Eur. Phys. J. C (2021) 81:540
https://doi.org/10.1140/epjc/s10052-021-09342-y

THE EUROPEAN PHYSICAL JOURNAL C

Regular Article - Experimental Physics

Secondary vertex finding in jets with neural networks

Jonathan Shlomi[1,a], **Sanmay Ganguly**[1], **Eilam Gross**[1], **Kyle Cranmer**[2], **Yaron Lipman**[1], **Hadar Serviansky**[1], **Haggai Maron**[3], **Nimrod Segol**[1]

[1] Weizmann Institute of Science, Rehovot, Israel
[2] NYU, New York, USA
[3] NVIDIA Research, Tel Aviv, Israel

Received: 1 October 2020 / Accepted: 14 June 2021 / Published online: 23 June 2021

Abstract Jet classification is an important ingredient in measurements and searches for new physics at particle colliders, and secondary vertex reconstruction is a key intermediate step in building powerful jet classifiers. We use a neural network to perform vertex finding inside jets in order to improve the classification performance, with a focus on separation of bottom vs. charm flavor tagging. We implement a novel, universal set-to-graph model, which takes into account information from all tracks in a jet to determine if pairs of tracks originated from a common vertex. We explore different performance metrics and find our method to outperform traditional approaches in accurate secondary vertex reconstruction. We also find that improved vertex finding leads to a significant improvement in jet classification performance.

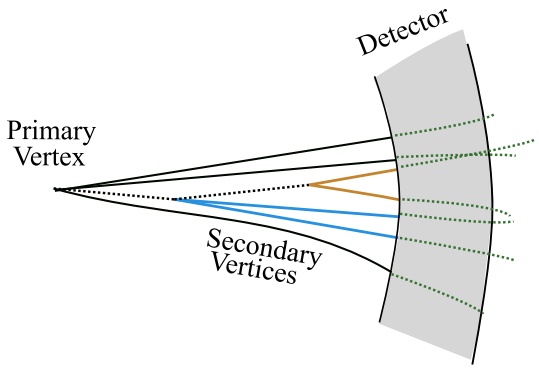

Fig. 1 Illustration of a jet with secondary decay vertices. In order to identify the flavor of the jet, vertex reconstruction aims to group together the tracks measured in the detector based on their point of origin

1 Introduction

Identifying jets containing bottom and charm hadrons and separating them from jets that originate from lighter quarks, is a critical task in the LHC physics program, referred to as "flavor tagging". Bottom and charm jets are characterized by the presence of secondary decays "inside" the jet - the bottom and charm hadrons will decay several millimeters past the primary interaction point (primary vertex), and only stable outgoing particles will be measured by the detector. Figure 1 illustrates a typical bottom jet decay, with two consecutive displaced vertices from a bottom decay (blue lines) and charm decay (yellow lines).

Existing flavor tagging algorithms use a combination of low-level variables (the charged particle tracks, reconstructed secondary vertices), and high-level features engineered by experts as input to neural networks of various architectures in order to perform jet flavor classification [1].

Vertex reconstruction can be separated into two tasks, *vertex finding*, and *vertex fitting* [2]. Vertex finding refers to the task of partitioning the set of tracks, and vertex fitting refers to estimating the vertex positions given each sub-set of tracks. Existing algorithms typically use an iterative procedure of finding and fitting to perform both tasks together. We focus on using a neural network for vertex finding only. Vertex finding is a challenging task because of two factors:

- Secondary vertices can be in close proximity to the primary vertex, and to each other, within the measurement resolution of the track trajectories.
- The charged particle multiplicity in each individual vertex is low, typically between 1 and 5 tracks.

Vertex reconstruction is in essence an inverse problem of a complicated noisy (forward) function:

$$\text{Particle Decay} \rightarrow \text{Particle Measurement in Detector} \quad (1)$$

[a] e-mail: jonathan.shlomi@weizmann.ac.il (corresponding author)

Neural networks can find a model for this inverse problem without expert intervention by using supervised learning, i.e., by providing many examples of the forward process, which can be provided by simulations. They can also be easily optimized by retraining without expert intervention. Particle colliders may have different modes of operation during their lifetime, such as the LHC increasing its collision energy over the years. Different data taking conditions require re-optimizing reconstruction algorithms, and neural networks provide a simple way to perform that re-optimization.

Since the set of tracks to be partitioned has no inherent order, we use an equivariant[1] neural network architecture. We show in this paper that this constraint on the model results in better performance.

We first describe the dataset on which we test our proposed algorithm in Sect. 2. The model architecture and the baseline algorithms are described in Sect. 3. Section 4 discusses the performance metrics defined for vertex finding. Section 5 describes how the impact of vertex finding on jet classification was assessed, and the results are presented in Sect. 6. Conclusions are given in Sect. 7.

1.1 Background

Standard vertex reconstruction algorithms. Existing vertex reconstruction techniques are based on the geometry of the tracks, or a combination of the geometry and constraints that are configured by hand to match a specific particle decay pattern [3]. In order to handle finding and fitting multiple vertices, a standard algorithm is adaptive vertex reconstruction (AVR) [2,4,5]. The basic concept of AVR is to perform a least squares fit of the vertex position given all the tracks, then remove less compatible tracks from the fit, and refit those tracks again to more vertices. This repeats until no tracks are left. AVR can be used to first fit the primary vertex with special considerations for its unique properties, and subsequently fit secondary vertices. In this paper it is used as a general multi-vertex fitter, applied only to tracks associated to a single jet.

Deep learning on sets and graphs. Following the successful application of deep learning to images [6,7], there is an ongoing research effort aimed at applying deep learning to other data structures such as unordered sets [8–10] and graphs [11–14]. Typical learning tasks for such domains are point-cloud classification for sets, or molecule property prediction, for graphs. A challenge in both scenarios stems from the arbitrary order of the elements in the set or the nodes in the graph. Fully connected, convolutional and recurrent networks do not have the correct inductive bias for learning tasks on unordered sets [15]. They assume a fixed size or an ordering in the data. A popular design principle for networks that process such unordered data is constraining layers to be equivariant or invariant to the reordering operation. By using only equivariant layers the neural networks is constrained to represent only equivariant functions.

Recently, the Set2Graph (S2G) model [16] was proposed as a simple, equivariant model for learning tasks in which the input is an arbitrarily ordered set of n elements and the output is an $n \times n$ matrix that represents their pairwise relations. The S2G model was proved to be universal, meaning it can approximate any equivariant function from a set to a graph. We use this model in this paper.

Deep learning for particle physics. Neural networks that operate on sets have been used recently in a number of particle physics applications [17]. The data structure of an unordered set is a natural description for most particle physics reconstruction tasks, and recent progress in the field of graph neural networks [15] has prompted many new applications. For the problem of track reconstruction, a graph neural network was used to classify the paths between adjacent detector "hits" [18,19]. This is a similar application to vertex finding since the end result must be a partition of the set of hits to different tracks. Other applications of graph neural networks to partitioning sets of objects include particle reconstruction in calorimeters and liquid argon time projection chambers [20–23]. Direct jet classification has also been proposed with a few different variants of message passing networks [24–31].

