2024-LDRD-2301: Graph Analytics for CEBAF Operations

C. Tennant, T. Larrieu, D. Moser
Jefferson Lab, Newport News, VA 23606

J. Li, S. Wang, Z. Xu
University of Virginia, Charlottesville, VA 22903

Abstract

We report on the progress achieved during a 2-year Laboratory Directed Research and Development (LDRD) project titled "Graph Analytics for CEBAF Operations". The objective of this project is to leverage deep learning on graph representations of CEBAF's injector beamline in order to create a tool for improving the efficiency of beam tuning tasks [1]. Specifically, we use graphs to represent the injector beamline at any arbitrary date and time and invoke a graph neural network (GNN) to extract a low-dimensional, informative representation that can be visualized in two-dimensions. By analyzing years of operational data from the CEBAF archiver, good and bad regions of parameter space can be identified. The goal is to exercise this framework as a real-time tool to aid beam tuning, which represents the dominant source of machine downtime.

Background

With access to information-rich data sources, an increase in compute power, and the availability of user-friendly, open source software, the field of artificial intelligence (AI) – and deep learning in particular – is making revolutionary impacts in a variety of fields and sectors. Arguably, the biggest advances in deep learning are applications for natural language processing and computer vision. The data for each of these domains (i.e. text and images) can each be considered a type of graph. For example, text can be represented by a line graph where a word is related to its predecessor and successor, while images are a regular 2D grid of pixel values.

In general, however, graphs are used to describe more complex relationships between entities. Examples include social, economic, communication, citation, and transportation networks, disease pathways, knowledge graphs and even molecules, among many others. In recent years, the field of network, or graph, analytics has been supplemented with the tools of deep learning. Specifically, a graph neural network (GNN) provides a framework for defining deep neural networks on arbitrary graph data. This has been an area of intense research and development in the last five years [2, 3, 4, 5]. To our knowledge, using a graph representation approach to better understand accelerator beamlines, with specific application for beam tuning, has never been done before and therefore represents a novel area of research with potential for significant benefit to CEBAF operations.

Motivation

Enormous efforts are expended creating high-fidelity simulations of accelerator beamlines. While these simulations provide guidance on how to set up a beamline, there always exists a gap between the simulated ideal and the real-world implementation. Bridging that gap often requires a laborious and time-consuming process known as beam tuning. This project develops a data-driven

approach to beam tuning in CEBAF that leverages deep learning over structured data (graphs) with the goal of reducing the effort and time required for this task.

The fundamental idea behind this proposal is to express the state of CEBAF beamline as a graph, find a suitable low-dimensional embedding such that the result can be visualized, and to leverage that framework for more efficient beam tuning. This process is illustrated in Fig. 1. Specifically, the goal is to construct a graph for the state of the CEBAF injector beamline at any arbitrary date and time, apply a suitable whole-graph embedding, and visualize low-dimensional representations. Providing an associated category label for each embedding (i.e. "good" setup or "bad" setup) would allow the identification of optimal regions of parameter space. In this way we provide a principled, data-driven approach to beam tuning in CEBAF that leverages deep learning over structured data.

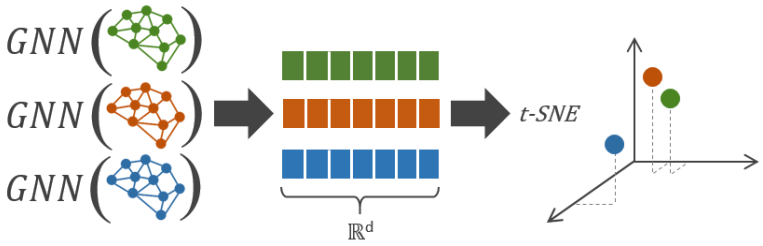

FIGURE 1: Illustration of whole-graph embedding. The goal is to find a low-dimension vector representation of a graph that preserves as much information as possible. The color of each graph/vector/marker denotes a unique CEBAF injector configuration, for example.

What makes this project novel is how we propose to use deep learning over this high-dimensional structured data to visualize what is happening in a low-dimensional latent space, and to utilize it as a tool for beam tuning. An illustration of how this might work is shown in Fig. 2. We first begin by embedding months – or even years – of historical injector setups and use a trained deep learning model to embed them in a two-dimensional space (Fig. 2, left). Next, by mining the archiver and labeling several of the setups we can map out the "good" and "bad" regions in parameter space (Fig. 2, middle). Lastly, we can track in real-time how an injector setup is moving through this latent space during beam tuning, with the benefit of seeing if changes are moving the system closer or further from regions marked by "good" setups. Furthermore, data collection is ongoing and passive, and it does not require investment in additional diagnostics and equipment.

