



Secondary Vertex Reconstruction at The High-Luminosity LHC with MaskFormer Architecture

Author: Sittipon Kumda

Supervisors: Doga Elitez, Jonathan Niklas Renusch,
Dr. Paul Gessinger-Befurt

CERN, CH-1211 Geneva, Switzerland

Keywords: Secondary Vertex Reconstruction, High-Luminosity LHC, MaskFormer, Machine Learning

Abstract

With the upcoming era of the High-Luminosity LHC, secondary vertex reconstruction will face significant computational challenges in dense environment, highlighting the need for novel and efficient identification algorithms. This report introduces a new approach, adapting the transformer-based MaskFormer architecture from computer vision to reframe SV finding as an end-to-end instance segmentation problem. A dedicated data processing pipeline was developed to convert simulated ROOT data into a uniform HDF5 format suitable for a multi-task model designed to simultaneously perform SV classification, track assignment (masking), and vertex property regression. The model, trained on a realistic, class-imbalanced dataset, demonstrated excellent performance on the classification and track assignment tasks, achieving results comparable to a baseline trained on a perfectly balanced reference sample. However, the vertex regression task proved highly sensitive to this imbalance, showing significant underperformance compared to the reference. These findings establish the query-based transformer architecture as a powerful and viable new direction for tackling the combinatorial challenge of vertex reconstruction. The results also highlight that achieving precise regression in a multi-task setting on imbalanced data is a key challenge, suggesting that future work must prioritize the optimization of the loss function weights and target scaling strategies to unlock the full potential of this approach.

Contents

1	Introduction	3
1.1	Background	3
1.2	Objectives	3
1.3	Pipeline Overview	4
2	Adaptation of MaskFormer	4
2.1	MaskFormer Architecture	4
2.2	Proposed Architecture for SV Reconstruction	5
2.3	Competitive Architectures	6
2.4	Simulation tools and Data format	6
2.5	Machine Learning preferred data format	7
2.6	Conversion Module	7
2.7	Dataset Summary	10
3	Model Result and Analysis	10
3.1	Secondary Vertex Classification	11
3.2	Track Assignment (Mask Prediction)	11
3.3	Secondary Vertex Regression	12
4	Conclusion	12
5	Discussion	13
5.1	Imbalance in Multi-Task Learning	13
5.2	Impact of Target Normalization on Performance	13
5.3	Hyperparameter Sensitivity Study	14
5.4	Recommendations for Future Work	15

1 Introduction

1.1 Background

The reconstruction of secondary vertices (SVs) is an essential task in high-energy physics experiments at CERN. These displaced vertices are key signatures in a wide range of physics analyses, from studies within the Standard Model to searches for long-lived particles in Beyond the Standard Model theories. While numerous established algorithms exist for SV reconstruction, the upcoming High-Luminosity Large Hadron Collider (HL-LHC) era presents a significant new challenge. The expected increase in the number of simultaneous proton-proton collisions, a phenomenon known as pileup, will drastically increase the combinatorial complexity of the detector environment, demanding more robust and efficient reconstruction techniques.

In response to such challenges, Machine Learning has emerged as a powerful paradigm. A particularly promising approach is the use of transformer-based architectures like MaskFormer[1]. Originally designed for the computer vision task of instance segmentation, MaskFormer excels at identifying complex, non-linear correlations in high-dimensional data by learning to group related inputs into distinct object instances. This inherent capability for grouping and pattern recognition makes the architecture exceptionally well-suited for secondary vertex (SV) reconstruction, a task that lacks a simple, deterministic solution and instead relies on interpreting intricate patterns from detector inputs such as particle tracks and jets.

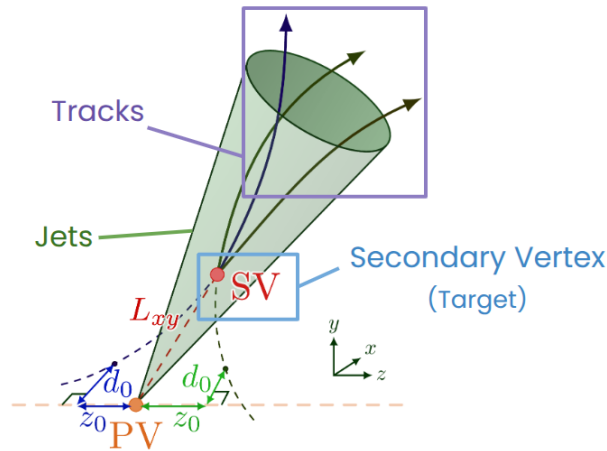


Figure 1: The visualization of track and secondary vertex inside a jet

1.2 Objectives

The ultimate goal of this project is to develop and validate a machine learning pipeline for SV reconstruction, tailored to the high pile-up environment of the HL-LHC. The goal is for this algorithm to effectively process track and jet data to achieve high performance in SV finding and fitting.

However, as official HL-LHC simulation datasets are still in development, the work presented in this report constitutes a foundational proof-of-concept. The dataset employed is a preliminary simulation designed to approximate the expected detector conditions. Therefore, the immediate objectives of this report are to:

- Establish a complete, end-to-end machine learning pipeline, from data pre-processing to model evaluation.
- Demonstrate the functionality of the pipeline on the available dataset.
- Conduct a preliminary evaluation of the model’s performance to confirm its potential for more advanced studies with future datasets.

1.3 Pipeline Overview

The end-to-end machine learning pipeline developed for this study comprises three principal stages: data format conversion and pre-processing, model training, and performance evaluation. Each stage is designed to be a modular component, facilitating independent development and testing. The overall workflow is illustrated in Figure 2.

