Transforming Pileup — Notes

Claudius Krause and Daohan Wang

February 2024

1 Introduction

The Large Hadron Collider (LHC) aims to explore both BSM processes and SM precision measurements in particle physics by operating at very high instantaneous luminosities. This high luminosity leads to an increase in the number of inelastic proton collisions within each bunch crossing, resulting in a phenomenon known as pileup. Pileup involves numerous low-energy, soft collisions that accompany the primary, hard collision of interest. See [1] for a review. These secondary collisions produce additional particles that complicate data analysis by adding extra energy to the physical observables being measured. Currently, ATLAS and CMS experiments encounter an average of about 80 pileup collisions per bunch crossing for LHC Run 3, with expectations of this number increasing significantly to 200 in HL-LHC in Runs 4-5.

Mitigating pileup's impact is crucial for accurate data analysis. Techniques have been developed to identify and subtract the contributions from pileup, particularly focusing on distinguishing between particles originating from the hard collision (leading vertex) and those from pileup collisions. Advanced methods involve precision measurements and the use of algorithms to subtract charged and neutral pileup contributions, including sophisticated pileup removal techniques like charged-hadron subtraction (CHS), jet area method, Soft Killer algorithm and PUPPI algorithm [2].

Furthermore, many novel approaches leveraging machine learning for pileup mitigation are proposed, which generally formulate the pileup mitigation problem in the language of a ML regression problem. For example, PUMML [3] utilizes the CNN and processes the inputs including the energy distributions (jet images) of charged LV particles, charged pileup particles and all the neutral particles and output the energy distributions of neutral particles coming from leading vertex alone. PUPPIML [4] utilizes graph neural network to enhance the PUPPI algorithm. It takes the energy distributions, charges, as well as the PUPPI weights of charged LV particles, charged pileup particles and all the neutral particles as inputs, and outputs a probability distribution indicating whether particles originating from the leading vertex. PUMA leverages the sparse self attention mechanism for effective pileup mitigation for LHC data analysis. It operates on raw reconstructed observables, including four-

momentum, PID and vertex ID, and outputs the scaling of each particle's 4-momentum by the estimated fraction of hard energy.

[5] [data driven] presents a semi-supervised learning approach using graph neural networks (GNN) for pileup mitigation at the LHC. The model trains on labeled charged particles and infers on unlabeled neutral particles, leveraging a graph-based structure to capture the spatial relationships among particles. The model outputs an N-dimensional array (where N is the total number of particles per event) indicating the probability that each particle originates from the leading vertex.

Claudius: just dropping some comments I remembered from the disussion with Robert:

- the "baseline" in which CMS has the best understanding and the most data is the ECal in the barrel. The HG-CAL (high-granularity calorimeter) is a future CMS upgrade that will give us a lot of information on the particles. If we want to work with that, it will be a lot more experimental (but nevertheless much needed).
- standard algorithm in CMS is PUPPI [2]. Standard in ATLAS is soft-killer [6].
- in principle, every observable has a different, optimal pileup mitigation stragety, so PUPPI is not used everywhere. Possible observables/ applications are
  - lepton isolation
  - jet energy
  - jet substructure
  - MET
  - photon isolation
- He can help us find the right setup, such that we are useful for the experiments.

Since all acronyms of codes working on pileup contain "PU", we could call our model PUMMBA (pileup mitigation with mamba) ;-).

2 Dataset creation

As a simple scenario, we employ the QCD dijet process as our research example. (In the future, we will study Z($\to \nu\nu$)+jets and Z($\to l^+l^-$)+jets). We generate 10K parton-level events with MadGraph_aMC@NLO [7] for pp collisions at $\sqrt{s}$ = 14 TeV. Parton Shower and hadronization are performed by Pythia 8. Delphes-3.5.0 [8] is implemented for detector simulation. To simulate the pileup effects, we utilizes Pythia 8 [9] to generate 6M soft pileup events. On

average, 150 soft-QCD events are mixed with the hard scattering event, sampled from the 6M soft pileup events.

