Electron-Proton Scattering Event Generation using Structured Tokenization

Steven Goldenberg
Data Science
Jefferson Lab
Newport News, VA 23606
sgolden@jlab.org

Diana McSpadden
Data Science
Jefferson Lab
Newport News, VA 23606
dianam@jlab.org

Nobuo Sato
Theory Division
Jefferson Lab
Newport News, VA 23606
nsato@jlab.org

Kostantinos Orginos
Department of Physics
The College of William and Mary
Williamsburg, VA 23185
kostas@jlab.org

Robert Edwards
Theory Division
Jefferson Lab
Newport News, VA 23606
edwards@jlab.org

Abstract

Recent work such as Omnijet-α has demonstrated that effective tokenization combined with transformer-based architectures can produce valuable foundation models for jet physics. While tokenization may help models capture generalizable event characteristics, it also introduces discretization errors that may compromise the precision required for downstream physics analyses. As the number and complexity of the particle features grow, these errors are likely to grow proportionally. In this study, we investigate new tokenization strategies to improve the application of generative transformer models to PYTHIA8 simulations of electron-proton scattering at the Electron-Ion Collider. Specifically, we propose a feature-based structured tokenization approach that utilizes multiple tokens per particle, improving expressivity, while reducing the total number of unique tokens needed. We evaluate this method against grid-based binning, K-means clustering, and vector-quantized variational auto-encoders on event simulations. Our results show that feature-based structured tokenization reduces discretization error, leading to more accurate generative modeling of particle-level events.

1 Introduction

With the experimental programs at Jefferson Lab's 12 GeV upgrade and the planned Electron-Ion Collider (EIC) [1], the nuclear physics community is entering a new era of exploration into the core of nuclear matter, aiming to understand confined systems of quarks and gluons inside protons and neutrons. The experiments at Jefferson Lab are collecting an unprecedented volume of scattering data, with even more expected from the EIC, operating at different energy regimes. These data must be processed and analyzed using the theory of strong interactions in order to infer the internal properties of hadronic systems.

Strictly speaking, the scientific task is an inference problem, and the challenge lies in performing inference using large-scale data. The classical approach has been to compress the data into lower-dimensional projections that can be directly compared with theoretical predictions. However, this approach suffers from a variety of limitations including the need to remove detector effects (which introduces model-dependent biases and irreducible systematic uncertainties) and the inherent loss of information associated with dimensionality reduction, to name a few.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

To address this challenge, we develop a particle-level tokenization scheme to enable the use of transformer-based models for inference tasks in nuclear particle physics. Rather than relying on traditional lower-dimensional projections, our approach operates directly on the particle-level outputs, such as those reconstructed from scattering experiments. In these experiments—such as collisions between incident electrons and a nuclear target at Jefferson Lab (or beams at the EIC)—collision energy is transformed into new particles, from which observables such as their mass, transverse momentum p_T, azimuthal angle ϕ (relative to the collision axis), and pseudorapidity η can be reconstructed from signals captured by the detectors. These reconstructions provide a natural basis for sequence modeling. We evaluate several tokenization strategies for representing these quantities and assess how different tokenizations affect transformer-based reconstruction and generation of realistic particle-level events.

To model these tokenized particle sequences, we utilize the transformer architecture, which has proven highly effective for sequence modeling tasks across various domains. Introduced by Vaswani et al. [2], the transformer model initiated the large language model revolution by replacing recurrence with multi-head self-attention, enabling efficient modeling of both local and global dependencies. When trained autoregressively, transformers factorize the joint probability of a sequence using the chain rule, $P(x) = \prod_{t=1}^{T} P(x_t \mid x_{1:t-1})$, where each token x_t is predicted from its predecessors. Through training, these models learn statistical patterns and contextual relationships between tokens, which enables them to standardize heterogeneous inputs by mapping to a shared embedding space. Because attention is global, transformers are able to generate across long particle sequences, preserving both local structure and global event-level dependencies.