2 Data

We test the proposed algorithm on a simulated dataset.[2] The dataset consists of jets sampled from $pp \to t\bar{t}$ events at $\sqrt{s} =$ 14 TeV. The events are generated with PYTHIA8 [32] and a basic detector simulation is performed with DELPHES [33], emulating a detector similar to ATLAS [34]. charged particle tracks are represented by 6 perigee parameters (d_0, z_0, ϕ, $\cot\theta$, p_T, q) and their covariance matrix. Noise is added to the track perigee parameters with Gaussian smearing. The track parameters resolution depends on the transverse momentum p_T and pseudorapidity η of the track in a qualitatively similar way to the measurements reported in [34]. The covariance matrix is diagonal in this simplified track smearing model –

[1] If x is an $n \times d$ tensor, and σ is a permutation on n elements, then a layer L is called equivariant if $L(\sigma x) = \sigma L(x)$ and invariant if $L(\sigma x) = L(x)$.

[2] The dataset and code used in this paper are available at https://zenodo.org/record/4044628 and https://github.com/jshlomi/SetToGraphPaper.

the smearing is done independently for each parameter with no correlated effects.

Jets are constructed from calorimeter energy deposits with the anti-k_T algorithm [35] with a distance parameter of $R = 0.4$. Charged tracks are cone associated to jets with a $\Delta R < 0.4$ cone around the jet axis. The flavor labeling of jets (as bottom, charm or light) is done by matching weakly decaying bottom and charm hadrons to the jet with a ΔR cone of size 0.3.

A basic jet selection is applied, requiring jets have $p_T > 20$ GeV and $|\eta| < 2.5$ The input to the vertex finding algorithms is the set of tracks associated to each jet, the jet p_T, η, ϕ and jet mass.

Dataset composition. The properties of secondary vertices, such as their distance from the primary vertex, depend on the jet flavor but also on p_T, η, and number of tracks (n_{tracks}). However, the distribution of those parameters is different for the different flavors, depending on the process used to generate the sample. The dataset is therefore built by sampling equal numbers of jets from each flavor in each (p_T, η, n_{tracks}) bin, as illustrated in Fig. 2a. For each bin, the flavor with the least amount of jets (usually c jets) in that bin determines the number of jets from the other flavors that are sampled. Figure 2b shows the resulting distribution of the number of vertices in each jet flavor, and Fig. 2c shows the distribution of p_T, η, and n_{tracks} for all the flavors. The dataset is split into training (500k jets), validation, and testing datasets (100k jets each).

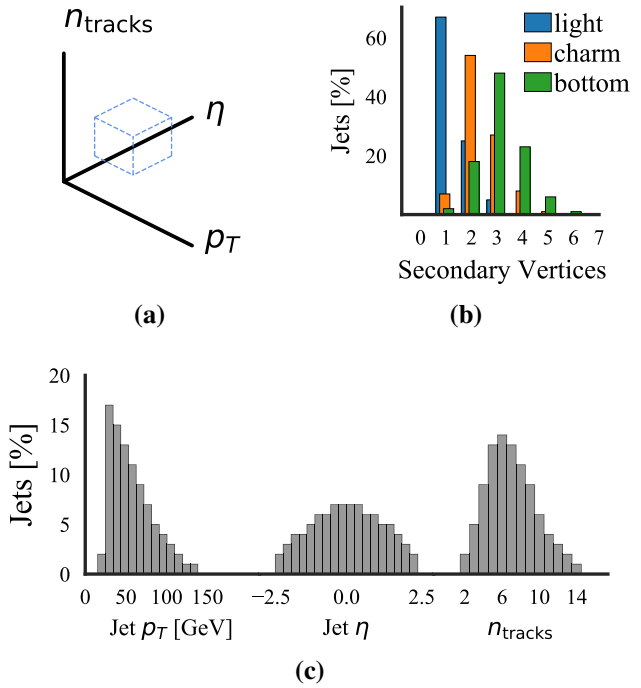

Fig. 2 **a** The dataset is composed by selecting equal numbers of jets from each flavor in each bin of p_T, η, and n_{tracks}. **b** Distribution of the number of secondary vertices for the different jet flavors. **c** The resulting distribution of p_T, η, and n_{tracks} in the dataset

3 Vertex finding algorithms

We compare four different algorithms.

- Adaptive vertex reconstruction (AVR).
- Set2Graph neural network.
- Track pair (TP) classifier.
- Recurrent neural network (RNN) model.

AVR serves as the baseline, and represents the existing vertex reconstruction algorithms. The S2G model is our universal equivariant model. The TP and RNN algorithms are baseline neural networks that are similar to S2G but remove one of its important properties: The TP algorithm is not universal, while the RNN is not equivariant. The architectures of all models are described below.

3.1 Adaptive vertex reconstruction

We use adaptive vertex reconstruction as implemented in the RAVE software package [4]. This algorithm is a representative of existing (non neural network based) methods. The input to the algorithm is the set of tracks associated to the jet and their covariance matrix. The output is a set of vertices, and a set of track-to-vertex association weights. The algorithm can associate a track to more than one vertex. To convert this output into an unambiguous partition, each track is assigned to the vertex to which it has the highest weight. There are hyperparameters that control the iterative fitting or finding procedure such as cuts on the track-to-vertex weight for removing outliers, and these were scanned to find the set of cuts resulting in the highest Rand index (defined in Sect. 4.1). Additional details about the hyper-parameter optimization are given in Appendix A.

3.2 Set2Graph neural network

For the neural network training, the vertex finding task is cast as an edge classification task, as illustrated in Fig. 3. The input consists of the tracks associated to a jet, represented as an array of $n_{\text{tracks}} \times d_{\text{in}}$ matrix, with the $d_{\text{in}} = 10$ features composed of the 6 track perigee parameters and the jet feature vector (the jet features are duplicated for each track). The output is a binary label attached to each pair of tracks indicating whether they originated from the same position in space.

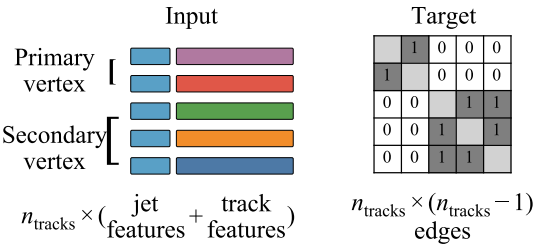

Fig. 3 The input and training target for the neural network algorithms. For a jet with n_{tracks}, the input is an array of $n_{\text{tracks}} \times d_{\text{in}}$ track and jet features (jet features are represented by the light blue boxes, track features by the colored boxes), and the target output is a binary classification label for each of the $n_{\text{tracks}} \times (n_{\text{tracks}} - 1)$ ordered pairs of tracks in the jet

The S2G network is built as a composition of 3 modules, $\psi \circ \beta \circ \phi$: a set-to-set component, ϕ, a broadcasting layer β and a final edge classifier ψ. Here we give only a high level description of what each module does and its purpose, the specific model details are given in Appendix B. The model architecture is illustrated in Fig. 4.

The set-to-set component ϕ takes as input the matrix of size $n_{\text{tracks}} \times d_{\text{in}}$. The output of ϕ is a hidden representation vector for each track, with size $n_{\text{tracks}} \times d_{\text{hidden}}$. ϕ is where information is exchanged between tracks and it is implemented as a deep sets [8] network.

