Improving Proton Beam Radiotherapy by Classifying Simulated Patient Data in Compton Camera Imaging with Neural Networks

REU Site: Online Interdisciplinary Big Data Analytics in Science and Engineering

Angelo Calingo[1*], Bikash Gautam[2*], Peter L. Jin[3*], Sidhya Pathak[4*], Michelle Zhao[5*],
Hussam Fateen[6], Harrison Lewis[6], Matthias K. Gobbert[6],
Vijay R. Sharma[7], Lei Ren[7], Ananta Chalise[7],
Stephen W. Peterson[8], and Jerimy C. Polf[9]

[1]Department of Computer Science & Engineering, University of Nevada, Reno, USA
[2]Department of Computer Science, Alabama Agricultural and Mechanical University, USA
[3]James M. Bennett High School, Salisbury, MD, USA
[4]Department of Computer Science, University of Virginia, USA
[5]Department of Computer Science, Cornell University, USA
[6]Department of Mathematics and Statistics, University of Maryland, Baltimore County, USA
[7]Department of Radiation Oncology, University of Maryland School of Medicine, USA
[8]Department of Physics, University of Cape Town, South Africa
[9]M3D, Inc., USA

Technical Report HPCF–2025–5, hpcf.umbc.edu > Publications

Abstract

Proton beam radiotherapy is an advanced cancer treatment utilizing high-energy protons to destroy tumor matter. When a proton beam interacts with a patient's body, it emits prompt gamma rays, which are detected by a Compton camera. However, image reconstruction of the beam path from these scatterings is often unusable due to mischaracterized scattering sequences and excessive image noise. To address this, machine learning models were developed to classify the scattering events. Multiple novel robust-volume datasets simulating particle interactions with human tissue were generated using Duke University CT scans and Geant4 and Monte-Carlo Detector Effects (MCDE) software. Novel implementations of a Event Classifier Transformer and a 1D Convolutional Neural Network (CNN) were developed to better address spatial scattering relationships. The prior models, along with a Fully-Connected Neural Networks (FCN) and Long Short-Term Memory Neural Network (LSTM), were optimized through large-scale hyperparameter studies using a new automated tuning framework built into the Big-Data REU Integrated Development and Experimentation (BRIDE) machine learning pipeline. From hyperparameter tuning and larger datasets, FCN and LSTM models achieved significant 17-18% numerical accuracy increases from any previous work. With minimal overfitting, these models offer much greater generalizability. Transformer and CNN models were not as well suited to patient data but still achieved accuracies comparable or greater than previous work.

1 Introduction

As a highly precise form of radiotherapy, proton beam therapy provides a more focused alternative to conventional cancer treatments. In this approach, proton beams deposit most of their energy at a specific depth, known as the Bragg peak, within the treatment area, rather than passing through a large body cross-section as X-rays do. This allows proton beam therapy to minimize the amount of radiation exposure to nearby healthy tissue.

However, to fully achieve the clinical potential of proton therapy, precise real-time tracking of prompt gamma rays is essential. Prompt gammas are emitted as the protons interact with

*These authors contributed equally to this work.

the target tissue. Accurate measurements of beam position relative to the tumor are crucial to ensure effective and safe treatment. To account for uncertainties surrounding beam positioning and external factors such as patient movement, a safety margin is incorporated. The safety margin aims to allow maximum exposure of treatment to the tumor, but could also result in unnecessary radiation to surrounding healthy tissue.

To improve targeting accuracy and allow for tighter safety margins, researchers are exploring real-time imaging techniques based on prompt gamma rays. These techniques allow for the determination of the proton range for the Bragg peak. A current promising method for capturing this radiation is through the use of Compton cameras, which can detect the gamma-ray interactions as the proton beams traverse through the patient's body. A major limitation, however, is the inability of Compton cameras to determine the precise sequence of gamma-ray scattering events. This leads to noise and distortion in the reconstructed images, making it difficult to locate the proton beam within the body, and in some cases rendering the images unusable.

To address the aforementioned challenges, machine learning techniques have been introduced to classify the sequences of the gamma-ray events and improve the quality of reconstructed images. Current studies have demonstrated the applicability of Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) models [5,6], deep Fully Connected Networks (FCNs) [1,21], and hybrid models that combine layers from both architectures. However, past studies relied primarily on Monte Carlo simulations of Water Phantom (WP) datasets, which are based on a constant-density water medium, and, in some cases, included testing on small simulated patient datasets or hybrid WP-patient datasets.

In contrast, for this study, there have been recent developments in data generation of large amounts of new WP and simulated patient data. Through the use of these new datasets, this study improved model accuracies with a particular focus on enhancing accuracy using newly generated large-scale simulated patient datasets, which accurately capture the much more complex tissue densities found in real patients. We conducted extensive hyperparameter testing not only on FCNs and LSTMs, but also on newly developed architectures, including a 1D Convolutional Neural Network (CNN) and an Event Classifier Transformer, to better capture spatial relationships in the data.

The remainder of this report is organized as follows: Section 2 elaborates on background information on proton beam radiotherapy, Compton camera imaging, scatter types, and the motivation for using machine learning models. Section 3 introduces the technical background of the machine learning techniques used in this study and details each specific model. It also discusses the hardware and software used in the experimental setup. Section 4 outlines the data generation methods including the use of the POLARIS J Monte Carlo (PJMC) framework, Geant4, and Monte Carlo Detector Effects (MCDE).This section further explains how this study's novel approach to simulating patient data differs from other existing techniques, describes the datasets that were generated, and highlights their distinctions. Additionally, it covers preprocessing within the Big-Data REU Integrated Development and Experimentation (BRIDE) platform and introduces the newly integrated grid search functionality for efficient hyperparameter tuning. Section 5 presents the results, especially relating to the new simulated patient datasets, the outcomes of hyperparameter optimization for each model using grid search, and the performance of each trained model. Finally, Section 6 summarizes the most significant findings and compares them to previous studies. This section also discusses potential future directions and models that may be promising for further research in this area.

2 Application Background

2.1 Proton Beam Radiotherapy

Radiation therapy is a widely used approach in cancer treatment; proton beam therapy is a more modern form of radiation treatment offers greater precision and safety during treatment as compared to conventional X-ray therapy. In X-ray therapy, high-energy X-rays are used to destroy cancer cells, but the rays deposit their maximum radiation dose shortly after entering the body and continue to release energy as they pass through. This pattern in radiation release causes unnecessary radiation exposure to health tissue surrounding the cancer tumor or cells in the path of the rays before the tumor.

This problem introduces a main advantage of proton therapy, as proton therapy uses charged particles (protons) that interact with tissue in a different manner. As the protons travel through the body, they are able to deposit relatively little amounts of energy until reaching a specific depth where the cancer site is located. There, they release the majority of their radiation, which then quickly drops in dosage as the protons leave the treatment area. This behavior of energy deposition is demonstrated in the Bragg peak. Beyond the Bragg peak, the radiation dose release rapidly decreases to nearly zero, which greatly helps prevent healthy tissue from receiving radiation. This behavior of proton beam therapy as compared to other types of treatment is demonstrated in Figure 1 [10].

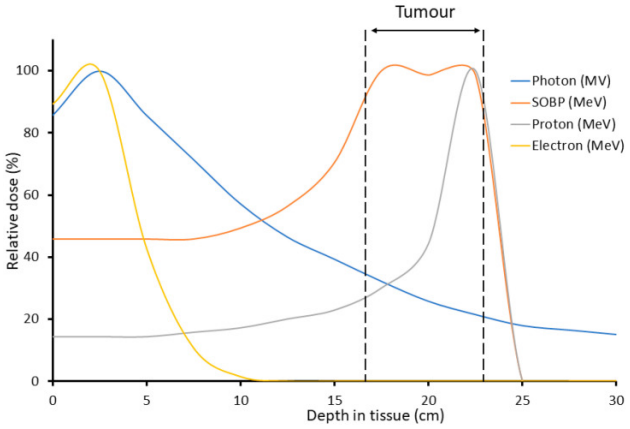


Figure 1: Spread-out Bragg peak (SOBP) relative to other treatment energy types. [10]

Locating the Bragg peak would allow clinicians to target tumors with a high level of precision, which can be especially beneficial for the treatment of cancers that are near critical structures. This is what makes determining the exact location of the Bragg peak so important. However, this is challenging because tumors are embedded in healthy tissue in close proximity to critical structures. Any misalignment can compromise optimal treatment as well as nearby healthy tissue.

As mentioned previously, to effectively utilize proton beam therapy while recognizing the difficulties in locating the Bragg peak, clinicians often create a safety margin. For example, a treatment area that may be larger than the actual tumor area ensures that the entire tumor will receive the dose of radiation. As shown in Figure 2, accurately determining this safety margin is critical. In the figure, the purple outline indicates the intended treatment area for a breast cancer patient. In panel (a), the region enclosed by the yellow outline represents a conservatively estimated treatment volume that incorporates a margin of uncertainty. However, this volume would encompass the heart, as shown in red, and the left anterior descending coronary artery (LAD), as outlined

in white, potentially exposing them to unnecessary radiation and increasing the patient's risk of radiation-induced heart disease. In contrast, panel (b) shows how an optimized safety margin can significantly reduce the dose administered to both the heart and the LAD [17].

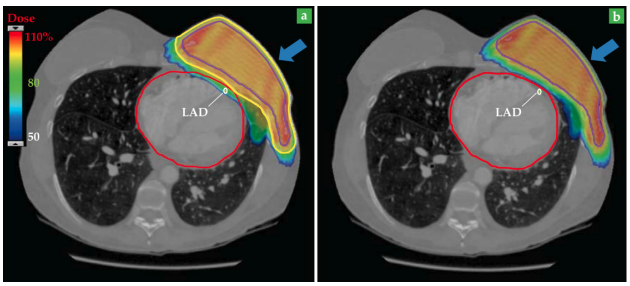


Figure 2: **(a):** Proton beam (blue arrow) targets the tumor (purple) with a safety margin (yellow) that overlaps healthy heart tissue (red) and the left anterior descending (LAD) coronary artery (white). **(b):** Improved safety margin avoids overlap with the heart and LAD [17].

.

Through capturing real-time data on the proton beam's trajectory during treatment, clinicians can more accurately deliver treatment and a narrower safety margin can be implemented as illustrated in Figure 2(b). One currently promising approach to acquire this real-time information is through the use of a detector known as the Compton camera. As the high-energy protons interact with atomic nuclei and the electrons in tissue, they emit prompt gamma rays, which can then be detected by the Compton camera. This provides further insights into the beam's path and thus offers more information on the location of the Bragg peak [17].

2.2 Compton Camera and Image Reconstruction

To monitor proton radiotherapy treatment, prompt gamma radiation can be detected and recorded using a Compton camera, a multistage detector designed to capture and reconstruct visualizations of prompt gamma radiation and associated scattering events. Prompt gamma rays are emitted at a specific angle of displacement, which is determined by the energy levels of the proton's interaction with atomic nuclei. As the prompt gamma rays interact with the detector, the Compton camera calculates the spatial coordinates (x_i, y_i, z_i) and the energy level (e_i) of each scatter. These measurements are used to construct a Compton cone of emission, which projects the potential trajectories that this collision could occur, as shown in Figure 3 [15]. Using this cone of emission, the origin of the gamma is mathematically determined [16].

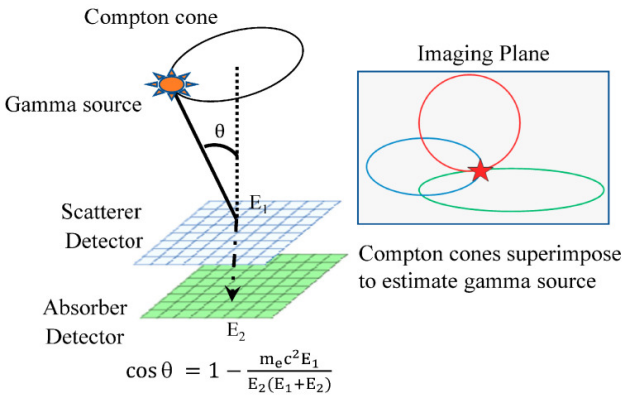


Figure 3: Compton cone of emission. [15]

After scattering events are recorded, image reconstruction algorithms are used to recover a visualization of the proton beam's path. However, a key limitation of the Compton camera is its finite time resolution. Because it cannot explicitly capture the sequential order of the prompt gamma interactions, the true sequence of the events is unknown. This uncertainty introduces noise in the reconstructed images, compromising their reliability and rendering them partially unusable in a medical setting [16].

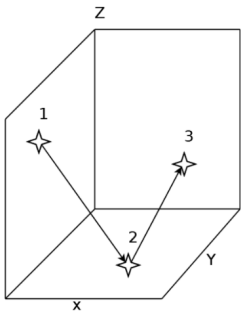
2.2.1 Scatter Types

Prompt gamma radiation is emitted at nearly the speed of light. Because of this, the ordering of the scattering events can be disrupted, introducing background noise into image reconstruction. In order to identify the false events that created the noise within the image, scattering events are classified into 13 distinct types, which are grouped into three categories: true triples, doubles to triples (DtoT), and false triples, as illustrated in Figure 4. The 13 scattering classes are shown by Figure 5.

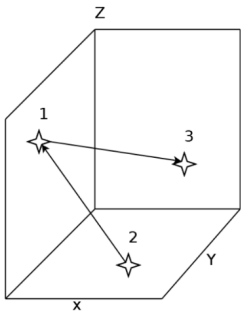
True Triples: True triples involve three sequential interactions with the Compton camera. There are six possible orderings for the interactions: 123, 132, 213, 231, 312, and 321. Out of these, only the 123 sequence accurately reflects the physical interaction order and is usable for image reconstruction purposes.