The mixing procedure is done via the PileUpMerger module in Delphes, which contains four parameters:

- **PileUpFile**: The event sample containing 6M pile-up events in binary format, which is generated by Pythia 8 and converted to binary format.
- **MeanPileUp**: The average amount of pile-up events per bunch-crossing. For each hard scattering, N pile-up events will be randomly chosen from the PileUpFile, where N is a random number following a distribution defined by the parameter PileUpDistribution with a mean MeanPileUp.
- **PileUpDistribution**: Defines the distribution of the number of pile-up events. 0 for Poisson, 1 for Uniform.
- **ZVertexSpread**: Pile-up and hard scattering events are randomly distributed in time and z position according to some parametrization specified by the user (in meters and second units). It can be either continuous or binned.

The events were clustered with FastJet 3.3.4 using the anti k_T algorithm with a jet radius of R = 0.4. A parton level p_T cut of 100 GeV was applied and up to two leading jets with $p_T > 100$ GeV and $\eta \in [-2.5, 2.5]$ were selected from each event.

For each event, we identify the primary vertex and we assume that for $|z| >$ ZVertexResolution (0.1mm) the hard interaction vertex can be distinguished from pile-up vertices. In this way, we can label all the charged pileup particles, which serves as the most important information for neutral pileup subtraction.

The training dataset encompasses 10K events, each event featuring the leading jets along with all particles contained within these jets, including both hard scattering particles and soft pileup particles. For the jets, we have exclusively retained the transverse momentum (p_T), pseudorapidity (η), azimuthal angle (ϕ), and jet mass. Following[10], each particle's attributes, such as p_T, η, ϕ, energy (E), Vertex ID, and Particle ID (PID), have been preserved. The PID classification, derived from the Energy Flow Algorithm, categorizes particles into EFlowTracks, EFlowNeutralHadrons, and EFlowPhotons.

This energy fraction regression task is distinct from previous ML approaches to the pile-up problem [5, 6], which treat the problem as one of classification: a particle is uniquely identified as arising from a hard or pile-up vertex. This definition is no longer valid when

Following [10], in this project, we consider the realistic scenario of a detector with finite spatial and energy resolutions. Based on particle flow algorithm [11], a measured PF candidate is frequently the result of several particles depositing energy in the same detector component. Accordingly, we train our models to decompose each detected particle into co-linear LV and PU components. We

Table 1: Variables for each particle that serve as input for the network.

p_{T}	particle transverse momentum
η	particle pseudorapidity
ϕ	particle azimuthal angle
E	particle energy
particle ID	EFlowPhoton (0), EFlowNeutralHadron (1), EFlowTrack (2)
vertex ID	charged LV (0), charged PU (1), neutral (-1)
labels	hard energy fraction of each particle

define the hard energy fraction of each particle as

$$\hat{y} = \frac{E_{LV}}{E_{LV} + E_{PU}} \tag{1}$$

In this regression task, we have labeled each particle within the leading jet of each event in the training set with the hard energy fraction. Our objective is for the model to accurately predict the energy fraction of each particle in the leading jets of each event in the validation set after training. And we aim to accomplish pileup subtraction through a rescaling process. In order to make our model focus more on accurately modeling particles with high p_T, we impose a rescaling factor to the MSE loss function:

$$\mathcal{L}\left(\{\mathbf{x}_1\}, \{\mathbf{x}_2\}, ..., \{\mathbf{x}_N\}, \{y_1\}, \{y_2\}, ..., \{y_N\}; \Theta\right) = \sum_i^N \frac{p_{\mathrm{T},i}^{\alpha}}{p_{\mathrm{T},J}^{\alpha}} ||g(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_i; \Theta) - y_i||^2 \tag{2}$$

where α is an adjustable hyperparameter, i is the index over all the particles contained in the leading jet and N is the total number of particles inside the leading jet, $\mathbf{x}_i$ is the feature vector of particle i, y_i is the hard energy fraction of particle i, Θ denotes free parameters of the model g, and g() is the prediction of the model for particle i.