The broadcasting layer β constructs a representation for each ordered pair of tracks (directed edge) using the output of ϕ. The edge representation is simply a concatenation of the representations of the two tracks, with the sum of all track representations, resulting in an output of size $(n_{\text{track}}(n_{\text{track}} - 1)) \times 3d_{\text{hidden}}$.

The edge classifier ψ is an MLP that operates on the edges to produce an edge score. This edge score is trained according to the target defined in Fig. 3. During inference (after the training is complete) the edge scores are symmetrized, so for an unordered track pair the edge score s_{ij} is:

$$s_{ij} = \sigma\left(\frac{1}{2}\left(\psi(\text{track}_i, \text{track}_j) + \psi(\text{track}_j, \text{track}_i)\right)\right) \quad (2)$$

where σ is the sigmoid function.

3.3 Neural network baselines

The neural network baselines are meant to check the importance of the properties of the S2G model. The models have a similar number of trainable parameters: 0.46M for S2G, 0.42M and 0.53M for TP and RNN respectively. They share the same architecture of $\psi \circ \beta \circ \phi$ as the S2G model, with some components replaced as described below. Their properties are summarized in Table 1.

The TP classifier is not a universal model. It will allow us to quantify the contribution of the information exchange between tracks to the overall vertex finding performance. As illustrated in Fig. 5, the hidden representation created for

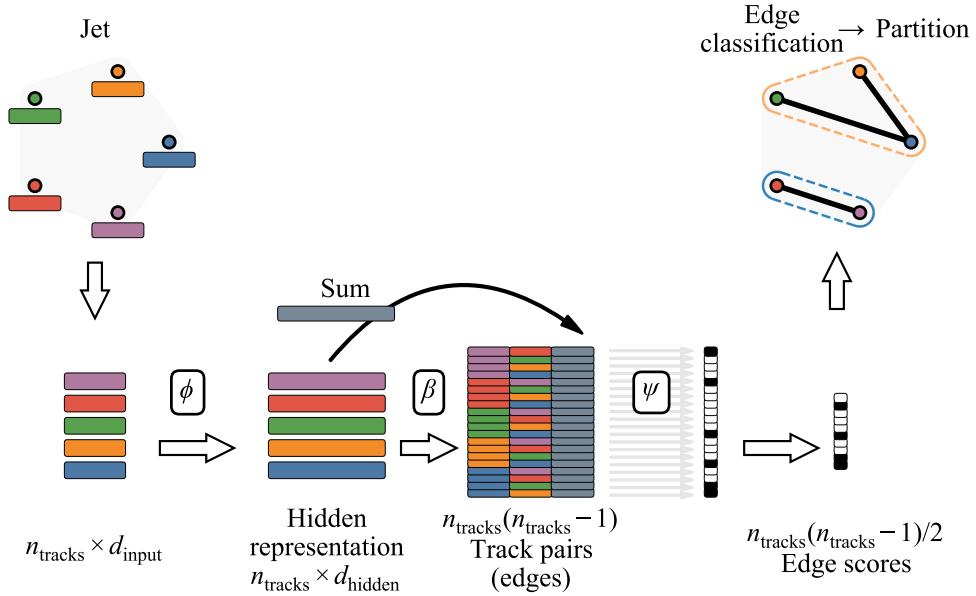

Fig. 4 Partitioning a set of jet tracks using a neural network. A set-to-set component, ϕ, creates a hidden representation of each track, with size d_{hidden}. A broadcasting layer β, then creates a representation for each directed edge (ordered pair of tracks in the jet) by combining the representation of the two tracks and the sum of all representations. An edge classifier ψ then operates on the directed edges. This output is used for training the model (see the target definition in Fig. 3). During inference the output of the edge classifier is symmetrized to produce an edge score. The edge scores are used to define the set partition by optimizing the partition score, as described in Sect. 3.4

Table 1 Comparison of the neural network models. The inference time and FLOPS are measured per single jet with 14 tracks. FLOPS were estimated with [36]

Model	Equivariant/universal	MFLOPS	Parameters	Inference time (ms)
Set2Graph	✓ ✓	7.7	4.6M	5.5
Track Pair	✓ X	6.9	4.5M	2.9
RNN	X ✓	9.1	5.3M	23.4

each track by the deep set module is conditional on the other tracks in the jet. We expect that for the task of vertex finding, being aware of all tracks is important, as the probability of a track pair being connected is conditional on the presence or lack of additional tracks nearby.

The TP classifier checks this assumption about the data. If the probability of each track pair is conditional only on the properties of the track pair, this algorithm will perform as well as the S2G model. It is still expected to perform reasonably well, as it can still learn to join together tracks based on their geometry alone.

The deep set based ϕ layer is replaced by an MLP applied to each track in the jet (independently from the other tracks) to produce some hidden vector representation of that track. While a deep set has been proven to be universal (can approximate any function from sets to sets) [37] applying element-wise MLP is not universal for permutation equivariant functions.

Additionally, the broadcasting layer β does not use the sum of the track hidden representations. The ψ network operates only on the pair of track hidden representations. Therefore in the TP classifier there is no information exchange between the track pairs – each track pair is classified independently.

In the RNN model the ϕ deep set component is replaced by a stack of bi-directional GRU layers [38]. Each GRU layer processes the sequence of track representations, sorted by the track transverse momentum. The layer output is a concatenation of the sequence of hidden representations from both directional passes of the GRU, therefore each track hidden representation still contains information from all other tracks in the jet. This model can theoretically learn any function that the S2G model can, but its architecture is not equivariant. This model will show if the equivariance is a useful inductive bias for this task. Additionally, the sequential nature of the RNN leads to a slower inference time compared to the S2G and TP models (see Table 1).

3.4 Inference

The network output needs to be converted into a cluster assignment for the tracks. If an edge tracks $i \to j$ is connected, and track j is connected to track k, then the edge between $i \to k$ must also be connected, regardless of its

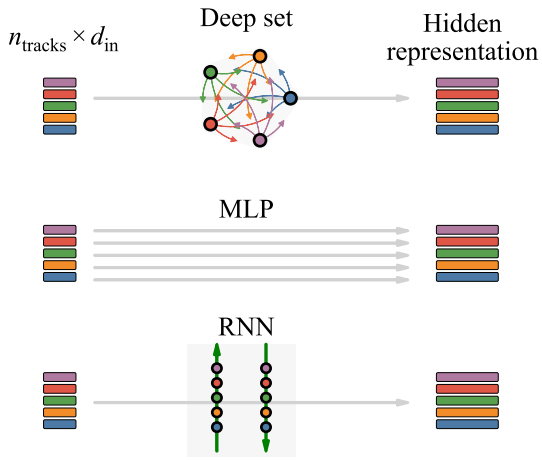

Fig. 5 The deep set module ϕ in the S2G model (top) creates the track hidden representation based on information exchange between the tracks in the jet. The TP classifier (center) however, creates the hidden representation with an MLP, which operates on each track individually. The RNN model (bottom) creates the hidden representation with a bi-directional GRU, which means the output depends on the order in which the tracks are sorted

edges score. This could lead to a situation where many edges with low edge scores are artificially connected. Therefore we utilize the partition score optimization algorithm proposed by the authors of [21]. Track pairs whose score (Eq. 2) is above a threshold of 0.5 are considered in sequence of decreasing score, and are "connected" only if their addition decreases the partition score:

$$\text{Partition score} = \sum \delta_{ij} \ln(s_{ij}) + (1 - \delta_{ij}) \ln(1 - s_{ij}) \quad (3)$$

where δ_{ij} is 1 if track$_i$ and track$_j$ are assigned to the same cluster. In other words, if the connection of two tracks leads to an indirect connection between tracks with low edge scores, the connection is rejected.

