PAPER • **OPEN ACCESS**

Learning tree structures from leaves for particle decay reconstruction

To cite this article: James Kahn *et al* 2022 *Mach. Learn.: Sci. Technol.* **3** 035012

IOP Publishing Mach. Learn.: Sci. Technol. 3 (2022) 035012 https://doi.org/10.1088/2632-2153/ac8de0

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
31 May 2022

REVISED
12 August 2022

ACCEPTED FOR PUBLICATION
30 August 2022

PUBLISHED
14 September 2022

PAPER

Learning tree structures from leaves for particle decay reconstruction

James Kahn[1,2,*], **Ilias Tsaklidis**[3,*], **Oskar Taubert**[1,2], **Lea Reuter**[4], **Giulio Dujany**[5], **Tobias Boeckh**[3], **Arthur Thaller**[6], **Pablo Goldenzweig**[4], **Florian Bernlochner**[3], **Achim Streit**[2] and **Markus Götz**[1,2]

[1] Helmholtz AI, Germany
[2] Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[3] Physikalisches Institut der Rheinischen Friedrich-Wilhelms-Universität Bonn, Bonn, Germany
[4] Insitute for Experimental Particle Physics (ETP), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[5] Université de Strasbourg, CNRS, IPHC, UMR 7178, 67037 Strasbourg, France
[6] Aix Marseille Université, CNRS, IN2P3, CPPM, 13288 Marseille, France
[*] Author to whom any correspondence should be addressed.

E-mail: james.kahn@kit.edu **and** itsaklid@uni-bonn.de

Keywords: particle physics, tree reconstruction, lowest common ancestor generation, graph neural networks, self-attention neural networks, transformer

Supplementary material for this article is available online

Abstract

In this work, we present a neural approach to reconstructing rooted tree graphs describing hierarchical interactions, using a novel representation we term the lowest common ancestor generations (LCAG) matrix. This compact formulation is equivalent to the adjacency matrix, but enables learning a tree's structure from its leaves alone without the prior assumptions required if using the adjacency matrix directly. Employing the LCAG therefore enables the first end-to-end trainable solution which learns the hierarchical structure of varying tree sizes directly, using only the terminal tree leaves to do so. In the case of high-energy particle physics, a particle decay forms a hierarchical tree structure of which only the final products can be observed experimentally, and the large combinatorial space of possible trees makes an analytic solution intractable. We demonstrate the use of the LCAG as a target in the task of predicting simulated particle physics decay structures using both a Transformer encoder and a neural relational inference encoder graph neural network. With this approach, we are able to correctly predict the LCAG purely from leaf features for a maximum tree-depth of 8 in 92.5% of cases for trees up to 6 leaves (including) and 59.7% for trees up to 10 in our simulated dataset.

1. Introduction

Can the structure of a tree graph be determined from its leaves alone? If all the information necessary to construct each parent node is contained in its children, then intuitively the answer is yes. This is often the scenario we are presented with in real world situations, where a tree represents the chronological interactions of objects. Such structures arise, for example, in semantic tree parsing, where the logic of a sequence like a sentence or mathematical equation is represented as a semantic tree; citation relevance hierarchies, where related publications are represented by a citation connection tree; or the decay of sub-atomic particles, where the topology of the decay process is determined by physical laws. In these situations, we are limited to observing the properties of the leaves of such trees and inferring details about its structure from the leaf properties alone.

Well understood composition laws can allow us to reconstruct tree relations by attempting all reasonable combinations of children and verifying the physical validity of parents formed. In situations involving a large number of tree topologies, however, the combinatorial complexity of such solutions makes them infeasible. The ability to efficiently learn the relational structure of a tree from its leaves alone provides a means of

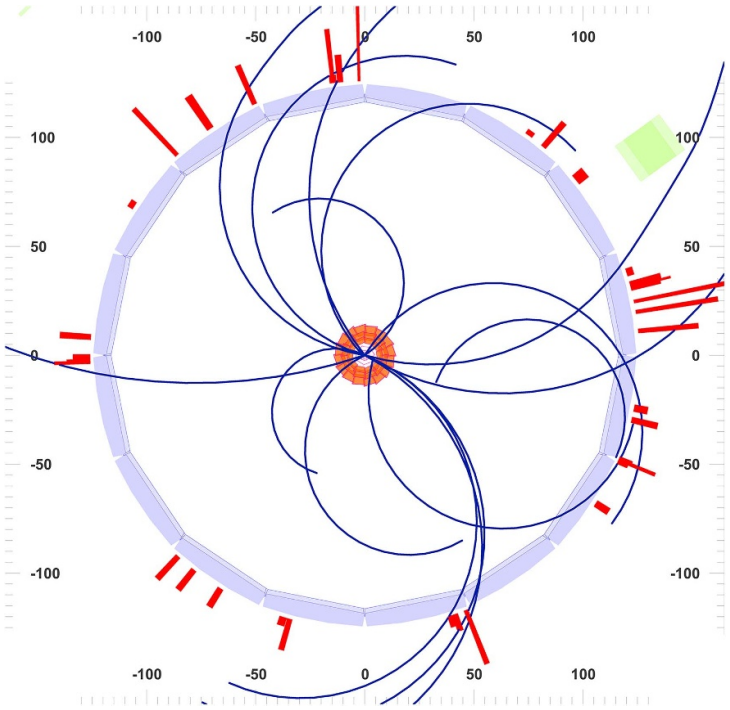


Figure 1. Example cross-sectional view (beam direction of the detector not shown) of a particle decay from the Belle II experiment [4]. Blue lines show the path of charged particles through the detector. The red and blue rings are detector components, and red and green towers represent energy deposits from detected particles.

reducing this complexity to a tractable level, where domain-specific knowledge can then be applied to fill in the constituent properties.

Applying deep learning to the task of tree reconstruction requires an effective representation of the structural information. This representation is essential to providing a tangible learning target. Furthermore, we are often presented with scenarios in which there is no prior knowledge about the depth of the tree or the degrees of nodes within. For this, we propose the lowest common ancestor generation (LCAG) matrix, a novel tree representation which encodes a tree's structure in a compact representation suitable for use as a training target. We demonstrate the use of the LCAG in learning how to reconstruct tree structures purely from leaves in one of the aforementioned examples: sub-atomic particle decays.

Particle colliders, such as the large Hadron collider (LHC) [1] and the Belle II experiment [2], study the fundamental laws of nature by accelerating and colliding particles at close to the speed of light. The collisions that occur produce heavy subatomic particles which rapidly decay into lighter particles travelling away from the collision point. Subsequent decays may follow, until lighter, stable particles live long enough to reach a detection device. In figure 1, an example of the trajectories of charged particles originating from a collision in the Belle II experiment is shown. This chronological structure can be naturally represented as a graph, or more specifically, as a rooted tree. The terminal leaves of the tree consist of the particles that reach the detector equipment, the intermediate nodes their ancestors, and the edges of the graph their parent-child relations. The topology and constituents of the tree are dictated by physical laws, for example conservation of energy and momentum. In order to learn the interactions of this particle decay tree, the employed model must infer these physical laws, and hence predict the structure of the tree.