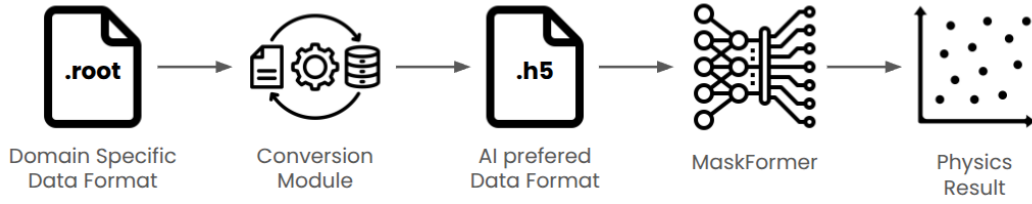


Figure 2: Schematic of the end-to-end machine learning pipeline, detailing the workflow from initial data conversion to final performance evaluation.

2 Adaptation of MaskFormer

2.1 MaskFormer Architecture

The original MaskFormer architecture, introduced by Cheng et al. in 2021[1], consists of three main modules: a pixel decoder, a transformer decoder, and segmentation heads. A schematic of this architecture is shown in Figure 3.

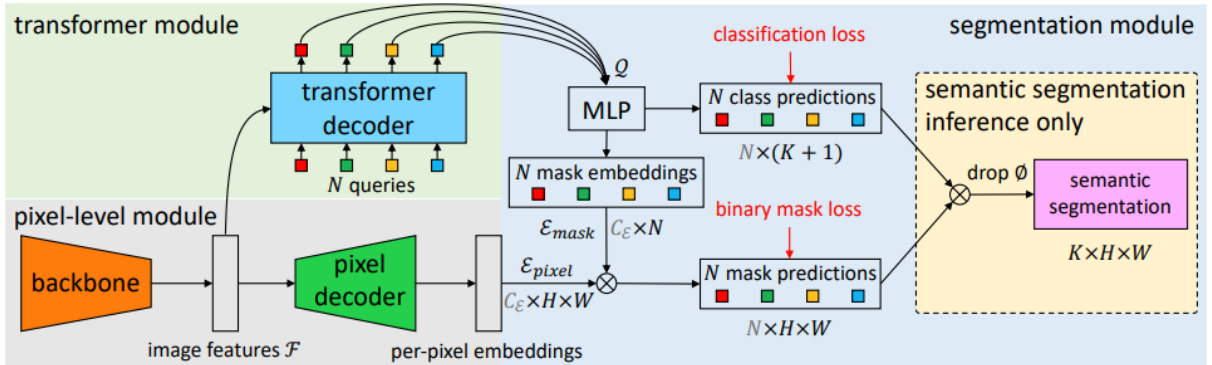


Figure 3: Overview of the original MaskFormer architecture.

The role of each module is described below:

Feature Embedding This module is a CNN-based feature extractor, often using a bottleneck design. A backbone network first extracts a low-resolution feature map

from the input image. This feature map is then fed to the transformer decoder, while a separate network generates high-resolution per-pixel embeddings used for the final mask prediction.

Transformer Decoder This is a standard transformer decoder module that processes a set of N learnable embeddings, known as object queries. It performs cross-attention between these queries and the image feature map from the pixel decoder. This process yields a set of N per-query embeddings, which serve as the final representations for potential objects or segments.

Segmentation Heads This final stage consists of two separate feed-forward networks (MLPs). One network takes the per-query embeddings to perform a classification, predicting the class label for each of the N queries. The second network is used to generate the final binary masks from the per-pixel embeddings.

2.2 Proposed Architecture for SV Reconstruction

In the context of secondary vertex reconstruction, two primary tasks must be accomplished. The first, *SV finding*, involves assigning constituent tracks to their originating secondary vertex within a jet. The second, *SV fitting*, uses these assigned tracks to calculate the properties of the vertex. These physics tasks can be framed as distinct machine learning problems: SV finding is analogous to an instance segmentation task, where each "instance" is an SV and the "pixels" are the tracks. SV fitting, in contrast, is a classic regression task. This framing makes the MaskFormer architecture, which excels at segmentation, a highly advantageous starting point for this work.

The proposed architecture, a novel adaptation of MaskFormer introduced by Samuel Van Stroud (2024)[2], is designed to perform these tasks simultaneously. A schematic of this model is shown in Figure 4, and its main components are detailed below.

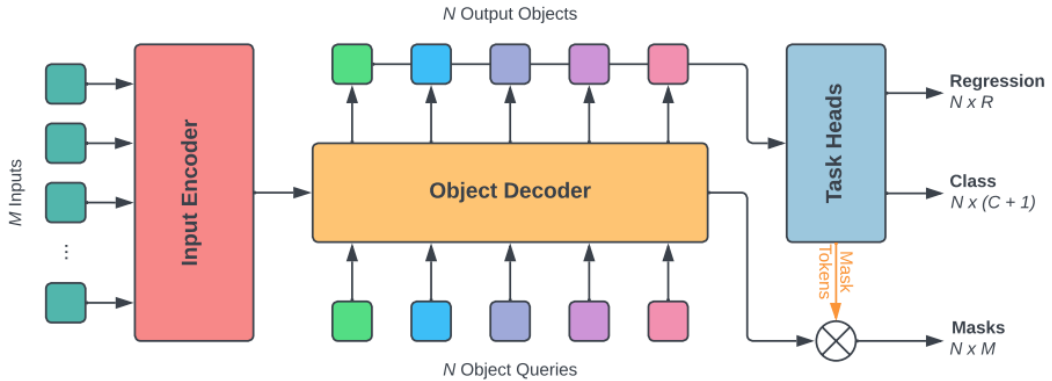


Figure 4: Overview of the architecture for SV reconstruction, adapted from MaskFormer.