Transformers require discrete inputs for categorical features, e.g. particle identifiers (pIDs), and may benefit from improved compression when generating complex, high-dimensional data. A key question in this domain is how to best tokenize continuous features. Significant research has gone into tokenization of continuous data for large language models, including techniques that utilze separate text and number "heads" such as xVal [3]. Formatting numbers consistently with scientific notation like P1000 [4] or by adding whitespace to enforce single digit number tokens [5] has shown to improve performance on a wide range of mathematical and scientific tasks. Specifically for particle physics, VQ-VAE models have been used to encode full particles into single tokens [6]. For the most direct comparison to prior particle physics work, we evaluate multiple tokenization strategies including grid-based binning, K-means clustering, VQ-VAE, and our novel feature-based structured tokenization method on generated electron-proton scattering events from PYTHIA8.2[7].

Our evaluation covers both reconstruction of training events to verify the tokenization methods, as well as statistics on model-generated events. For tokenization metrics, we evaluate reconstruction errors using per-feature mean squared error (MSE), codebook utilization, normalized perplexity $\|P\|$, and the Jensen-Shannon (JS) distances between binned distributions of the three continuous features. Statistics for pID reconstruction are omitted as our tokenization strategies are specifically designed to reproduce them with 100% accuracy. For generated events, reconstruction losses cannot be computed, but distribution shifts for all features, including pIDs, are measured using JS distances on their histograms. We additionally show the distribution of event lengths for each tokenization strategy.

2 Dataset

We test our tokenization methods with particle level information from the PYTHIA8 event generator. We simulate electron-proton collisions events at a EIC configurtation with a center of mass energy of 140 GeV. For each event, we collected all possible final state asymptotically stable particles with pseudorapidity $\eta < 5$ and recorded a 4-dimensional feature array $(\mathrm{pID}, p_T, \eta, \phi)$ for each particle. The largest event in our training set contains 65 particles, so we designed each model's context window to handle at least 100 particles.

The particles for each event are ordered such that the electron with the largest transverse momentum (p_T), known as the leading electron, is placed first. After assigning the leading electron, the remaining particles are placed in order of descending p_T. The azimuthal angle, ϕ, for the remaining particles is rotated relative to the leading electron. Finally, for methods other than VQ-VAE, we tokenize using $\log(p_T)$ rather than p_T to ensure positive momentum values when decoded. After initial testing, we

observed VQ-VAE performance degradation when training with $\log(p_T)$ (MSE of 0.176 vs. 0.023); therefore, we present results trained with p_T for that method.

3 Methods

3.1 Baseline tokenization methods

We compare our structured tokenization to three baseline strategies: uniform, K-means, and VQ-VAE. To ensure accruate particle IDs (pIDs), all baseline strategies first tokenize the continous features (p_T, η, ϕ) into a token $C \in [0, d^3)$ for $d \in \{10, 15\}$. Then we add Td^3 to each token where T is the tokenized pID (0-19) leading to a final token ID of $C + d^3T$. The uniform strategy for continous features maps each particle to a cell in a 3D (d x d x d) grid and uses the cell ID as the token. The K-means strategy uses a standard K-means algorithm to fit the particle features with d^3 clusters.

Inspired by Omnijet-α [6], we implement a VQ-VAE[8] that tokenizes the three continuous particle features into a discrete code, C, from a learnable embedding codebook. This allows the model to map the features into an embedding space that may better cluster information as compared to K-means. Our VQ-VAE model is only tested with d^3 codes where $d = 10$. A description of our model and training is given in Appendix B.1.

3.2 Feature-based structured tokenization

Instead of mapping each particle to a single token, feature-based structured tokenization tokenizes each feature individually resulting in four tokens per particle. We also add an indicator token for `<particle_start>` to guide the sampling process. This idea more closely mimics tokenization of language, where each token may be meaningless on its own, but contains richer meaning in context.

While this requires longer sequences to generate full events, the increase in tokens is linear while the increase in expressivity is exponential. This allows us to utilize significantly fewer tokens, with 20 reserved tokens for particle IDs and $d^3 = 1000$ (i.e. $d = 10$) tokens for continous features. The continous features utilize the naive grid-based binning strategy, but we can reuse the same 1000 tokens for each feature. This would be equivalent to using 1000^3 uniform bins. An example of these sequences is given in Appendix A. These sequences are five times longer, and therefore training and inference times are proportionately larger: around two and a half hours for training compared to 37 minutes using a single Nvidia A800 GPU and the Accelerate library for PyTorch [9]. However, the structured sequences are still significantly shorter than the context window of standard large language models.