3.5 Training procedure and loss function

We train the network f to perform edge predictions, i.e., predicting the probability of each pair of input tracks to originate from the same vertex. For a jet with n_{tracks} we therefore predict $n_{\text{tracks}}(n_{\text{tracks}} - 1)$ edge scores. We train the network

f with the edge predictions before the symmetrization step, which results in $n_{\text{tracks}}(n_{\text{tracks}}-1)/2$ edge scores.

In terms of edge classification, it is import to balance the false positive and negative rates. We initially trained the network with a standard binary cross entropy (BCE) loss function:

$$\text{BCE} = \sum_{\text{edges}} -y_{\text{edge}} \ln(\hat{y}_{\text{edge}}) - (1-y_{\text{edge}}) \ln(1-\hat{y}_{\text{edge}}) \quad (4)$$

where $\hat{y}_{\text{edge}}$ is the edge predicted value, between 0 and 1, and y_{edge} is the truth edge label (0 or 1). The sum is over all edges in a single jet.

Training with BCE loss function resulted in a high number of false negatives. We therefore introduced a loss function based on the F_β score, defined as:

$$F_\beta = \frac{(1+\beta^2)\cdot \text{TP}}{(1+\beta^2)\cdot \text{TP} + \text{FP} + \beta^2 \cdot \text{FN}} \quad (5)$$

with TP, FP, FN the true positives, false positives and false negatives respectively. The F_β score is not differentiable. Quantities such as *true positives* are defined by functions that contain non differentiable conditions, for example:

$$\text{true positives} \equiv \sum_{\text{edges}} (\hat{y}_{\text{edge}} > \text{threshold})\, y_{\text{edge}} \quad (6)$$

To compute a differentiable F_β loss, denoted as F_β^* these quantities are approximated as differentiable functions:

$$\begin{aligned} \text{true positives}^* &\equiv \sum_{\text{edges}} \hat{y}_{\text{edge}} \cdot y_{\text{edge}} \\ \text{false positives}^* &\equiv \sum_{\text{edges}} \hat{y}_{\text{edge}} \cdot (1-y_{\text{edge}}) \\ \text{false negatives}^* &\equiv \sum_{\text{edges}} (1-\hat{y}_{\text{edge}}) \cdot y_{\text{edge}}. \end{aligned} \quad (7)$$

However, training with the F_β^* loss only was unstable. Given the random weight initialization of the network, the training would sometimes fail to converge. A combined loss of BCE and F1 was finally used:

$$\text{Loss} = \text{BCE} - \lambda \sum_{\text{jets}} F_\beta^* \quad (8)$$

λ and β are hyperparameters that control the balance between false negatives and false positives.

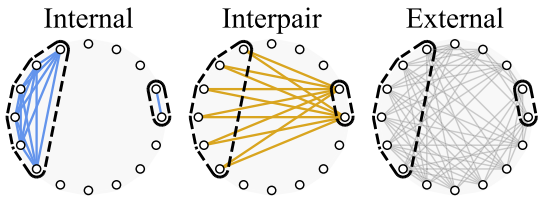

Fig. 6 Definition of internal, interpair and external edges for a pair of vertices

4 Performance metrics for vertex finding

We quantify the vertex finding performance from 3 different perspectives: The entire jet, individual vertices and pairs of vertices. The motivation for defining multiple metrics is that vertex finding is an intermediate step which is used for a number of other tasks related to event reconstruction. Therefore it is important to quantify the performance for a wide variety of jets with different kind of decay topologies.

4.1 Overall jet performance

For jets as a whole, we consider the adjusted Rand index (ARI) [39]. ARI is a measure of the similarity between two set partitions. For vertex finding where the ground truth is well defined, we can treat the ARI of a jet as a "score" that tells us how well our vertex finding algorithm reproduced the ground truth partition. ARI is a normalized form of the Rand index, defined as:

$$\text{RI} = \frac{\text{number of correct edges}}{\text{number of edges in the set}}. \quad (9)$$

Correct edges are edges whose label matches the label they have in the ground truth (true positives and true negatives). The adjustment of the RI is done by normalizing relative to the expectation value or the RI:

$$\text{ARI} = \frac{\text{RI} - \mathbb{E}[\text{RI}]}{1-\mathbb{E}[\text{RI}]}. \quad (10)$$

The expectation value of the RI is defined by a choice of a random clustering model. There are several models one can adopt, described in Ref. [40]. In our case a suitable choice is the "one-sided" comparison, where the true vertex assignment is considered fixed, and the expectation value is computed assuming one draws a completely random vertex assignment for the algorithm prediction. The expression for the expectation value is therefore:

$$\mathbb{E}[\text{RI}] = \frac{B_{N-1}}{B_N} \frac{\sum_i \binom{g_i}{2}}{\binom{N}{2}} + \left(1 - \frac{B_{N-1}}{B_N}\right)\left(1 - \frac{\sum_i \binom{g_i}{2}}{\binom{N}{2}}\right) \quad (11)$$

where $N \equiv n_{\text{tracks}}$, B_N is the bell number (the number of possible partitions of a set with N elements), the sum is over the i vertices in the jet and g_i is the number of tracks in the i-th vertex.

An ARI score of 1 means the algorithm found the correct cluster assignment, while 0 represents a cluster assignment that is as good as random guessing. We consider the ARI score in 3 categories: perfect (ARI of 1), intermediate (ARI between 0.5 and 1), and poor (ARI lower than 0.5).

4.2 Vertices and vertex-pairs performance

Instead of looking at an entire jet, we can consider subsets of the jet – individual vertices and all possible vertex pairs. We distinguish between *internal*, *external*, and *inter-pair* edges. Figure 6 illustrates the definition. Internal edges connect tracks inside a vertex, Interpair edges connect tracks in one vertex to tracks in the other vertex (this definition is only relevant for vertex pairs), and external edges connect tracks from the vertex/vertex pair to other tracks in the jet. Note that "external edges" refers to edges that are connected only at one end to one of the tracks in the subset under consideration (vertex or vertex pair) – not to all edges that are external to the subset. Considering a specific vertex, or a pair of vertices, we can compute separately the accuracy for each type of edge:

$$\text{Accuracy}_{\text{edge type}} = \frac{\text{correct edges}}{\text{number of edges of that type}} \tag{12}$$

where for internal edges, *correct edges* are those predicted to be connected by the algorithm, and for the other types, correct edges are those predicted to be disconnected.