The contributions of this work are:

- We introduce the LCAG matrix, a compact representation of rooted tree graph structures suitable for machine learning-based reconstruction of tree structures from leaves.
- We present a modification of neural relational inference (NRI) for graph learning, tailored to predicting entire graph structures in a supervised setting.
- We use the LCAG as a training target for the neural reconstruction of simulated particle physics decays. In doing so, we demonstrate how the LCAG enables the first end-to-end trainable solution which learns the hierarchical structure of varying tree sizes directly, using only the terminal tree leaves to do so.
- We open-source the PyTorch implementation for this work, experiments, and trained models used[7], as well as the data [3].

[7] https://github.com/Helmholtz-AI-Energy/BaumBauen.

2. Related work

Learning and exploiting the hierarchy of graphs has drawn attention in a variety of machine learning domains in recent years. Works in the field of neural machine translation and semantic parsing [5, 6], or citation count prediction and review models [7], learn the hierarchy of rooted tree structures. Graph pooling [8] was introduced to create hierarchical representations of graph neural networks (GNNs) themselves to boost performance on graph classification tasks. However, these all address the problem of predicting hierarchical structures of known and fixed depth. In [9], the authors utilized message passing functions to create hierarchies within a graph for community detection, but the case of a predefined tree structure remains.

The field of relational inference concerns the task of learning the interaction dynamics within a system from observational data. Work in the field includes the NRI model [10], and the hierarchical relational inference [11]. Both works adopt an unsupervised approach to infer the relational structure of graphs, with the former operating on graphs representing dynamical systems, and the latter extending it to images of the systems. Other supervised approaches take the form of edge labelling tasks or edge weight prediction, and aim at binary edge classification [12], node clustering [13, 14], or graph classification [15]. These supervised methods mainly focus on pairs of edges without dealing with global hierarchies inside the graphs.

For the special case that a graph is fully connected, i.e. all nodes connected to all others, an approach that relates all inputs to all others is appropriate. The self-attention mechanism employed by the Transformer neural network [16] operates as a GNN for such a case, modelling the pair-wise interactions between inputs.

In the field of particle physics, machine learning has begun to play a prominent role [17–19], and more recently GNNs have received a lot of attention [20, 21].

GNNs have been investigated for improving direct particle reconstruction methods[8]. A GNN-based implementation of the Particle Flow algorithm [22, 23], a key piece of software at the LHC which combines information from multiple detector hardware components to identify individual particles, is now being explored as an end-to-end trainable alternative to the existing rule-based approach [24, 25]. The HEP.TrkX [26], and subsequent Exa.TrkX [27] have demonstrated the power of GNNs in the task of tracking individual charged particles at the LHC. Ju *et al* [28] further reinforced the ability of GNNs to learn the kinematics of particles based on detected information alone. These works focus on individual particle tracks through detector equipment only, and do not deal with the decay tree structures of the particles themselves.

Jet vertex finding has been of particular interest to the LHC experiments and involves the labelling of detected particles according to their vertex in hadronic particle decays (jets). Henrion *et al* [29] used message-passing neural networks to learn the adjacency matrices for jet decays directly. Shlomi *et al* [30] applied a range of GNNs to what was effectively an edge classification task, and used the predicted binary edge labels to identify siblings in the jet decay tree structure. However, these approaches presuppose the graph structure of the jets being reconstructed, for example as containing a primary and multiple secondary vertices. JEDI-net [31] attempts to identify the topology of a given jet, but only goes as far as classifying which topology, and not assigning specific particles vertices within. ParticleNet [32] applies a Dynamic Graph Convolutional Neural Network [33] to identify the root particle of jets, treating the detected particles as a point cloud. The transformer-based ParT model [34] explored using the set of individual particle kinematic and interaction information directly for the same task. The above works recognize the importance of a meaningful input data representation, namely, treating detected particles as an unordered set, but fail to leverage a similarly meaningful representation for the labels as well. However, they all again reinforce that GNNs are an appropriate choice for learning the complex dynamics of particle decays.

The Full Event Interpretation (FEI) [35] software used by the Belle II collaboration attempts to reconstruct entire particle decay trees using a sequence of individually trained Boosted Decision Trees (BDTs). However, the specific decay processes targeted by the BDTs are hard-coded. This both limits the total possible decay processes that can be reconstructed by the FEI, and requires domain-driven optimisation in every step of the tree reconstruction. Furthermore, as the BDTs are trained independently from one another, this approach is prone to error accumulation, unlike a single, end-to-end trainable solution.

Our work attempts to resolve the constraints of the aforementioned works by introducing a new, compact representation of hierarchical trees that can be used as a single target label. The goal being to no longer require assumptions about the structure, i.e. the depth or number of vertices, and leverage the power of a fully end-to-end trainable approach. We do so in the context of particle decays, using GNNs inspired by previous works.

[8] Particle reconstruction is the task of using raw information about the energy deposited in the detector devices to obtain the physical properties of a particle candidate.

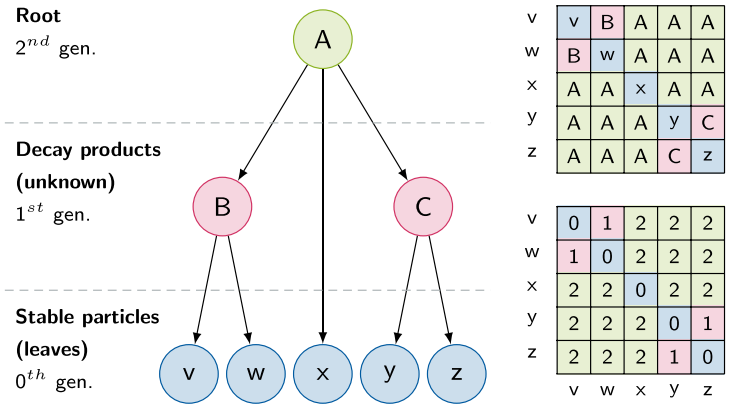


Figure 2. Example of a hierarchical tree representing a particle decay (left), its corresponding LCA (top right), and LCAG matrix (bottom right). Colours are added for visual clarity only, and node labels are simply examples to uniquely identify nodes and do not correspond to real particle names.

3. Learning tree structures from leaves

A graph is a data structure that can be used to model a set of interdependent entities (nodes) and their relationships (edges). More formally, adopting the notation of [36], given a set of objects and their relations, a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of v_i nodes and $\mathcal{E}$ the set of $e_{ij} = (v_i, v_j)$ edges, expresses the relational structure of the system. The adjacency matrix $\mathbf{A}$ of a graph of n nodes is an $n \times n$ binary matrix, with $A_{ij} = 1$ if $e_{ij} \in \mathcal{E}$ and $A_{ij} = 0$ if $e_{ij} \notin \mathcal{E}$. Nodes are described by the feature vector $\mathbf{x}_i \in \mathbf{R}^d$, given d features of node v_i, and edges by the feature vector $\mathbf{x}_{(i,j)} \in \mathbf{R}^c$ for c edge features between nodes v_i and v_j. Rooted trees are a special case of graphs that are acyclic and have one node designated as the root.

