to collide protons at center-of-mass energy of $\sqrt{s}$ = 14 TeV every 25 ns. Protons are accelerated to velocities near the speed of light, which allows us to measure things like: production rates, masses lifetimes, and decay rates. The LHC is the largest and most advanced particle collider in the world, and arguably, the largest data generation machine ever built. Proton collisions occur at four different interaction points in the LHC ring where detectors are placed to record the collision data for physics analyses. The largest detectors, ATLAS [6] and CMS [7], are so-called multipurpose detectors designed to measure and search for a wide range of physical processes.

At high enough energies, protons clash and create sprays of particles traversing in different directions. These particles, and their trajectories, are then reconstructed through sequences of data analysis techniques, which might involve ML algorithms. A visualization of the CERN accelerator complex is shown in Figure 2. In the next part, we will dive a little more into the ATLAS detector, simply because the data we are analyzing for this project was modelled based on the ATLAS detector response, and published by an ATLAS team.

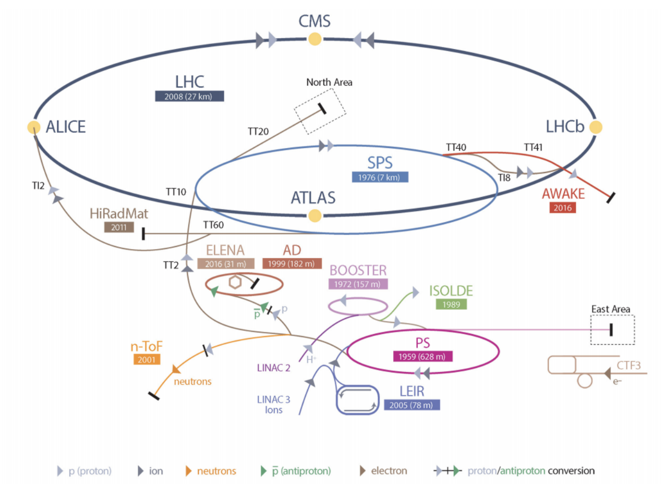

FIG. 2: The CERN accelerator complex. The LHC (dark blue) is the last in a chain of several smaller particle accelerators. Yellow dots represent the four main detector experiments: ATLAS, ALICE, CMS and LHCb. Taken from [8].

The ATLAS detector

The ATLAS detector is largest of the four main LHC experiments, and is the apparatus on which this project is based on. It is shaped like cylinder around the LHC beam axis and is roughly 44 meters long with a diameter of 25 meters. The weight of the machine is approximately 7000 tons. An illustration showing the dimensions and some subsystems of the ATLAS detector is shown in Figure 3.

The detector is a complicated instrument with several layers designed to take advantage of the high energies available at the LHC and observe physical phenomena occurring in this energy regime. One such observation is probably one of the greatest triumphs in modern physics, namely the discovery of the Higgs boson in 2012 [9].

The coordinate system in ATLAS is defined such that the nominal interaction point (IP) in the center of the detector is at the origin, and the beam direction is parallel to the z−axis. An illustration is shown in Figure 4. The x−y plane is transverse to the beam direction, where the positive x−direction points inward to the center of the LHC ring, and positive y upwards. The coordinate sys-

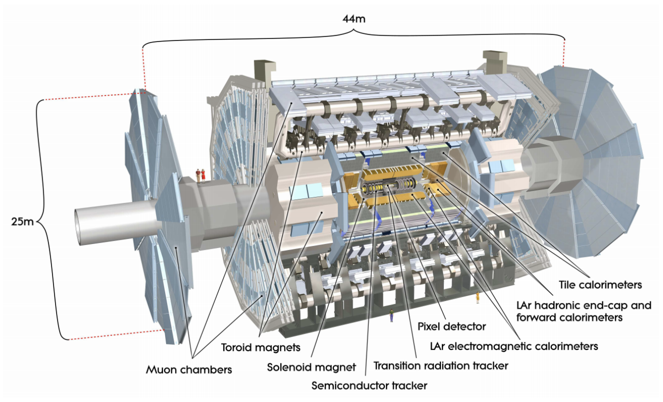
44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Muon chambers
Solenoid magnet
Transition radiation tracker
Semiconductor tracker

FIG. 3: An illustration of the ATLAS detector. Taken from [6]

tem can also be expressed in polar coordinates, where the azimuthal angle ϕ goes around the beam axis, and the polar angle θ is the angle from the IP to the x−y plane [10].

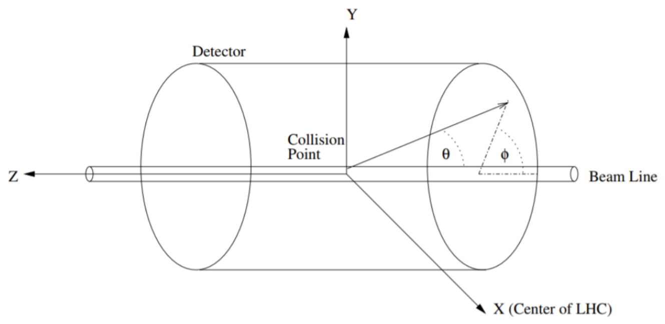

FIG. 4: An illustration of the coordinate system in ATLAS. Taken from [10].

Instead of θ however, polar angle measurements are more often made using pseudo-rapidity, which is defined as: $-ln(tan(\frac{\theta}{2}))$. Thus, it is often the case that we measure position coordinates in terms of (η, ϕ) coordinates.