Doubles to Triples (DtoT): DtoT events consist of one double and one single interaction that occur independently but are mistakenly interpreted as a single triple event. There are six possible orderings for this event: 124, 134, 214, 234, 324, and 314, where the "4" denotes the secondary prompt gamma interaction causing the misclassifications.

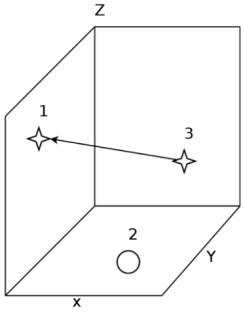
False Triples: False triples are events that appear to be true triples, but are actually composed of three independent gamma interactions. These false events may introduce noise into image reconstruction and must be discarded [4, 12, 16].



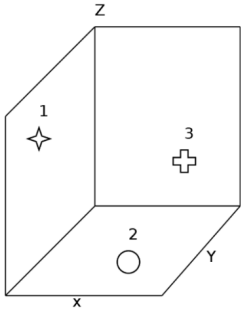
(a) True Triple scatter path of prompt gamma detected by Compton camera.



(b) Possible True Triple scatter path detected by Compton camera.



(c) Possible Double-to-Triple path of prompt gamma detected by Compton camera.



(d) False triple path of prompt gamma detected by Compton camera.

Figure 4: Visualization of the types of scattering events [7].

Class	Interaction 1				Interaction 2				Interaction 3			
123	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2	e_3	x_3	y_3	z_3
132	e_1	x_1	y_1	z_1	e_3	x_3	y_3	z_3	e_2	x_2	y_2	z_2
213	e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1	e_3	x_3	y_3	z_3
231	e_2	x_2	y_2	z_2	e_3	x_3	y_3	z_3	e_1	x_1	y_1	z_1
312	e_3	x_3	y_3	z_3	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2
321	e_3	x_3	y_3	z_3	e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1
124	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2	single			
214	e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1	single			
134	e_1	x_1	y_1	z_1	single				e_2	x_2	y_2	z_2
314	e_2	x_2	y_2	z_2	single				e_1	x_1	y_1	z_1
234	single				e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2
324	single				e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1
444	single				single				single			

Figure 5: The 13 distinct types of scatterings in the data. [6].

2.2.2 The Need for Machine Learning

Real-time imaging is vital when it comes to verifying the location of the Bragg Peak in order to increase the safety of proton beam therapy. There have been many classical methods that were created and used as a de-noising attempt to enhance the reconstructed images by applying filters to decrease the blurring and artifacts. In [13], the distance from an image pixel's center to the Compton cone was computed in order to obtain threshold values. These threshold values were applied to each of the pixels values to reduce the noise and give a much cleaner image. However, filters in particular are limited in that they may not be able to adequately capture complex data patterns and hard-to-identify false events.

In contrast, machine learning in computer vision is able to identify and obtain much more complex patterns than a filter can. Machine learning models have the capability of classifying the various scatter events from the Compton camera data. Once the false events are found, they can be removed, which will result in a much cleaner image that can be used towards verifying the location of the Bragg peak.

3 Machine Learning

3.1 Technical Background on Machine Learning

Machine learning involves using algorithms to analyze data, identify patterns, and perform predictions. The main type of machine learning used in this research is Neural Networks (NN), due to their capability to learn deep patterns from the data and provide knowledgeable insights through predictions. We use supervised learning in that the NN is trained on labeled data with a specific variable of interest: the class. The model then predicts this class from training by finding relationships between it and the predictor variables.

Neural networks are designed to mimic the human brain's neuron connections for adapting to new information. They are made up of layers of neurons that each have a set of weighted inputs from the layer before it and give outputs by using a nonlinear activation function into the next layer of neurons. During training, the NN updates the weights of the matrices throughout the layers in order to control the scaling and shifting of the activation function outputs.

The combination of these scaling and shifting matrices that affect the transformations of input data through the use of activation functions allows NN to approximate any continuous function that deals with relationships between a data feature and its output, which in our case is the class of a specific scatter type. This concept is known as the Universal Approximation Theorem [8]. However, there is no guarantee that a certain training algorithm is able to find the exact weights and functional compositions necessary to show a perfect relationship between the input and output data. Still, the optimizer and loss function are designed to approximate these theoretical relationships to a numerical precision.

The four main types of neural networks applied in this work are Fully-Connected Neural Networks (FCN), Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Event Classifier Transformers.

3.1.1 Fully-Connected Neural Networks

A Fully-Connected Neural Network is a simple deep learning model that handles information going one direction coming from the input and passing through the hidden activation layers to finally reach the output layer [19]. This process is seen in Figure 6. In a fully-connected layer, every neuron in a layer is connected to every neuron in the following layer; in other words, the output

for a layer is the input for the next layer. FCNs can be used in tasks that deal with classification and regression where the data structure's spatial and temporal dependencies are not crucial to the task at hand. In addition, they are suited for learning complex relationships due to an ability to have a large number of parameters. The model is trained using a custom loss function that adds a penalty to discourages misclassification outside of the scattering group (true triple, DtoT, or false triple) more than misclassification purely outside of the wrong class.

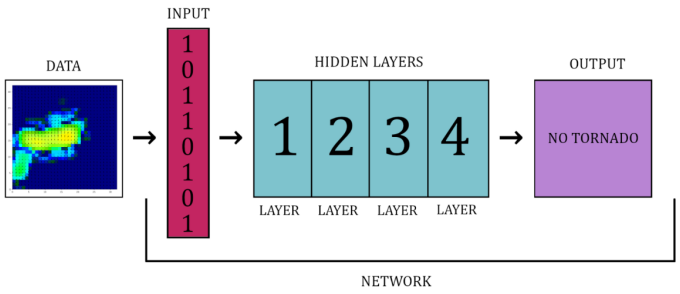


Figure 6: An example structure of a Feed Forward Neural Network [1].

3.1.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are a type of neural network that handles sequential data. RNNs have recurrent connections that allow them to hold onto information while moving through time steps in order to obtain the dependencies. They have a hidden memory state that updates when new information is given, while still keeping track of the previous input [20]. However, RNNs can be sensitive to changes in hyperparameters and have a likelihood in running into vanishing and exploding gradients. Previous studies have showed that the FNN and RNN models perform well on simulated data [19]. Clark et al. reported on a finding of 74.6% accuracy for a water phantom-based scattering classification problem when using Tensorflow-implemented Long-Short Term Memory (LSTM) model. In addition, many other recurrent models were investigated in previous work with the goal of attempting to obtain the clinical standard of 90% or greater model testing accuracy [7].

Long-Short Term Memory (LSTM) Neural Network A Long Short Term Memory (LSTM) is a type of RNN that is specifically designed to handle long-term dependencies. LSTMs use input, forget, and output gates to handle memory as shown in Figure 7 [20]. These gates are part of a component called a memory cell; a memory cell is able to store information over a period of time as it may be needed in the later stages of training. LSTMs were thought to do well with our datasets since the scatter interactions are sequential. In this research, the LSTM that was implemented gives the LSTM output to the fully connected layers to perform predictions. The entire layout of this architecture is shown in Figure 8, which shows how one of the 13 scatter types is predicted. The model is trained using a custom loss function that adds a penalty to discourages misclassification outside of the scattering group (true triple, DtoT, or false triple) more than misclassification purely outside of the wrong class.

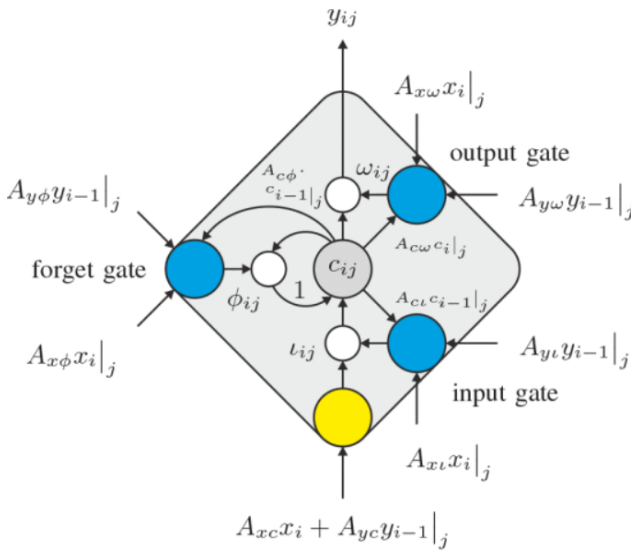


Figure 7: Gate structure of an LSTM model [20].

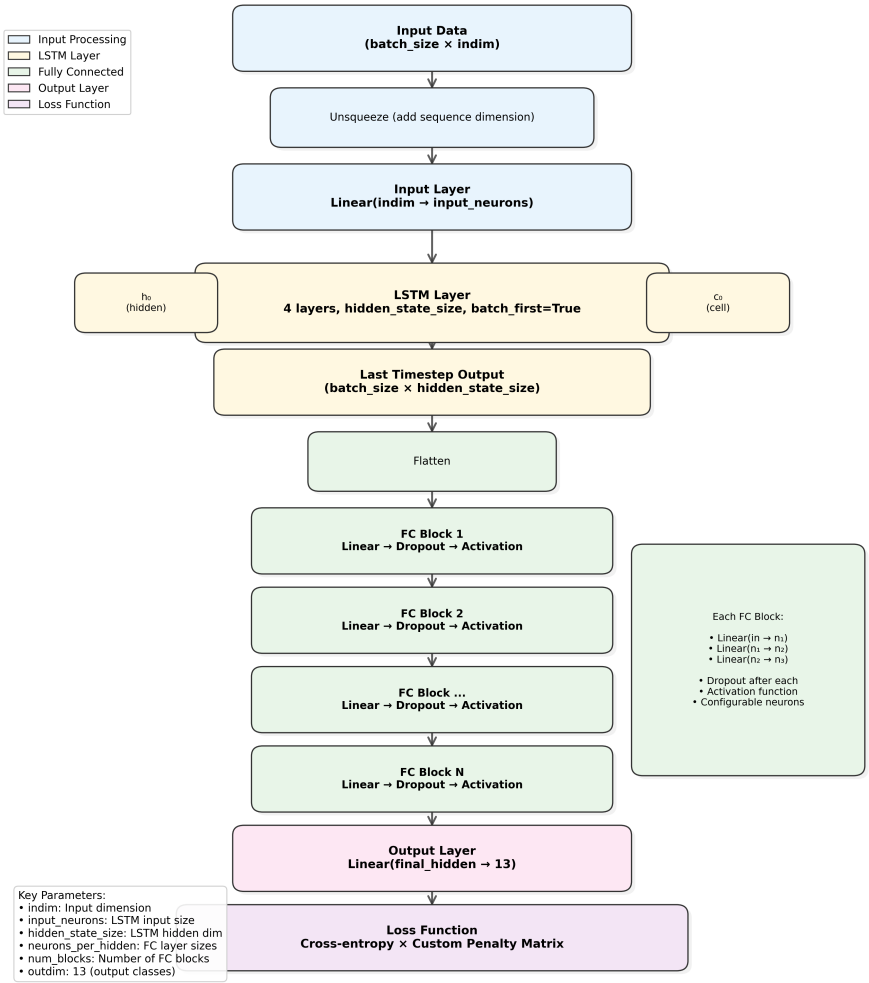


Figure 8: Architecture of ImprovedLSTM2024 for 13-class scattering classification.

3.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a class of deep learning models originally designed for image recognition; however, they are also widely used in a variety of tasks involving structured grid-like data. CNN models leverage local patterns in data using convolutional layers that apply a set of learnable filters (kernels) across the input. Each filter captures specific features of the input space, including edges, curves, or more abstract patterns depending on the depth of the network. Unlike FCNs, which connect every input to every output, CNNs preserve spatial locality and drastically reduce the number of parameters, making them efficient and suitable for high-dimensional input data.

CNNs typically consist of convolutional layers followed by non-linear activation functions, pooling of layers for downsampling, and fully-connected layers for classification. The strength of CNN models lies in their ability to automatically and adaptively learn spatial hierarchies of features, enabling them to extract useful representations from data.

One-Dimensional Convolutional Neural Network (1D CNN) A One-Dimensional Convolutional Neural Network (1D CNN) is a variant of a CNN designed to process sequential or structured data with a single spatial dimension, such as time series or ordered feature vectors. Unlike traditional 2D CNNs that operate on images, 1D CNNs apply convolutional filters along one axis to capture local patterns across adjacent features.

In our case, the input to the 1D CNN is a fixed-length vector of 15 values representing the spatial and energy features from three detected gamma interactions:

$$(e_1, x_1, y_1, z_1, \mathrm{euc}_1,\ e_2, x_2, y_2, z_2, \mathrm{euc}_2,\ e_3, x_3, y_3, z_3, \mathrm{euc}_3).$$

The 1D CNN learns to identify meaningful local features such as energy differences, spatial distances, or alignment patterns relevant to scattering classification. The model consists of seven convolutional layers with ReLU activation, followed by max pooling and fully connected output head.

Figure 9 shows the full architecture of our novel 1D CNN model. It includes seven stacked Conv1D layers with progressively increasing channels and kernel sizes, followed by max pooling, flattening, and dense layers. The final output is a 13-class softmax prediction representing the scatter type. The architecture is specifically designed to balance depth and resolution, allowing the model to learn spatial and energetic patterns efficiently from the structured input. The model is trained using a custom loss function that adds a penalty to discourages misclassification outside of the scattering group (true triple, DtoT, or false triple) more than misclassification purely outside of the wrong class.