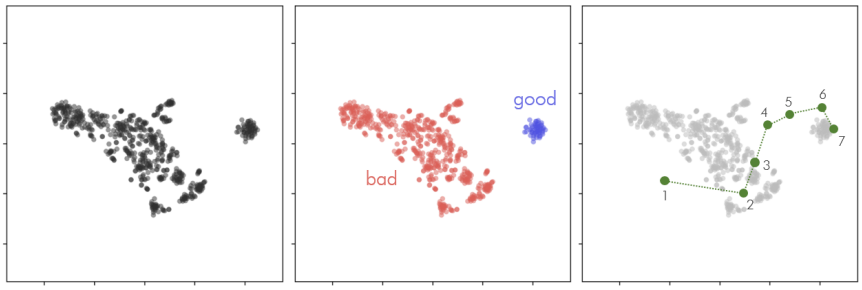

FIGURE 2: An example of a 2D visualization where each marker represents a low-dimensional embedding of a complex graph (left), the result of associating a label for the goodness of the setup in CEBAF (middle), and using the results to guide beam tuning in a control room setting (right).

Method

Developing the workflow constituted the majority effort for the first year of the project. At a high level, we are pre-training a deep learning model on a large set of unlabeled data using a technique called self-supervised learning (SSL). Methods for self-supervised learning try to learn as much as possible from the data alone, so a model can be fine-tuned for a specific downstream classification task. In this way we can take advantage of years of operational data stored in the archiver without having to do the laborious and expensive task of hand labeling the data. We implement a graph neural network to learn rich feature vectors for each graph. A special class of loss function, known as contrastive loss, is implemented which maximizes agreement between latent representations from similar graph pairs ("positive pairs") while minimizing agreement from unlike pairs ("negative pairs"). We then fine-tune the model on the downstream task of classifying good and bad setups using a smaller, labeled dataset. A high-level view of this workflow is illustrated in Fig. 3. Finally, a dimensionality reduction technique like principal component analysis (PCA) or UMAP [6] is used to visualize the results. It should be noted that this same basic workflow is being leveraged to produce state-of-the-art results in natural language processing and also in vision tasks.

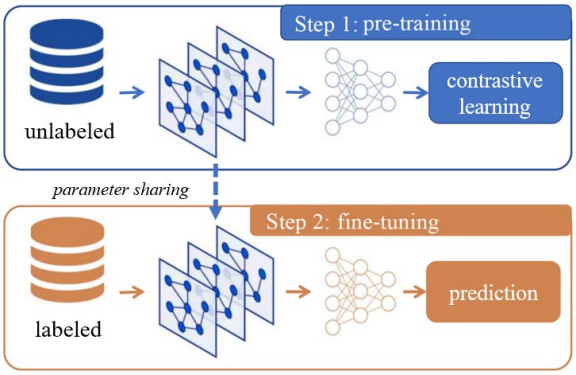

FIGURE 3: Schematic showing the workflow starting from pre-training a model on unlabeled data using contrastive learning (top) and then fine-tuning the model on a smaller set of labeled data using supervised learning (bottom). Figure modified from Ref. [7].

Beamline-to-Graph

To make the idea of representing a beamline as a graph more concrete, consider the following example shown in Fig. 4. The (fictitious) beamline consists of several element types (beam current monitor (BCM), beam position monitor (BPM), quadrupole, solenoid, corrector) which are mapped to nodes. Each node type has a unique set of features; quadrupoles, correctors, and solenoids have a single value corresponding to their field strength, a BCM reports measured beam current, and a BPM has two features which correspond to the horizontal and vertical beam position readings. It should be noted that in addition to scalar quantities, features can include other information-rich data sources, including but not limited to, text and/or images. For instance, a graph might contain a node that represents a beamline viewer with one its features being the image recorded by the diagnostic. The resulting graph for this simple example is a heterogeneous, directed graph. It is heterogeneous because it consists of nodes of different type and is directed because the edges have a sense of direction. A user-defined "window" defines the edges. In this

example a window of 2 is used which means each *setting* element/node is connected to the two setting elements/nodes immediately downstream and any *reading* nodes in between. A setting node is one in which the operator can modify the setpoint and is used for routine beam tuning tasks. In this example these would include the magnetic elements (quadrupole, solenoid, corrector). Reading nodes, on the other hand, are passive readbacks and represent diagnostics in the beamline (BPM, BCM). The edges are directed, since in this non-recirculating beamline topology, an element cannot affect anything upstream of itself. Depending on the downstream task and the beamline, other graph representations will benefit from a different window size. Though not explicitly shown in this example, using a graph framework allows global beamline parameters to be incorporated as well. For instance, a "master node" that has connections to each node in the graph could contain readings from temperature sensors in the beamline enclosure, outdoor temperature and humidity, date and time, and/or electronic log entries, among other things.