We can therefore abstract the hierarchical structure problem described in section 1 using rooted trees as follows: given a set of k leaf node, represented by a corresponding set of vertices $\{v_i | 1 \leqslant i \leqslant k\}$ and features $\{x_i\}$, find the missing intermediate nodes and edges in the form of an adjacency matrix A. If we are to learn A, we must use a solution representation that satisfies the following: first, since there is in general no implicit order to the leaf nodes in graph-based problems, the solution representation must be permutation invariant (or at least permutation aware); second, as the total number of nodes in a given tree is initially unknown, we also require a data representation for the solution whose size depends only on the number of leaves. The adjacency matrix itself only satisfies the first condition, but requires the total number of nodes in $\mathcal{G}$ to be known *a priori* in order to define the correct number of rows and columns. In the following section we present a compact representation, equivalent to the adjacency matrix for the case of hierarchical trees, that satisfies these requirements with minimal constraints.

In this work we limit ourselves strictly to learning the hierarchical tree structure. We make the explicit assumption that once the structure is predicted, either non-leaf-node labels can be inferred in a later step or are not of interest. Our goal is to provide a solution for cases in which the combinatorial explosion in finding the tree structure is a major limiting factor preventing simply attempting all feasible combinations.

3.1. Lowest common ancestor generation (LCAG) tree representation

To satisfy the problem requirements outlined above, we employ a modified representation of the lowest common ancestor (LCA) matrix representation of rooted trees [37]. Given two nodes in a graph, a and b, the LCA is defined as the deepest node that is an ancestor of both a and b, deepest in this case meaning farthest graph distance, dist, from the root node r, i.e.:

$$\mathrm{LCA}(a, b) = \mathrm{argmax}_p \{\mathrm{dist}(p, r) | p \in \mathrm{path}(a, r) \wedge p \in \mathrm{path}(b, r)\}.$$

Figure 2 shows an example tree representing an arbitrary particle decay process (left) and its corresponding LCA matrix (top right), with generations up from the leaves labelled on the far left. Letters indicate arbitrary node labels, and the colours are simply added for clarity and do not hold further meaning[9]. In order for this representation to describe a graph unambiguously, it must be a tree where any non-leaf node has *two or more* children. This way the set of nodes that are elements of the LCA matrix is identical to $\mathcal{V}$. This

[9] We note here that the node labels in an LCA are unique identifiers of each node, and do not correspond, in the physics example, to the particle types.

assumption bounds the maximum depth of the tree to $\mathcal{O}(\log_2 k)$ for k leaves. Note that each entry in the LCA matrix encodes information about an unordered pair of vertices, and as such is symmetric.

In order to express the LCA matrix in a form appropriate for machine learning, we propose the LCAG matrix, in which each entry of the LCA matrix is replaced with its corresponding generation in the tree. Figure 2(bottom right) shows the LCAG corresponding to the example tree. We see here a clear distinction between the LCA and LCAG: while the LCA requires unique node labels to identify ancestors, the LCAG is a relaxed form which allows the same identifier (generation) to represent multiple ancestors. In other words, the LCA is surjective to the LCAG. The underlying assumption for making this modification is that if there exist structural rules which allow the inference of the tree structure from the leaf nodes alone, then these rules can also be used on the inferred structure to deduce unseen node labels and hence recover the LCA. Relating this to the example particle decay process in figure 2, if we know that two particles v and w come from the same parent, summing these energies would tell us the mass, and hence the node label of the parent B.

For the LCAG matrix, we adopt a pull-down convention: every node in the tree resides at the lowest possible generation, with all leaves fixed at the zeroth generation. Formally:

$$\mathrm{LCAG}_{ij}(\mathcal{G}) = g(\mathrm{LCA}_{ij}(\mathcal{G})),$$
$$g(u) = \begin{cases} 0, & \text{if u is a leaf} \\ \max\limits_{v \in \mathcal{C}(u)} \{g(v)\} + 1, & \text{otherwise} \end{cases}$$

where $\mathcal{C}(u)$ is the set of children of node u.

Lemma 1. *The LCAG matrix uniquely describes any rooted tree isomorphism class for trees in which every non-leaf vertex has two or more children. I.e. if two trees $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are described by the LCAG L, then G and H are isomorphic ($G \simeq H$).*

We prove this by complete induction. Lemma 1 is trivially true for trees consisting of a single root node, described by the LCAG [0]. We will now show that it also holds for any tree of height $m+1$ if it holds for trees of height m. Consider two trees G and H of height $m+1$, that are described by the LCAG L. Given any subtree G_i of G that includes all descendants of root(G_i) in $V(G)$, G_i is described by L_i containing all entries of L corresponding to the leaves of G_i. The L_i of all subtrees whose roots are siblings in G and are direct descendants of $root(G)$ must be disjoint in L, since their LCA describes the parent of the roots of all G_i in G. The remaining entries of L not in any L_i are all by definition $m+1$. The same set of L_i correspond analogously to subtrees of H. Therefore, for each G_i we can find a corresponding subtree H_i that is described the same L_i in such a way that $\dot{\bigcup}_i V(G_i) \cup \mathrm{root}(G) = V(G)$ and $\dot{\bigcup}_i V(H_i) \cup \mathrm{root}(H) = V(H)$. Because of lemma 1, G_i and H_i are isomorphic, as their height is m and there is an isomorphism $f_i : V(G_i) \to V(H_i)$. We can find an isomorphism $f : V(G) \to V(H)$ by $f(\mathrm{root}(G)) = \mathrm{root}(H)$ and $f(u) = f_i(u)$ for $u \in V(G_i)$. Therefore, lemma 1 also holds for trees of height $m+1$.

3.2. Converting LCAG from and to adjacency matrix

In order to generate the adjacency matrix of a tree given its LCAG matrix, we proceed level by level, starting from the leaves. We generate a list of leaves from the dimensions of the input LCAG. We then compute the vertices in the next level by collecting all sets of siblings in the list, i.e. vertices whose LCA is in the next level. For each set of siblings, we remove them from the list and add their parent vertex to the list, keeping track of the LCA of the newly added vertices and all other vertices in the list. Algorithm 1 shows the pseudocode of the conversion algorithm.

To generate the LCAG from an adjacency matrix, we first find the set of leaves, and then fill the values of the matrix with the longest graph distance to the LCA of the corresponding pair of leaf vertices.

4. Experimental setup and results

In this work, we demonstrate the method of learning the LCAG as a training target using a GNN to predict a collection of simulated particle decays. To the best of our knowledge this is the first end-to-end trainable solution which learns the hierarchical structure of varying tree sizes directly, using only the terminal tree leaves to do so.