We can also multiply the different kinds of accuracies to compute an overall accuracy for the vertex/vertex-pair in question.[3]

For individual vertices, we can evaluate the accuracy as a function of any vertex property we deem important, for example the number of tracks in the vertex. For vertex pairs, an important metric is the performance as a function of the distance between the two vertices. It is expected that as the distance between vertices decreases, accurate vertex finding becomes more difficult, and nearby vertices will be merged. The vertex pair performance metrics allow us to quantify that.

[3] For vertices without one kind of edge (e.g. vertex with 1 track and no internal edges) the accuracy for that type is set to 1.

5 Impact on jet classification

In order to asses the impact of improved vertex finding on jet classification, we trained a classifier that took the edge classification prediction of the different algorithms as input, along with the tracks and jet features. The classifier predicts if the jet is a bottom, charm or light jet. The architecture for jet classification is illustrated in Fig. 7. A vertex finding module (either AVR, or one of the neural network models) is used to produce an edge prediction for the input set of tracks, which is added to a hidden representation created by a deep set. The resulting graph is processed by a graph network [15] and the resulting graph representation is classified by an MLP. Details about the architecture and training are given in Appendix C. In this scenario, the edge predictions can be considered as a form of supervised attention for the jet classifier. The weights of the vertex finding module are frozen during training.

The baseline classification performance is given by training the same model with an untrained S2G vertex finding module. This baseline model has the ability to reach the same performance as the model with the pre-trained S2G network, as it is an identical network. However it is trained only with the classification objective, where both vertex finding module and the rest of the network are trained together. This baseline therefore shows if an unsupervised attention mechanism can reach similar classification performance, which would require it to identify the relevant features in the data without guidance.

6 Results

The vertex finding results are summarized in Table 2. The S2G model outperforms AVR in all jet performance metrics. The improvement is significant (about 20% increase in ARI) for b and c jets, while for light jets the same high performance is maintained. The ARI distribution for the different flavors is shown in Fig. 8 – while there is still a substantial amount of poorly reconstructed jets (with ARI < 0.5) there are more than twice as many perfectly reconstructed b and c jets compared to AVR. In Fig. 9 the mean ARI is shown as a function of both the number of tracks, and the number of vertices in the jet. For b jets, there is a very large improvement in jets with a small number of tracks, but the advantage over AVR is maintained across the entire range. The AVR algorithm outperforms S2G only in b and c jets which have only one vertex, which are very rare in the dataset.

When considering vertex and vertex-pair metrics, for bottom and charm jets the mean internal accuracy for S2G is within 1% of the baseline, and a large increase (between 10 to 20%) is achieved for external and inter-pair accuracy. Figure 10 shows the performance for vertices, as a function of

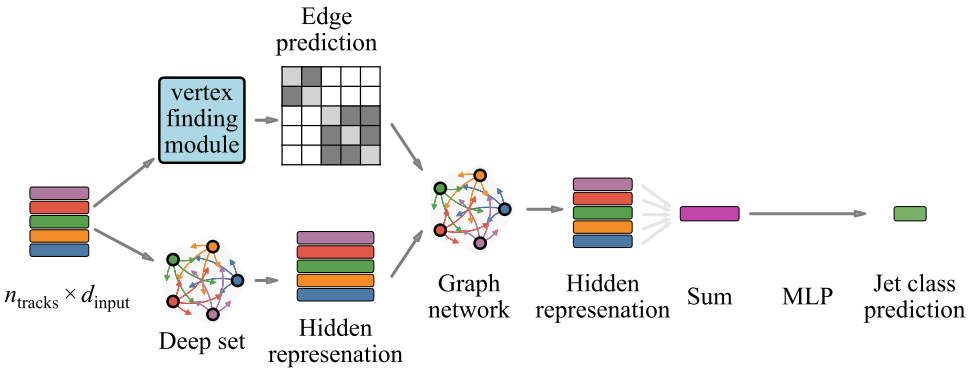

Fig. 7 Jet classification model. The vertex finding module contains either one of the neural network models described in Sect. 3, or the predictions produced by the baseline AVR algorithms, pre-computed on the training dataset. If a pre-trained network in used in the vertex finding module, its weights are frozen during the training of the jet classifier

Table 2 Comparing vertex finding performance from three perspectives: jet, vertex and vertex-pair. See Sect. 4 for the definitions of the various metrics. The mean for each metric, split by jet flavor, is shown for the S2G, AVR and TP algorithms. The S2G model outperforms or equals the other algorithms, maintaining the baseline AVR high accuracy for light jets with significant improvements for b and c jets

	Algorithm	Jet			Vertex			Vertex-pair				
		F1	RI	ARI	Internal	External	Combined	Internal_1	Internal_2	Interpair	External	Combined
b jets	AVR	0.56	0.61	−0.01	**0.91**	0.51	0.46	**0.59**	**0.90**	0.54	0.58	0.18
	Track pair	0.62	0.74	0.32	0.86	0.71	0.60	0.55	0.87	0.72	0.74	0.29
	RNN	0.59	0.75	0.37	0.79	**0.77**	0.60	0.48	0.84	**0.78**	**0.80**	0.27
	Set2Graph	**0.66**	**0.78**	**0.43**	0.86	0.76	**0.65**	0.54	0.88	**0.78**	0.79	**0.33**
c jets	AVR	0.70	0.65	0.22	**0.95**	0.41	0.39	**0.49**	**0.91**	0.49	0.66	0.14
	Track Pair	0.74	0.73	0.40	0.92	0.58	0.52	0.47	0.88	0.65	0.76	0.24
	RNN	0.71	0.72	0.40	0.86	**0.60**	0.50	0.39	0.85	0.65	0.77	0.19
	Set2Graph	**0.75**	**0.75**	**0.45**	0.94	**0.60**	**0.56**	0.47	**0.91**	**0.67**	**0.78**	**0.26**
light jets	AVR	**0.97**	**0.96**	0.93	**0.99**	0.89	0.88	**0.33**	**0.98**	0.73	0.89	0.14
	Track Pair	0.96	**0.96**	0.93	0.97	**0.93**	0.90	0.32	0.97	0.87	**0.95**	**0.26**
	RNN	0.93	0.92	0.87	0.93	0.90	0.84	0.25	0.94	0.82	0.93	0.18
	Set2Graph	**0.97**	**0.96**	**0.94**	0.98	**0.93**	**0.91**	0.32	**0.98**	**0.88**	**0.95**	**0.26**

The result of the highest performing algorithm in each category is marked in bold

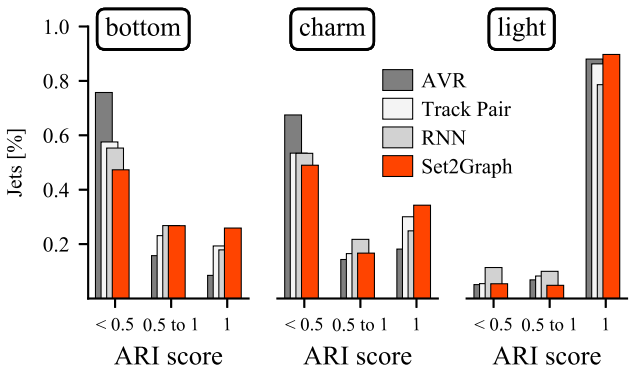

Fig. 8 ARI scores for the different flavors of jets. We consider 3 categories: *Perfect* – jets with an ARI score of exactly 1, *Intermediate* – a score between 0.5 and 1 and *Poor* – scores below 0.5

vertex size (i.e., number of tracks in the vertex). The S2G algorithm maintains an advantage over the full range of vertex sizes. The S2G model has a similar internal accuracy to the baseline, but a 10% increase in external accuracy for smaller vertices.