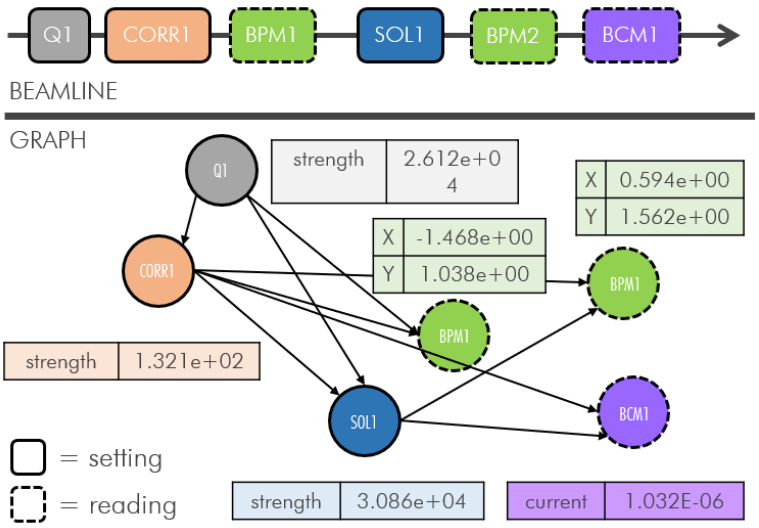

FIGURE 4: Illustration depicting an arbitrary accelerator beamline (top) and one possible way to construct a corresponding graph representation (bottom). Here the nodes represent individual elements, the node features correspond to appropriate element parameters, a user-specified window of 2 elements (see text for details) defines edges between nodes. The edges are directed to capture the fact that an element cannot influence upstream elements of the beamline.

Injector Beamline

For the purpose of the LDRD, we are focusing on the CEBAF injector. Specifically, we consider the beamline starting with MFA0I03 (s = 6.65 m) and extending to the 2 kW insertable dump (s = 101.58 m). Recent upgrades have modified the first several meters of the beamline, and in order to leverage historical data that predates those changes, we omit the region upstream of MFA0I03 from consideration. The injector beamline is appealing for a variety of reasons:

- manageable size: dealing with an entire pass of CEBAF for this proof-of-concept work would be daunting, however the injector is a manageable size both physically (approximately 95 m of beamline) and in the number of process variables (PVs).
- lots of beam tuning: because the formation and evolution of the beam at low energy is so critical to performance, the injector represents a region where a lot of beam

tuning happens, which translates to varied data, and is a necessary component for training models.
- diversity of beamline components: the injector contains a variety of elements, including warm and cold radio-frequency (RF) cavities, correctors, solenoids, dipoles, quadrupoles, as well as many diagnostics such as BPMs, BCMs, vacuum readbacks and beam loss monitors (BLM).
- suited for beam studies: because the injector can be operated even when other portions of the machine are down, it makes it ideally suited for maximizing dedicated beam studies opportunities.
- The injector is unidirectional and therefore simpler to study than the main accelerator ring where recirculation complicates the notion of upstream and downstream directionality.

`ced2graph` *Software*

In practice, creating a graph as outlined previously requires mining data from several CEBAF databases. Given a date and time stamp, a beamline is defined from the CEBAF Element Database (CED) [8]. Secondly, node features are populated by loading appropriate control system process variables (PVs) stored in the MySQL Archiver (Mya) [9]. Each setting nodes is connected to n downstream setting nodes and any reading node in between, where n is set by the user (edges never originate on reading nodes, they only end). The data is then written to four files, which together constitute all the information needed to construct the graph [10]. For additional details on the file format see Appendix A. As discussed previously, nodes in the graph are broadly divided into setting and reading nodes. For the injector beamline the nodes and their associated features are:

- **settings**
  - corrector: (magnetic field strength, current)
  - dipole: (magnetic field strength, current)
  - quadrupole: (magnetic field strength, current)
  - solenoid: (magnetic field strength, current)
  - RF cavity: (phase, gradient, gang phase)
- **readings**
  - beam loss monitor: (current)
  - beam position monitor: (horizontal position, vertical position, wire sum)
  - beam current monitor: (current)
  - vacuum readback: (pressure)

By way of example, a single graph that represents the state of the injector is comprised of 11 different node types, 206 total nodes, 296 node features, and 409 edges (for $n = 2$).

Graph Embedding

In our framework, we leverage a GNN to generate graph embeddings of the injector beamline. Due to the heterogeneity of nodes (i.e. nodes can have different length feature vectors according to their type), we model each injector configuration as a heterogeneous graph. The model first utilizes a linear layer for each node type so that the input node features of differing node types are transformed into vectors of the same hidden size. Then the model processes these hidden vectors

via standard graph attention network (GAT) layers to generate output embeddings of specified output size [11]. This has been implemented using the PyTorch Geometric library [12]. The latest code is available at: https://github.com/SongW-SW/cebaf-graph-analyze (note, at present the repository is private and only allows access to members of the project). Below we discuss the details of how the GNN is leveraged to train a model using self-supervision and also in a supervised way.