The ATLAS detector itself is comprised of several subdetectors, each specializing in measuring certain kinematic properties of these particles. By combining the measurements of the subsystems it is possible to reconstruct the physics that occurred in the detector.

What is a jet?

In this project we're implementing a particular type of DNN called an LGN to the task of jet-classification. A jet is a spray of particles going in the same direction. We detect the particles, their direction (momenta), and reconstruct the "jet" via some sort of "cone" algorithm that sums the momentum in that cone. Simply put, when we say we input a jet to the LGN, we mean that we input the set of 4-momenta of all particles in the jet.

Figure 5 gives a visual representation of two jets (two orange cones). Each jet is simply a collection of particle tracks all originating from a common mother particle. For example, a top quark jet, or simply a top jet, refers to the spray of particles that originate when a top quark decays. These product particles will also decay and hence a chain like sequence of created particles, all of which have related kinematics because they all come from the same mother particle, a top quark.

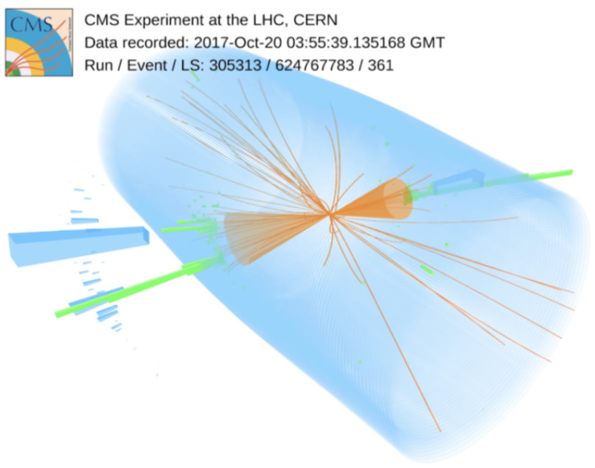
CMS Experiment at the LHC, CERN
Data recorded: 2017-Oct-20 03:55:39.135168 GMT
Run / Event / LS: 305313 / 624767783 / 361

FIG. 5: An illustration of two jets, each given by yellow cone which combines tracks/particles together.

III. PHYSICS EQUIVARIANT MODELS

In section I, we have mentioned examples of equivariant networks like CNNs and GNNs. We discussed how each of these models are invariant with respect to a set of transformations, where CNNs were robust to T(N) (translations in N dimensions) while GNNs were robust to S_N (permutations of N objects). It is natural to ask whether we can come up with models which are robust to some other set of transformations that depends on the domain of application. In fact, there are some theoretical attempts to build such models. For a fruitful discussion on some artificial neural network architectures designed to be equivariant to certain symmetry group transformations like E(2) and E(3); see this minireview [11].

The LGN, the topic of this work, is not the earliest attempt to develop a Lorenz group equivariant network. There is a work published in 2018 which builds a so called Lorentz Layer [12]; suggesting that such a network can also achieve benchmark performance.

The main reluctance as to why physicists haven't been very active in implementing symmetry preserving network architectures is that, not only is it difficult to develop, but also because no one wanted to interfere with how the networks work. For some time, networks which are not equivariant have been achieving state-of-the-art performances, and as with DNNs in general, we often think of them as a black box. You don't necessarily know why they work; you just give the network the full freedom to learn the best way it could. Thus, it is not really clear whether purposely enforcing analytic constraints on DNNs would work better. There will always be this trade-off between giving the network the freedom to learn while imposing restrictions. The more you restrict, the potentially worse it will learn. Because at the end of the day, if you build a very complicated DNN, it should be able, in principle, to find the best way to solve this loss optimization problem. And any restrictions you impose on the network can only make it worse. But the caveat here is that, we are assuming this DNN is very complicated so that it can basically learn anything, even the restrictions you are imposing to help it learn better, but in practice, this DNN will not be the most complicated model there is.

IV. LGN MODEL ARCHITECTURE

Before jumping to discuss the model architecture, it is probably better to first clearly explain the task we are expecting the model to tackle. This project is concerned with the task of jet-classification, also called *jet tagging*, which refers to the challenge of classifying jets as coming from top quarks (which is our signal, denoted by a label 1) or QCD (which is our background, denoted by a label 0). In ML terms, we are building a DNN classifier for the task of binary classification. But not only that, we want our classifier to be equivariant with respect to Lorentz symmetries. This in turn restricts the possible operations our model can make on the input jets during the forward pass. Technically, any of the model operations throughout the forward pass should be invariant to Lorentz boosts (and rotations) of the input jets.

Network Architecture overview

The components of the LGN architecture as shown in Figure 6 are as the following:

1. **Inputs:** The inputs to the network consist of the 4-momenta vector (E, p_x, p_y, p_z) of N_{obj} number of particles in the jet. Here E denotes the energy and p_j denotes the momentum of the particle in j direction. The input also consists of Lorentz scalars such as spins, charges and labels. In this case only two scalars are considered with the momenta vector.

2. **Input layer:** The input layer W_{in} is a fully connected layer that divides the input into multiple channels N_{ch}.

3. **Clebsch Gordan (CG) Layer:** The CG layers, L_{CG}, are the layers with nonlinear mappings which update the features stored in the nodes while preserving Lorentz invariance.

4. **Multi-layer Perceptron (MLP):** The MLP layer, MLP_{inv}, (here *inv* stands for invariant), act on the scalar components to ensure the nonlinearity while doing nothing to other representations.