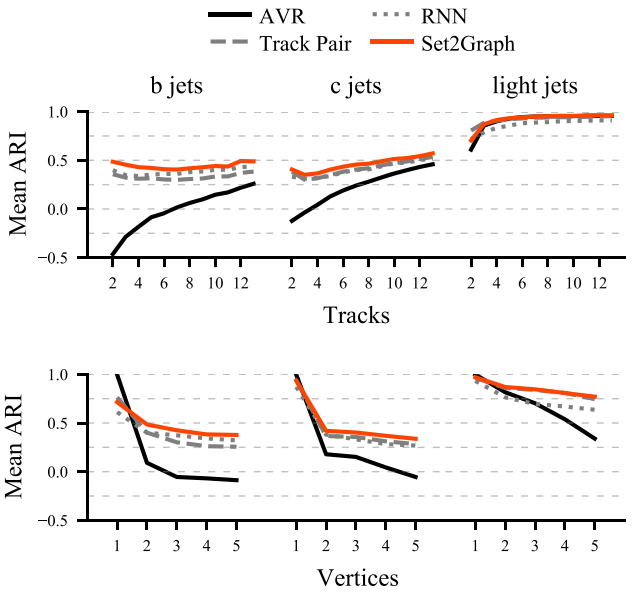

Fig. 9 Mean ARI scores for the different flavors of jets as a function of the jet properties

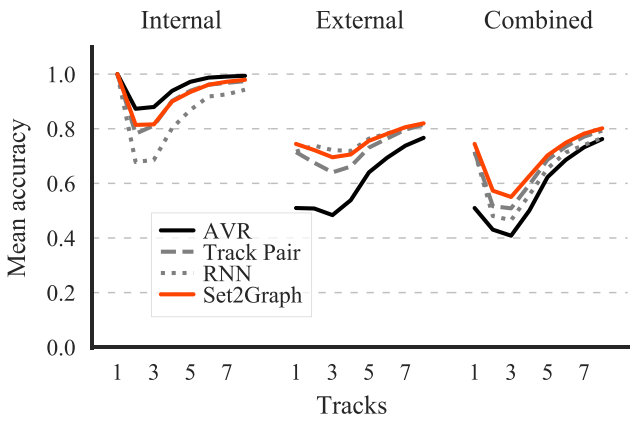

Fig. 10 Vertex performance as a function of the vertex size. Internal, external and combined accuracy are defined in Sect. 4.2

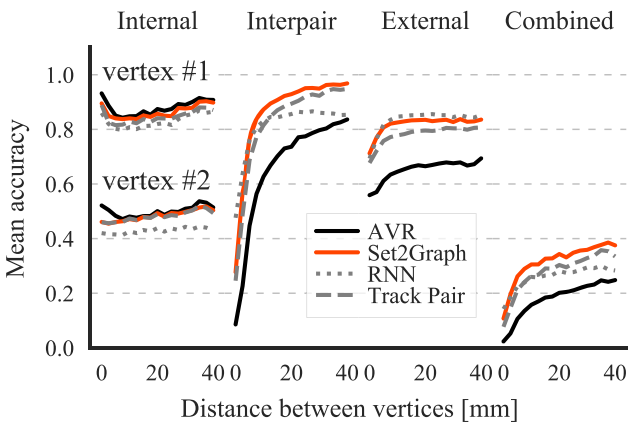

Fig. 11 Vertex pair accuracy as a function of distance between the vertices. The internal accuracy is shown for both smaller vertex (the vertex with fewer tracks, vertex #1) and the larger vertex (vertex #2)

Figure 11 show the performance for vertex pairs, as a function of the distance between the vertices. Again the S2G shows a promising ability to separate vertices even when the distance between them approaches 0. The performance increase of about 10% in combined accuracy comes from the improvement in interpair and external accuracy, i.e., less merging of vertices.

Comparison to neural network baselines Both the TP and RNN algorithms have a lower ARI by about 5–10% compared to the S2G model for b and c jets. S2G also outperforms both baselines in vertex and vertex-pair combined accuracy. From Fig. 8 we can see that S2G has the highest percentage of perfectly reconstructed jets, and Figs. 9, 10 and 11 show that this advantage is maintained across the entire dataset.

Impact on jet classification The results for jet classification are shown in Table 3. The pre-trained S2G classifier outperforms the AVR based classifier by over 10% in terms of overall accuracy with the most significant gain coming from the increased rejection of light jets (an increase in light jet F1 from 40% to 69%). The neural network baseline with an S2G based vertexing module that is trained only towards the classification objective shows better performance than the AVR and track pair based algorithms. This indicates that the network is able to learn some important features of the data by itself. The RNN and S2G based models have similar performance, with the S2G model outperforming the RNN in particular in c jet identification.

7 Conclusions

We proposed training a neural network to perform vertex finding, using supervised learning. We found that it outperforms standard techniques for multiple performance metrics of vertex reconstruction, and shows promising increase in performance for nearby vertices.

We utilized the Set2Graph model, a simple equivariant and universal model of functions from sets to graphs. We showed that the model's universality and equivariance were both important. The universality was needed to properly learn the vertex finding task, by taking into account information from all tracks in the jet. Equivariance was a useful inductive bias, resulting in better performance compared to recurrent neural network which could in theory learn the same function as the S2G model. We evaluated the impact of the improved accuracy in vertex reconstruction on jet classification by training a classifier that used the vertex finding predic-

Table 3 Jet flavor classification performance metrics. The model with pre-trained S2G vertex finding module outperforms the other algorithms in overall

Vertex finding module	Accuracy	F1	b jets F1	c jets F1	Light jets F1
AVR	0.50	0.49	0.62	0.44	0.40
Baseline	0.57	0.56	0.67	0.40	0.60
Track pair	0.56	0.57	0.65	**0.48**	0.57
RNN	0.62	0.60	**0.74**	0.37	**0.69**
Set2Graph	**0.63**	**0.62**	0.72	0.44	**0.69**

The result of the highest performing algorithm in each category is marked in bold

tions as input, as a sort of supervised attention mechanism. We found that improved vertex finding leads to improved classification. The supervised attention mechanism lead to better results compared to an identical model with unsupervised attention. The universal models (S2G and RNN) had the best performance, however the equivariance of S2G gave it a slight advantage over the RNN.

Future work may explore the application of this technique to more complicated decays such as boosted Higgs to (bb/cc), and apply it to more realistic datasets that include full detector simulation and pileup interactions.

Acknowledgements EG and JS are supported by the NSF-BSF Grant 2017600 and the ISF Grant 125756. This research was partially supported by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center. KC is supported by the National Science Foundation under the awards ACI-1450310, OAC-1836650, and OAC-1841471 and by the Moore-Sloan data science environment at NYU. HS, NS and YL were supported in part by the European Research Council (ERC Consolidator Grant, "LiftMatch" 771136), the Israel Science Foundation (Grant no. 1830/17) and by a research grant from the Carolito Stiftung (WAIC).