3.3 Generation

To generate new events, we train a transformer-based machine learning model for each tokenization strategy. The model architecture and training parameters are given in Appendix B.2. After training, we generate new events by auto-regressively sampling the output token distribution. For all models except the structured tokenization, we sample from a standard multinomial distribution. For structured tokenization, we constrain the generation process based on the expected token type. Therefore particle IDs are always generated after a particle start token, followed by three tokens that correspond to the three continuous features. Generation of each event stops once an `<end_event>` token is reached.

4 Results and Discussion

In Table 1, we provide a variety of metrics on the tokenization and reconstruction of the training data. Metrics for the particle IDs are not shown as all six strategies by definition ensured IDs maintained fidelity after tokenization. Even though the structured tokenization method utilizes an order of magnitude fewer tokens, we see it outperforms all other strategies for codebook utilization, reconstruction errors, and JS distance on the continous features.

		Codebook		MSE			JS Distance		
	Tokens	Util.	$\|P\|$	p_T	η	ϕ	$\log(p_T)$	η	ϕ
Uniform	20000	23.5%	0.056	2.890	0.336	0.132	0.731	0.728	0.729
	67500	16.0%	0.050	0.907	0.149	0.058	0.690	0.689	0.692
K-means	20000	43.7%	0.173	0.106	0.031	0.030	0.086	0.104	0.145
	67500	31.6%	0.161	0.047	0.013	0.013	0.041	0.058	0.093
VQ-VAE	20000	47.1%	0.164	0.023	0.013	0.016	0.149	0.086	0.142
Structured	1020	**100%**	**0.384**	**1.2e-4**	**3.3e-5**	**1.3e-5**	**0.016**	**0.002**	**0.020**

Table 1: Tokenization metrics on the 8,000 training samples. $\|P\|$ is the normalized perplexity defined by the total codebook perplexity divided by the number of codebook entries. The JS distance is calculated over 100 bins.

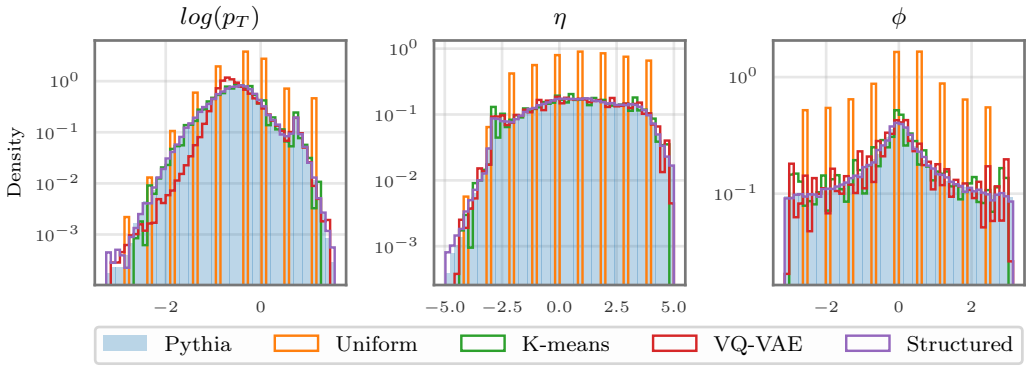

Figure 1: Histograms of the $\log_{10}$ transverse momentum (pT), pseudorapidity (η), and azimuthal angle (ϕ), comparing PYTHIA8 events over 50 bins with each tokenization method using $d = 10$.

	$\log(p_T)$	η	ϕ	pID
Uniform	0.731	0.728	0.729	0.004
	0.690	0.689	0.692	**0.003**
K-means	0.085	0.103	0.145	0.005
	0.041	0.058	0.094	**0.003**
VQ-VAE	0.148	0.086	0.142	0.004
Structured	**0.015**	**0.010**	**0.023**	0.006

Table 2: JS distance between 8,000 generated and training events calculated over 100 bins except pID, which is calculated over the 20 particle species.