5. **Output layer:** The final P_{inv} projects the features in the final layer into scalars. These scalars are then added up to ensure permutation invariance. The final layer of the network is the output layer, W_{out} , which produces scalar weights for binary classification **[is_background_weight, is_signal_weight]** for each jet.

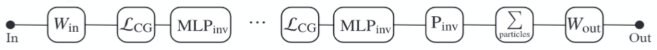

FIG. 6: LGN Architecture used in the classification problem.Here CG stands for Clebsch Gordan in $\mathcal{L}_{CG}$

Input to LGN

The input consists of the 4-momenta vector (E, p_x, p_y, p_z) of N_{obj} particles in the jet from a collision event with Lorentz scalars with them such as label, charge, spin, etc. Thus, the input vectors $\in T^{(0,0)^{\oplus \tau_0}} \bigoplus T^{(1,1)}$ representation of the Lorentz group. The irreducible representations of the Lorentz group are the tensor products of representations of $SU(2)$ which is given by

$$SU(2) = \left\{ \begin{pmatrix} \alpha & -\bar{\beta} \\ \beta & \bar{\alpha} \end{pmatrix} : \alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1 \right\} \quad (0.1)$$

Hence, $T^{(0,0)}$ represents the scalar part of the tensor and τ_0 is the number of scalar components. In this case, $\tau_0 = 2$. Similarly $T^{(1,1)}$ represent the 4-momenta vector.

Input layer

The input layer W_{in} is fully connected linear layer which produces $N_{ch}^{(0)}$ (here 0 means that these channels are the input to the first CG layer) channels given by vectors in each irreducible components. The layer acts on the momenta and lorentz scalar vector of each particle separately but the weights are shared to enforce permutation invariance. The input layer produces N_{obj} features $F_i^{(0)}$ for $i \in \{1, ..., N_{obj}\}$. Hence the dimension of the output is

$$N_{obj} \times (T^{(0,0)^{\oplus \tau_0}} \bigoplus T^{(1,1)})^{N_{ch}^{(0)}} \tag{0.2}$$

which turns out to be $N_{obj} \times 6^{N_{ch}^{(0)}}$.

Clebsch Gordan Layer $\mathcal{L}_{CG}$

The input to the p-th CG layer (starting with p=0) are N_{obj} features $\mathcal{F}_i^{(p)}$ belonging to some representations of the Lorentz group, where $i \in \{1, ..., N_{obj}\}$. The CG layers update the features according to the update rule

$$\mathcal{F}_i^{(p+1)} = \mathcal{L}_{CG}(\mathcal{F}_i^{(p)}) \tag{0.3}$$

where $\mathcal{L}_{CG}(\mathcal{F}_i^{(p)}) =$

$$W. \left(\mathcal{F}_i^{(p)} \bigoplus CG[\mathcal{F}_i^{(p)}]^{\bigotimes 2} \bigoplus CG[\textstyle\sum_j f(p_{ij}^2)p_{ij} \bigotimes \mathcal{F}_j^{(p)}]\right) \tag{0.4}$$

Here the $\bigoplus$ means stacking the tensors together. The components of eq.0.4 can be interpreted as

1. W is a linear operation which mixes each irreducible representation component to a specific $N_{ch}^{(p+1)}$ channels.
2. $\mathcal{F}_i^{(p)}$ stores the features in the previous CG layer.
3. $CG[\mathcal{F}_i^{(p)}]^{\bigotimes 2}$ models self-interaction of each particle.
4. $CG[\sum_j f(p_{ij}^2)p_{ij} \bigotimes \mathcal{F}_j^{(p)}]$ models pair-wise two-particle interactions. Since indices i and j are permutable, this term ensures permutation invariance.

The learnable matrix W belongs to the set of matrices that parametrize all linear equivariant map between irreducible representation. the function $f(p_{ij}) : \mathbb{R} \longrightarrow \mathbb{R}$ in eq.0.4 is a function with learned parameters and weights the interaction. This function is a linear combination of the Lorentz bell shaped curve

$$f(p_{ij}) = \sum_{i=1}^{10} \lambda_i (a_i + \frac{1}{b_i + c_i^2 p_{ij}^2}) \tag{0.5}$$

where a_i, b_i and c_i are learned parameters.The term $pij = p_i - p_j$ is the distance between two particles in momentum space, where p_i refers to the momentum of the i-th particle. This basically the edge features in the fully connected message passing.
The $CG[.]$ component in eq.0.4 is the Clebsch Gordan decomposition. The main non-linearity in this architecture occurs in this layer where both tensor product and Clebsch Gordan decompostion takes place. The $CG[.]$ function decomposes the tensor product into isotypic components. The coefficients of this decomposition in certain canonical basis are called CG coefficients.
Only the first few irreducible representation components are kept to control memory storage. Tensor products are performed channel-wise to minimize computations.In total there are 4 CG layers, and the number of channels chosen as the input of each CG layer are $N_{ch}^{(0)} = 2$, $N_{ch}^{(1)} = 3$, $N_{ch}^{(2)} = 4$, and $N_{ch}^{(3)} = 3$.