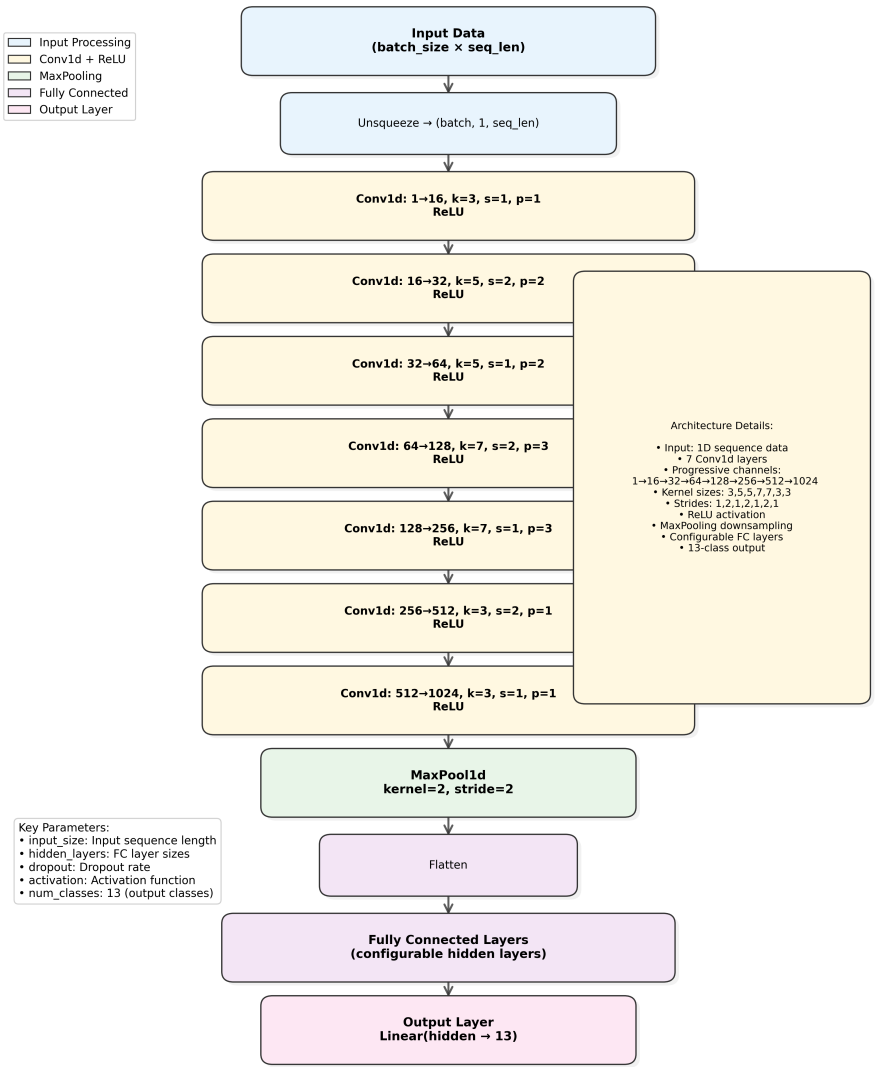


Figure 9: Architecture of the novel implementation of a 1D CNN for 13-class scattering classification.

3.1.4 Event Classifier Transformer

Transformer neural networks are the leading architecture in natural language processing and have been successfully adapted for particle physics [9, 11]. This motivated us to experiment with using transformers for event classification in our studies.

Traditional neural networks, like Multi-Layer Perceptrons (MLPs), often process input features as a simple, flat vector. This approach can make it difficult to learn complex non-linear interactions simultaneously. Our goal was to better capture interactions between features across the full event representation. Instead of processing the 15 input features as a flat vector, we treated each scalar value as an individual token. These tokens were embedded separately and processed as a sequence through several transformer encoder blocks. Each block used self-attention and feedforward layers with residual connections, allowing the model to learn global relationships between all features.

To produce a final classification result, a learnable class token is prepended to the processed feature tokens. This class token attends to the entire sequence to create a summary representation, which is then passed to a linear layer to generate a class prediction. The model is trained using

a custom loss function that adds a penalty based on a predefined distance matrix; this penalty discourages misclassification outside of the scattering group (true triple, DtoT, or false triple) more than misclassification purely outside of the wrong class.

This architecture is designed to handle structured data by leveraging the power of self-attention to model long-range dependencies between features, a task where traditional architectures can struggle. The complete model design is shown in Figure 10.

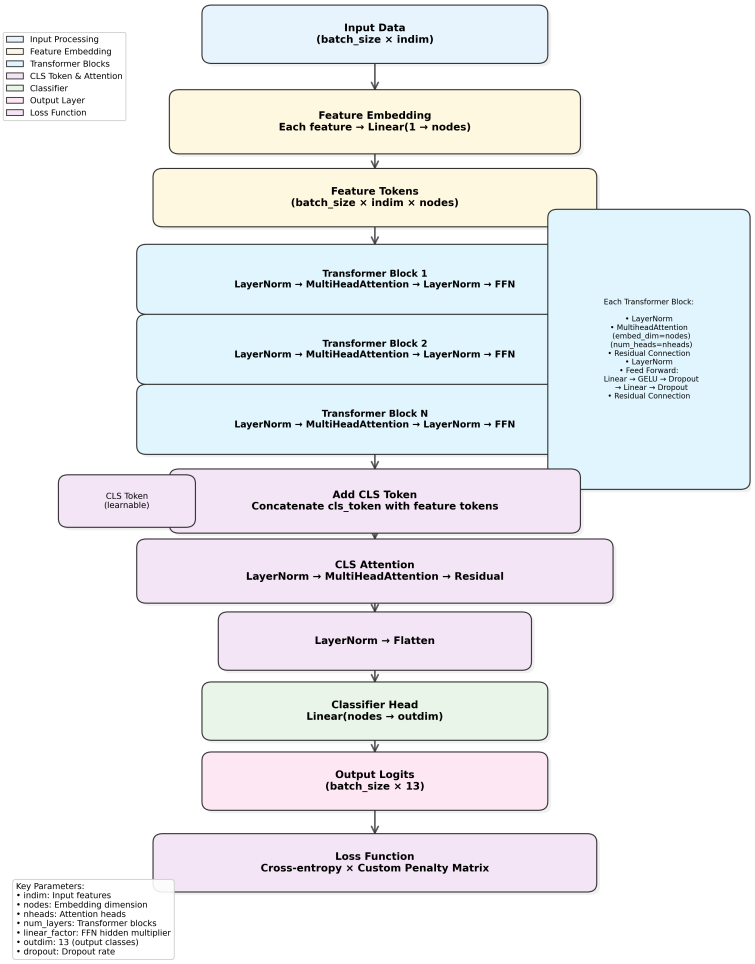


Figure 10: Architecture of the Event Classifier Transformer Neural Network for 13-class scatter classification.

3.2 Related Works

In recent years, significant progress has been made in the application of machine learning to image reconstruction from prompt gamma radiation. Four relevant works are the research in [1], [7], [14], and [5].

The work in [1] focuses on optimizing deep fully connected neural networks (FCNs) for image reconstruction. The authors demonstrated that FCNs have a reasonable ability to classify events across various classes. The work was based on the relatively simpler, constant-density water phantom data. They showed a greatest accuracy of around 75% by conducting a comprehensive hyperparameter study involving 288 experiments. These studies identified promising configurations that could lead to better performance in future applications. There is also a strong emphasis on

developing more compact neural networks that might achieve similar or superior performance while utilizing fewer parameters. RNNs were briefly tested. It is important to note that [1] utilized a *residual* architecture for their FCNs. This means they were able to train very deep networks with hundreds of layers without running into exploding gradients or other such problems. The best FCNs *without* residuals, which [2] explored, achieved a peak accuracy of 63.9% on water phantom data.

The research in [7] had a primary contribution related to RNNs. While deep residual FCNs achieved slightly higher accuracy, RNNs demonstrated comparable performance with a key advantage: they have a simpler architecture and fewer hidden layers. This allows for faster loading times and enhances usability. This also makes it more efficient for real-time applications in a clinical setting. In [5], they implemented a Long Short Term Memory (LSTM) model (their best performing model) and had a testing accuracy of 55.6% on the complex simulated patient data. However, their simulated patient dataset was quite small at only 500,000 rows (as compared to their water phantom or hybrid datasets of multiple million rows).

To build on the research [5], this work looks to improve upon their accuracy results. Their research was limited due to time constraints on hyperparameter tuning, and, as they noted, a small simulated patient dataset size. In this work, we aim to run a large breadth of hyperparameter studies on realistic simulated patient datasets in order to increase model accuracy, which did not approach clinical standards in previous work; these datasets will be much larger than previous patient datasets. We are also looking into exploring other neural network architectures including transformers and CNNs to potentially increase the overall model accuracy.

3.3 Hardware and Software

For this research, we utilized the Graphics Processing Units (GPU) in the chip cluster maintained by the UMBC High Performance Computing Facility (`hpcf.umbc.edu`). The GPU portion of the chip cluster consists of four nodes with eight RTX 2020Ti GPUs (11 GB of GDDR6 memory each), seven nodes with eight RTX 6000 GPUs (24 GB of GDDR6 memory each), two nodes with eight RTX 6000 GPUs (48 GB of GDDR6 memory each), two nodes with two H100 GPUs (100 GB of HBM3 memory each), and ten nodes with four L40S GPUs (48 GB of GDDR6 memory each).

The machine learning models were built and implemented using PyTorch v2.3.1 (`https://PyTorch.org`). For data preprocessing and manipulation, we used scikit-learn v1.3.0 (`https://scikit-learn.org/stable/`), pandas v2.2.2 (`https://pandas.pydata.org/`), and numpy v1.26.4 (`https://numpy.org/`). To visually display our results, we used matplotlib v3.8.4 (`https://matplotlib.org/`) and seaborn v0.13.2 (`https://seaborn.pydata.org/`). Our models were built inside of the python environment Anaconda3 (`https://www.anaconda.com/`).

4 Methods

Figure 11 shows the overall method pipeline used in this study. It outlines the full process from gamma simulation to model evaluation, including preprocessing, training, and hyperparameter tuning. Each stage of the pipeline is discussed in more detail in the following subsections of this Methods section.

Figure 11: Overview of the full ML pipeline from prompt gamma simulation to model evaluation.

4.1 Polaris J Monte-Carlo (PJMC)

The POLARIS J Monte Carlo (PJMC) framework is a custom simulation-based tool used to generate the high-fidelity data required for this study. PJMC functions as a wrapper for the Geant4 toolkit, automating the entire process of simulating charged-particle transport and detector response. Its primary purpose is to model prompt gamma emissions from a water phantom during proton therapy and simulate their detection by a novel, snout-mounted Polaris J3 Compton Camera (PJ3CC) array.

Previous water phantom data generation often involved backward-facing detectors positioned underneath the patient. Our current data, in contrast, simulates detectors placed at the snout, directly facing the proton beam's entry point. This novel configuration offers several advantages, including a potentially higher signal-to-noise ratio due to the closer proximity of the detectors to the proton interaction site, and a more direct view of the gamma-ray emissions originating from the Bragg peak.

To generate a comprehensive dataset, simulations were executed across a range of clinically relevant proton beam energies: 75 MeV, 110 MeV, 140 MeV, 150 MeV, 160 MeV, 180 MeV, and 190 MeV. For each energy level, the beam was laterally displaced to multiple positions within the phantom at x = 25 mm and y = -25, -15, -5, 5, 15, and 25 mm. This multi-parameter approach ensures our dataset is representative of a variety of clinical scenarios. The simulations were run on an array of 360 processes, thus we obtained 360 data files in `ROOT` format, which were combined into a single file after checking for any `Zombie` files, using the `hadd` command, which is used on `ROOT` data analysis framework to merge multiple `ROOT` files into a single output file, which we called `combined.root`.

We specifically chose these parameters to form a detailed contour of the Bragg peak and its surrounding region. This contour is crucial for accurately correcting for patient movement, internal organ deflection, and other unpredictable factors that can lead to inaccurate reconstruction of the proton range and dose deposition. By generating a high-resolution map of the expected gamma-ray emissions, we can better identify and compensate for discrepancies during real-time image guidance. The contoured dataset was created by concatenating the combined root files for various energies and spot locations, which were then processed by Python scripts to output `csv` files.

4.2 Geant4

A virtual experimental setup is designed by incorporating patient CT images from Duke University into the Geant4 simulation toolkit. As per design, a Compton camera detector placed below the patient bed captures prompt gamma events from the patient [16]. Data generation has been carried out at different energies and locations, referred to as layers and spots, respectively. For instance, layer 1 denotes a proton beam energy of 198.7 MeV, consisting of 16 spots. The data is stored in a tree named "scatterData", with 11 tuples packed in ROOT format. Furthermore, the prompt gamma events captured by the Compton camera are organized based on their energy deposition, distinguishing events that originate from positron annihilation (511 keV), and events from singles, doubles and triples.

Each spot generates two types of data files: Prompt Gamma Emissions and 511 keV Annihilation Gammas. Prompt Gamma Emissions result from the proton interactions with patient anatomy, while 511 keV Annihilation Gammas are produced when positrons annihilate with electrons, emitting gamma rays at 511 keV.

4.3 Monte-Carlo Detector Effects (MCDE)

The prompt gamma interactions generated by the Geant4 software are then fed into the Monte-Carlo Detector Effects (MCDE) model. MCDE conducts processing of the data by adding detector timing and trigger effects to make the simulated Geant4 data more similar to actual data captured by the Compton camera. In previous literature, it has been shown that MCDE provides a realistic simulation of the actual effects of the Compton camera, rendering the Geant4 simulated data usable as a similar representation of data captured during the clinical proton beam therapy process by the Compton camera [12].