Data Availability Statement This manuscript has associated data in a data repository. [Authors' comment: The dataset and code used in this paper are available at https://zenodo.org/record/4044628 and https://github.com/jshlomi/SetToGraphPaper. [![DOI](https://zenodo.org/badge/DOI/10.5281/zenodo.4044628.svg)] (https://doi.org/10.5281/zenodo.4044628).]

Funded by SCOAP[3].

Appendix A: Hyperparameter optimization for AVR

The AVR algorithm in RAVE [4] has three main parameters that can be adjusted by the user -

- Primary vertex significance cut
- Secondary vertex significance cut
- minimum weight for a track to stay in a fitted vertex

The values for there parameters were scanned in a grid between 0.1 to 10 for the significance cuts (33 equally spaced values) and between 0.1 to 0.8 for the minimum weight (10 values). For each possible value of the parameters, the mean RI was computed for each of the 3 flavors in the training dataset. The values of the b, c and light jet RI are shown in Fig. 12. The working point that was chosen had the highest b jet RI with a mean light jet RI above 0.95:

- Primary cut: 2.5
- Secondary cut: 2.5
- minimum weight: 0.2.

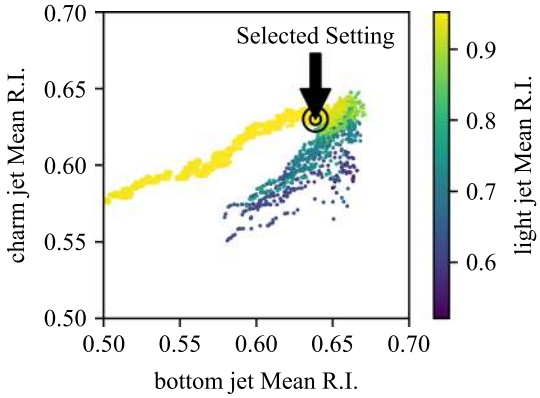

Fig. 12 AVR parameter scan

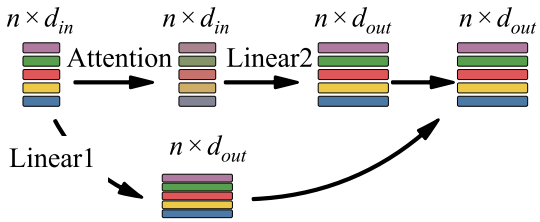

Fig. 13 A single deep set layer in the ϕ module

Appendix B: Model architecture and training details

Hyperparameter tuning and ablation studies The optimization of the model hyperparameters and architecture used in this paper are described in detail in the supplementary material of [16]. Below we describe the architecture for the final optimized model.

S2G model. The ϕ component of the S2G model is composed of a sequence of deep set layers [8], each of which contain a self-attention mechanism and two linear $d_{in} \to d_{out}$ layers, in a structure shown in Fig. 13. A ReLU non-linearity is used between the layers.

The attention block in the deep set layer is a key/query attention [41,42]:

$$\text{Attention}(X) = \text{softmax}\left(\frac{\tanh f_1(X) \cdot f_2(X)^T}{\sqrt{d_{small}}}\right) \cdot X \quad \text{(B.1)}$$

where X is the $n \times d_{in}$ input, f_1, f_2 are the key and query MLPs of width $d_{small} = d_{in}/10$.

If we describe the stack of deep set layers by their output dimension d_{out}, the ϕ module layer dimensions are:

$$\phi \text{ output dimensions} = (256, 256, 256, 256, 5). \quad \text{(B.2)}$$

The edge classifier component ψ takes in the $n \cdot (n-1) \times (5 \cdot 3)$ output of the broadcasting layer, and uses a single hidden layer MLP with output dimensions (256, 1).

Baseline TP Classifier.

The MLP that replaces the deep set layers has the following output sizes:

$$\phi_{\text{TP}} \text{ output dimensions} = (384, 384, 384, 384, 5). \quad \text{(B.3)}$$

The edge classifier component ψ is identical expect its input size is now $5 \cdot 2$ instead of $5 \cdot 3$ due to the absence of the sum in the broadcasting layer.

Baseline RNN

The GRU layer output sizes are:

$$\phi_{\text{RNN}} \text{ output dimensions} = (256, 256, 128, 6). \quad \text{(B.4)}$$

Each GRU layer is bi directional. Each direction results in a hidden representation of size $d_{out}/2$, and the results are concatenated.

Training hyperparameters

We used a batch size of 2048, Adam optimizer [43] with learning rate of 10^{-3}. Training takes place in less than 2 h on a single Tesla V100 GPU. The training is stopped when the validation loss stops does not decrease for 20 epochs.

Appendix C: Jet classification model architecture

The model, illustrated in Fig. 6 is composed of four components:

- Deep set network
- Vertex finding module,
- Graph network [15].
- Jet classifier MLP.

Deep set The deep set network is described in Appendix B. In the classification model it has dimensions of:

$$\text{Deep set output dimensions} = (126, 126, 126, 126). \quad \text{(C.1)}$$

The deep set creates a hidden representation for each track in the input.

Vertex finding module This is either the AVR pre-computed vertex assignment, or one of the vertex finding networks. The output of this module is an edge prediction e_{ij} between any two tracks in the input set.

The graph network creates a hidden representation for the tracks based on the output of the deep set and the vertex finding module, which is treated as edge features for the fully connected graph of tracks.

The graph network is composed of a sequence of GN blocks, each with an edge update and node update MLP.

$$g^t = \sum_i h_i^t \quad \text{(C.2)}$$

$$m_i^{t+1} = \sum_{j \in N(i)} E_t(h_i^t, h_j^t, e_{ij}, g^t) \quad \text{(C.3)}$$

$$h_i^{t+1} = U_t(h_i^t, m_i^{t+1}) \quad \text{(C.4)}$$

where h_i^t is the ith node hidden representation at step t, g^t is the global representation of the graph (sum of all node hidden representations), E_t and U_t are the edge and node update MLPs for layer t of the graph network and e_{ij} is the edge prediction given by the vertex finding module for the edge between node i and j. $N(i)$ is the node neighborhood. In this model the graph is always fully connected, so the node neighborhood contains all the nodes in the graph. The edge update MLP has linear layers with sizes:

$$E_t \text{ dimensions} = (126 \cdot 3 + 1, 100, 20). \quad \text{(C.5)}$$

The node update MLP has linear layers with sizes:

$$U_t \text{ dimensions} = (126 + 20, 100, 126). \quad \text{(C.6)}$$

The graph network has 3 such GN blocks.

The jet classifier MLP takes as input the sum of track hidden representations and the jet features (p_T, η, ϕ, jet mass). It predicts if the jet is a b,c or light jet.

$$\text{Jet classifier dimensions} = (126 + 4, 100, 50, 3). \quad \text{(C.7)}$$

C.1 Jet classifier training

The model is trained with a batch size of 1000, Adam optimizer and a learning rate of 5×10^{-4}, and cross entropy loss. Training takes less than 2 h on single Tesla V100 GPU. The training is stopped when the validation loss stops does not decrease for 20 epochs.