Multilayer Perceptron MLP_{inv}

The MLP layer is only applied to the $N_{ch}^{(0)}$ scalar component channels in the node features given by $(T^{(0,0)})^{\bigoplus N_{ch}^{(p)}}$. Here p index is with respect to the index of the CG layer applied to the output of the MLP layer. The number of channels are conserved in this transformation which means $N_{ch}^{(p)}$ input channels will result into $N_{ch}^{(p)}$ output channels. The MLP layer acts on the scalar values of each particle separately but the parameters are shared across all (N_{obj}) particles for permutation invariance. The activation function used is a linear combination of basis Lorentzian bell-shaped curve

$$a + \frac{1}{b + c^2 x^2} \tag{0.6}$$

where a, b and c are learnable parameters. In this architecture the learning function takes in ten parameters of a, b and c.Thus the learnable function is given by

$$f(x) = \sum_{i=1}^{10} \lambda_i (a_i + \frac{1}{b_i + c_i^2 x^2}) \tag{0.7}$$

Output layer

The output layer takes the arithmetic sum of N_{obj} features produced after the last to maintain permutation invariance.For classification task, the Lorentz-invariant outputs are the only vital result. Hence as given in 0.8, the output only extracts the invariant isotypic component of this sum and applies a fully connected linear layer W_{out} to the $N_{ch}^{(3)}$ channels producing two scalar weights for binary classification.

$$\vec{w}_{out} = W_{out} \cdot (\sum_i \mathcal{F}_i^{(N_{CG})})_{(0,0)} \tag{0.8}$$

The final output of the LGN is [**is_background_weight, is_signal_weight**] which assigns a classification label to each jet.

V. LGN MODEL TRAINING

As explained in section IV, the LGN is a complicated DNN with built-in internal symmetry operations that preserve elements of the Lorentz group. In ML terms, it is an equivariant model implemented to the task of binary classification. Simply put, the LGN takes as input the set of 4-momenta of all particles in a jet (alongside Lorentz scalars like the spin and charge) and outputs a binary label that classifies the jet. It is a standard signal vs background classifier, where our signal is defined to be top quark jets (with label 1), and our background is defined to be QCD jets (with label 0).

In this section, we will give a brief overview on the dataset we used to train the LGN, followed by a discussion on the training process of the LGN as a classifier in terms of (a) computational cost, and (b) training result.

Dataset

We have trained the LGN model on the problem of *top tagging*. This is a classification task that aims to identify top quark jets among a background of lighter quark jets, known as QCD jets. Since the classification task is independent of the inertial frame of the observer, the outputs of the classifier should be Lorentz invariants. In other words, we expect the classifier output (which is a label between 0 and 1) to be immune to Lorentz boosts/rotations of the input jets.

We perform top tagging classification experiments on the publicly available reference dataset [13], the same dataset used to produce the results of the paper [3]. The dataset contains 1.2M training entries (1.6M including validation entries), and 400k testing entries. Each of these entries represents a single jet whose origin is either a top quark or QCD. The events were produced with center-of-mass energy $\sqrt{s} = 14$ TeV, using the PYTHIA Monte Carlo event generator [14]. The ATLAS detector response was modeled with the DELPHES software package [15].

The jets are clustered using the anti-kt algorithm [8], with a radius of R = 1, where R is measured in (η, ϕ) coordinates. For each jet, the 4-momenta of the particles are saved in Cartesian coordinates as (E, p_x, p_y, p_z) for up to 200 constituent particles sorted by the highest transverse momentum $\sqrt{p_x^2 + p_y^2}$ where the colliding particle beams are aligned along the z-axis. On average, each jet contains 50 particles, so events with less than 200 particles are zero-padded (zero-padding means manually adding extra particles with 4-momentum equals to 0). The 4-momenta in the dataset are all scaled by a uniform factor at the input to the network to avoid overflows and losses.

Computational cost

In practice, training a DNN to have benchmark performance is tricky. Not only does it require fine-tuning of hyperparameters by the developers, but also requires highly parallelizable machines (such as GPUs) to speed up the training process. For example, DNN models trained for the ImageNet Large Scale Visual Recognition Challenge take many days for optimal training. Finishing a 90-epoch ImageNet-1k training with *ResNet-50* on a NVIDIA M40 GPU can take up to 14 days [16].

Since we're training the model on quite a large dataset ($\sim$ 1.5M entries), in addition to the model's complicated forward pass, composed of tensor products and Clebsch-Gordon decompositions (as explained in section III), it is no surprise that the training over 20 epochs lasted for around 6 days, putting an average of about 7.2 hours per epoch. This is comparable to the performance stated in their paper [3], as they have trained the model for 53 epochs, with a wall-clock time of about 7.5 hours per epoch, on the same machine we were using (RTX 2080Ti). The reason why we stopped the training after only 20 epochs was mainly because of time constraints to finish the project, and also because the model's improvement was very slow at this mark.

One thing we have noticed during the model training is the non-uniform time per epoch. Figure 7 shows a plot of the time the model took to train per epoch, and it looks like the model had a harder time training near the end (around epoch number 16). So far, the reason for this non-uniformity is not clear. One reason might be a hardware failure (like GPU overheating so minimizing its utilization). Another reason might be that some Clebsch-Gordon decompositions become much more difficult as you train the model. In any case, to test the real reason requires another big training (maybe even on another GPU machine) which we didn't have time for right now.

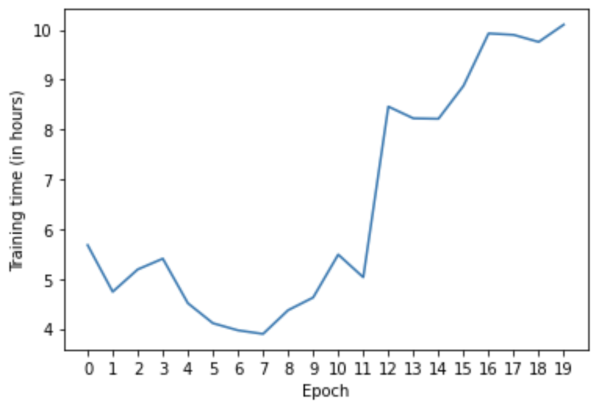

FIG. 7: A plot showing the non-uniformity of time per epoch during training.