4.1. Data

Both the LHC and Belle II simulated collision data are restricted to internal member access only, and existing, publicly accessible benchmarks such as the TrackML challenge [38] and the PD4ML [39] focus only on identifying individual particles or whole-graph classification tasks, not any form of decay tree reconstruction. Therefore, we simulate our own experimental dataset using the Phasespace library [40]. This

Algorithm 1. Pseudocode for the conversion of a LCAG gram matrices L into a tree T, error handling for ill-formatted matrices omitted. Indices and ranges are zero-indexed.

Input: Symmetric LCA(G) gram matrix L $(n \times n)$
Output: Root of tree T described by L
$leaves \leftarrow [\,]$
$levels \leftarrow sorted(unique(L))$
$total\_nodes \leftarrow n$
for all n **do** $leaves.append(Node(level = 0))$
{level by level}
for $level$ **in** $[0, \ldots, levels.length]$ **do**
  for $column$ **in** $[0, \ldots, n]$ **do**
    for row **in** $[column + 1, \ldots, n]$ **do**
      if $L[row, column] \neq level$ **then** continue
      $node\_a \leftarrow leaves[row]$
      $node\_b \leftarrow leaves[column]$
      $root\_a \leftarrow get\_root(node\_a)$
      $root\_b \leftarrow get\_root(node\_b)$
      {same level, new node}
      if $root\_a.level < level + 1$ **and** $root\_b.level < level + 1$ **then**
        $parent \leftarrow Node(level = level + 1)$
        $parent.children.append([node\_a, node\_b])$
        $total\_nodes \leftarrow total\_nodes + 1$ root_b is older
      else if $root\_a.level < level + 1$ **and** $root\_b == level + 1$ **then**
        $root\_a.children.append(root\_b)$ vice versa
      else
        $root\_b.children.append(root\_a)$
      end if
    end for
  end for
end for
$root \leftarrow get\_root(leaves[0])$

library takes as input a tree structure in the form of parent-daughter relations, as well as the mass of each participating particle (node). Decays are then simulated according to Monte Carlo phase space [41], which obey the usual physical laws (conservation of energy, momentum, etc). The output is the four-momentum (energy and 3D momentum) of each participating particle, of which we use the leaf nodes for input to the network. In a high-energy particle physics experiment, these leaf nodes correspond to particles detected by the experimental hardware, which a physicist would then use to reconstruct the originating decay. We refer to different tree structures as topologies, and we refer to individual simulated trees (of any topology) as samples.

The simulated datasets used by particle physics experiments for building similar types of reconstruction algorithms include only known topologies. For the type of usage of such algorithms, it is intentional that only those topologies that are well understood are reconstructed to avoid discrepancies between simulation (i.e. training) and measured data (i.e. inference). Therefore, it is reasonable to expect all possible topologies intended for reconstruction be present in the training dataset, with no generalisation to new, unseen topologies.

We simulate a synthetic particle decay dataset for our experiments, consisting of topologies with a common root particle of mass 100 (arbitrary units). Intermediate particles are selected at random with replacement from the following masses: $[90, 80, 70, 50, 25, 20, 10]$. Final state particles, which make up the leaf nodes of generated topologies, are drawn with replacement from the following masses: $[1, 2, 3, 5, 12]$. For each intermediate particle (including the root), we limit the minimum number of children to two, and the maximum five. These masses and ranges of child particles are chosen to correspond to the relative mass scales and decay multiplicities observed in nature [42]. For particle physics decay processes seen at the Belle II experiment, for example, the number of leaves is typically around ten or fewer, with existing decay reconstruction algorithms reflecting this[10].

[10] We discuss the case at the LHC, which produces significantly more leaves per collision event, later in section 4.6.

Tree topology creation is then as follows: starting from the root particle a set of children are selected from the available intermediate and final state particles such that the sum of their masses totals less than the root, this process is then repeated for each child particle which is not a final state particle and so on until only final state particles remain.

The synthetic dataset consists of 200 topologies in total, with 2000 samples per topology for each of training, validation, and testing. Figure 5(top) shows the distribution of simulated topologies, grouped according to the number of leaves and the total depth of the tree, which we use as a surrogate for topology complexity. All leaf node features are normalized to a normal distribution of mean zero, standard deviation one. We do not enforce any ordering of the nodes and leave them unsorted as created in the dataset.

4.2. Models

For the choice of neural network models, we require architectures that operate on unordered sets of vertices. The architectures must be able to learn the inter-dependencies between vertices and model the underlying dynamics of the physical system. We therefore select the encoder components of both the Transformer and NRI models. All model implementations were made in *PyTorch v1.8.1* [43] and trained using *PyTorch-Ignite v0.4.4* [44].

4.2.1. Transformer

Transformer models have dominated natural language processing in recent years due to their ability to model complex, pair-wise interactions. In the following experiments we employ the encoder component of the transformer, followed by an outer-concatenation operation[11] to project from node representations into pair-wise edge representations. This is then fed through a fully-connected output head to produce the LCAG prediction. We can interpret the LCAG as being a weighted adjacency matrix describing a graph of the leaves alone, where the integer weights represent the generation of the LCA each pair of leaves. Therefore we can apply the Transformer directly to the task of learning the LCAG.

4.2.2. NRI

NRI was originally proposed by [10] as an unsupervised GNN in the form of a variational autoencoder. In our experiments we use only the encoder component of the NRI, as we are performing supervised training. The building blocks of an NRI layer consist of node-to-edge and edge-to-node message passing modules, followed by a fully-connected module each. The latent representation then alternates between node-wise and edge-wise as it propagates through the network. The encoder produces a discrete categorical distribution of the edge label predictions, which in our case are the entries of the LCAG. Each LCAG entry prediction is treated as an individual classification task with a softmax followed by an argmax function after the last layer. The individual losses are averaged to produce the total loss.

Figure 3 shows the complete NRI encoder architecture used in experiments. To update the state embeddings between edge-to-node and node-to-edge transition layers, the NRI employs sequences of multilayer perceptrons (MLPs) containing two linear layers with ELU activations [45] and batch normalisation [46]. We extend the original NRI implementation to include a configurable number of additional MLPs both within the NRI blocks and at the beginning and end of the model to allow for added learning capacity.

The *neural message-passing mechanism* in the NRI, used in the edge-to-node and node-to-edge layers, updates state embeddings by aggregating information across all connected edges in the graph (we adopt the notation of [10]):

$$\text{node-to-edge} : \mathbf{h}^{l}_{(i,j)} = f^{l}_{e}\left(\left[\mathbf{h}^{l}_{i}, \mathbf{h}^{l}_{j}\right]\right)$$

$$\text{edge-to-node} : \mathbf{h}^{l+1}_{i} = f^{l}_{v}\left(\sum_{j} \mathbf{h}^{l}_{(i,j)}\right),$$

where $\mathbf{h}^{l}_{i}$ represents the state embedding of node i at layer l, and $\mathbf{h}^{l}_{(i,j)}$ the embedding of the edge connecting nodes i and j. f_e and f_v are feed-forward neural networks used to produce the edge and node embeddings. In contrast to [10], we allow for self-interactions $\left(\mathbf{h}^{l}_{(i,i)}\right)$ in our experiments as we found this empirically to perform better.