Self-Supervised Learning

Self-supervision generates a supervisory signal from the data itself, thereby avoiding the laborious task of hand labeling. The motivation is that a model pre-trained on a large body of unlabeled data will learn robust embeddings that can more easily be fine-tuned with a small labeled dataset in the standard supervised way. Below we highlight the main steps of this workflow and which is illustrated in Fig. 5.

- A graph, G_1, is chosen at random and one of three graph augmentations is applied to create a corrupted version of the graph. The user specifies which augmentation should be used. They are:
  - *node masking*: randomly removes a percentage of graph nodes (default value is set to 30%)
  - *edge removal*: randomly removes a percentage of graph edges (default value is set to 20%)
  - *feature shuffling*: within each node type, all features are randomly shuffled
- A linear layer is applied to each node type to transform feature vectors into the same size so we can treat the graph as homogeneous. For the results in this note, the hidden and output size is 128.
- A GAT layer is used to aggregate information from a node's neighbors. As the name suggests, self-attention is used so rather than each node contributing uniformly, the model learns the neighbors which are more important and weights them differently during aggregation. Unless otherwise noted, the results reported use 3 GAT layers with a single attention head.
- The loss function follows the implementation from Ref. [13]. First, the similarity score is computed between a random node representation of the corrupted graph and the representation of the entire, uncorrupted graph. A second similarity score is computed between a random node representation of the uncorrupted graph and the representation of the entire, uncorrupted graph. A cross-entropy loss is used to maximize agreement between latent representations from similar graph pairs while minimizing agreement from unlike pairs.
- Training is performed with mini-batches of 5 graphs using the Adam optimizer with the learning rate set to 5E-03.

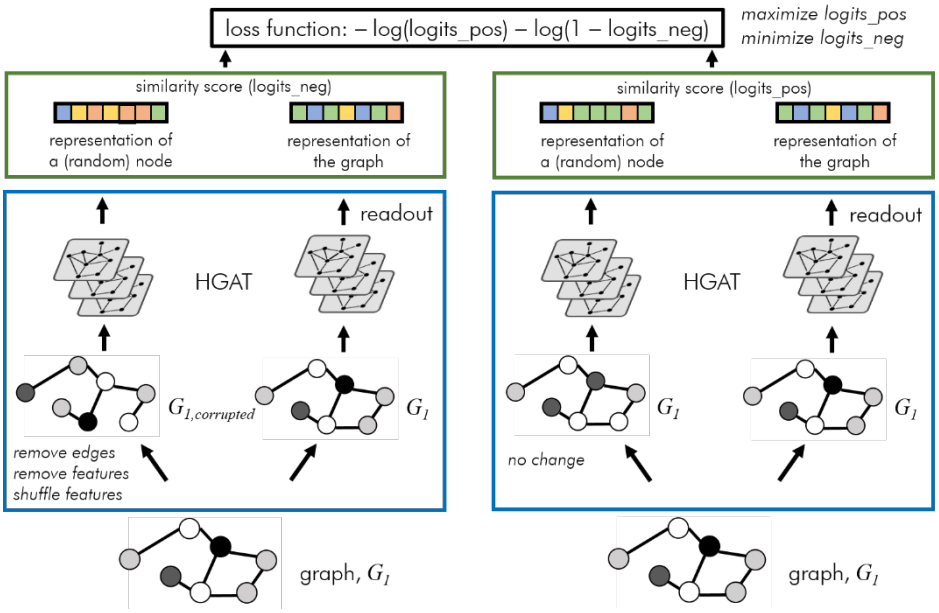

FIGURE 5: Illustration of the workflow for training a model with self-supervised learning. One of three graph augmentation strategies are used to create a corrupted version of a graph, G_1. The similarity score is computed between a random node representation of the corrupted graph and the representation of the entire, uncorrupted graph (left). A separate similarity score is computed between a random node representation of the uncorrupted graph and the representation of the entire, uncorrupted graph (right). Cross-entropy loss is used to maximize agreement between latent representations from similar graph pairs ("logits_pos") while minimizing agreement from unlike pairs ("logits_neg"). Figure modified from Ref. [7].

Supervised Learning

Training a model in a supervised way leverages the same node embedding and GAT workflow, but does not use graph augmentations or compute similarity scores. Rather, the training is done in the conventional way by associating graph embeddings with their ground truth label and using cross-entropy loss. See Fig. 6.

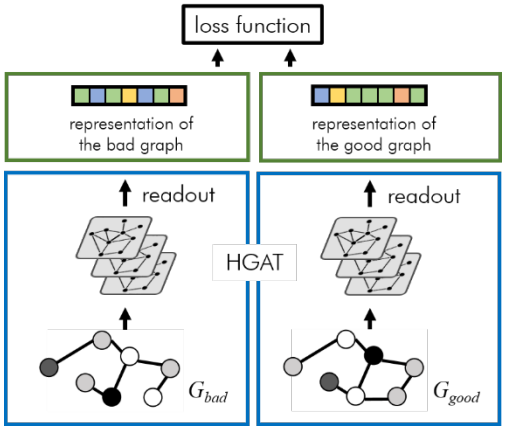

FIGURE 6: Illustration of the workflow for training a model with supervised learning.