The primary adaptations to the original MaskFormer model are as follows:

Input Encoder The original CNN-based pixel decoder is replaced with a transformer-based encoder. This module is designed to process an input set of unordered track and jet kinematic features as presented in the Figure 1, producing a contextualized embedding for each track.

Transformer Decoder This module remains largely unchanged. It performs cross-attention between the contextualized track embeddings (from the Input Encoder) and the N learnable object queries. The output is a set of N embeddings, each representing a candidate secondary vertex.

Task Heads The original segmentation heads are replaced by three distinct, task-specific heads that take the SV candidate embeddings as input:

- **Classification Head:** A feed-forward network (FFN) that predicts the class of each SV candidate (originating from a b-quark, c-quark, or light-quark fragmentation).
- **Masking Head:** An FFN that produces the track-to-vertex association mask, assigning tracks to each SV candidate.
- **Regression Head:** A new, dedicated FFN that predicts the precise geometric properties, Angular displacement from Jet-Axis (dR) and Radial flight length from Primary Vertex (L_{xy}), for each SV candidate.

2.3 Competitive Architectures

While Graph Neural Networks (GNNs) are a strong baseline for reconstruction tasks, their reliance on iterative message passing across a predefined graph can be a limitation. In contrast, the transformer-based architecture provides a more direct and powerful solution. Its attention mechanisms inherently operate on a dynamic, fully-connected graph of all input tracks, allowing a set of SV queries to perform a global, top-down grouping of tracks into vertices in a single, end-to-end step.

2.4 Simulation tools and Data format

A detailed description of the physics event generation and detector simulation is beyond the scope of this report. However, the primary software tools employed to produce the dataset are summarized here. Each component is a standard tool in the High-Energy Physics community, and their specific roles in the data generation pipeline were as follows:

- The event generation, detector simulation, digitization, and dataset production were performed within the ACTS framework.
- The Open Data Detector (ODD) [3] was used to define the detector geometry.
- Event generation was carried out with Pythia8 [4], while detector simulation employed Fatras [5] with the ODD geometry, both integrated into ACTS [6].
- Track reconstruction, navigation, jet-track matching, truth labeling, secondary vertex reconstruction, and plotting were handled using ACTS libraries, whereas FastJet [7] was employed for jet clustering.
- The final datasets were written to ROOT [8] files using ACTS as the overarching toolkit.

The raw data retrieved from the detector simulation is stored in ROOT files, a domain-specific format widely used in high-energy physics. Within these ROOT files, simulation data is typically organized into separate branches, containing event-wise non-homogeneous 2-dimensional arrays. This structure, while efficient for physics analysis, is generally incompatible with standard machine learning frameworks, which often expect uniform, vectorized inputs.

The non-homogeneous nature of this data, where array sizes can vary significantly from one event to another, poses a direct challenge for ingestion by neural networks. A conceptual visualization of this non-uniform data state is shown in Figure 5, highlighting the need for a dedicated conversion and pre-processing step to transform the contents into a structured, machine learning-compatible data format.

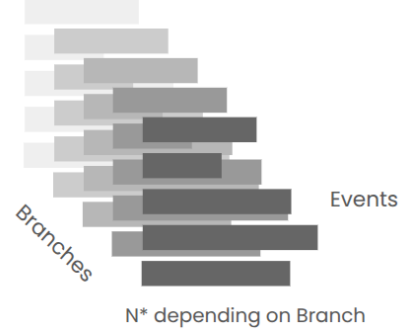


Figure 5: Conceptual visualization of the non-homogeneous data structure.

The details of conversion module will be elaborated in the Conversion Module sections.

2.5 Machine Learning preferred data format

Machine learning frameworks, such as PyTorch[9] used in this project, typically require a uniform, vectorized data format to perform essential matrix algebra for operations like gradient optimization.

HDF5 is used in this project because it stores data in dense, multi-dimensional, and homogeneous arrays. This structure allows for direct and efficient ingestion by machine learning libraries. Furthermore, HDF5 is a self-describing format that supports a hierarchical structure, which is excellent for organizing large datasets, and it is optimized for fast I/O slicing, enabling efficient training on datasets that may be too large to fit into memory.

The simulation data from ROOT file is restructured into Jet-wise uniformed array. These arrays are then systematically stored within a single HDF5 file, organized under a hierarchical group structure to maintain clarity and ease of access. For each jet, the data is partitioned into three distinct groups:

- **Jets:** An array containing the kinematic properties of the jet itself.
- **Tracks:** A fixed-size 2D array containing the properties of every track associated with the jet. Jets with fewer tracks than the maximum are padded.
- **SV:** A fixed-size 2D array containing the truth-level properties of the secondary vertices within the jet.

2.6 Conversion Module

A dedicated conversion module was developed to systematically restructure the event-wise, non-homogeneous arrays from the source ROOT files into the structured, jet-wise arrays

required by the model. This module simultaneously applies pre-processing steps according to physics-based requirements. The operation of this module was illustrated in Figure 6.