[11] An outer-concatenation involves duplicating the leaves along a new axis to produce a $(k \times d) \rightarrow (k \times k \times d)$ tensor and concatenating this with the transpose of itself. This creates a $(k \times k \times 2d)$ dimension tensor of node pairs.

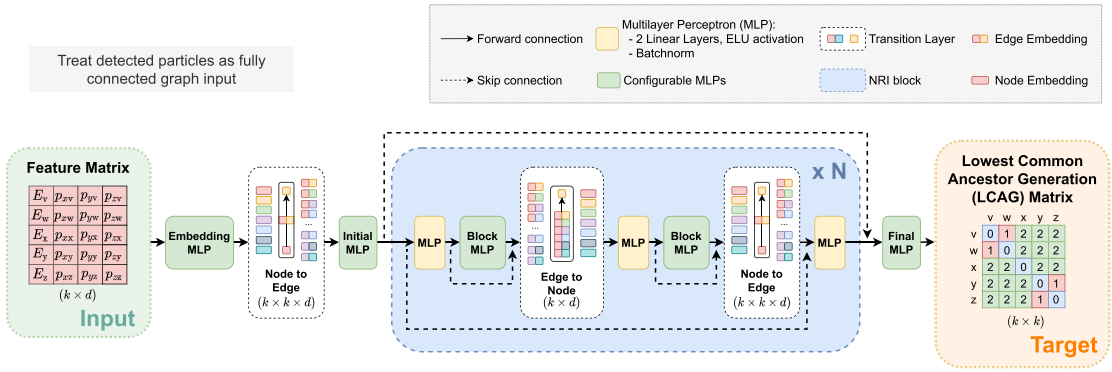


Figure 3. NRI encoder model architecture. The *node-to-edge* and *edge-to-node* layers shown are from [10], tensor shapes are shown only when dimension changes. The input feature matrix contains the four-momentum (E, p_x, p_y, p_z) of each leaf (v, w, ...).

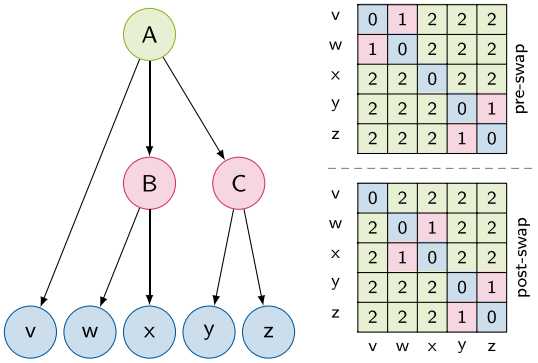


Figure 4. Exchanging the parent of two leaves from the tree in figure 2 modifies two entries (in the upper/lower triangle) in the its LCAG matrix.

4.3. Evaluation metrics

Similar to (non-ordinal) classification tasks, a predicted LCAG has no clear concept of an *almost correct* tree, for example in the case when only one entry prediction is incorrect. To see why this is the case, we can look at an example of what we would intuitively identify as two similar trees, i.e. almost isomorphic, and see that they are not related by valid LCAG representations. Consider the simple operation of exchanging the parent of the nodes x and v in the example from figure 2. Figure 4 demonstrates how this alters two LCAG entries in the process (considering only the upper or lower triangle due to symmetry). Changing only one of the LCAG entries would result in what we consider an *invalid* LCAG: one that does not represent a coherent tree structure. We therefore introduce the metric perfectLCAG to evaluate the rate of correct LCAG predictions, which is the ratio of correctly predicted LCAGs to the total number of samples.

4.4. Experiments

We optimized and evaluated both the Transformer and the NRI encoder on the simulated dataset described above (cf section 4.1). To find the optimal hyperparameters for both models, we used the *Optuna v2.10* [47] library with a Tree-structured Parzen Estimator [48] optimisation approach. The objective value maximized is the perfectLCAG score on validation samples. All trainings were performed on a single NVIDIA A100 GPU with 40 GB memory.

We performed an initial hyperparameter search on the parameters shown in table 1. Namely, we explored how the network performance changes when varying the building blocks, i.e. the attention layers in the Transformer or the blocks in the NRI, as well as the overall capacity of the models, i.e. the number of neurons in fully-connected layers, and additional initial/final MLPs. As this is a multi-target classification task, i.e. one target per LCAG entry, where getting all LCAG entry predictions correct is more important than getting only a subset correct with a high confidence (as cross-entropy loss does) we also included focal loss [49] in the search.

Table 1. Phasespace hyperparameter search parameters. Asterisks indicate parameters used exclusively in the larger-dimension, follow-up search performed on the NRI.

Model	Parameter	Values
NRI	No. NRI blocks	[1, 2, 4, 6]
	Block MLP layers	[0, 2, 4]
	Feed-forward width	[128, 256, 512, 768*, 1024*, 2048*]
	Initial MLP layers	[0*, 1, 2, 4]
	Final MLP layers	[0*, 1, 2, 4]
	Loss	[cross-entropy, focal]
Transformer	No. attentions	[1, 2, 4, 6]
	No. attention heads	[4, 8, 16]
	Feed-forward width	[256, 512, 1024]
	Final MLP layers	[1, 2, 4]
	Loss	[cross-entropy, focal]

All hyperparameter search training were performed for a maximum of 100 epochs, a batch size of 128, dropout rate of 0.3, random seed of 100, and with class weights applied, using the Adam optimizer with a learning rate of 0.001.

The results of the searches, shown for each individual hyperparameter, are shown in the supplementary material in figures 1 and 2. Our experiments show the NRI to significantly outperform the Transformer encoder, reaching an average validation perfectLCAG of around 46%, compared to the Transformer's 11%. No significant deviations are observed between the training and the validation scores. Two NRI blocks are consistently found to produce optimal results, with larger feed-forward dimensions, favouring 512, and no additional block MLPs within.

Based on these results, we performed a longer training for both the NRI and Transformer encoders using the optimal hyperparameters found and a random seed of 101 until the validation perfectLCAG score no longer improved (500 and 323 epochs, respectively). The results on the test dataset are shown in figure 5, where the NRI and Transformer achieved an average perfectLCAG of 46.6% and 5.80%, and a corresponding average accuracy of 91.6% and 38.6%, respectively. As with the hyperparameter optimisation, we observe no significant difference between the training and validation perfectLCAG scores, around 54% and 10% for the NRI and Transformer, respectively, but a clear sign of overfitting with respect to the test dataset. Overall, the Transformer struggles to learn more than the most trivial topologies, whereas the NRI achieves reasonable perfectLCAG scores on most of those below 10 leaves. Looking closer, for trees up to and including 6 leaves, the average perfectLCAG for the NRI is 92.5%, and for cases of trees up to and including 10 leaves 59.7%. The scores for all other subdivisions are shown in table 2. This result demonstrates not only that the NRI is an appropriate choice of architecture for learning physical interactions for a variety of tree sizes, but also that it behaves as expected, namely by learning what we intuitively identify as simpler topologies first.