The input of MCDE includes the Prompt Gamma Emmissions and 511 keV Annihilation Gamma files generated by Geant4, as previously mentioned. After processing by MCDE, the user receives text file outputs of singles, singles' initial energies, doubles, dType, doubles' initial energies, triples, tType, and triples' initial energies. These outputs are dependent on the parameters given to the MCDE model, such as beam energy, dosage rate, and irradiation time. Only the triples scatter data is usable for our machine learning process. This generated data from MCDE has 12 columns consisting of 3 sets of energy and associated spatial coordinates (e_1, x_1, y_1, z_1). Due to the nature of scatterings, we cannot directly feed the entirety of its generated data into the machine learning models. As we can only use the triples data, through preprocessing with the BRIDE platform as further discussed below, we need to discard doubles and singles data. Additionally, MCDE does not generate the same proportion of different classes of data: false triples, DtoTs, and true triples vary at different proportions in the data. We can balance this data in further preprocessing steps as discussed below. Importantly, MCDE can only be run with integer dosage rate values.

An important distinction is that for our novel simulated patient dataset of 1.1 million rows (`patient_2025` based on patient CT measurements as discussed in Section 4.4, MCDE was run on each Geant4 energy layer separately. The MCDE energy layers with their beam energies can be seen in Table 1. However, for our novel simulated patient dataset of 4.1 million rows (`al_patient_2025`, MCDE was run after all Geant4 energy layer data files were combined together. This was with a constant beam energy of 100 MeV.

Layer	**Beam Energy (MeV)**
1	199
3	195
5	191
7	186
9	182
11	178
13	174
15	170
17	166
19	162

Table 1: MCDE Parameters for the `patient_2025` and `al_patient_2025` simulated patient data.

4.4 Data

The data we worked with on this project were of two types: water phantom datasets and simulated patient datasets. The process of generating the water phantom dataset is elaborated on in Section 4.1. The process of generating the simulated patient data is covered in Sections 4.2 and 4.3. Below,

we introduce the physical datasets that were introduced or used in this work.

1. `patient_2025`: this novel dataset is based on patient CT measurements and was generated by processing data through MCDE at 1 kMU, 20kMU, 40kMU, 60kMU, 140kMU, and 180kMU dosage rates at the variable proton beam energies shown in Table 1. MCDE was run on each Geant4 energy layer file separately. It contains approximately 1.1 million observations.

2. `al_patient_2025`: this novel dataset is based on patient CT measurements and was generated at 1 kMU, 20kMU, 40kMU, 60kMU, 140kMU, and 180kMU dosage rates at a constant proton beam energy of 100 MeV. MCDE was run after all Geant4 data files of varying energy layers were combined together. This dataset contains around 4.1 million observations.

3. `shakeri-obe`: this dataset is based on patient CT measurements. The data was generated by previous researchers in [6] and contains around 500,000 observations.

4. `barajas`: this is data based on a constant-density water phantom. It was created by the work in [3] with dosage rates of 20kMU, 100kMU, and 180kMU with a 150MeV proton beam. It contains around 1.8 million observations.

This work introduced the `patient_2025` and `al_patient_2025` simulated patient datasets based on actual patient CT measurements. These datasets are of distinction as they are around 2 times and 8 times greater than the simulated patient data used in previous research, respectively [6].

4.5 BRIDE Platform

The Big-Data REU Integrated Development and Experimentation (BRIDE) platform is a data and machine learning development platform that ensures reproducibility, expandibility, and rigorous testing methods; it was originally developed by the researchers in [6]. It contains the processes of data preprocessing, model implementation, model tuning, model testing, and model selection. The platform has modularity in that each process is standalone from other processes. Importantly, BRIDE allows for proper separation of training, validation, and tests sets while using parallelization for computing and prevents data leaks. In this research, we expanded on the BRIDE platform to enhance its effectiveness and usefulness.

The structure of the BRIDE platform is contained in Figure 12. BRIDE consists of multiple distinct parts. First, raw data output files of triples, tType, triples' initial energies, doubles, dType, doubles' initial energies, singles, and singles' initial energies are inserted into the **Raw Data** directory of BRIDE. Through **pp1** preprocessing, the raw data is preprocessed to include only triples, creating a final dataset CSV file through the creation of class labels for true triples, dtoTs, and false triples. Then, the data CSV file is fed into **pp2**, which performs a train, validation, and test set split; it also separates the training columns and the target variable for the machine learning model process. After the processes in pp2, the data is utilized to train and test the machine learning models. Model checkpoints and output logs containing training and validation metrics are outputted in the **logs** directory. The model is then tested on a held-out test set, producing metrics and figures in the **eval** directory.

To run on BRIDE, a directory named `base/` in a Linux-like file system must contain the entire BRIDE structure. Each distinct model run must have a unique `run_id`; the recommended naming scheme for the `run_id` is `initialsMonthDigits.DateDigits.YearDigits_id-string`. Now, in `base/`, there are various directories:

- `base/config/`: contains `run_id`.yaml files, each of which is the configuration for one run.

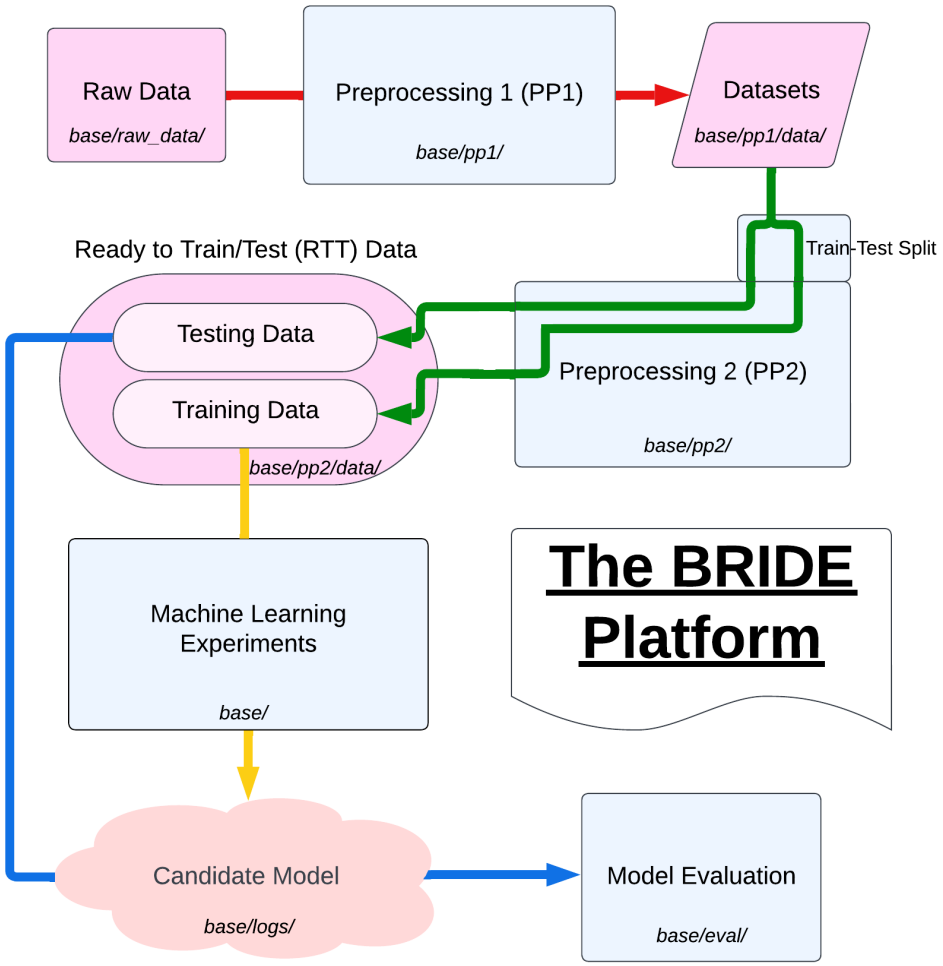


Figure 12: Flowchart of the structure of the BRIDE platform.

- `base/runs/`: contains `run_id` directories, each of which has a `train.sh` script for machine learning experiments, a `predict.sh` script for model evaluation, and a `slurm_output` directory for slurm output files.
- `base/logs/`: holds the logs of each run, including CSV logs and model checkpoints.
- `base/eval/`: contains `run_id` directories, each of which has files from model evaluation including test set metrics and confusion matrices. It also contains a `eval.py` file that conducts the model evaluation.
- `base/raw_data/`: contains folders for the raw post-MCDE data files.
- `base/pp1/`: contains Python preprocessing scripts and a `pp1.sh` file to conduct processing of post-MCDE data into a trainable CSV file. Intermediate files from the processing rest in an intermediate-output folder. In particular, pp1 processing performs merging, shuffling, optional class balancing, and data normalization. In the `pp1` folder, there is also a `visualize.sh` and a `visualize.py` file to conduct visualizations of the data distribution.

- `base/pp2/`: contains Python scripts and `pp2.sh` to perform a train-test split of the data. Ready-to-train data is held in the `data` folder, and each dataset contains training and testing data folders.
- `base/grid_search/`: holds the Python and YAML templates for each model for conducting grid searches.

In order for BRIDE to function, the `base/` directory must include the following Python scripts:

- `model.py` contains the Python code of the models.
- `train.py` contains the Python code for the training of the models.
- `predict.py` contains the Python code to do model evaluation.
- `utils.py` contains all auxiliary and utility Python code.

After pp1 and pp2 preprocessing, the resulting structured dataset contains 15 numerical input features per event. These features capture spatial and energy properties of the three detected gamma interactions. These features are seen in Table 2.

Feature Category	**Description**
Energy Features	e_1, e_2, e_3: Energy deposited by gamma interactions 1, 2, and 3 respectively (MeV)
	Total of 3 features representing energy deposition at each interaction point
Spatial Coordinates	x_1, y_1, z_1: Spatial coordinates of hit 1 in detector frame (mm)
	x_2, y_2, z_2: Spatial coordinates of hit 2 in detector frame (mm)
	x_3, y_3, z_3: Spatial coordinates of hit 3 in detector frame (mm)
	Total of 9 features representing 3D positions of interaction points
Distance Features	euc_1, euc_2, euc_3: Euclidean distances between interaction points (mm)
	Total of 3 features capturing geometric relationships between hits
Total: 15 input features per prompt gamma scattering	

Table 2: 15 input features capturing energy, position, and geometry of gamma interactions.

4.5.1 Grid Search

While testing out models and tuning hyperparameters to improve their performance, we encountered a significant inefficiency: hyperparameter tuning was being done manually for every single combination. Previous years' teams had typically adjusted one parameter at a time, observed the results, then adjusted and trained another set of parameters. This was a process that was both

time-consuming and suboptimal. Given that some models had over 10 hyperparameters, even testing just three values for each would result in over 59,000 combinations. Manually exploring this search space was impractical. To address this and streamline the process, we developed a Grid Search script to automate and accelerate the hyperparameter tuning process.

In our Grid Search script, we first define a hyperparameter grid by creating a dictionary corresponding all the hyperparameters with their possible values. Using `sklearn.model_selection`, `ParameterGrid`, our script generates every possible combination of the parameters. For each hyperparameter configuration, the script copies a base YAML configuration template, updates it with the different combinations of hyperparameters, and saves it as a new config file.

A corresponding SLURM script is generated to train the model using these parameters. The script creates and stores each run into a new subdirectory within the `runs/` folder named according to a unique runID. The script then submits each SLURM job to the cluster using `subprocess.run(['sbatch', slurm_script])`, which allows the cluster to run many jobs in parallel. Lastly, a `used_hyperparams.yaml` is saved in each run directory to allow for tracking of the specific hyperparameters used in that run.

5 Results

5.1 Simulated Patient Data

As stated in Section 4.4, this research generated two novel simulated patient datasets: one of 1.1 million rows and the other of 4.1 million rows. These datasets are around 2x and 8x as large as the patient data used in previous research, respectively [6]. As larger datasets usually lead to an increase in model robustness and generalizability up to a certain point, we trained machine learning models on these novel datasets.

5.2 Patient_2025 Hyperparameter Studies

In order to increase the generalizability of machine learning models on the novel simulated patient datasets, we conducted multiple hyperparameter studies with different models on the 1.1 million row `patient_2025` dataset.

5.2.1 LSTM Hyperparameter Study

We conducted a hyperparameter study on the 1.1 million row `patient_2025` dataset using an LSTM model from [6]. This study was conducted using the automated grid search mechanism developed in this work.

For this grid search, 18 combinations of hyperparameters were tested. We varied the neuron configurations of the fully-connected layers, dropout, and learning rate, as seen in Table 3. The constant hyperparameters of the runs are also shown in Table 4. For the studies, we ran for a total of 4000 epochs and set the patience for early stopping to 1500 epochs. Model checkpoints were saved every 400 epochs to allow for model evaluation at various locations in the training process.

Hyperparameter	Value
Neuron Configuration	[128,128,128,128], [128,64]
Dropout	0.05, 0.15, 0.3
Learning Rate	0.001, 0.0005, 0.0001

Table 3: LSTM grid search candidate hyperparameters.

Hyperparameter	Value
Hardware	4 rtx6000 GPUs
Validation Split	0.1
Batch Size	256
Learning Rate Gamma	0.1
Learning Rate Step	1000
L2	0.0000001
Loss Function	CrossEntropy + Custom Pairwise Loss (p=1)
Optimizer	Adam
Activation Function	ReLU
Max Epochs	4000

Table 4: LSTM grid search constant hyperparameters.

The results of this LSTM hyperparameter study are shown in Table 5. As seen, the best model with the hyperparameters shown in Table 6 achieved a maximum training accuracy of 73%, a maximum validation accuracy of 70%, and a testing set accuracy of 71%. The training and validation accuracy and loss curves of the best model and the confusion matrix on the testing set are shown in Figures 13 and 14, respectively.

In general, from this hyperparameter study, we found that having two fully-connected layers with a neuron configuration of [128, 64] usually led to an increased accuracy of around 3-5% compared to the 4-layer neuron configuration. However, the best run had 4 fully-connected layers with a neuron configuration of [128, 128, 128, 128] suggesting that this 4-layer configuration is best with the specific learning rates and dropout values studied. The best model's hyperparameters suggest that having a small dropout value and a small learning rate leads to the best accuracy with a minimal amount of overfitting, as only a 3% numerical difference between training and validation accuracy was observed.