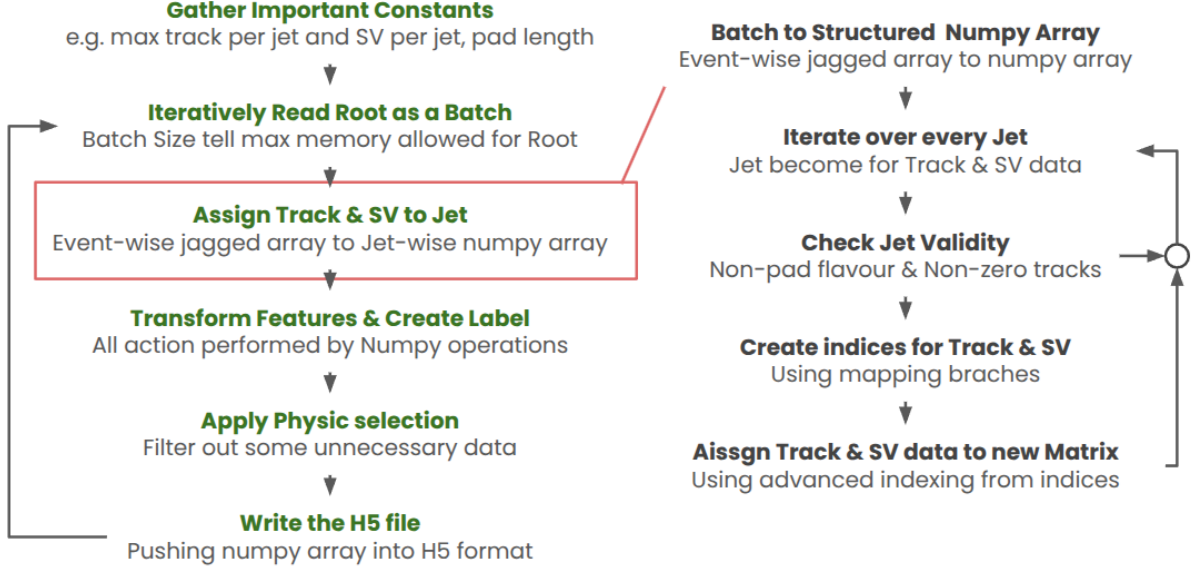


Figure 6: High-level diagram of the data conversion module's operation.

Developed in Python, the module is designed for batch processing to ensure both memory and time efficiency when handling large datasets. The core of the algorithm processes each batch of events within a single main loop. Inside this loop, instead of slow, iterative appending, advanced NumPy indexing techniques are used to directly map and assign track and SV data into pre-allocated, fixed-size, jet-wise arrays. This approach provides stable and predictable memory usage while leveraging the highly optimized, vectorized nature of NumPy. By minimizing native Python loops, this strategy mitigates the performance overhead typically associated with interpreted languages like Python.

The conversion module successfully processes ROOT files into the HDF5 format, demonstrating excellent performance and scalability. The processing time scales with the number of events; for a sample of 250k events (containing 1.6M jets), the operation took approximately 6 minutes, while a larger sample of 1M events (64M jets) was processed in approximately 30 minutes.

Data integrity is preserved throughout this process. A direct comparison between the original ROOT file and the final HDF5 file confirms that the conversion and pre-processing steps have not introduced biases. **Figure 7** validates the distributions for key jet and track features, while **Figure 8** and **Figure 9** demonstrate that the track-to-jet and SV-to-jet assignments, respectively, are perfectly preserved. All validation plots shown were produced using a sample of 250,000 events.

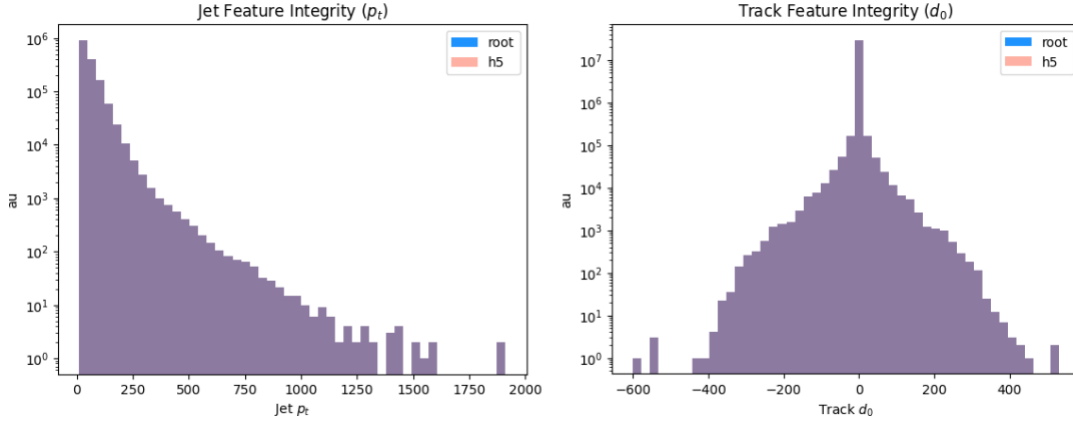


Figure 7: Validation of jet and track feature distributions, comparing the source ROOT data with the processed HDF5 data.

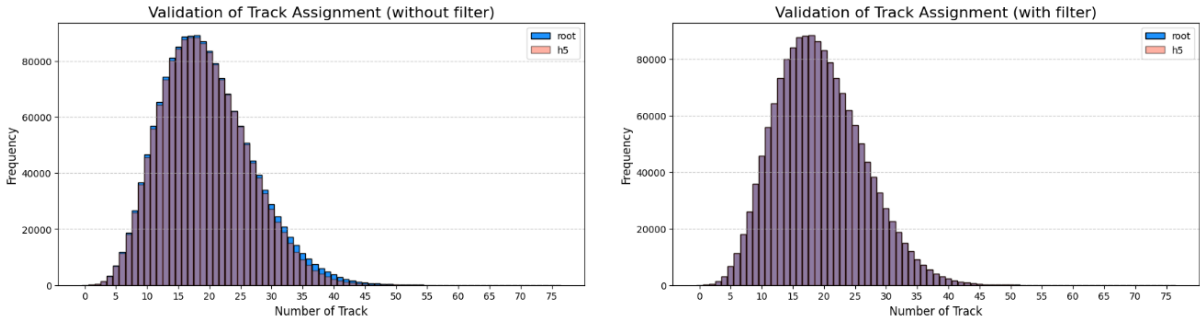


Figure 8: Validation of the track-to-jet assignment, showing a perfect correlation between the number of tracks per jet in the source and processed files.