Training result

A main plot to assess the training of a ML model is the loss function vs number of epochs (basically number of iterations). As explained in section III, the loss function is a measure of the deviation of the prediction of the model from the truth label. Thus, it is instructive to check how this function behaves as the model is training. Not only that, but it is also a good idea to distinguish the training loss from the validation loss, also called generalization loss, to make side by side comparison plots. This allows us to closely monitor how well the model is generalizing to prevent the model from overfitting on the training data. Usually, we stop the training once both the training loss and validation loss converge. Leaving the training for too long usually means a decrease in the training loss but at the expense of an increase in the validation loss (a clear mark of overfitting). Figure 8 visually illustrates what happens as you train a ML model.

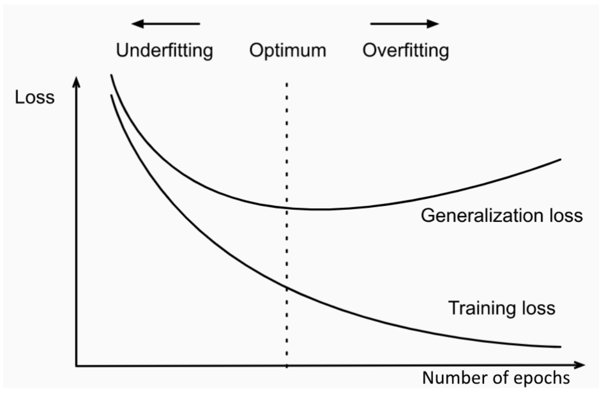

FIG. 8: A visual illustration of a typical training of a ML model.

In our case, we have trained the model for 20 epochs and here are our loss plots shown in Figure 9.

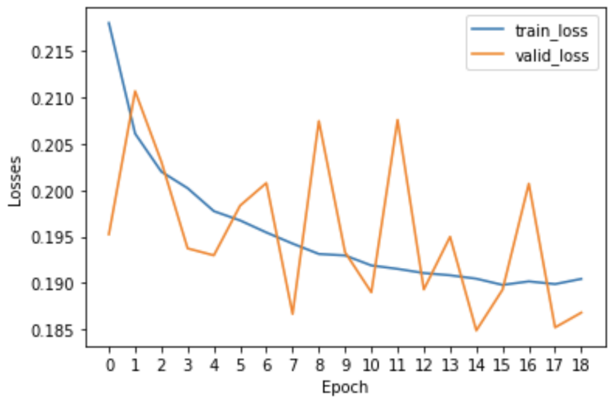

FIG. 9: Training plots of the training loss and validation loss per number of epochs.

We can see that the training loss indeed converges which is a sign that the model is indeed training. The validation loss has also converged but the orange curve in Figure 9 is not smooth simply because of a technical detail of how we calculate these loss functions. Basically, during one epoch (one iteration over 1.2M jets) the model updates its parameters many times, specifically, each time it passes a batch. Since we chose a batch size of 32 jets, this meant that the model updated its parameters around 37.5k times per epoch. Training loss values are calculated after each batch of the training sample is passed and then we take an average over all batches as the model updates its parameters during the epoch. This in effect means that the training loss from one epoch to the next is smooth. But for the validation loss, the story is a bit different because we are calculating the validation loss after each whole epoch of training. This meant that we are calculating the validation loss values after the model has updated it parameters so many times. Thus, the validation loss curve doesn't seem as smooth as the training loss curve. Aside from this technical detail, the take-away is that both curves, the blue one (for the training loss) and the orange one (for the validation loss) are both converging towards a minimum value, which is what you would expect from a ML model during training.

Usually, the main metric to assess the performance of a ML model is the accuracy of the model. We calculate the accuracy of the LGN model by predicting the class of jets from the, so far untouched, test dataset, after each epoch of training. Also for better monitoring of the training, it is often useful to consider the training accuracy and validation accuracy as well. Figure 10 shows accuracy plots during training of the model.

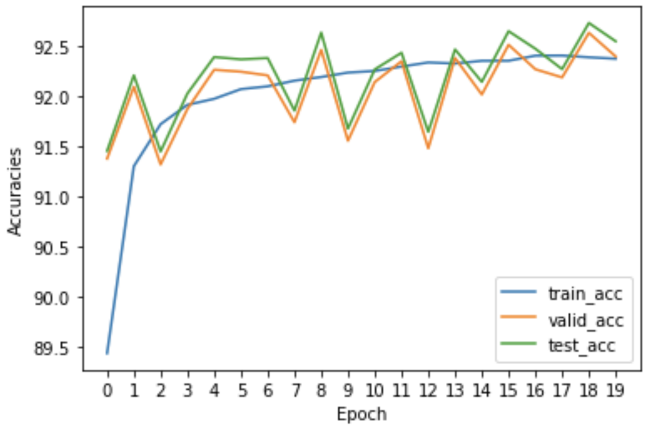

FIG. 10: Training plots of the training accuracy, validation accuracy, and test accuracy per number of epochs.