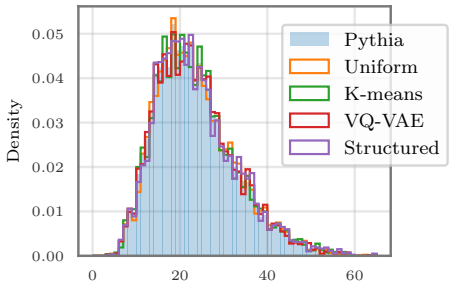

Figure 2: Comparison of the real PYTHIA8 event lengths with the generated event lengths for each model.

Table 2 shows remarkably similar results when compared to pure reconstruction, indicating that the transformer model was capable of learning the proper token distributions for generation, and that the limiting factor was discretization error. Again, we see the structured tokenization method outperforms our baseline strategies, except for pID generation, which shows a slight decrease in performance. The differences in pID generation are small enough to be negligible given the inherent noise in the generation process. As a visual proof of performance, Figure 1 and 2 show histograms over the continuous features and the event sequence lengths respectively, comparing the training data and generated events from each model. Figure 1 shows a shift below $p_T = 1$ for the VQ-VAE $\log(p_T)$ distribution, which could be explained by our tokenizer training on p_T directly as described in section 2.

4.1 Future work

While our work shows significant improvements for the tokenization of electron-proton scattering events, numerous opportunities remain for future investigation. First, we hope to show in a future publication improved performance for downstream inference tasks, beyond generation of accurate particle kinematic distributions. To enhance reproducibility, the associated code will be made publicly available alongside that work.

Second, the way we handle tokenization of the pID for our baseline events results in far more tokens than are necessary based on our reported utilization and normalized perplexity metrics. For example, the VQ-VAE model could use an embedding layer for the pIDs to produce a vector that could be concatenated with the continuous features. Alternatively, there is interesting work on utilizing continous features directly without tokenization that should be evaluated [10].

Finally, we would like to extend these ideas to allow for conditional generation. One clear benefit of structured tokenization is the ability to add a single structure token for conditioning that indicates a new type of value, such as center of mass energy. Our work also indicates that current large language models could possibly be repurposed for these tasks by fine-tuning them on highly structured and normalized data.

Acknowledgments and Disclosure of Funding

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

References

[1] R. Abdul Khalek et al. Science requirements and detector concepts for the electron-ion collider: EIC yellow report. Nuclear Physics A, 1026:122447, 2022.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in Neural Information Processing Systems, 30, 2017.

[3] Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, et al. xVal: A continuous number encoding for large language models. arXiv preprint arXiv:2310.02989, 2023.

[4] François Charton. Linear algebra with transformers. arXiv preprint arXiv:2112.01898, 2021.

[5] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. 2021.

[6] Joschka Birk, Anna Hallin, and Gregor Kasieczka. Omnijet-α: the first cross-task foundation model for particle physics. Machine Learning: Science and Technology, 5(3):035031, 2024.

[7] Torbjörn Sjöstrand, Stefan Ask, Jesper R Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O Rasmussen, and Peter Z Skands. An introduction to PYTHIA 8.2. Computer Physics Communications, 191:159–177, 2015.

[8] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. Advances in Neural Information Processing Systems, 30, 2017.

[9] Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, and Sourab Mangrulkar. Accelerate: Training and inference at scale made simple, efficient and adaptable. `https://github.com/huggingface/accelerate`, 2022.

[10] Michael Tschannen, Cian Eastwood, and Fabian Mentzer. GIVT: Generative infinite-vocabulary transformers. In European Conference on Computer Vision, pages 292–309. Springer, 2024.

[11] Minyoung Huh, Brian Cheung, Pulkit Agrawal, and Phillip Isola. Straightening out the straight-through estimator: Overcoming optimization challenges in vector quantized networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, Proceedings of the 40th International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 14096–14113. PMLR, 23–29 Jul 2023.