Data

To generate a dataset with `ced2graph`, a user can either provide a list of specific date and time stamps or provide a range of dates and the interval at which to output a graph. We leveraged both methods for the work reported in this note. The `ced2graph` software also allows for user-defined filtering so that a graph is generated only if specific constraints are met. For the datasets described below, we only require the current monitor at 0L02 (IBC0L02Current) read greater than 0.1 μA. The user also sets the number of edges each setting node should make with downstream setting nodes. Our baseline dataset uses a value of 2 (this does not mean that each node has only two connections, it means that each setting node is connected to the next two downstream setting nodes *and any reading node(s) in between*). A summary of the graph datasets is shown in Table 1, with more detailed descriptions of each given below.

TABLE 1: Summary of the baseline graph datasets.

Dataset Name	Number of Graphs	Number of Edges	Labeled?
2021	2,129	2	N
Good Setups	354	2	Y
Bad Setups	254	2	Y
January 2022	353	2	N

2021 Data

The injector state was recorded every hour for the calendar year of 2021 which, after filtering, yields 2,129 graphs (the first half of the year was a Scheduled Accelerator Down (SAD) and operations did not start in earnest until August). This dataset is used for model pre-training.

Good Setups Data

In order to identify periods of good injector running, we used the MyaPlot Event Search feature. Given specific PVs and logical conditions to search on, Event Search returns events in a given time period that meet those constraints. Good injector configurations were identified if there had not been a fast shutdown (FSD) trip for 30 minutes, the laser modes (master and Hall) were reading continuous wave (CW) operation, and the beam current at 0R08 exceeded 5 μA [14]. While Event Search can automate this search, each event still must be examined by an experienced operator to confirm that a Hall was not changing current or coming on- or off-line. Following this procedure returns a dataset of 354 graphs labeled as good for supervised learning.

Bad Setups Data

Calling these "bad" setups is a misnomer – it would more correct to label them "non-ideal". And it turns out, identifying these kinds of setups is very difficult. A setup in which beam cannot be cleanly transported through the injector is clearly bad, but the aim of this work to identify optimal setups and be able to distinguish them from non-ideal configurations. If the beam is deemed good by users in the experimental halls, then it's safe to assume the injector setup is good. However, if the beam is deemed bad by the users, it does not necessarily follow that the injector configuration is bad. The issue could originate from anywhere in the several kilometers of beamline between the injector and the end stations. Therefore, to create a collection of bad setups we leveraged data from a January 9, 2022 beam studies that took data for a different project involving the injector [15]. In

the study a variety of beamline components were varied, one at a time, from their nominal values and the downstream response of the system recorded. The changes to each element were sufficient to cause a downstream response, but small enough for beam to be transported cleanly. The time stamps corresponding to each system change were used as input to `ced2graph`. In the end we generated a dataset of 254 graphs labeled as bad for supervised learning.

January 2022 Data

Lastly, we generated an unlabeled dataset for gauging the performance of the trained model, as a way to mimic a deployed system. The data was collected between January 10, 2022 (so as not to include the beam studies the previous day) through the end of the month, at intervals of every hour. This results in 353 graphs. Note that January 2022 is part of the *same* operational run from which the unlabeled 2021 graphs were extracted. This is important because we know that from one operational run to another there are significant underlying data drifts (though not yet well defined or identified). We will return to this issue in the "Future Work" section.

By considering this many PVs (296), at such a fine granularity (every hour, or even every several seconds), over such a long time (several months), we are utilizing data from the archiver in novel ways. It's not surprising, therefore, that several issues had to be resolved before it all worked smoothly. For example, one of the issues was the discovery that several of the vacuum readback PVs were reading current drawn (Amperes) rather than pressure (Torr). It's critical that nodes of the same type report their features in the consistent units. Fortunately, the scaling to convert from current to pressure is known [16] and straightforward to implement in the configuration file of `ced2graph`.

Ablation Studies

With the datasets described in the previous section, and using the code developed to train deep learning models over structured data, we proceed to perform a variety of studies aimed at gauging performance of the model. Unless otherwise noted, the graphs are assumed to be undirected and constructed with 2 edges. Each model was pre-trained on the 2021 unlabeled graphs using self-supervised learning and fine-tuned on the labeled graphs using supervised learning. The latter used 5% of the graphs for training and the remainder for testing.