Our achieved 71% testing accuracy is particularly notable for this patient data process. This accuracy result is a significant 15% numerical increase from the highest accuracy achieved on simulated patient data in any previous work [6]. As simulated patient data is based on variable-density patient tissue and thus may be more realistic compared to constant-density water phantom, a large accuracy increase is especially notable.

Test	Max Train Acc.	Max Val. Acc.	Test Acc.
1	0.7445	0.6226	-
2	0.7466	0.6283	-
3	0.7336	0.7039	0.71
4	0.7366	0.6382	-
5	0.7445	0.6328	-
6	0.71	0.6128	-
7	0.6804	0.588	-
8	0.71	0.6128	-
9	0.7126	0.663	-
10	0.7364	0.6069	-
11	0.7749	0.6579	-
12	0.7429	0.7011	-
13	0.7341	0.6159	-
14	0.7507	0.6464	-
15	0.7305	0.692	-
16	0.7209	0.609	-
17	0.754	0.667	-
18	0.7351	0.693	-

Table 5: LSTM patient_2025 hyperparameter study results.

Hyperparameter	Value
Neuron Configuration	[128,128,128,128]
Dropout	0.05
Learning Rate	0.0001

Table 6: LSTM patient_2025 hyperparameter study best hyperparameters.

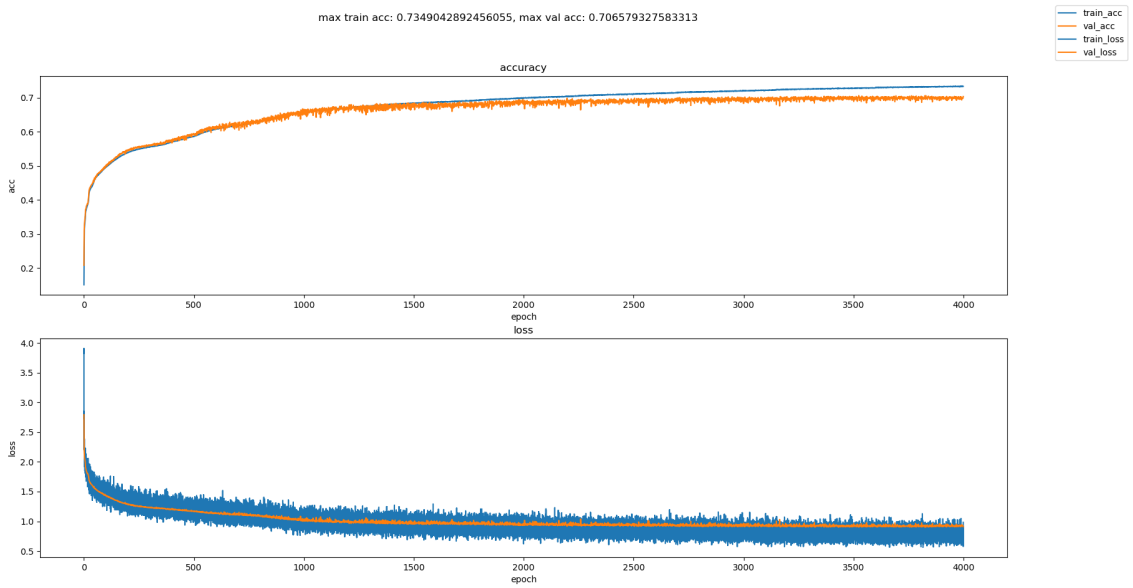


Figure 13: Training and validation accuracy and loss curves for the best LSTM model on patient_2025 hyperparameter study.

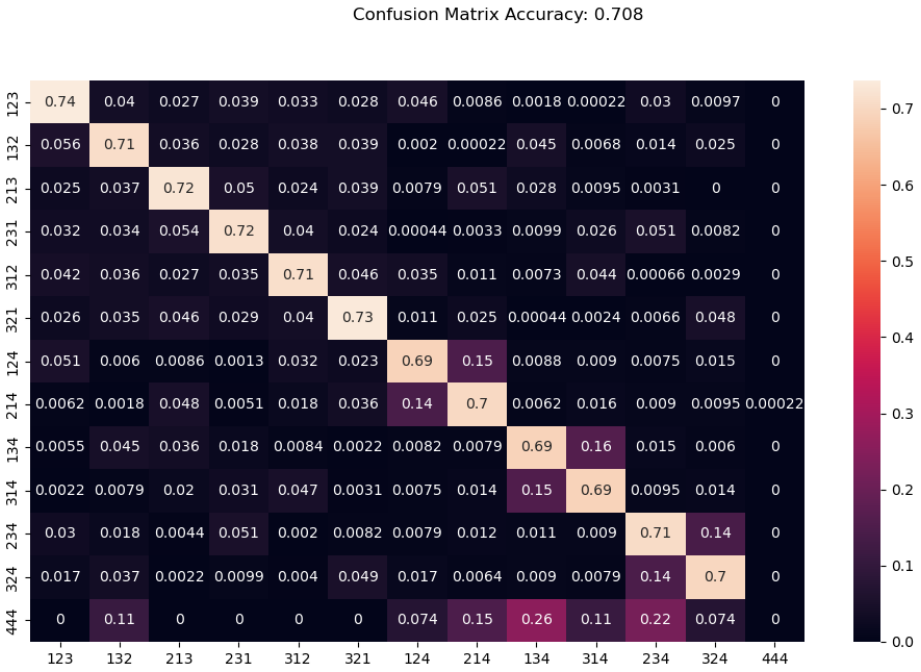


Figure 14: Confusion matrix on the test set for the best LSTM model on patient_2025 hyperparameter study.

5.2.2 FCN Hyperparameter Study

We implemented a hyperparameter study for the fully-connected neural network model from [6] on the 1.1. million row `patient_2025` dataset. The study was conducted using the grid search mechanism on BRIDE.

With this grid search, 18 combinations of hyperparameters were tested. The neuron configuration of the hidden layers, starting learning rate, and dropout rate were varied as in Table 7. The constant parameters of the search are shown in Table 8. Models ran for a maximum of 4000 epochs with early stopping set to 1500 epochs. The results of the FCN hyperparameter study are shown in Table 9. The model with the best accuracy had a training accuracy of 72%, a validation accuracy of 73%, and a testing accuracy of 73%. The training and validation accuracy and loss curves of the model and the test set confusion matrix are shown in Figures 15 and 16, respectively. The hyperparameters of this best model can be found in Table 10.

From this hyperparameter study, we found that having a shallower model led to better accuracy results. The 4-layer FCN model performed much better than the 8-layer FCN model, suggesting that a greater number of layers led to overfitting and a poorer generalization performance. The difference between these two architectures' accuracy results was large (around 20% numerically). We found that dropout and learning rate did not have as great of an impact on performance, while the impact of learning rate was negligible too. There was practically no overfitting for all of the FCN models, likely due to the dropout implemented.

Hyperparameter	**Value**
Neuron Configuration	[512, 256, 256, 256, 256, 256, 256, 128], [256, 256, 128, 128]
Dropout	0.05, 0.2, 0.4
Learning Rate	0.001, 0.0005, 0.0001

Table 7: FCN grid search candidate hyperparameters.

Hyperparameter	**Value**
Hardware	4 rtx6000 GPUs
Validation Split	0.1
Batch Size	256
Learning Rate Gamma	0.9
Learning Rate Step	100
L2	0.01
Loss Function	CrossEntropy + Custom Pairwise Loss (p=1)
Optimizer	Adam
Activation Function	ReLU

Table 8: FCN grid search constant hyperparameters.

Test	Max Train Acc.	Max Val. Acc.	Test Acc.
1	0.736	0.705	-
2	0.741	0.7076	-
3	0.7502	0.7115	-
4	0.7199	0.7234	-
5	0.721	0.72625	0.730
6	0.7067	0.71897	-
7	0.568	0.578	-
8	0.5977	0.6185	-
9	0.68785	0.71265	-
10	0.558	0.57375	-
11	0.5595	0.5771	-
12	0.6105	0.6444	-
13	0.5306	0.5537	-
14	0.5374	0.5598	-
15	0.5354	0.5586	-
16	0.5211	0.5497	-
17	0.5229	0.552	-
18	0.5174	0.5481	-

Table 9: FCN patient_2025 hyperparameter study results.

Hyperparameter	Value
Neuron Configuration	[256,256,128,128]
Dropout	0.05
Learning Rate	0.0005

Table 10: FCN Patient_2025 hyperparameter study best hyperparameters.

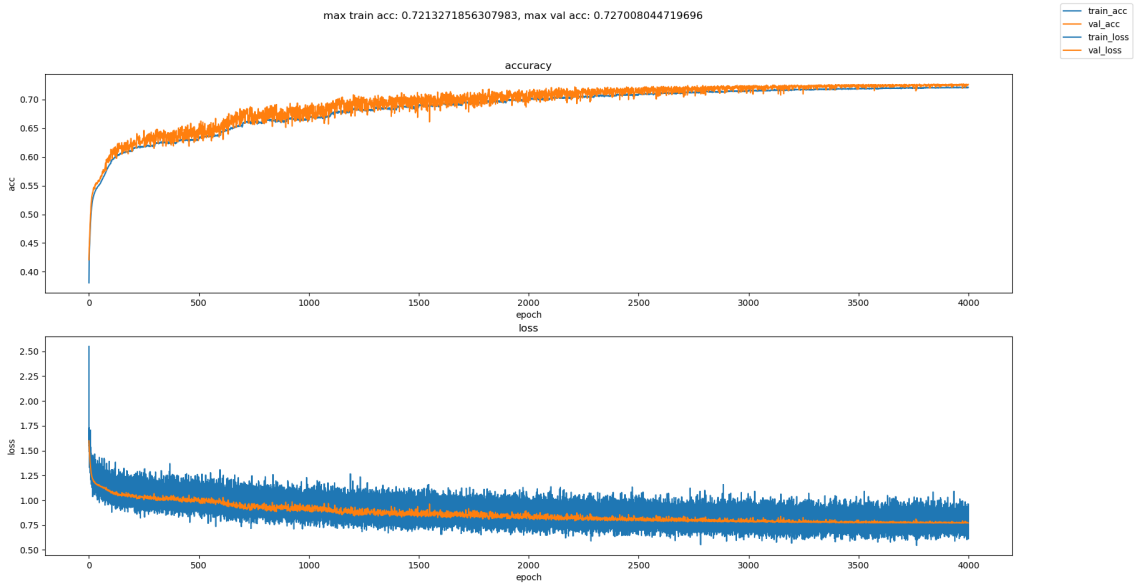


Figure 15: Training and validation accuracy and loss curves for the best FCN model on patient_2025 hyperparameter study.

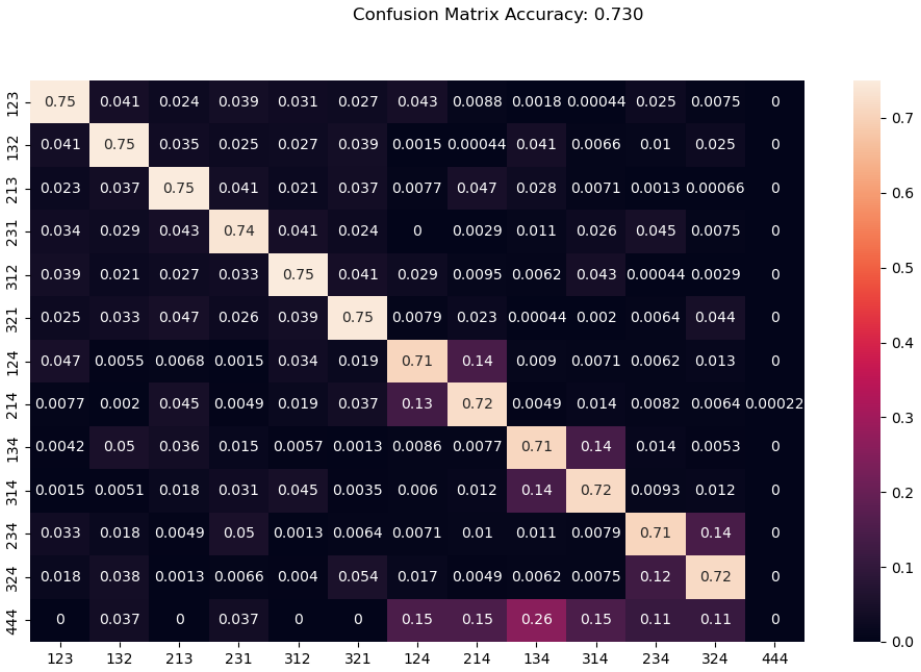


Figure 16: Confusion matrix on the test set for the best FCN model on patient_2025 hyperparameter study.

5.2.3 1D CNN Hyperparameter Study

A hyperparameter study was conducted using the novel implementation of a 1D CNN model developed in this work. This implementation consisted of a deeper number of convolutional layers than the 1D CNN model tested in previous work, as a deeper model was thought to be able to better capture patterns in the data [6]. The grid search was conducted using the built-in feature in BRIDE.

A total of 24 hyperparameter combinations were tested in the grid. The batch size, neuron configuration of the hidden layers, initial learning rate, and dropout rate were varied as displayed in Table 11. Some hyperparameters were held constant throughout the grid search and are shown in Table 12. The results of the CNN hyperparameter study are listed in Table 13. The model with the highest validation accuracy had a training accuracy of 62%, a validation accuracy of 55%, and a testing accuracy of 55%. Training and validation curves are shown in Figure 17 and the model's confusion matrix on the test set is shown by Figure 18. The best model's hyperparameters are found in Table 14.