Interestingly, we observe no significant advantage to an increased number of topologies at a given complexity. For example the large number of topologies at depth 5 with leaves 9 or 10 in the dataset show no noticeable performance improvement over those at depth 6 with the same number of leaves. This indicates that the model is only able to learn each topology individually, and is unable to generalize to topologies of similar complexity.

4.5. Ablation studies

Given the significantly better performance of the NRI over the Transformer, we explored the importance of the sequential edge-to-node and node-to-edge encoding layers within the NRI blocks. To do so, we repeated the previous training with all blocks removed, leaving only the initial node-to-edge encoding layer to infer the LCAG from pair-wise combinations of the embeddings alone. The results are shown in figure 6(a), which reached an average test perfectLCAG of 8.7%, only slightly outperforming the Transformer. We therefore conclude that the ability to iteratively combine node pairs is an essential feature of the NRI's ability to predict the LCAG, and similar features should be considered in any other GNN approaches explored in future.

The hyperparameter optimisations demonstrated a clear preference for larger feed-forward dimensions and a lower number of initial/final feed-forward layers. We therefore performed an additional hyperparameter search on these parameters up to a maximum feed-forward width of 2048,[12] with all other hyperparameters set to the optimal found in the previous search. The results are shown in figure 3 in the

[12] Due to hardware constraints we were unable to explore larger widths.

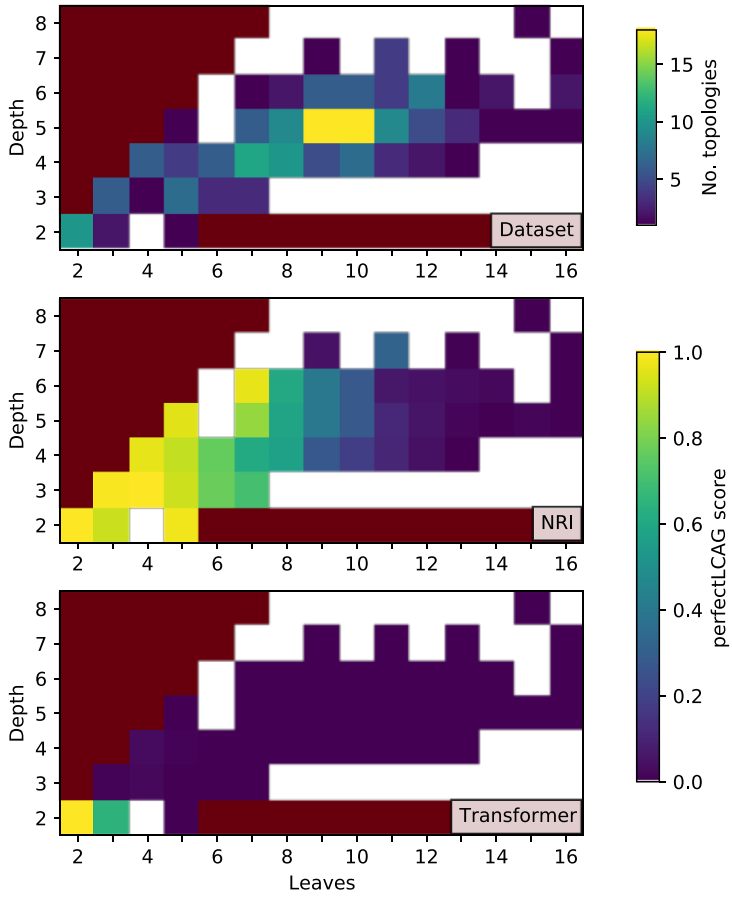


Figure 5. Dataset coverage (top) and perfectLCAG scores (middle, bottom) of the first experiment. Dark red indicates regions disallowed by the min/max children restrictions on topologies, white indicates regions with no training samples. Colours in the top plot show the number of topologies with the given depth/leaves configuration, colours in the lower two plots show the average perfectLCAG scores achieved on the given topologies' configurations.

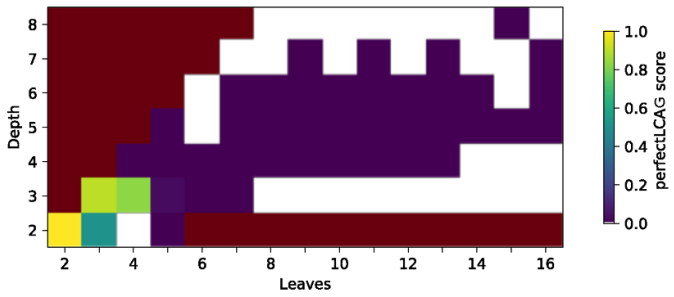
Table 2. Average perfectLCAG scores for subsets of topologies of increasing complexity. Bold numbers indicate the best score at each complexity.

	Average perfectLCAG (%) score for leaves up to and including														
Model	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Transformer	**100.0**	63.1	46.3	30.5	24.7	17.1	13.0	9.8	7.7	6.8	6.3	6.1	6.0	5.9	5.8
NRI	**100.0**	98.8	98.2	96.2	92.5	85.7	79.2	68.5	59.7	54.3	50.3	48.8	48.1	47.6	46.6
NRI_{2048}	**100.0**	**99.3**	**98.7**	**97.5**	**94.2**	**87.1**	**79.7**	**70.1**	**60.9**	**55.7**	**51.5**	**50.0**	**49.2**	**48.7**	**47.7**
$NRI_{noblocks}$	**100.0**	91.2	69.1	46.0	37.2	25.7	19.6	14.7	11.6	10.3	9.4	9.1	9.0	8.9	8.7

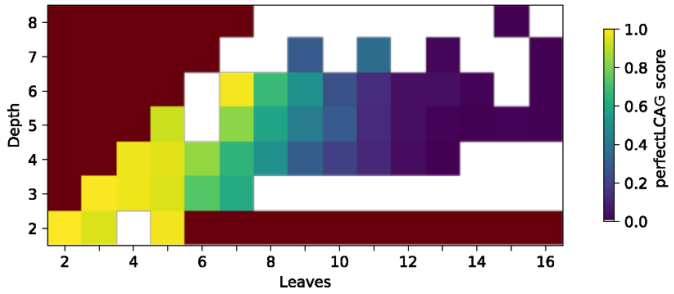
supplementary material, where no additional initial/final layers were found to be ideal, along with the maximum feed-forward width.

Following this finding, we repeated the training for 177 epochs (until validation perfectLCAG no longer improved) with these hyperparameters and the same random seed as in the trainings in section 4.4. The results of are shown in figure 6(b). We observe an average perfectLCAG of 47.7% (94.2% and 60.9% for the $\leqslant$6 and $\leqslant$10 regions, respectively) and accuracy of 92.6%, which represents only a small improvement over the previous results. Detailed results are shown in table 2. We see a relatively large discrepancy between this score and the maximum hyperparameter optimisation perfectLCAG reached in figure 3 in the supplementary material of 59.6% on the validation dataset. Similar to the experiments in section 4.4, we also find a comparable discrepancy in the validation perfectLCAG score of the 177 epoch training, which achieved 58.4%.