GNN- Versus MLP-encoder

The premise of this project is to leverage GNNs in order to preserve topological/structural information when representing a beamline. The hypothesis being that this would provide more expressive low-dimensional representations, which in turn could be used for the benefit of CEBAF operations. But given the relatively linear nature of the graph structure, it is not clear a priori if this is the case. To address the issue, we compared the performance of the model using a GNN and then comparing that when using a simple multi-layer perceptron (MLP). The results are given in the first two columns of Table 2. Model accuracies were computed for the three different graph augmentations (see the "Graph Embedding" section). The results demonstrate that a GNN outperforms an MLP – an indication that preserving structural information greatly enhances performance.

Table 2: Summary of model performance under a variety of conditions (see text for more detail).

Setting	**GNN** *(2-edges)*	**MLP** *(2-edges)*	**GNN** *(5-edges)*	**GNN** *(5-edges, directed)*
Edge Removal	0.9225	0.8612	0.9008	0.7798
Feature Removal	0.9037	0.8483	0.6992	0.7302
Feature Shuffling	0.9876	0.8417	0.7752	0.7307
No SSL	0.8663	0.8512	0.8047	0.7380

Effect of the Number of Edges

The baseline graph datasets were generated with 2 edges, by which we mean each setting node is connected to the two downstream setting nodes along with any reading node(s) in between. We also explored the effect of pre-training and fine-tuning models on graphs generated with 5 edges. The results of this study are shown in the third column of Table 2. Because we start with a greater number of edges, it's not surprising that using edge removal during self-supervised learning does not impact performance as much as the other two augmentations. It is not entirely clear why there is such a significant drop in performance, though it may be attributed to feature over-smoothing – a common problem with GNNs.

Effect of Directed Edges

Another hypothesis posited that graphs with directed edges may prove useful as it builds-in the notion of causality (i.e. a beamline component cannot impact upstream of itself). We tested this with the 5-edges graph data and the results are given in the fourth column of Table 2. Except for the feature removal augmentation, we see a fairly significant drop in performance. We believe this is due to the lack of information flow. By construction, edges end on reading nodes. Therefore, with directed edges their information is not conveyed to other nodes.

Effectiveness of Self-Supervised Learning

The effect of bypassing pre-training and directly training a model on the smaller set of labeled graphs was also investigated. These results are given in the last row ("No SSL") of Table 2. Consider the GNN trained with 2-edge graphs (first column) as this exhibits the best performance. Pre-training in a self-supervised way leads to a more accurate final model, especially when limited, labeled data is available (recall that only 5% of the training data is used for fine-tuning). Given the difficulty and time required for a subject matter expert to label data, the ability to achieve good model performance by pre-training on easily accessible, unlabeled data conveys a huge benefit.

Beam Studies Results

In this section we consider two results of graph embeddings with specific application to CEBAF operations.

Graph Embedding Test

Preliminary results of applying the graph embedding workflow to the CEBAF injector beamline are summarized in Fig. 7. A GNN encoder was first pre-trained on the 2,129 unlabeled graphs from 2021 and then fine-tuned on the smaller set of labeled (354 good, 254 bad) graphs. Recall

that a bad setup is sufficient to transport beam but would not deliver physics-quality beam to users. The good and bad regions in Fig. 7 are denoted by the green and red contours, respectively. After fine-tuning, a set of 353 unlabeled graphs representing operations in January 2022 were input to the model and their resulting latent representations plotted (denoted by black markers).

There are several things to note in this result. First, the model cleanly separates good and bad regions of parameter space. Secondly, the majority of the unlabeled graphs from January 2022 (75%) are clustered around the good region in parameter space. Intuitively this makes sense, in that we were several months into a run, things were running relatively smoothly, and we were delivering beam to users. We expect that most of the setups were good.

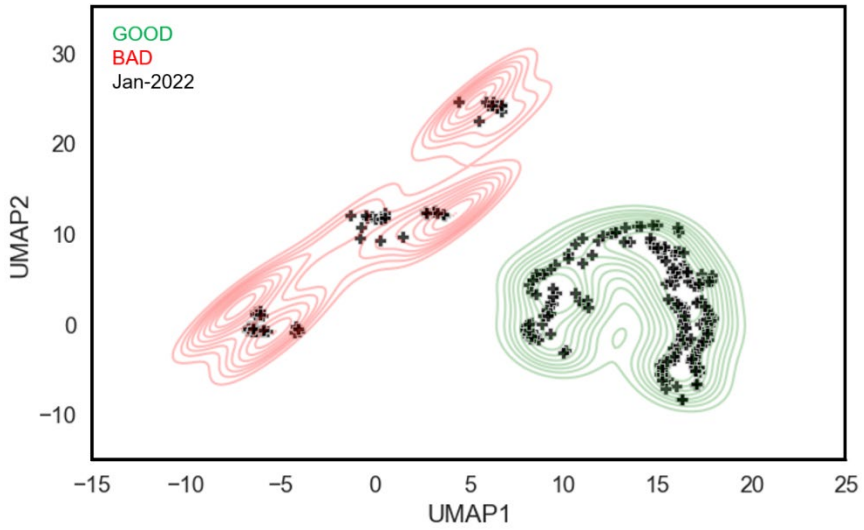

FIGURE 7: Visualization showing regions of parameter space for good setups (green contours) and bad setups (red contours) based on 2021 data (left). The black markers represent unlabeled states of the injector during January 2022. The model assigns a majority of the setups during this time (75%) to the region represented by good setups (right).