Hyperparameter	**Value**
Batch Size	64, 128
Neuron Configuration	[2048, 1024, 512, 256], [1024, 512, 256], [512, 256, 128]
Dropout	0.2, 0.3
Learning Rate	0.0005, 0.001

Table 11: 1D CNN grid search candidate hyperparameters.

Hyperparameter	**Value**
Hardware	4 rtx6000 GPUs
Validation Split	0.1
Learning Rate Gamma	0.9
Learning Rate Step	10
L2	0.00001
Loss Function	CrossEntropy + Custom Pairwise Loss (p=0)
Optimizer	Adam
Activation Function	ReLU
Max Epochs	1000

Table 12: 1D CNN grid search constant hyperparameters.

Test	Max Train Acc.	Max Val. Acc.	Test Acc.
1	0.795	0.52	-
2	0.7062	0.5287	-
3	0.7672	0.5177	-
4	0.6709	0.5395	-
5	0.083	0.083	-
6	0.6689	0.5402	-
7	0.7946	0.5113	-
8	0.6964	0.5343	-
9	0.8039	0.5117	-
10	0.6823	0.5351	-
11	0.7594	0.5183	-
12	0.6773	0.5406	-
13	0.8307	0.5034	-
14	0.7861	0.5143	-
15	0.7743	0.5091	-
16	0.7781	0.518	-
17	0.8225	0.506	-
18	0.6924	0.5351	-
19	0.8303	0.5051	-
20	0.7638	0.517	-
21	0.8211	0.5041	-
22	0.7427	0.5243	-
23	0.8037	0.508	-
24	0.6219	0.5469	0.551

Table 13: 1D CNN patient_2025 hyperparameter study results.

Hyperparameter	Value
Batch Size	128
Neuron Configuration	[2048,1024,512,256]
Dropout	0.3
Learning Rate	0.001

Table 14: 1D CNN patient_2025 hyperparameter study best hyperparameters.

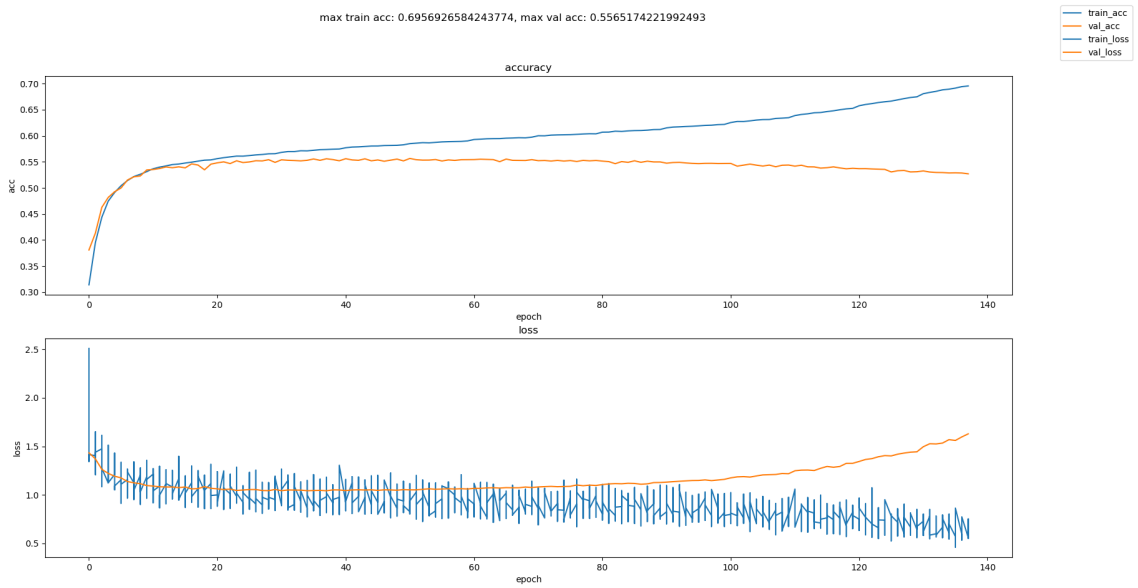


Figure 17: Training and validation accuracy and loss curves for the best 1D CNN model on the patient_2025 hyperparameter study.

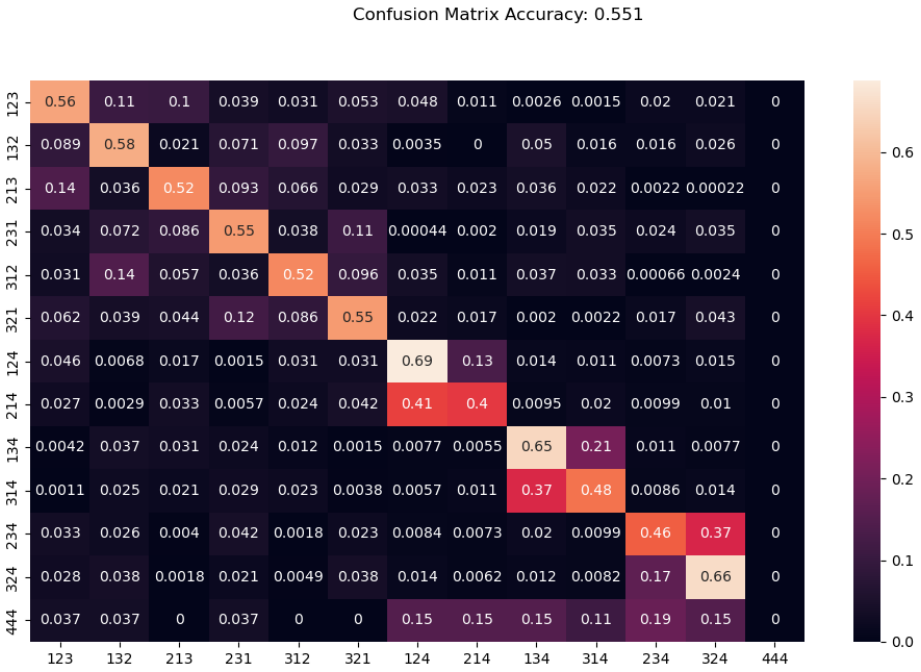


Figure 18: Confusion matrix on the test set for the best 1D CNN model on patient_2025 hyperparameter study.

The achieved 55.2% testing accuracy is comparable to previous years' results on simulated patient data; previous work achieved 55% testing accuracy with FCNs and a 55.6% testing accuracy

with LSTMs [6]. However, this CNN model was trained on much larger datasets compared to previous years. As larger datasets usually mean more robust models up to a certain point, this suggests that the 1D CNN has a decreased pattern recognition ability on simulated patient data compared to other models. Additionally, this CNN model performed around 20% worse compared to the best LSTMs and FCNs developed in this work, also suggesting that the CNN has a decreased learning performance on the data. We also found that the CNN model is extremely sensitive to its hyperparameters. For example, test number 5 in this hyperparameter study achieved only 8% maximum validation accuracy, while all other runs achieved more than 50% validation accuracy. Certain combinations of hyperparameters could result in much lower accuracy and a model unable to learn any patterns in data, as 8% accuracy is around the level of random guessing with a 13-class dataset.

5.3 Al_Patient_2025 Hyperparameter Studies

In addition, we conducted multiple hyperparameter studies to increase model performance on the 4.1 million row `al_patient_2025` simulated patient dataset.

5.3.1 LSTM Hyperparameter Study

This research conducted a hyperparameter study for the 4.1 million row dataset using an LSTM machine learning model from [6]. We tested 18 combinations of hyperparameters in this grid. The hyperparameters that were varied were the neuron configurations of the fully-connected layers, dropout rate, and starting learning rate; these choices were the same as those in the `patient_2025` LSTM hyperparameter study and can be seen back in Table 3. For all studies, the models ran for a maximum of 4000 epochs with early stopping set to 1500 epochs. Model checkpoints were saved every 400 epochs. The full constant parameters were also the same as those in `patient_2025` and can be seen in Table 4.

The results of the LSTM hyperparameter study are shown in Table 15. The model with the best validation accuracy achieved a training accuracy of 75%, a validation accuracy of 74%, and a testing accuracy of 74%. The hyperparameters of this model are displayed in Table 16. The training and validation curves are seen in Figure 19, and the test set confusion matrix is seen in Figure 20. These results are especially notable as they constitute a significant numerical increase of 18% in testing accuracy compared to any previous work on simulated patient data. This suggests that with hyperparameter tuning, LSTM models may be suitable for cleaning Compton camera-captured data. With this hyperparameter study, we found that a shallower neuron configuration with fewer layers and smaller neuron sizes usually led to better performance. The dropout rate did not have a large effect on performance (less than a 5% accuracy difference), although it did assist in minimizing overfitting with this model. A low learning rate was seen to give the best results.

Test	Max Train Acc.	Max Val. Acc.	Test Acc.
1	0.75	0.7233	-
2	0.753	0.7128	-
3	0.7346	0.729	-
4	0.747	0.7152	-
5	0.749	0.728	-
6	0.749	0.731	-
7	0.736	0.721	-
8	0.739	0.729	-
9	0.747	0.737	0.737
10	0.748	0.723	-
11	0.751	0.726	-
12	0.747	0.735	-
13	0.745	0.721	-
14	0.749	0.726	-
15	0.750	0.732	-
16	0.732	0.721	-
17	0.741	0.718	-
18	0.741	0.725	-

Table 15: LSTM al_Patient_2025 hyperparameter study results.

Hyperparameter	Value
Neuron Configuration	[128,64]
Dropout	0.3
Learning Rate	0.0001

Table 16: LSTM al_Patient_2025 hyperparameter study best hyperparameters.

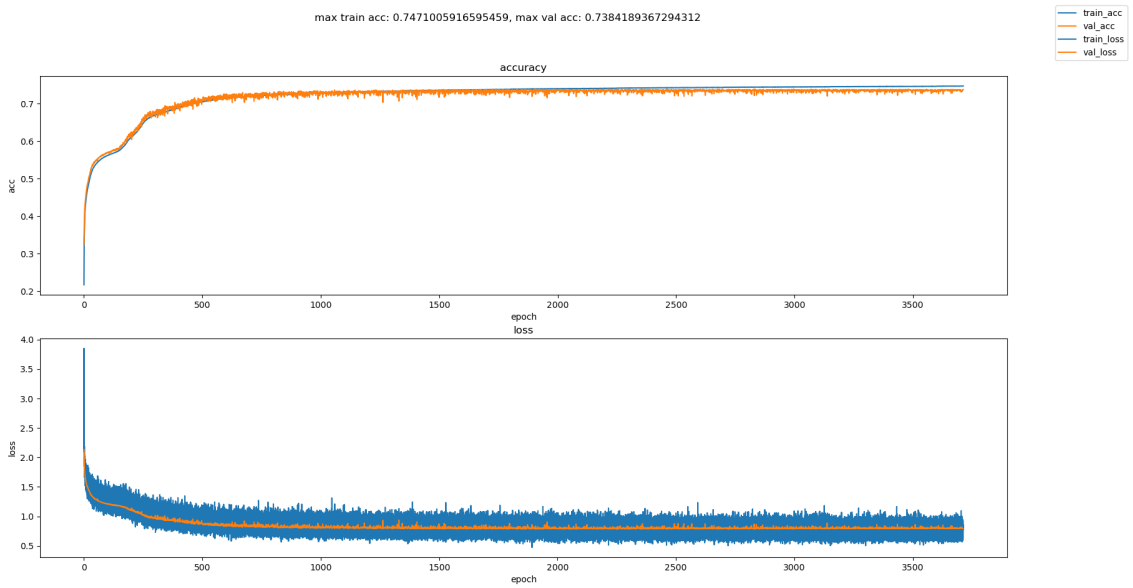


Figure 19: Training and validation accuracy and loss curves for the best LSTM model on the al_patient_2025 hyperparameter study.

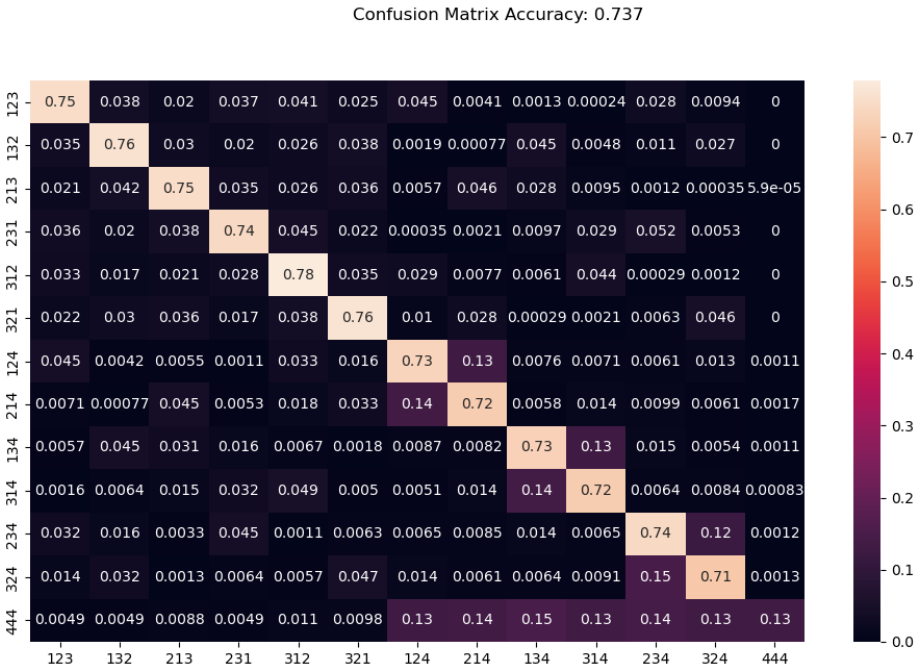


Figure 20: Confusion matrix on the test set for the best LSTM model on al_patient_2025 hyperparameter study.