We see a smooth increase in the training accuracy curve (blue curve) which indicates that the model is learning the data. We also see a somewhat slow and steady increase in the validation accuracy curve (orange curve) and test accuracy curve (green curve) which means the model is indeed improving. We also reach a benchmark test accuracy of 92.7% at epoch number 19, which is comparable to the test accuracy they have achieved in the paper [3] of around 92.9%. Table I shows a comparison of the accuracy our LGN model achieved, the accuracy the paper's LGN model achieved [3], and the accuracy of another state-of-the-art model called ParticleNet (which is not Lorentz group equivariant). It is no surprise that the ParticleNet model has higher accuracy than the LGN, as the LGN is just the earliest major attempt of a Lorentz group equivariant model.

TABLE I: Accuracy results

DNN model	Accuracy
ParticleNet	93.8%
LGN (paper[3])	92.9%
LGN (us)	92.7%

VI. LGN MODEL EVALUATION

The development of ML algorithms necessitates a search for metrics that evaluate the performance of those algorithms. These metrics might change depending on the task at hand. For example, metrics like the mean squared-error (MSE), or mean average error (MAE), are found to be better suited for regression tasks. While other metrics like: accuracy, receiver operating characteristic (ROC) curves, confusion matrix, F1-score and others; are found better suited for classification tasks. In section V, we already explored one metric, namely the accuracy. In this section, we will explore two other popular ML metrics, that are mostly used for classification tasks, the ROC curves and the confusion matrix. Furthermore, we will show some tests we did to test the equivariance of the LGN as compared to a basic DNN we have built.

ML metrics

First, let us begin by a brief overview on what a ROC curve is. It is a graphical plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied. After training the classifier to discriminate between signal and background, we make two histogram plots, the first histogram refers to all signal events and their classified output, while the second histogram refers to all background events and their classified output. Figure 11 shows a basic plot of what these histograms might look like after you train a binary classifier (this plot is only for illustrative purposes, it is not a result of our actual LGN model). For each classifier cut (x-axis value), we start counting the fraction of events to the right of each histogram. In Figure 11 we pick a random classifier cut of 0.8, also just for illustrative purposes. The fraction of events to the right of the signal (orange) histogram at this cut is what we call the True Positive Rate (TPR) at a cut of 0.8. The fraction of events to the right of the background (blue) histogram at this cut is what we call the False Positive Rate (FPR) at a cut of 0.8. We then use those two values (FPR,TPR) to make a point on the ROC curve. So essentially, a ROC curve is a curve plotted with x-axis being FPR, and y-axis being TPR, and each point on this plot correspond to a different classifier cut (chosen between 0 and 1, because this is the range of our binary classifier).

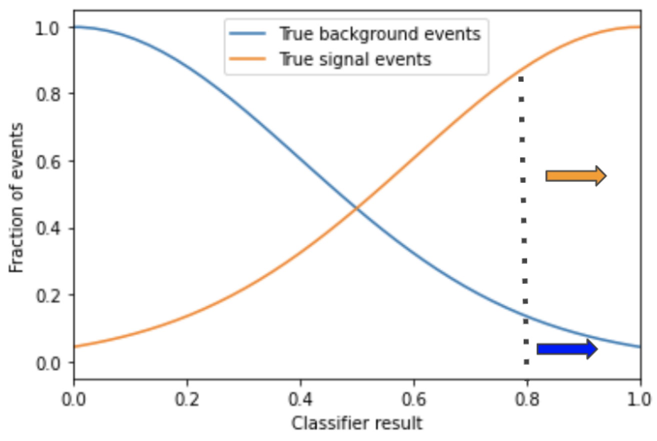

FIG. 11: A typical result of a binary classifier after training (for illustration purposes only).

Figure 12 shows the ROC curve plot for our model after 19 epochs of training. A very useful metric to extrapolate from the ROC curve is called the Area Under the Curve (AUC). AUC provides an aggregate measure of performance across all possible classification thresholds. The closer the AUC is to 100% the better.

From the legend in Figure 12, one can read that we have achieved an AUC of 96.9% which is higher than the AUC achieved in the paper [3] by the LGN model (96.4%). This result is really nice considering our resources and the time constraint we had to produce this work. Table II shows a comparison of the AUC our LGN model achieved, the AUC the paper's LGN model achieved [3], and the AUC of another state-of-the-art model the ParticleNet (which is not Lorentz group equivariant).

The next ML metric we consider is the confusion matrix plot. It has a specific table layout that allows visualization of the performance of a classifier at one classifier cut. Typically, we pick a classifier cut at 0.5 for binary classification models (unless there is a reason not to). An 0.5 classifier cut means that any event classified with a classifier output greater than 0.5 is considered a signal,

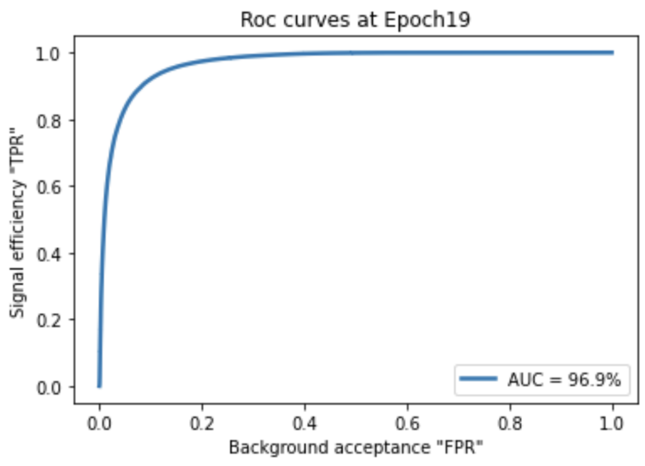

FIG. 12: ROC curves plot after 19 epochs of training.