A Example of structured tokenization

For a given sequence of particles in an event, our baseline tokenization methods simply add tokens for the start and end of the event as seen below where p_i represents the encoded token for particle i:

$$(\texttt{e\_start}, p_1, p_2, \cdots, p_n, \texttt{e\_end}). \tag{1}$$

Structured tokenization splits each particle into its four components with an additional particle start token. Therefore, our event sequences are represented as:

$$(\texttt{e\_start}, \texttt{p\_start}, \text{pID}_i, p_{T_i}, \eta_i, \phi_i, \texttt{p\_start}, \cdots, \phi_n, \texttt{e\_end}). \tag{2}$$

This means that each token sequence is approximately five times longer for the structured tokenization method. Utilizing longer sequences can lead to increased training and inference times, however the sequence lengths are still far smaller than the context window for standard large language models ($\approx 500 \ll O(1e5)$)

B Model architectures and training

B.1 Vector-quantized variational auto-encoder

Encoder-decoder backbone:

- Input-projection layer: a linear projection maps the 3-dimensional input into a hidden space dimension of $H = 100$.
- Transformer blocks: the encoder and decoder are composed of NormFormer layers (an improved transformer variant with pre-layer normalization and activation scaling). Each stack contains $L = 4$ layers, with $B = 4$ MLP sub-blocks per layer.
- Multi-head self-attention: each layer uses $n_h = 10$ attention heads, so each head processes $H/n_h = 10$ features. Attention is causal, enforcing autoregressive sequence modeling.

Vector quantization: The encoder output is passed to the `vqtorch` `VectorQuant` layer that maps the 100-dimension continuous embeddings to discrete codebook entries:

- **Codebook size**: 1000 entries
- **Latent dimension**: 4
- **Codebook Initialization**: k-means clustering of training samples
- **Commitment loss**: weighted by β= 0.9. This parameter encourages the stability of codebook assignments while promoting codebook usage through the `VectorQuant` loss computation: $\mathcal{L}_{\text{VQ}} = (1-\beta)\,\|z_e - \text{sg}[z_q]\|_2^2 + \beta\,\|\text{sg}[z_e] - z_q\|_2^2$, where z_e is the encoder output, z_q is the quantized codebook vector, β is the commitment weight, and $sg[\cdot]$ is the stop-gradient operator used to treat the input tensor as a constant for backpropagation. For commitment loss, $(1-\beta)\|z_e - \text{sg}[z_q]\|_2^2$, updates only the codebook z_e since gradients do not flow into z_q. Codebook loss, $\beta\|\text{sg}[z_e] - z_q\|_2^2$, updates only the codebook z_q since gradients do not flow into z_e.
- **Training objective**: The model minimizes a standard VQ-VAE objective: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \beta\mathcal{L}_{\text{commit}}$, where $\mathcal{L}_{\text{recon}}$ is the reconstruction loss between the input and the decoder output, and $\mathcal{L}_{\text{commit}}$ penalizes the distance between the encoder and its assigned codebook vectors. $\beta\mathcal{L}_{\text{commit}}$ is $\mathcal{L}_{\text{VQ}}$ in the Commitment loss definition.

- **Codebook maintenance**: Code vectors are synchronized every step with replacement every 7 steps.
- **Codebook grouping**: 100 affine groups [11], each with 10 codes, are used.
- **Optimization**: AdamW with learning rate 10^{-3}
- **Scheduler**: constant learning rate decay with $\gamma = 0.5$
- **Training**: up to 200 epochs with batch size 512 on an A800 GPU. The best checkpoint on a validation set is used.

B.2 Generation: Transformer model architecture

Given a total number of tokens `n` and a sequence length `s`, each model uses the following layers:

- **Token embedding**: `nn.Embedding(n, 512)`
- **Positional encoding**: `nn.Embedding(s, 512)`
- Six transformer blocks made of:
  - **Multi-head Attention**: `nn.MultiHeadAttention(512, num_heads=8)`
  - **Layer normalization**: `nn.LayerNorm(512)`
  - **Feedforward MLP**: Linear layers with a feed-forward dimension of 2048, internal ReLU activation, and layer normalization

All models used the same sequence length of $s = 102$ except the structured tokenization model, which requires $s = 502$ as each particle requires five tokens instead of one. Therefore, the maximum number of particles per event is 100 for all models. The models were trained with an initial learning rate of 2e-4 for 200 epochs, a batch size of 128 events, and a linear warmup cosine scheduler with a warm up length of 5%.