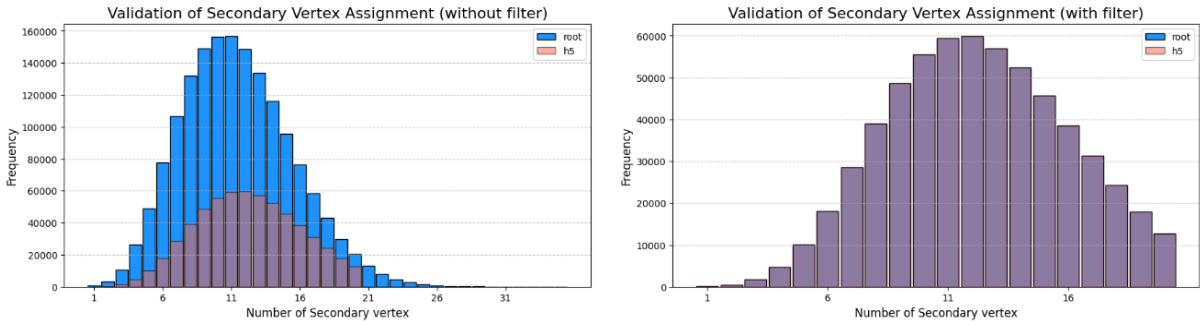


Figure 9: Validation of the SV-to-jet assignment, showing a perfect correlation in the number of secondary vertices per jet. Note that the distribution's shape is affected by the exclusion of light-flavor jets during pre-processing.

During data pre-processing, the following filtering and selection criteria were applied:

- **Jet Filtering:** Jets were removed if they were identified as background (with a truth flavor label of -99) or if they contained no reconstructed tracks.
- **Track Selection:** For each jet, a maximum of the 50 tracks with the highest transverse momentum (p_T) were retained.

- **Secondary Vertex Limit:** A limit was placed on the number of secondary vertices stored per jet. While this was set to 20 for the validation figures to retain 95% of the data, a tighter limit of 5 is planned for the final training dataset. Additionally, SVs within light-jets are flagged and excluded from certain calculations.

2.7 Dataset Summary

The final dataset used for this work is derived from a simulation of **1M events**, which initially contained approximately 64M jets. It should be noted that while this dataset is suitable for the development and validation of our reconstruction algorithm, it is a preliminary sample and has not yet undergone full physics validation.

After applying the selection criteria described in the previous section, the total number of jets available for the study was reduced to approximately **12.6M jets as a final number** used to produce the plots in this report. This final dataset was then partitioned into three distinct subsets for training, validation, and testing. A detailed summary of the dataset splits and properties are provided in Table 1 and Table 2 respectively.

Table 1: Summary of the dataset used for training and evaluation. The table shows the number of jets in each partitioned subset after all filtering and pre-processing steps have been applied, with a breakdown by jet flavor.

Split	Total Jets	B Jets	C Jets	L Jets
Training	10 000 000	1 400 000	600 000	8 000 000
Validation	1 300 000	185 000	75 000	1 040 000
Testing	1 300 000	185 000	75 000	1 040 000
Total	12 600 000	1 770 000	750 000	10 080 000

Table 2: Overall statistical properties of the final dataset used in this project. Note: These statistics exclude light-flavor jets (L-Jets) that do not contain any secondary vertices.

Property	Mean	Minimum	Maximum
Tracks per Jet	3.26	1	20
SVs per Jet	1.56	1	5
<i>b</i> -SVs per Jet	1.23	0	5
<i>c</i> -SVs per Jet	0.33	0	5

All input features were transformed using standard normalization. For the regression targets (the SV properties), a logarithmic transformation ($\log(x)$) was used to compress the range of the values. Following this, the resulting distribution was also transformed using the standard normalization.

3 Model Result and Analysis

The MaskFormer model was trained on the final dataset using the same hyperparameter settings as the reference architecture [2]. To benchmark the performance and understand the impact of our more realistic, imbalanced dataset, the model was also trained on the reference dataset from [2], which is a balanced sample of 13 million jets with an equal

number of b-, c-, and light-flavor jets. The model’s performance was then evaluated for each of the three primary tasks: classification, track assignment, and vertex regression.

3.1 Secondary Vertex Classification

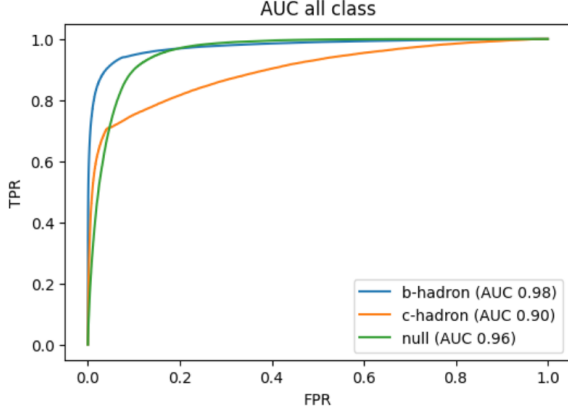


Figure 10: ROC curves for the model trained on our dataset.