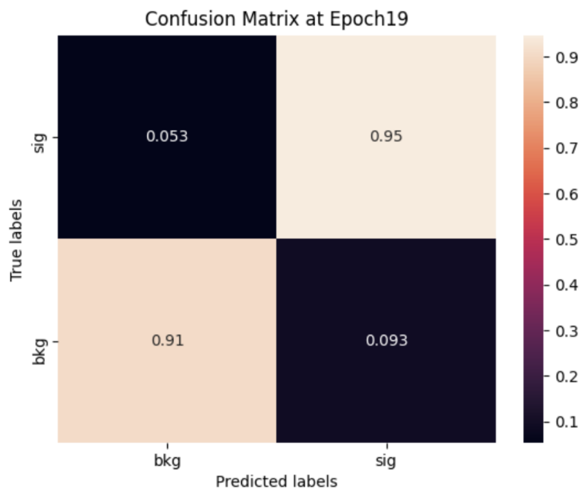

FIG. 13: Confusion matrix plot after 19 epochs of training.

TABLE II: AUC results

DNN model	AUC
ParticleNet	98.5%
LGN (paper[3])	96.4%
LGN (us)	96.9%

and any classifier output less than 0.5 is considered a background. We can then calculate the TPR and FPR at a cut of 0.5 as we explained in the last paragraph, by counting the fraction of events in the histograms. Furthermore, if we also calculate the True Negative Rate (TNR), by counting events to number of events to the left of the cut at 0.5 of the background (blue) histogram, as well as the False Negative Rate (FNR) by counting events to number of events to the left of the cut at 0.5 of the signal (orange) histogram; we can then build this confusion matrix table shown in Figure 13.

From Figure 13, we can see read off four quantities from each of the four squares. The FNR is about 5% (top left square); which indicates that 5% of truly labeled signal jets are wrongly classified as background. The TPR is about 95% (top right square); which indicates that 95% of truly labeled signal jets are correctly classified as signal. The TNR is about 91% (lower left square); which indicates that 91% of truly labeled background jets are correctly classified as background. The FPR is about 9% (lower right square); which indicates that 9% of truly labeled background jets are wrongly classified as signal. Simply put, what we expect from a well trained binary classifier is a confusion matrix plot that has greater values on the diagonal (lower left and top right squares). This confusion matrix plot is more often used for multiclass classification tasks, where this matrix would be an $N_c \times N_c$ matrix (where N_c is the number of classes). In such tasks, one can easily reflect on which classes are harder to classify, and which pairs of classes are very similar and so confusing to the classifier.

Equivariance tests

So far, we have discussed common ML metrics such as: accuracy, ROC curves, and confusion matrix plots; and we showed how our LGN model is performing really well with respect to them. In this section, we will show how, in addition to those metrics, our model is performing really well from a physicist point of view. We explore two tests to check the equivariance of the model: rotation invariance test, and Lorentz boost invariance test.

For the *rotation invariance test*, we first pass a jet to the model and predict its output label (Y). Then, we rotate the same jet and pass it again to the model to predict its label (Y_{rot}). We then compute the deviation between the output of the model for the jet and its rotated version $|Y - Y_{rot}|$. Figure 14 shows a plot for the deviation in the output with the angle of rotation. The green curve presents the result for the LGN model, while the red curve presents the result for a basic DNN model that we set up only for testing purposes. We can see how the LGN model is shown to be somewhat robust with respect to the increase in the angle of rotation, compared to the result of the DNN model which is shown to have a much higher deviation. Note that the reason why the deviation values are low (of the order 10^{-7}) is probably because this is a binary classification task, and so there is little room to go wrong with the prediction (after all a random prediction is correct 50% of the time). We believe that for multiclass classification tasks, the deviation values will be much more telling.

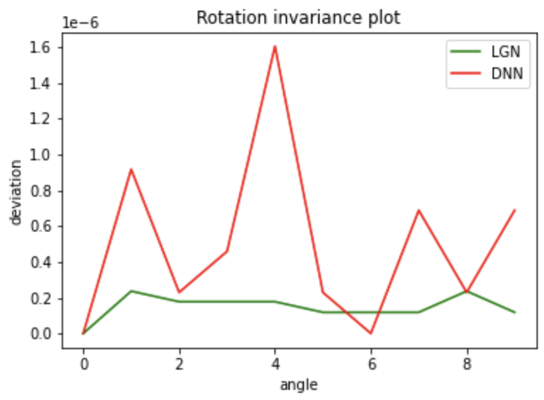

FIG. 14: Rotation Invariance test plot. The green curve presents the deviation in the prediction of the LGN when passing jets and rotated versions of them. The red curve presents the deviation in the prediction of a basic DNN model that we set up only for testing purposes. The angle is measured in degrees.

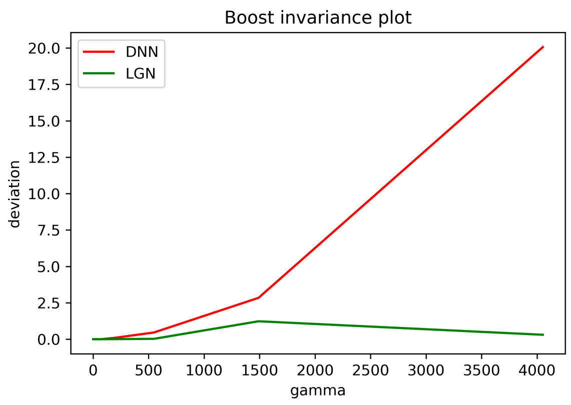

FIG. 15: Boost Invariance test plot. The green curve presents the deviation in the prediction of the LGN when passing jets and boosted versions of them. The red curve presents the deviation in the prediction of a basic DNN model that we set up only for testing purposes.