To summarise, with a width of 1024, the model achieved 46.6% test and 46% validation perfectLCAG, and for 2048 with all else equal it reached 47.7% test and 59.6% validation perfectLCAG. We therefore conclude that the hyperparameter optimisation has caused overfitting to the validation dataset, and that an increase in the fully-connected widths within the NRI only serves to exacerbate this problem.



(a) NRI encoder without NRI blocks (trained using a single `Node to Edge`).



(b) NRI encoder with a feed-forward layer width of 2048 for all MLP components.

Figure 6. Ablation study results for the NRI encoder on the test dataset shown in figure 5(top). The network failed to learn any complex topologies.

4.6. Discussion

While the Belle II experiment typically has ten or fewer leaves, the noisy experimental environments in hadron colliders like the LHC mean that many other particles can be produced, raising the total number of leaves significantly. However, it is often the case that only a subset of leaves that belong to the particle decay of interest are selected. For example, the LHCb triggers [50, 51] have a specific focus on filtering for events with *b*-hadrons whose decay products are subsequently analysed. As our results demonstrate high performance for scenarios with up to ten leaves, our proposed approach already offers utility in existing experimental scenarios. Nevertheless, there is still significant room for improvement, the key to which we believe will be finding exactly which hyperparameters can be modified to improve performance for more leaves.

We have already explored some of the hyperparameters in the experiments and shown the essential role of the NRI blocks (figure 6). We also demonstrated that a larger model capacity (larger width of feed-forward layers) corresponds to improved performance (table 2), but saw that this quickly leads to overfitting. While a further investigation into this overfitting is outside the scope of this work, as our primary interest was to perform an initial exploration into what appear to be the essential NRI hyperparameters, based on these findings we recommend considering the inclusion of cross-validation, stronger regularization, or ensemble methods to identify and mitigate this.

Looking further, we saw that the choice of architecture has the largest impact on performance, with the Transformer appearing unable to learn a meaningful solution (figure 5). We concluded in section 4.5 that the NRI's ability to iteratively combine node pairs was an essential component for its performance, however graph nodes do not exist only as child pairs, but also triplets and more. Therefore, a broad investigation into various other architectures, especially GNNs, with potentially better inductive biases is likely the most promising avenue for increasing performance to handle more leaves and larger trees.

5. Conclusion

Learning tree structures from leaves is a highly relevant, yet largely unexplored problem from a machine learning point of view, particularly for physical processes such as those in the field of particle physics. Recent years have seen not only the rise of machine learning in modeling the dynamics of physical systems, but more specifically the use of GNNs to do so. This shift represents an understanding that the representation of input

data, as unordered sets with relational information, is key to enabling such learning. Despite this, there has been little focus on constructing equally meaningfully representations of the prediction targets.

In this work, we have introduced the LCAG matrix as a novel, compact representation of tree-structured data. We have shown that the LCAG, while equivalent to the adjacency matrix, resolves the constraints that make learning the adjacency matrix directly impractical. By doing so, the LCAG enables the first end-to-end trainable solution which learns the hierarchical structure of varying tree sizes directly, using only the terminal tree leaves to do so.

We have demonstrated the use of the LCAG in the task of learning to predict a variety of particle decay processes on a simulated physics dataset. Our results show that when selecting an appropriate GNN, in this case the NRI encoder, the network is able to correctly predict the LCAG for 92.5% of decay trees up to and in including 6 leaves, and 59.7% of trees up to and including 10 leaves. We showed how the NRI blocks are an essential contributor to these results, and that the Transformer was unable to converge on a meaningful solution. Our results hint at the possibility of other GNNs with potentially better inductive biases enabling the performance to scale to larger trees with more leaves.

Our results pave the way to construct end-to-end trainable neural approaches to particle decay reconstruction at large collaborations such as the LHC or Belle II that may eventually replace existing algorithms. While existing particle physics reconstruction algorithms could in theory be adapted to the dataset used in this work for a direct comparison, this would entail very significant work as they have been heavily tailored towards the context of their respective experiments. It is both simpler and more meaningful for the experiments to instead explore the LCAG approach on real-world physics datasets directly.

While this work has only scratched the surface of what can be achieved when predicting the LCAG, particularly when exploring other graph interaction learning neural network architectures, we believe it to be a powerful representation tool as it resolves many of the constraints on existing hierarchical learning approaches. We expect that this work will serve as inspiration not only to high-energy particle physics experiments, but other domains working with similar problems and tree-structured data as well.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://doi.org/10.5281/zenodo.6983258.

Acknowledgments

This work is supported by the Helmholtz Association Initiative and Networking Fund under the Helmholtz AI platform grant and the HAICORE@KIT partition, the Bundesministerium für Bildung und Forschung (BMBF) under the Grant 05H21PDKBA, and the L'Institut National de Physique Nucléaire et de Physique des Particules (IN2P3) du CNRS (France) and the French Agence Nationale de la Recherche (ANR) under Grant ANR-21-CE31-0009 (Project FIDDLE) and the Seed Money programme of Eucor—The European Campus. We wish to also thank Hosein Hashemi, Julián García Pardiñas, and Isabelle Ripp-Baudot for the fruitful discussions.

ORCID iDs

James Kahn https://orcid.org/0000-0002-8517-2359
Ilias Tsaklidis https://orcid.org/0000-0003-3584-4484
Oskar Taubert https://orcid.org/0000-0002-3707-499X
Lea Reuter https://orcid.org/0000-0002-5930-6237
Giulio Dujany https://orcid.org/0000-0002-1345-8163
Florian Bernlochner https://orcid.org/0000-0001-8153-2719
Achim Streit https://orcid.org/0000-0002-5065-469X
Markus Götz https://orcid.org/0000-0002-2233-1041

References

[1] Evans L and Bryant P 2008 LHC machine *J. Instrum.* **3** S08001
[2] Abe T *et al* 2010 Belle II technical design report (arXiV:1011.0352)
[3] Kahn J *et al* 2022 *Lowest Common Ancestor Generations (LCAG) Phasespace Particle Decay Reconstruction Dataset* (Zenodo) (https://doi.org/10.5281/zenodo.6983258)
[4] Belle II Collaboration 2021 *Belle II Analysis Software Framework (basf2): Release-06-00-00* (Zenodo) (availabale at: https://zenodo.org/record/5574116/export/geojson)