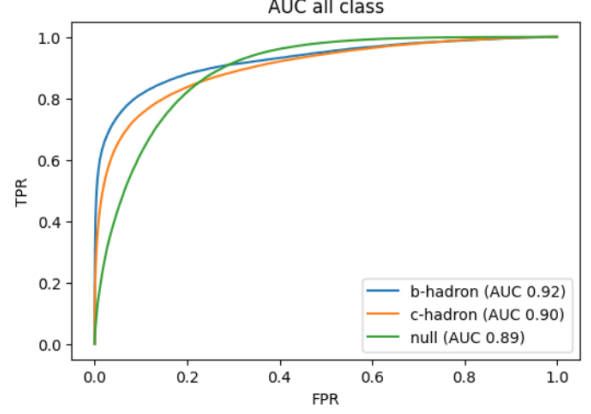


Figure 11: ROC curves for the model trained on the balanced reference dataset.

Despite the significant class imbalance in our dataset (a much smaller proportion of b- and c-jets), the model achieves impressive classification performance, as shown by the ROC curves in Figure 10. This performance is comparable to that of the model trained on the balanced reference dataset (Figure 11). However, a performance degradation is visible for the c-jet classification in our sample, likely attributable to the extremely low statistics for this class.

3.2 Track Assignment (Mask Prediction)

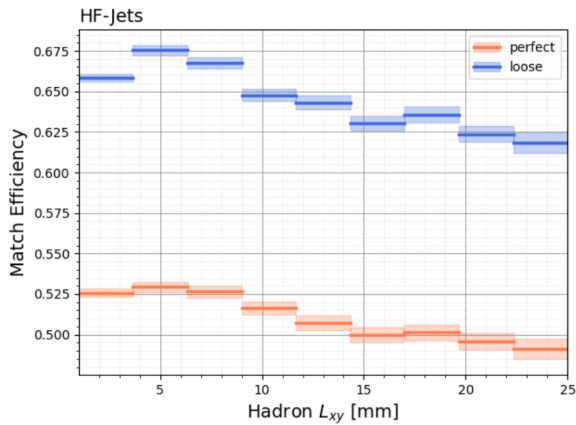


Figure 12: Masking efficiency as a function of transverse displacement (L_{xy}) for the model trained on our dataset.

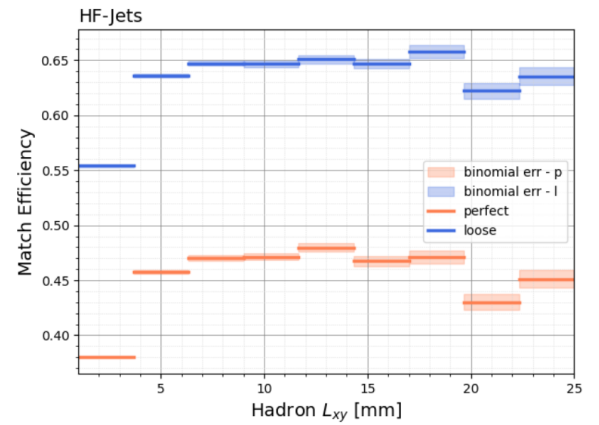


Figure 13: Masking efficiency as a function of transverse displacement (L_{xy}) for the model trained on the reference dataset.

The track assignment performance, or masking efficiency, is comparable between the two models (Figures 12 and 13). The model trained on our dataset shows a slightly higher

'perfect match' efficiency, while the model trained on the reference set exhibits a more stable performance with smaller uncertainty bands.

3.3 Secondary Vertex Regression

The vertex regression task proved to be the most challenging. Figure 14 shows the correlation between the true and predicted vertex properties for the model trained on our dataset. While the model shows some ability to predict the angular displacement (ΔR), it largely fails to capture the transverse displacement (L_{xy}), indicating significant under-performance. In stark contrast, the same model trained on the balanced reference dataset achieves a near-perfect correlation for both properties, as shown in Figure 15. This suggests that the regression task is highly sensitive to the dataset's characteristics, representing a key area for future improvement. The poor regression performance was also likely exacerbated by the use of a preliminary dataset, as time constraints prevented the generation of a more realistic, experimentally-validated simulation sample.

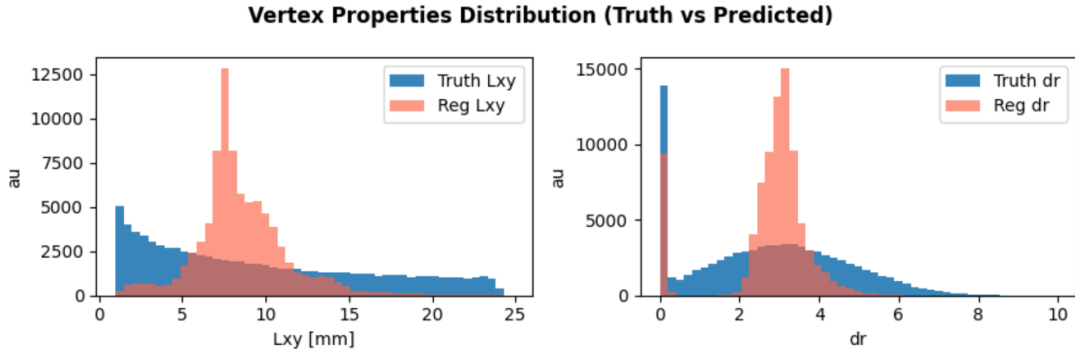


Figure 14: Truth vs. Prediction distributions for vertex properties on our dataset.