Another test to check the equivariance of the model is the *Lorentz boost invariance* test. For this test, we also pass a jet to the model and predict its output label (Y), but then instead of passing a rotated version of the jet, we pass a boosted version of the jet to the model to predict its label ($Y_{boosted}$). A boosted version simply means that we apply a Lorentz transformation on all the 4-momenta of the particles in the jet. It basically refers to a version of the jet as observed from another inertial frame moving with velocity β. It is more instructive to speak in terms of the factor γ, where $\gamma = \frac{1}{\sqrt{1-\beta^2}}$. We then compute the deviation between the output of the model for the jet and its boosted version $|Y - Y_{boosted}|$.

Figure 15 shows a plot for the deviation in the output with the factor γ. The green curve presents the result for the LGN model, while the red curve presents the result for a the same basic DNN model that we had set up for the rotation invariance test. We can see how the LGN model is showing little deviation with increasing values of γ, compared to the result of the DNN model, indicating that indeed the LGN is a successfully built physics equivariant model. Note that the increase in deviation for the DNN (red) curve with γ is expected because the larger the γ the larger the values in the Lorentz matrix we use to boost the jets, and so the bigger the change in the actual values of the 4-momenta.

CONCLUSION

We have successfully tested the LGN model for a binary classification task in HEP. We believe the test accuracy our model achieved is a success when compared to the actual results of the paper [3]. As claimed in the paper, this is believed to be the first application of a fully Fourier space equivariant architecture in physics, following an application in chemistry [17]. This approach is an important stepping-stone in building a family of physics-informed ML algorithms based on group theory. From a physics point of view, symmetries have always been central to conservation laws and the way we formulate our theories, and so it feels interpretable to build a ML model which respects physics symmetries.

A possible shortcoming of a physics equivariant network, like LGN, is the amount of time it takes to develop one. However, once developed for a specific symmetry group, such as the Lorentz group, SO(1, 3), it is broadly applicable to many problems with the same symmetry at a very low development cost.

Future work might explore the addition of extra particle kinematic information. Another exciting problem is applying the network to regression tasks such as regressing the 4-momenta of particles in a jet. Finally, one might try to include the multiple symmetries in SM, which include U(1), SU(2) and SU(3), in one network architecture.

Acknowledgements

We would like to thank Zichun Hao for his input to understanding the forward pass of the model. We also want to thank Raghav Kansal and Javier Duarte for the weekly check-up on the project.

[1] A. Ivakhnenko and L. G., “Cybernetic Predicting Devices,” *CCM Information Corporation* (1965). 1
[2] CERN, “Storage,” https://home.cern/science/computing/storage. 1

[3] A. Bogatskiy *et. al.*, “Lorentz Group Equivariant Neural Network for Particle Physics,” *37th International Conference on Machine Learning* (2020) 2006.04780. 1, 6, 8, 9, 10

[4] S. B. Oliver *et. al.*, “LHC Design Report,” *CERN Yellow Reports: Monographs* (2004) http://cds.cern.ch/record/782076. 2

[5] J. T. Boyd, “LHC Run-2 and Future Prospects,” 2001.04370, https://cds.cern.ch/record/2197559. 2

[6] G. Aad *et. al.*, “The ATLAS Experiment at the CERN Large Hadron Collider,” *Journal of Instrumentation* (2008) http://nordberg.web.cern.ch/nordberg/PAPERS/JINST08.pdf. 2, 3

[7] S. Chatrchyan *et. al.*, “The CMS Experiment at the CERN LHC,” http://dx.doi.org/10.1016/j.physletb.2012.08.020. 2

[8] M. Esma, “The CERN accelerator complex. Complexe des accélérateurs du CERN,” https://cds.cern.ch/record/2197559. 2

[9] G. Aad *et. al.*, “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC,” *Physics Letters B 716.1* (2012) http://dx.doi.org/10.1016/j.physletb.2012.08.020. 2

[10] E. Aakvaag, “Search for Dark Matter produced in association with a Higgs boson decaying to tau leptons at $\sqrt{s} = 13$ TeV with the ATLAS detector,” *CERN-THESIS-2020-293* (2021) https://cds.cern.ch/record/2753626/files/CERN-THESIS-2020-293.pdf. 3

[11] R. Kansal, “Symmetry Group Equivariant Neural Networks,” https://mcgreevy.physics.ucsd.edu/f20/final-papers/2020F-220-Kansal-Raghav.pdf. 3

[12] A. Butter, G. Kasieczka, T. Plehn, and M. Russell, “Deep-learned Top Tagging with a Lorentz Layer,” https://scipost.org/10.21468/SciPostPhys.5.3.028. 3

[13] G. Kasieczka, T. Plehn, J. Thompson, and M. Russel, “Top Quark Tagging Reference Dataset,” https://zenodo.org/record/2603256.YEdwA10zY-Q. 6

[14] T. Sjostrand *et. al.*, “An Introduction to PYTHIA 8.2,” *Comput. Phys. Commun.* (2015). 6

[15] J. Faverau *et. al.*, “A modular framework for fast simulation of a generic collider experiment,” *JHEP* (2014). 6

[16] Y. You *et. al.*, “ImageNet Training in Minutes,” 1709.05011. 6

[17] B. M. Anderson *et. al.*, “Cormorant: Covariant Molecular Neural Networks,” *NeurIPS 2019* (2019). 10