[5] Li S, Wu L, Feng S, Xu F, Xu F and Zhong S 2020 Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem *Findings of the Association for Computational Linguistics: EMNLP 2020* (Association for Computational Linguistics) pp 2841–52
[6] Miura Y, Kano R, Taniguchi M, Taniguchi T, Misawa S and Ohkuma T 2018 Integrating tree structures and graph structures with neural networks to classify discussion discourse acts *Proc. 27th Int. Conf. on Computational Linguistics* (Association for Computational Linguistics) pp 3806–18
[7] Qiao Z, Wang P, Fu Y, Du Y, Wang P and Zhou Y 2020 Tree structure-aware graph representation learning via integrated hierarchical aggregation and relational metric learning *2020 IEEE Int. Conf. on Data Mining (ICDM)* pp 432–41
[8] Ying R *et al* 2018 Hierarchical graph representation learning with differentiable pooling *Advances in Neural Information Processing Systems* vol 31 (Curran Associates Inc.) pp 4805–15
[9] Zhong Z, Li C T and Pang J 2020 Hierarchical message-passing graph neural networks (arXiv:2009.03717)
[10] Kipf T *et al* 2018 Neural relational inference for interacting systems *Proc. 35th Int. Conf. on Machine Learning (ICML '18) (Proc. Machine Learning Research)* vol 80 (PMLR) pp 2688–97
[11] Stanić A, van Steenkiste S and Schmidhuber J 2020 Hierarchical relational inference (arXiv:2010.03635)
[12] Kim J *et al* 2019 Edge-labeling graph neural network for few-shot learning *Proc. 2019 IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR 2019)* (Institute of Electrical and Electronics Engineers (IEEE)) pp 11–20
[13] Finley T and Joachims T 2005 Supervised clustering with support vector machines *Proc. 22nd Int. Conf. on Machine Learning (ICML '05)* (Association for Computing Machinery (ACM)) pp 217–24
[14] Wang H and Leskovec J 2020 Unifying graph convolutional neural networks and label propagation (arXiv:2002.06755)
[15] Ranjan E, Sanyal S and Talukdar P 2020 ASAP: adaptive structure aware pooling for learning hierarchical graph representations *Proc. AAAI Conf. on Artificial Intelligence* vol 34 pp 5470–7
[16] Vaswani A *et al* 2017 Attention is all you need *Advances in Neural Information Processing Systems* vol 30 (Curran Associates, Inc.)
[17] Guest D, Cranmer K and Whiteson D 2018 Deep learning and its application to LHC physics *Annu. Rev. Nucl. Part. Sci.* **68** 161–81
[18] Larkoski A J, Moult I and Nachman B 2020 Jet substructure at the Large Hadron Collider: a review of recent advances in theory and machine learning *Phys. Rep.* **841** 1–63
[19] Albertsson K *et al* 2018 Machine learning in high energy physics community white paper *J. Phys.: Conf. Ser.* **1085** 022008
[20] Shlomi J, Battaglia P and Vlimant J R 2021 Graph neural networks in particle physics *Mach. Learn.: Sci. Technol.* **2** 021001
[21] Duarte J and Vlimant J R 2020 Graph neural networks for particle tracking and reconstruction (arXiv:2012.01249)
[22] Sirunyan A M *et al* 2017 Particle-flow reconstruction and global event description with the CMS detector *J. Instrum.* **12** P10003
[23] Aaboud M *et al* (ATLAS Collaboration) 2017 Jet reconstruction and performance using particle flow with the ATLAS Detector *Eur. Phys. J.* C **77** 466
[24] Pata J, Duarte J, Vlimant J R, Pierini M and Spiropulu M 2021 MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks *Eur. Phys. J.* C **81** 381
[25] Mokhtar F, Kansal R, Diaz D, Duarte J, Pata J, Pierini M and Vlimant J R 2021 Explaining machine-learned particle-flow reconstruction (arXiv:2111.12840)
[26] Farrell S *et al* 2017 The HEP.TrkX project: deep neural networks for HL-LHC online and offline tracking *EPJ Web Conf.* **150** 00003
[27] Ju X *et al* 2021 Performance of a geometric deep learning pipeline for HL-LHC particle tracking *Eur. Phys. J.* C **81** 876
[28] Ju X *et al* 2020 Graph neural networks for particle reconstruction in high energy physics detectors (arXiv:2003.11603)
[29] Henrion I *et al* 2017 Neural message passing for jet physics *Workshop on Deep Learning for Physical Sciences (DLPS 2017)* (Curran Associates, Inc.) pp 1–6
[30] Shlomi J *et al* 2021 Secondary vertex finding in jets with neural networks (arXiv:2008.02831)
[31] Moreno E A *et al* 2020 JEDI-net: a jet identification algorithm based on interaction networks *Eur. Phys. J.* C **80** 58
[32] Qu H and Gouskos L 2020 Jet tagging via particle clouds *Phys. Rev.* D **101** 056019
[33] Wang Y, Sun Y, Liu Z, Sarma S E, Bronstein M M and Solomon J M 2019 Dynamic graph CNN for learning on point clouds *ACM Trans. Graph.* **38** 146:1–12
[34] Qu H, Li C and Qian S 2022 Particle transformer for jet tagging *Proc. 39th Int. Conf. on Machine Learning* (PMLR) pp 18281–92
[35] Keck T *et al* 2019 The full event interpretation *Comput. Softw. Big Sci.* **3** 6
[36] Wu Z *et al* 2021 A comprehensive survey on graph neural networks *IEEE Trans. Neural Netw. Learn. Syst.* **32** 4–24
[37] Aho A V, Hopcroft J E and Ullman J D 1973 On finding lowest common ancestors in trees *Proc. 5th Annual ACM Symp. on Theory of Computing STOC'73* (Association for Computing Machinery) pp 253–65
[38] Amrouche S *et al* 2020 The tracking machine learning challenge: accuracy phase (arXiv:1904.06778)
[39] Benato L *et al* 2021 Shared data and algorithms for deep learning in fundamental physics (arXiv:2107.00656)
[40] Navarro A P and Eschle J 2019 phasespace: *n*-body phase space generation in Python *J. Open Source Softw.* **4** 1570
[41] James F E 1968 *Monte Carlo phase space* CERN-68-15 CERN (https://doi.org/10.5170/CERN-1968-015)
[42] Zyla P A *et al* Particle Data Group 2020 Review of particle physics, 2020–2021 *Prog. Theor. Exp. Phys.* **2020** 083C01
[43] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32, ed H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox and R Garnett (Curran Associates, Inc.) pp 8024–35
[44] Fomin V, Anmol J, Desroziers S, Kriss J and Tejani A 2020 High-level library to help with training neural networks in PyTorch (availabale at: https://github.com/pytorch/ignite)
[45] Clevert D A, Unterthiner T and Hochreiter S 2016 Fast and accurate deep network learning by exponential linear units (ELUs) (arXiv:1511.07289)
[46] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift (arXiv:1502.03167)
[47] Akiba T, Sano S, Yanase T, Ohta T and Koyama M 2019 Optuna: a next-generation hyperparameter optimization framework *Proc. 25th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining KDD '19* (Association for Computing Machinery) pp 2623–31
[48] Bergstra J *et al* 2011 Algorithms for hyper-parameter optimization *Proc. 24th Int. Conf. on Neural Information Processing Systems NIPS'11* pp 2546–54
[49] Lin T Y, Goyal P, Girshick R, He K and Dollár P 2017 Focal loss for dense object detection *Proc. IEEE Int. Conf. on Computer Vision* pp 2980–8
[50] Aaij R *et al* 2013 The LHCb trigger and its performance in 2011 *J. Instrum.* **8** P04022
[51] Gligorov V V and Williams W M 2013 Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree *J. Instrum.* **8** P02013