5.3.2 FCN Hyperparameter Study

We conducted a hyperparameter study using the fully-connected neural network model from [6] on this novel simulated patient dataset. Similar to before, 18 combinations of hyperparameters were searched through. The hidden layer neuron configuration, learning rate, and dropout value were varied; these variable hyperparameters' choices were the same as those in the `patient_2025` FCN hyperparameter study and can be seen in Table 7. The constant parameters are also the same as those of `patient_2025` and are shown in Table 8. Models ran for a maximum of 4000 epochs with early stopping set to 1500 epochs.

The accuracy results of this hyperparameter study are displayed in Table 17. As seen, the best FCN model reached a training accuracy of 74%, a validation accuracy of 74%, and a testing accuracy of 74%. Its hyperparameters are shown in Table 10. Training and validation curves and the testing set confusion matrix are displayed in Figures 21 and 22, respectively. The testing accuracy of this model (74.2%) is slightly higher than the testing accuracy of the LSTM `al_patient_2025` grid search best model (73.7%) by 0.5%. Although this is a small difference in accuracies, it may suggest that FCNs may have the same or slightly better performance on simulated patient data compared to LSTM models. Notably, we found that having a shallower FCN architecture significantly increased validation accuracy by up to 20%, similar to our findings from the FCN `patient_2025` grid search. In particular, a shallower architecture may have led to a better generalizability performance due to a simpler model. These shallow FCN models are a significant area of improvement compared to previous work, which had lower model performance with deep FCN models [6].

Test	Max Train Acc.	Max Val. Acc.	Test Acc.
1	0.7115	0.7162	-
2	0.7228	0.7268	-
3	0.7381	0.7423	0.742
4	0.7078	0.7278	-
5	0.714	0.7284	-
6	0.7202	0.7355	-
7	0.5599	0.5775	-
8	0.6817	0.7126	-
9	0.7021	0.7269	-
10	0.5549	0.576	-
11	0.557	0.5783	-
12	0.6732	0.7158	-
13	0.5225	0.5539	-
14	0.5356	0.5638	-
15	0.5406	0.5689	-
16	0.5182	0.5515	-
17	0.5192	0.5532	-
18	0.5208	0.5558	-

Table 17: FCN al_Patient_2025 hyperparameter study results.

Hyperparameter	Value
Neuron Configuration	[512, 256, 256, 256, 256, 256, 256, 128]
Dropout	0.05
Learning Rate	0.0001

Table 18: FCN al_Patient_2025 hyperparameter study best hyperparameters.

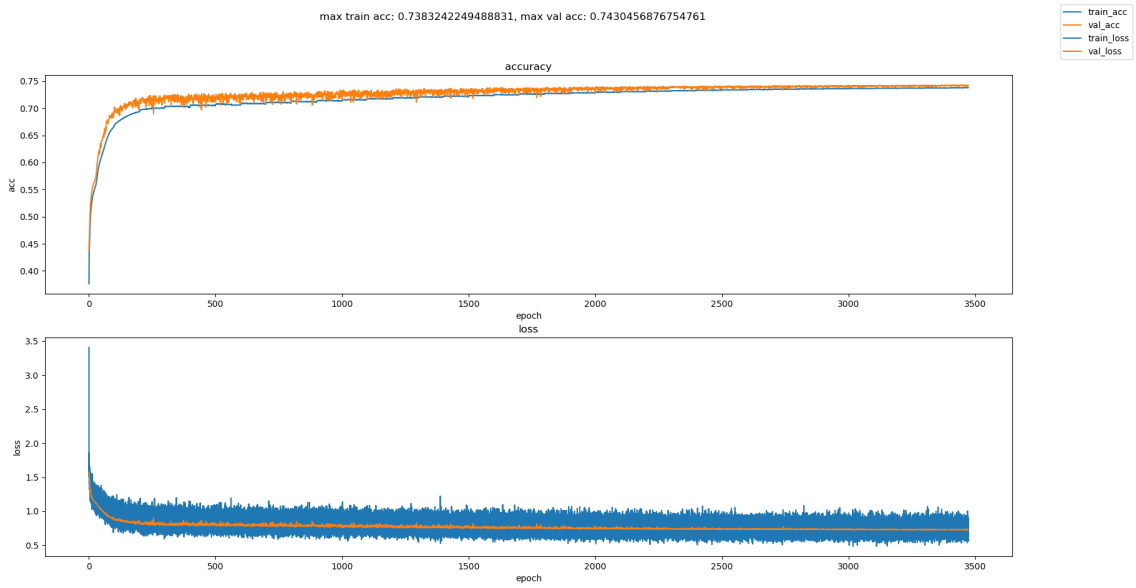


Figure 21: Training and validation accuracy and loss curves for the best FCN model on the al_patient_2025 hyperparameter study.

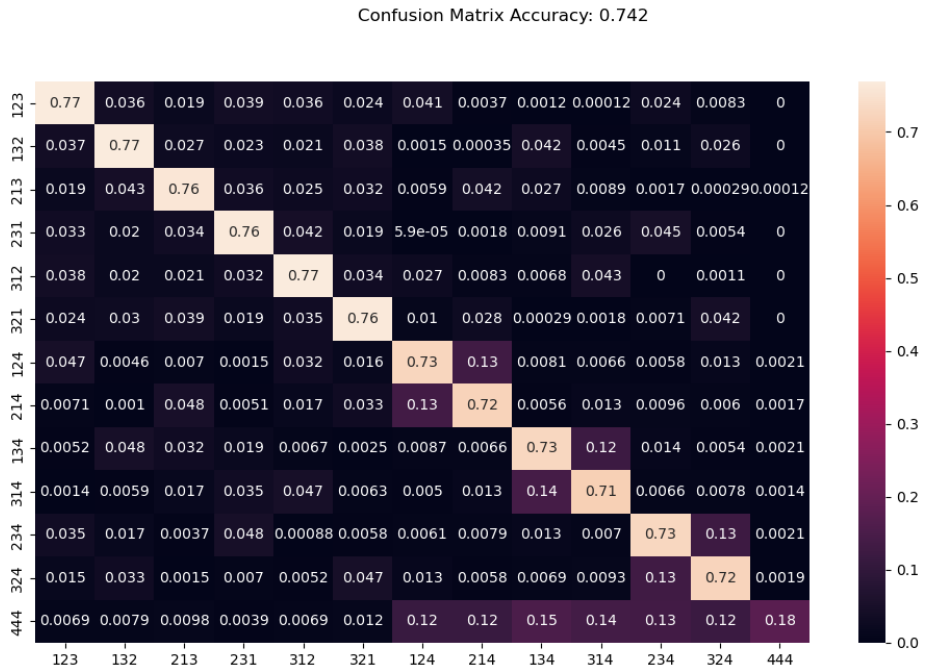


Figure 22: Confusion matrix on the test set for the best FCN model on the al_patient_2025 hyperparameter study.

5.3.3 Transformer Hyperparameter Study

A hyperparameter study using the novel implementation of an Event Classifier Transformer that we developed was conducted on the 4.1 million row simulated patient dataset. In total, 32 combinations of hyperparameters were tested by varying the initial learning rate, dropout, number of model nodes, number of layers, and number of heads, as displayed by Table 19. Certain hyperparameters were held constant throughout the grid search and are shown in Table 20. Models ran with early stopping of 1500 epochs for a maximum of 4000 epochs.

The results of the study are listed in Table 21. We found that the best performing transformer model had a training, validation, and testing accuracy of 75.1%, 70.5%, and 70.4%, respectively. Its hyperparameters are listed in Table 22. The training and validation accuracy and loss curves are shown by Figure 23, and the confusion matrix on the test set is shown by Figure 24. Significantly, the best model's testing accuracy is a 14% numerical increase from the best result on simulated patient data from previous work [6]. Additionally, we found that the accuracy of the transformer model in particular was very variable; a slight change in hyperparameters could change validation accuracy by up to 60% numerically. The models also experience severe overfitting with up to a 30% numerical difference between training and validation accuracy. We found that using higher values of dropout did reduce the overfitting, though.

Hyperparameter	Value
Model Nodes	128,256
Number of Layers	4, 6
Number of Heads	4, 8
Dropout	0.15, 0.3
Learning Rate	0.0005, 0.0001

Table 19: Transformer grid search candidate hyperparameters.

Hyperparameter	Value
Hardware	4 rtx6000 GPUs
Validation Split	0.1
Batch Size	256
Learning Rate Gamma	0.1
Learning Rate Step	1000
L2	0.0000001
Loss Function	CrossEntropy + Custom Pairwise Loss
Optimizer	Adam
Activation Function	ReLU
Max Epochs	4000

Table 20: Transformer grid search constant hyperparameters.

Test	Max Train Acc.	Max Val. Acc.	Test Acc.
1	0.751	0.705	0.704
2	0.775	0.664	-
3	0.780	0.663	-
4	0.900	0.646	-
5	0.762	0.675	-
6	0.781	0.644	-
7	0.844	0.641	-
8	0.917	0.639	-
9	0.721	0.679	-
10	0.638	0.658	-
11	0.473	0.653	-
12	0.253	0.615	-
13	0.697	0.659	-
14	0.613	0.657	-
15	0.393	0.618	-
16	0.314	0.612	-
17	0.707	0.653	-
18	0.717	0.614	-
19	0.731	0.658	-
20	0.752	0.601	-
21	0.713	0.614	-
22	0.721	0.568	-
23	0.741	0.625	-
24	0.782	0.638	-
25	0.713	0.617	-
26	0.743	0.593	-
27	0.769	0.610	-
28	0.814	0.579	-
29	0.736	0.585	-
30	0.747	0.609	-
31	0.781	0.599	-
32	0.828	0.592	-

Table 21: Transformer al_patient_2025 hyperparameter study results.

Hyperparameter	Value
Model Nodes	128
Number of Layers	4
Number of Heads	4
Dropout	0.15
Learning Rate	0.0005

Table 22: Trnasformer al_Patient_2025 hyperparameter study best hyperparameters.

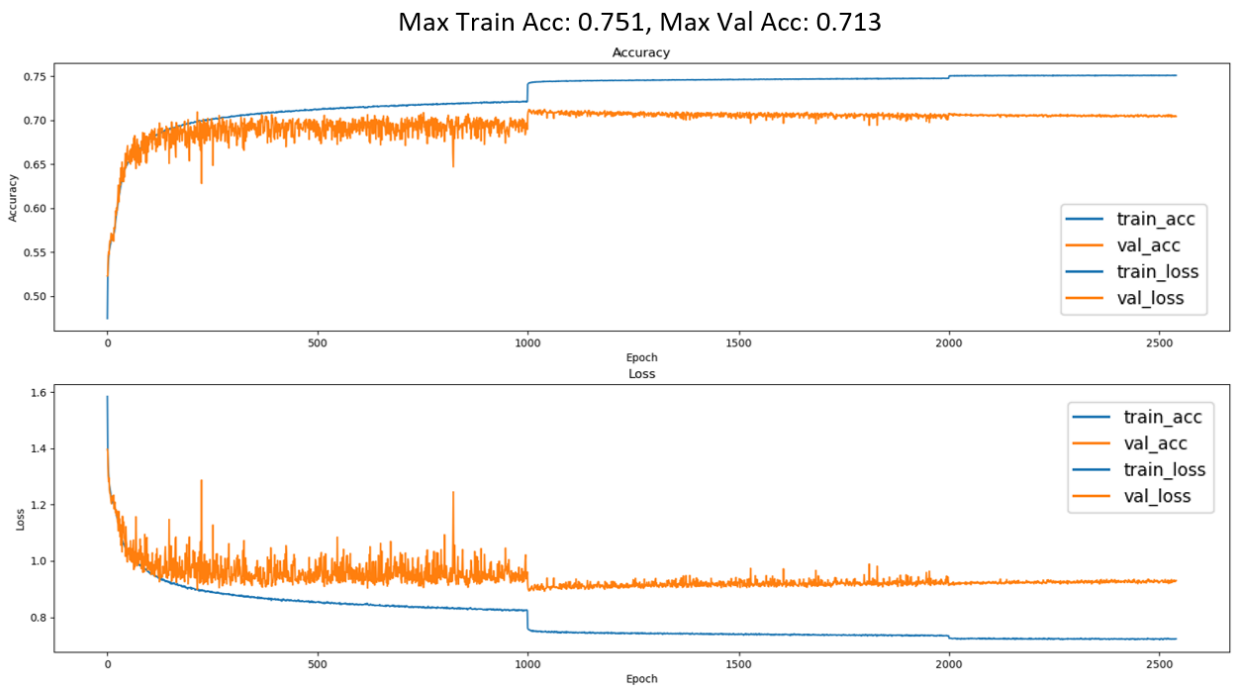


Figure 23: Training and validation accuracy and loss curves for the best transformer model on the al_patient_2025 hyperparameter study.