References

1. D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban, D. Whiteson, Jet flavor classification in high-energy physics with deep neural networks. Phys. Rev. D **94**(11), 112002 (2016)

2. A. Strandlie, R. Fruhwirth, Track and vertex reconstruction: from classical to adaptive methods. Rev. Mod. Phys. **82**, 1419–1458 (2010)
3. G. Piacquadio, C. Weiser, A new inclusive secondary vertex algorithm for b-jet tagging in ATLAS. J. Phys. Conf. Ser. **119**(3), 032032 (2008)
4. W. Waltenberger, RAVE: a detector-independent toolkit to reconstruct vertices. IEEE Trans. Nucl. Sci. **58**, 434–444 (2011)
5. W. Waltenberger, Adaptive vertex reconstruction. Technical Report CMS-NOTE-2008-033, CERN, Geneva, Jul 2008
6. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
7. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
8. M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R.R. Salakhutdinov, A.J. Smola, Deep sets, in *Advances in Neural Information Processing Systems* (2017), pp. 3391–3401
9. C.R. Qi, H. Su, K. Mo, L.J. Guibas, Pointnet: deep learning on point sets for 3d classification and segmentation, in *Proceedings of the Computer Vision and Pattern Recognition (CVPR)*, vol. 1(2). IEEE (2017), p. 4
10. H. Maron, O. Litany, G. Chechik, E. Fetaya, On learning sets of symmetric elements (2020). arXiv preprint. arXiv:2002.08599
11. J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs (2013), pp. 1–14
12. T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks (2016). arXiv preprint. arXiv:1609.02907
13. J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, in *International Conference on Machine Learning* (2017), pp. 1263–1272
14. H. Maron, H. Ben-Hamu, N. Shamir, Y. Lipman, Invariant and equivariant graph networks (2018). arXiv preprint. arXiv:1812.09902
15. P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi et al. Relational inductive biases, deep learning, and graph networks (2018). arXiv preprint. arXiv:1806.01261
16. H. Serviansky, N. Segol, J. Shlomi, K. Cranmer, E. Gross, H. Maron, Y. Lipman, Set2Graph: learning graphs from sets (2020). arXiv preprint. arXiv:2002.08772
17. J. Shlomi, P. Battaglia, J.-R. Vlimant, Graph neural networks in particle physics, in *Machine Learning: Science and Technology* (2020)
18. S. Farrell et al. Novel deep learning methods for track reconstruction, in *4th International Workshop Connecting the Dots 2018 (CTD2018) Seattle, Washington, USA, March 20–22, 2018* (2018)
19. X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, L. Gray, T. Klijnsma, K. Pedro, G. Cerati, J. Kowalkowski, G. Perdue, P. Spentzouris, N. Tran, J.-R. Vlimant, A. Zlokapa, J. Pata, M. Spiropulu, S. An, A. Aurisano, J. Hewes, A. Tsaris, K. Terao, T. Usher, Graph neural networks for particle reconstruction in high energy physics detectors (2020). arXiv preprint. arXiv:2003.11603
20. J. Kieseler, Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph, and image data. Eur. Phys. J. C **80**(9) (2020)
21. F. Drielsma, Q. Lin, P. Côte de Soux, L. Dominé, R. Itay, D.H. Koh, B.J. Nelson, K. Terao, K.V. Tsang, T.L. Usher, Clustering of electromagnetic showers and particle interactions with graph neural networks in liquid argon time projection chambers data (2020). arXiv preprint. arXiv:2007.01335
22. F.A. Di Bello, S. Ganguly, E. Gross, M. Kado, M. Pitt, L. Santi, J. Shlomi, Towards a computer vision particle flow. Eur. Phys. J. C **81**(2) (2021)
23. J. Pata, J. Duarte, J.-R. Vlimant, M. Pierini, M. Spiropulu, MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks. Eur. Phys. J. C **81**(5) (2021)
24. E.A. Moreno, O. Cerri, J.M. Duarte, H.B. Newman, T.Q. Nguyen, A. Periwal, M. Pierini, A. Serikova, M. Spiropulu, J.-R. Vlimant, JEDI-net: a jet identification algorithm based on interaction networks. Eur. Phys. J. C **80**(1), 58 (2020)
25. Q. Huilin, L. Gouskos, Jet tagging via particle clouds. Phys. Rev. D **101**(5) (2020)
26. J. Bruna, K. Cho, K. Cranmer, G. Louppe, I. Henrion, J. Brehmer et al., Neural message passing for jet physics, in *Deep Learning for Physical Sciences Workshop at the 31st Conference on Neural Information Processing Systems (NIPS)* (2017)
27. P.T. Komiske, E.M. Metodiev, J. Thaler, Energy flow networks: deep sets for particle jets. J. High Energy Phys. **2019**(1) (2019)
28. E.A. Moreno, T.Q. Nguyen, J.-R. Vlimant, O. Cerri, H.B. Newman, A. Periwal, M. Spiropulu, J.M. Duarte, M. Pierini, Interaction networks for the identification of boosted $h \to b\bar{b}$ decays. Phys. Rev. D **102**(1) (2020)
29. E. Bernreuther, T. Finke, F. Kahlhoefer, M. Krämer, A. Mück, Casting a graph net to catch dark showers (2020). arXiv preprint. arXiv:2006.08639
30. V. Mikuni, F. Canelli, ABCNet: an attention-based method for particle tagging. Eur. Phys. J. Plus **135**(6) (2020)
31. J. Guo, J. Li, T. Li, The boosted Higgs jet reconstruction via graph neural network (2020). arXiv preprint. arXiv:2010.05464
32. T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, An introduction to Pythia 8.2. Comput. Phys. Commun. **191**, 159–177 (2015)
33. J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, M. Selvaggi, Delphes 3: a modular framework for fast simulation of a generic collider experiment. J. High Energy Phys. **2014**(2) (2014)
34. G. Aad et al., The ATLAS experiment at the CERN large hadron collider. J. Instrum. **3**(08), S08003–S08003 (2008)
35. M. Cacciari, G.P. Salam, G. Soyez, The anti-k_t jet clustering algorithm. JHEP **04**, 063 (2008)
36. V. Sovrasov, Flops counter for convolutional networks in pytorch framework (2019)
37. N. Segol, Y. Lipman, On universal equivariant set networks (2019). arXiv preprint. arXiv:1910.02421
38. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014). arXiv preprint. arXiv:1406.1078
39. L. Hubert, P. Arabie, Comparing partitions. J. Classif. **2**(1), 193–218 (1985)
40. A. Gates, Y.-Y. Ahn, The impact of random models on clustering similarity. J. Mach. Learn. Res. **18**, 01 (2017)
41. M. Ilse, J.M. Tomczak, M. Welling, Attention-based deep multiple instance learning (2018). arXiv preprint. arXiv:1802.04712
42. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in *Advances in Neural Information Processing Systems 30* (2017), pp. 5998–6008
43. D.P Kingma, J. Ba, Adam: a method for stochastic optimization (2014). arXiv preprint. arXiv:1412.6980