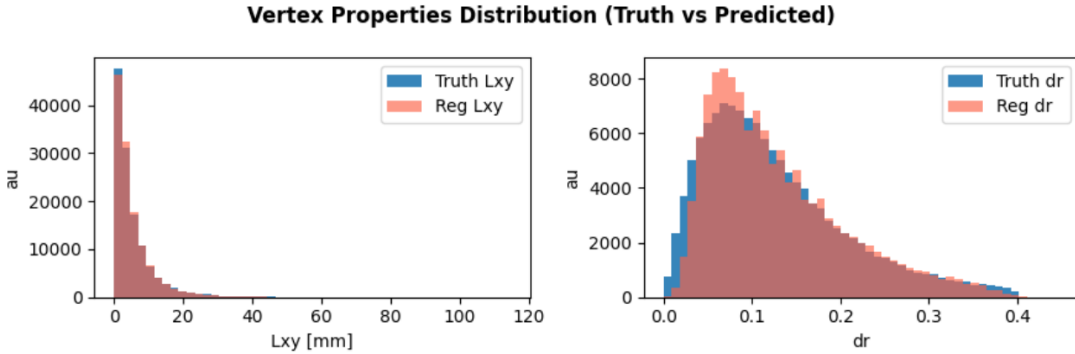


Figure 15: Truth vs. Prediction distributions for vertex properties on the reference dataset.

4 Conclusion

This report addressed the critical challenge of secondary vertex (SV) reconstruction in the high-occupancy environment expected at the High-Luminosity Large Hadron Collider (HL-LHC). We introduced and evaluated a novel approach based on the MaskFormer architecture. A key component of this work was the development of a dedicated data

processing pipeline to convert domain-specific ROOT files into a uniform, jet-wise HDF5 format suitable for modern machine learning frameworks.

The adapted MaskFormer architecture demonstrated significant promise, particularly in the classification and track assignment (masking) tasks. Despite being trained on a preliminary, highly imbalanced dataset with non-optimized hyperparameters, the model achieved classification performance comparable to a baseline trained on a perfectly balanced reference sample. This result validates the strength of the transformer’s attention mechanisms for learning powerful representations even with limited statistics for certain classes. However, the study also revealed a key challenge: the vertex regression task proved to be highly sensitive to the dataset’s composition. The model struggled on our imbalanced sample but performed exceptionally well on the balanced reference dataset. This discrepancy provides strong evidence that the model’s regression capabilities could be significantly improved by training on a more realistic, fully validated physics dataset.

In summary, this work successfully establishes the MaskFormer architecture as a powerful and viable new direction for secondary vertex reconstruction. It excels at the difficult task of correctly grouping tracks to vertices and provides a strong foundation for future research and optimization aimed at delivering a complete, high-performance SV-finding algorithm for the HL-LHC era.

5 Discussion

The results presented in the previous section highlight several key aspects of the adapted MaskFormer’s performance on the realistic, imbalanced SV reconstruction dataset. This section discusses the most salient observations, including the challenges of multi-task learning, the impact of data normalization, and a brief study of hyperparameter sensitivity.

5.1 Imbalance in Multi-Task Learning

A key observation during training was the disparity in learning speeds across the different tasks. The losses for the classification and mask prediction tasks converged significantly faster and reached lower plateaus than the loss for the vertex regression. This suggests an imbalance in the gradient magnitudes contributed by each task head, where the regression task may be under-weighted during the optimization process. Consequently, the relative weights of each component in the total loss function are critical hyperparameters that require careful tuning to ensure balanced and effective training across all objectives.

5.2 Impact of Target Normalization on Performance

The choice of normalization for the regression targets was found to have a significant and somewhat counter-intuitive cross-task impact, particularly on the performance of the SV classification head. As shown in Figure 16, standard normalization (Z-score) yielded the most stable and lowest loss for the regression task itself when compared to Min-Max scaling.

However, this choice had a notable effect on the classification task. Figure 17 illustrates that using Min-Max scaling, which bounds the regression targets to a smaller magnitude (e.g., within $[0, 1]$), resulted in a significantly lower and more stable classification loss. This suggests a delicate interplay between the tasks, where the scale of the regression

targets can affect the stability and performance of the classification head, revealing a crucial trade-off to consider during model optimization.

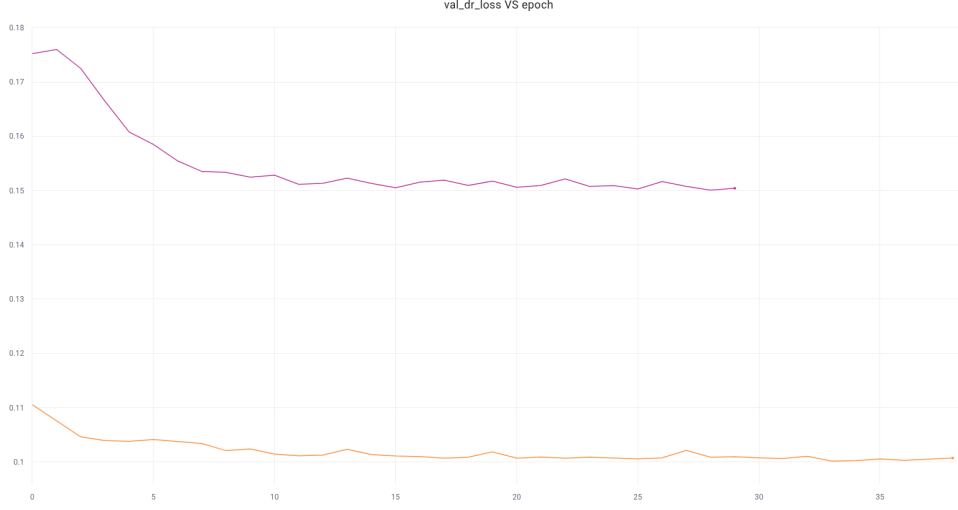


Figure 16: Comparison of the validation loss for the ΔR regression task using two different target scaling methods: Standard Normalization (pink) and Min-Max Scaling (orange).