Visualizing Switching Injector Configurations

At the end of August 2022, we secured beam studies time to switch injector configurations in CEBAF. The purpose was to start from an initial injector configuration (call it, *A*), load in different configuration (call it, *B*), and migrate from setup *B* to setup *A* in a methodical, incremental way. During the beam study we made 64 incremental steps, changing only one beamline component at a time, to successfully migrate from setup *B* to *A*. Offline, we created embeddings of those 64 discrete steps to visualize the effect of tuning a beamline by looking at a low-dimensional, latent space. This result is display in Fig. 8. Analysis is still ongoing and this dataset will be critical for developing an explainability framework – the topic of recently funded FOA project (see Appendix B).

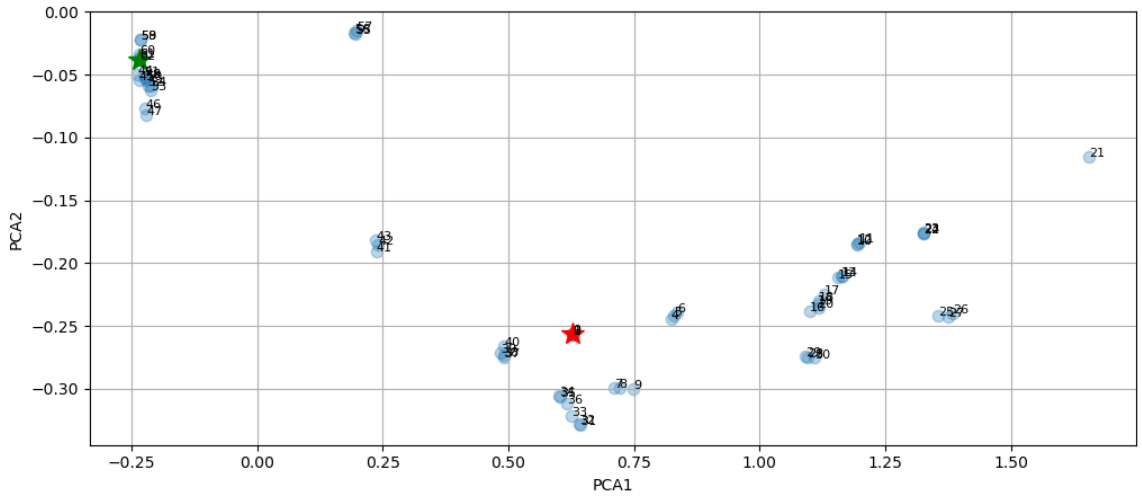

FIGURE 8: Visualization showing the transition from one injector setup in 64 discrete steps. The starting (final) configuration is denoted by the red (green) star. The numbered blue markers indicate the steps is migrating from one setup to another.

Summary

The last several years have witnessed a meteoric rise in applications of machine learning to the field of accelerator physics. Much of the published work represents leveraging ML on very narrow tasks. We believe this work addresses an issue inextricably linked to all user facilities (beam tuning) while taking a larger, global approach to the solution.

Excellent progress has been made in the 2 years of support from the LDRD. The ability to quickly generate a graph representation of CEBAF along with the ability to pre-train a deep learning model on unlabeled data, opens up exciting opportunities to explore additional use cases. In fact, this has potential application beyond particle accelerators to other high-dimensional, systems that require human-in-the-loop tuning more generally. In principle, these tools could easily be adapted for use in other operational facilities.

The project is an intersection of graph analysis, deep learning, self-supervised learning, dimensionality reduction, and visualization, all within the context of a complex, operational system. This combination of multiple cutting-edge research topics along with an abundance of historical, real-world data, makes this a unique research opportunity. We see this project as laying the foundation for a highly relevant research program in a rapidly developing field that emerged within the last decade, as the tools of deep learning are combined with the tools of graph analysis. See Appendix B for a list of papers, proposals, and publications related to this work.

Acknowledgement

The research described in this paper was conducted under the Laboratory Directed Research and Development Program at Thomas Jefferson National Accelerator Facility for the U.S. Department of Energy.

References

[1] C. Tennant, “Graph Neural Networks for Particle Accelerator Facilities”, Provisional Patent, October 11, 2021.

[2] Z. Wu et al., "A Comprehensive Survey on Graph Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. **32**, no. 1, pp. 4-24 (2021). doi: 10.1109/TNNLS.2020.2978386

[3] L. Wu, P. Cui, J. Pei, and L. Zhao, “Graph Neural Networks: Foundations, Frontiers, and Applications”, Springer, Singapore (2022).