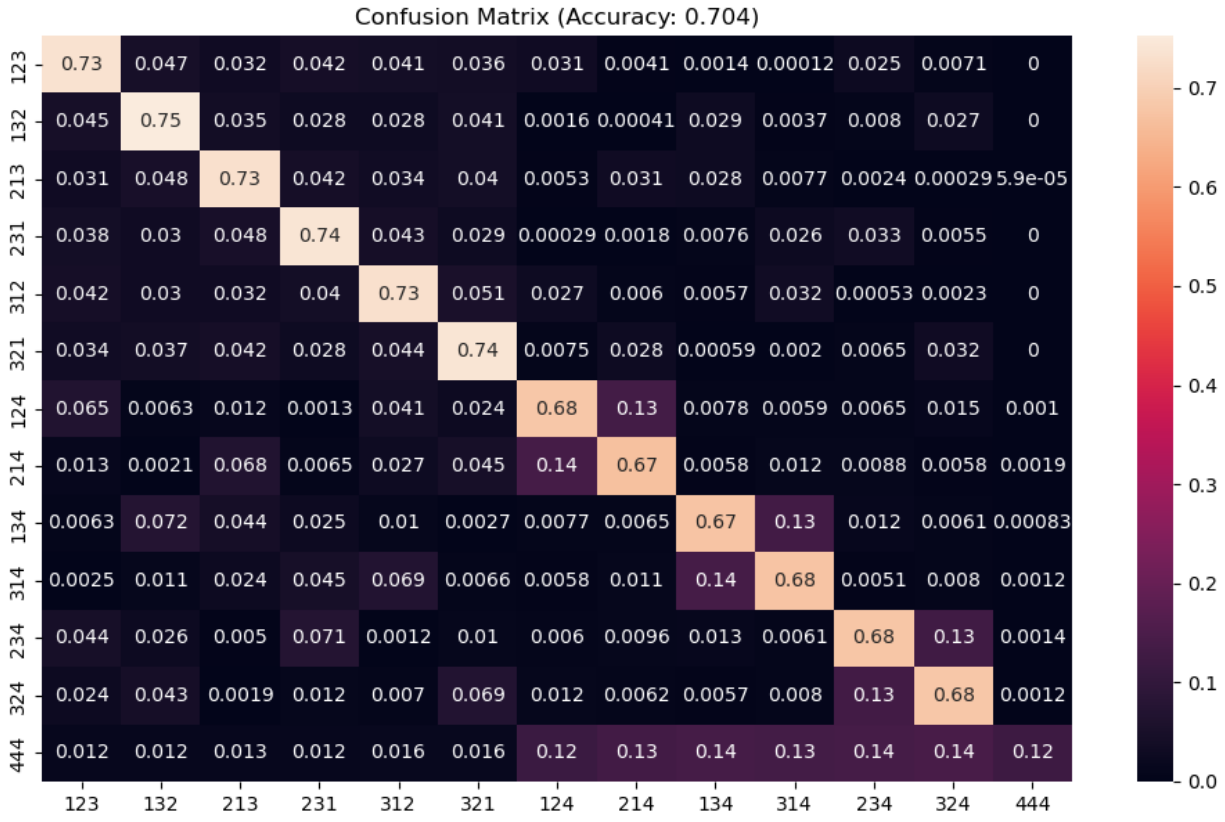


Figure 24: Confusion matrix on the test set for the best transformer model on the al_patient_2025 hyperparameter study.

6 Conclusion and Future Work

In this research, we found that for simulated patient data, the best-performing FCN model achieved a 74.2% testing accuracy while the best-performing LSTM model achieved a testing accuracy of 73.7%, as shown in Table 23. Both these models constitute a significant 18% numerical increase in accuracy from the highest result on patient data from any previous work. Along with the prior fact that these models offer a robust improvement on classifying simulated patient data, these models' results constitute an approximate 1-2% numerical accuracy increase compared to the highest achieved result on the less complex and easier-to-learn water phantom data on the BRIDE platform. These models offer greatly improved robustness in a Compton camera real-time imaging scenario as they achieve higher accuracy than previous research on both complex variable-density patient data and the simple constant-density water phantom, suggesting that the models are more generalizable even when trained on harder-to-learn data. Despite this, we found that the 1D CNN model did not perform well compared to the FCN, LSTM, or event classifier transformer. Even though it approximated previous results on patient data, its performance was likely due to an increased dataset size, suggesting that the model may not be suitable for a clinical setting without further experimentation.

Through this work, we introduced two novel simulated patient datasets based upon actual variable-density CT scan measurements. These datasets were up to 8 times larger than the simulated patient datasets used in previous researchers' work. This vastly increased dataset size likely had an influence on the higher accuracy results, as larger datasets often lead to a greater model robustness up to a certain point. With our conducted hyperparameter studies, it was found that having a shallower model architecture with fewer layers and fewer neurons per layer usually led to better performance, even though deeper architectures still sometimes performed better with strong dropout regularization techniques. However, without enough regularization, deeper architecture models tended to overfit on the training data. We found that training the models on the larger 4.1 million simulated patient data led to a moderate increase in accuracy ($\sim$5%) compared to training them on the smaller 1.1 million row dataset.

Several novel implementations of machine learning models were developed to allow for the models' better understanding of the spatial dependency patterns between various scatterings as detected by the Compton camera. In particular, a novel event classifier transformer neural network and a 1D CNN were developed on BRIDE. A grid search mechanism to allow for the large-scale training of models was newly implemented. The mechanism greatly improved the extensibility of the BRIDE model experimentation process by allowing for the automatic and efficient submitting of model runs.

Model	Dataset	Train Acc.	Val Acc.	Test Acc.	Notes
LSTM	patient_2025	0.73	0.70	0.71	15% gain over past results
	al_patient_2025	0.73	0.75	0.73	Grid search on 4.1M dataset
1D CNN	patient_2025	0.62	0.54	0.55	Performed worse than LSTM and FCN on this dataset
FCN	patient_2025	0.72	0.73	0.73	17% improvement over previous results
	al_patient_2025	0.73	0.74	**0.74**	**Best result overall**
Event Classifier Transformer	al_patient_2025	0.75	0.71	0.70	Slightly overfitting was experienced
Note: patient_2025 = 1.1M rows, al_patient_2025 = 4.1M rows					

Table 23: Model performance comparison across simulated patient datasets.

With regard to future work, the incorporation of more advanced models and architectures to better find patterns and relationships within the given data may be beneficial. Specifically, graph neural networks (GNNs) and hierarchical classification may be promising in this area of study.

6.1 Graph Neural Networks

A possible future direction is implementing graph neural networks, specifically Graph Attention Networks (GATs), to classify the scatter events. In our current pipeline, the scatter events are fed as a flat input into the models. However, the scatter events have a natural spatial and relational structure that the models do not take into account. There is a strong possibility that GNNs will work well because a triple of hits already naturally forms a graph. For example, model nodes could be seen as the individual hits; these interactions have spatial and temporal features including position, energy, and time. Model edges could be considered the relationships between the hits.

Each event in our dataset consists of a small number of interaction points and belongs to one of thirteen classes. Each scatter contains features such as deposited energy, spatial coordinates, and Euclidean distances. These hits are not independent. They are spatially related through Compton scattering, and the order, spacing, and energy differences between them often contain information relevant to the classification task. For example, the first hit in a triple carries more physical meaning such as indicating the origin or direction of the gamma, while subsequent hits help define the scatter pattern and energy loss trajectory.

GAT models may be a particularly suitable type of GNN for this task because they update the features of each node by aggregating information from its neighbors using learned attention weights, allowing the model the neighbors that are more relevant [22]. GATs are able to capture the dependencies between the hits by representing a hit as a node with edges linking them to show their relationships. Instead of flattening the data into one vector, GATs are able to maintain the spatial data structure and observe how the hits relate to one another. This method has the capability of improving classification accuracy when it comes to the scatter events.

6.2 Hierarchical Classification

Another direction that can be explored in future research is to replace the current flat classifier with a hierarchical one. The current models maps scattering events to one of the 13 classes directly, requiring it to distinguish among all of them during training. However as noted earlier, of the 13 classes, there are 3 distinct groups of classes: true triples, DtoTs, and false triples.

Given this, a possibly more efficient strategy for scatter classification could be to first use a preliminary model to classify a scattering as a true triple, DtoT, or false triple; then, for each scattering group, a secondary model would classify the scatter into one of the classes. This would simplify the classification process as it would allow the model to bypass any unnecessary computation and possible confusion between different scattering groups when classifying the scatter class by reducing the number of classes at a particular classification step [18]. In other words, this approach has the potential to improve the accuracy of the current classification pipeline by focusing more resources on the scatterings that are most relevant at that particular step.

Acknowledgments

This work is supported by the grant "REU Site: Online Interdisciplinary Big Data Analytics in Science and Engineering" from the National Science Foundation (grant no. OAC–2348755). Coauthors Sharma and Ren additionally acknowledge support by NIH. We acknowledge the UMBC High Performance Computing Facility and the financial contributions from NIH, NSF, CIRC, and UMBC for this work.

References

[1] Alina M. Ali, David Lashbrooke, Rodrigo Yepez-Lopez, Sokhna A. York, Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Towards optimal configurations for deep fully connected neural networks to improve image reconstruction in proton radiotherapy. Technical Report HPCF–2021–12, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2021.

[2] Kaelen Baird, Sam Kadel, Brandt Kaufmann, Ruth Obe, Yasmin Soltani, Mostafa Cham, Matthias K. Gobbert, Carlos A. Barajas, Zhuoran Jiang, Vijay R. Sharma, Lei Ren, Stephen W. Peterson, and Jerimy C. Polf. Enhancing real-time imaging for radiotherapy: Leveraging hyperparameter tuning with PyTorch. Technical Report HPCF–2023–12, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2023.

[3] Carlos A. Barajas. *Neural Networks for the Sanitization of Compton Camera Based Prompt Gamma Imaging Data for Proton Radiotherapy.* Ph.D. Thesis, Department of Mathematics and Statistics, University of Maryland, Baltimore County, 2022.

[4] Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Deep residual fully connected neural network classification of Compton camera based prompt gamma imaging for proton radiotherapy. *Front. Phys.*, 11:903929, 2023.

[5] Michael O. Chen, Julian Hodge, Peter L. Jin, Ella Protz, Elizabeth Wong, Ruth Obe, Ehsan Shakeri, Mostafa Cham, Matthias K. Gobbert, Carlos A. Barajas, Zhuoran Jiang, Vijay R/ Sharma, Lei Ren, Sina Mossahebi, Stephen W. Peterson, and Jerimy C. Polf. Improving gamma imaging in proton therapy by sanitizing compton camera simulated patient data using neural networks through the bride pipeline. Technical Report HPCF–2024–5, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2024.

[6] Michael O. Chen, Julian Hodge, Peter L. Jin, Ella Protz, Elizabeth Wong, Ruth Obe, Ehsan Shakeri, Mostafa Cham, Matthias K. Gobbert, Carlos A. Barajas, Vijay R. Sharma, Sina Mossahebi, Lei Ren, Stephen W. Peterson, and Jerimy C. Polf. Improving gamma imaging in

proton therapy by sanitizing compton camera simulated patient data using neural networks through the bride pipeline. In *2024 IEEE International Conference on Big Data (Big Data 2024)*, pages 7463–7470, 2024.

[7] Joseph Clark, Anaise Gaillard, Justin Koe, Nithya Navarathna, Daniel J. Kelly, Matthias K. Gobbert, Carlos A. Barajas, and Jerimy C. Polf. Sequence-based models for the classification of Compton camera prompt gamma imaging data for proton radiotherapy on the GPU clusters taki and ada. Technical Report HPCF–2022–12, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2022.

[8] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1984.

[9] Ahmed Hammad, Stefano Moretti, and M Nojiri. Multi-scale cross-attention transformer encoder for event classification. *Journal of High Energy Physics*, 2024(3):1–31, 2024.

[10] Jonathan R. Hughes and Jason L. Parsons. FLASH radiotherapy: Current knowledge and future insights using proton-beam therapy. *Int. J. Mol. Sci.*, 21(18):6492, 2020.

[11] Jaebak Kim. Training toward significance with the decorrelated event classifier transformer neural network. *Physical Review D*, 109(9):096035, 2024.

[12] Paul Maggi, Steve Peterson, Rajesh Panthi, Dennis Mackin, Hao Yang, Zhong He, Sam Beddar, and Jerimy Polf. Computational model for detector timing effects in Compton-camera based prompt-gamma imaging for proton radiotherapy. *Phys. Med. Biol.*, 65(12), 2020.

[13] Daniel W. Mundy and Michael G. Herman. An accelerated threshold-based back-projection algorithm for compton camera image reconstruction. *Medical Physics*, 38(1):15–22, 2011.

[14] Ruth Obe, Brandt Kaufmann, Kaelen Baird, Sam Kadel, Yasmin Soltani, Mostafa Cham, Matthias K. Gobbert, Carlos A. Barajas, Zhuoran Jiang, Vijay R. Sharma, Lei Ren, Stephen W. Peterson, and Jerimy C. Polf. Accelerating real-time imaging for radiotherapy: Leveraging multi-GPU training with PyTorch. In *2023 International Conference on Machine Learning and Applications (ICMLA 2023)*, pages 1735–1742, 2023.

[15] Raj Kumar Parajuli, Makoto Sakai, Ramila Parajuli, and Mutsumi Tashiro. Development and applications of Compton camera —– a review. *Sensors*, 22:7374, 2022.

[16] Jerimy C. Polf, Carlos A. Barajas, Stephen W. Peterson, Dennis S. Mackin, Sam Beddar, Lei Ren, and Matthias K. Gobbert. Applications of machine learning to improve the clinical viability of Compton camera based in vivo range verification in proton radiotherapy. *Front. Phys.*, 10:838273, 2022.

[17] Jerimy C. Polf and Katia Parodi. Imaging particle beams for cancer treatment. *Phys. Today*, 68(10):28–33, 2015.

[18] P. M. Rezende, J. S. Xavier, D. B. Ascher, G. R. Fernandes, and Pires D. E. V. Evaluating hierarchical machine learning approaches to classify biological databases. *Briefings in bioinformatics*, 23(4), 2022.

[19] Michael J. Smith and James E. Geach. Astronomia ex machina: a history, primer and outlook on neural networks in astronomy. *R. Soc. Open Sci.*, 10:221454, 2023.

[20] Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):517–529, 2015.

[21] Sokhna A. York, Alina M. Ali, David C. Lashbrooke Jr, Rodrigo Yepez-Lopez, Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Promising hyperparameter configurations for deep fully connected neural networks to improve image reconstruction in proton radiotherapy. In *2021 IEEE International Conference on Big Data (Big Data 2021)*, pages 5648–5657, 2021.

[22] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.