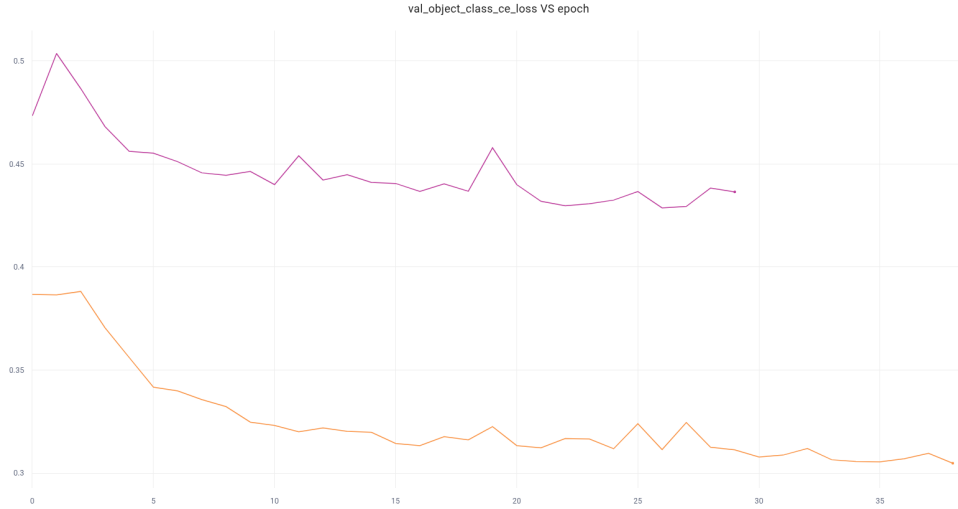


Figure 17: Comparison of the validation loss for the SV classification task, demonstrating the significant performance improvement when using Min-Max scaling for the regression targets.

5.3 Hyperparameter Sensitivity Study

A brief sensitivity study was conducted on key hyperparameters to understand their impact. The model’s performance showed little sensitivity to the batch size, with similar results obtained for values of 5,000, 10,000, and 20,000. Conversely, the learning rate had a more noticeable effect. While the regression task was largely insensitive across the tested range, the mask prediction and classification tasks benefited from smaller values. A maximum learning rate of 2.5×10^{-4} yielded the best results compared to 5×10^{-4} and 1×10^{-3} .

5.4 Recommendations for Future Work

Based on the observations in this report, several promising directions for future study emerge. First, in terms of hyperparameter optimization, future work should prioritize a large batch size for efficiency, paired with a relatively small learning rate. The most critical area for tuning, however, appears to be the careful adjustment of the individual loss weights and the choice of target normalization, given the strong coupling observed between the regression and classification tasks.

Beyond parameter tuning, a promising architectural direction is to explore a two-stage training regimen. This could involve a self-supervised pre-training phase where the transformer encoder-decoder is trained on a pretext task, such as masking and reconstructing input track features. Following this, the pre-trained backbone could be fine-tuned with the task-specific MLP heads for classification, masking, and regression. This approach could help the model learn more robust feature representations before tackling the imbalanced, multi-task downstream problem.

References

- [1] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. “Per-Pixel Classification is Not All You Need for Semantic Segmentation”. In: *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*. 2021. arXiv: 2107.06278. URL: <https://arxiv.org/abs/2107.06278>.
- [2] Samuel Van Stroud. *Secondary Vertex Reconstruction with MaskFormers*. 2024. arXiv: 2312.12272. URL: <https://arxiv.org/pdf/2312.12272>.
- [3] L. Sailer et al. *Open Data Detector*. Version v2.1.1. Oct. 2021. DOI: 10.5281/zenodo.5572213. URL: <https://doi.org/10.5281/zenodo.5572213>.
- [4] Christian Bierlich et al. “A comprehensive guide to the physics and usage of PYTHIA 8.3”. In: *SciPost Phys. Codebases* (2022), p. 1. DOI: 10.21468/SciPostPhysCodeb.1. arXiv: 2203.11601.
- [5] K Edmonds et al. *The Fast ATLAS Track Simulation (FATRAS)*. Tech. rep. Geneva: CERN, 2008. URL: <https://cds.cern.ch/record/1091969>.
- [6] Xiacong Ai et al. “A Common Tracking Software (ACTS) for the LHC and future experiments”. In: *J. Instrum.* 18.01 (2023), P01021. DOI: 10.1088/1748-0221/18/01/P01021. arXiv: 2209.08693.
- [7] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. “FastJet user manual”. In: *Eur. Phys. J. C* 72 (2012), p. 1896. DOI: 10.1140/epjc/s10052-012-1896-2. arXiv: 1111.6097.
- [8] R. Brun and F. Rademakers. “ROOT - An object oriented data analysis framework”. In: *Nucl. Instrum. Meth. A* 389 (1997), pp. 81–86. DOI: 10.1016/S0168-9002(97)00048-X.
- [9] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019, pp. 8026–8037. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.