[4] J. Zhou et al., “Graph Neural Networks: A Review of Methods and Applications”, AI Open, vol. **1**, pp. 57-81 (2020). doi: 10.1016/j.aiopen.2021.01.001

[5] Z. Shang, P. Cui, and W. Zhu, “Deep Learning on Graphs: A Survey”, arXiv:1812.04202v3 (2020).

[6] L. McInnes, J. Healy, and J. Melville, “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”, arXiv (2018). doi: 10.48550/ARXIV.1802.03426

[7] Y. Wang, J. Wang, Z. Cao, et al. “Molecular Contrastive Learning of Representations via Graph Neural Networks”, Nat. Mach. Intell. **4**, pp. 279–287 (2022). doi: 10.1038/s42256-022-00447-x

[8] T. Larrieu et al., “The CEBAF Element Database and Relational Operational Software”, in Proc. of 6th International Particle Accelerator Conference, MOPWI045, pp. 1256-1258 (2015).

[9] C. Slominski, “A MySQL Based EPICS Archiver”, in Proc. 12th International Conference on Accelerator and Large Experimental Physics Control Systems, WEP021, pp. 447-449 (2009).

[10] Code is available at: https://github.com/JeffersonLab/ced2graph

[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks” 6th International Conference on Learning Representations (2017).

[12] PyTorch Geometric: https://pytorch-geometric.readthedocs.io/en/latest/

[13] P. Velickovic, W. Fedus, W. Hamilton, P. Liò, Y. Bengio, and R. Hjelm, “Deep Graph Infomax” CoRR abs/1809.10341 (2018).

[14] D. Moser e-mail (March, 24, 2022)

[15] “A Browser Based Toolkit for Improved Particle Accelerator Controls” SBIR Phase I project with RadiaSoft, LLC (2021).

[16] D. Moser spreadsheet (February 2022)

Appendix A: Graph File Format

Figure A1 illustrates the four files that are written by the `ced2graph` software and Fig. A2 shows the directory structure. The file format is based on the one used for datasets in Ref. [21]. Together they constitute all the information needed to construct an attributed, heterogeneous graph. They are:

- `info.dat`: defines the node types
- `meta.dat`: lists the number of each node type in the graph
- `node.dat`: lists (node ID, node name, node type, node attributes)
- `link.dat`: lists (starting node ID, ending node ID, edge type, edge weight)

info.dat

```
TYPE    MEANING
0       quadrupole
1       corrector
2       BPM
3       solenoid
4       BCM
```

meta.dat

```
Node Total: Count 11
Node Type_0: Count 3
Node Type_1: Count 2
Node Type_2: Count 3
Node_Type_3: Count 2
Node_Type_4: Count 1
Edge Total: Count 19
Edge Type_0: Count 19
```

node.dat

```
0       Q1      0       3.25
1       CORR1   1       -110.00
2       BPM1    2       -0.8598, 1.6324
3       SOL1    3       1250.00
4       BPM2    2       0.5851, 0.2315
5       BCM1    4       0.12
6       Q2      0       -3.25
7       SOL2    3       1125.00
```

link.dat

```
0       1       0       1
0       2       0       1
0       3       0       1
1       2       0       1
1       3       0       1
1       4       0       1
1       5       0       1
1       6       0       1
```

FIGURE A1: Example showing the four files used to define a single graph.

```
2021 # Year
|-- 01 # Day
|   |-- 03
|   |   |-- 02
|   |   |   |-- info.dat
|   |   |   |-- link.dat
|   |   |   |-- meta.dat
|   |   |   |-- node.dat
|   |   |-- 04
|   |   |   |-- info.dat
|   |   |   |-- link.dat
|   |   |   |-- meta.dat
|   |   |   |-- node.dat
```

FIGURE A2: The directory structure implemented for writing graph data to file. The timestamp is used to create folders according to year, month, day, hour, minute (and second, if need be).

Appendix B: Papers, Proposals, and Presentations

Below is a list of papers, proposals, and presentations related to the LDRD project:

- Manuscript submitted ("Graph Learning for Particle Accelerator Operations") to *Frontiers in Big Data, Data Mining, and Management*, currently under review
- Successful FOA proposal ("Graph Learning for Efficient and Explainable Operation of Particle Accelerators") to continue work and provides funding through FY25
- 2023 Artificial Intelligence for Robust Engineering & Science (AIRES) Workshop, "Graph Embeddings for CEBAF Operations", poster presentation
- 2022 Accelerator Reliability Workshop (ARW), "Toward More Efficient Accelerator Tuning with Deep Learning", presented talk
- 2023 and 2024 Jefferson Lab Accelerator Advisory Committee (JLAAC), highlighted work in talks to external committee and received favorable feedback