3 ML setup

3.1 Basic Introduction of the Model Architecture

The whole model architecture will be implemented in the deep learning framework of PyTorch [12] with GPU acceleration. The loss function will be minimized using the Adam optimizer [13] with no weight decay.

The Selective State Space Models [14] are suitable for handling time series of variable length. The conventional State Space models are defined with four parameters (Δ, A, B, C), which define a sequence-to-sequence transformation in two stages.

- **Discretization** transforms the "continuous parameters" (Δ, A, B) to "discrete parameters" (A, B) through a fixed rule. For example, zero-order hold (ZOH) rule defined as follows:

$$\overline{A} = exp(\Delta A); \overline{B} = (\Delta A)^{-1}(exp(\Delta A) - I) \cdot \Delta B \tag{3}$$

where the most popular form of structure for matrix A is diagonal. $A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1}, C \in \mathbb{R}^{1\times N}$. N is the dimension of the latent state which is another hyperparameter.

- **Computation**. After the transformation, the model can be computed in two ways, either as a linear recurrence or a global convolution. The former way is used for efficient autoregressive inference (where the inputs are seen one timestep at a time). The latter way is used for efficient parallelizable training (where the whole input sequence is seen ahead of time). We choose the recurrence mode for our pileup subtraction task, which is defined as follows:

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \tag{4}$$

$$y_t = Ch_t \tag{5}$$

3.2 Explanation of selective SSM layer.

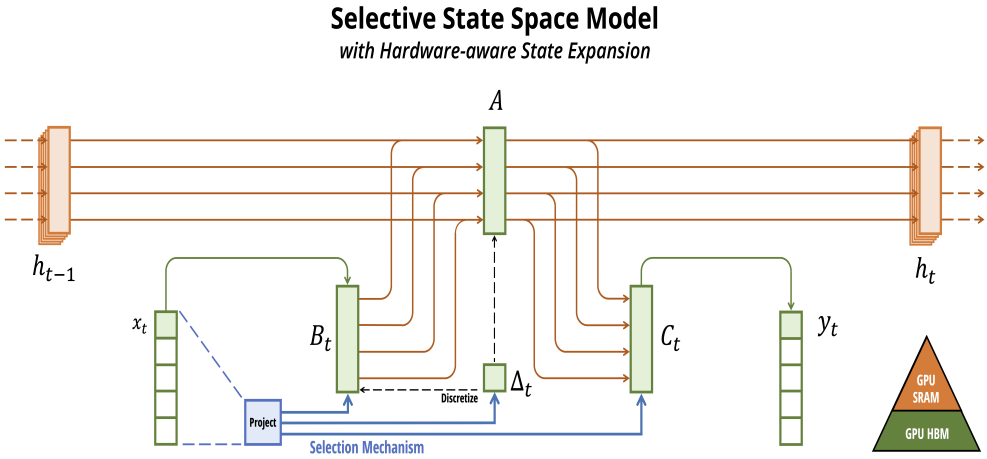

Figure 1: Illustration of Selective SSM Layer. Taken from Mamba Paper.

The critical property of selectivity is filtering out irrelevant information, thus enabling the compression of a sequence model's context into a more manageable and effective state. Within the framework of State Space Model (SSM), adjusting the parameter matrix B and C for selectivity allows finer-grained control over whether to let an input x_t into the state h_t or the state into the output y_t. "These can be interpreted as allowing the model to modulate the recurrent

Table 2: Selective State Space Model Algorithm.

Input	x:(B,L,D); Batch=32, Length(maximum=200), Dimension=6
Output	y:(B,L,1); Batch=32, Length(maximum=200), 1 means hard energy fraction
Structured N×N matrix A	←Parameter
B:(B,L,N)	← $\text{Linear}_N(x)$ (B,L,D)→(B,L,N)
C:(B,L,N)	← $\text{Linear}_N(x)$ (B,L,D)→(B,L,N)
Δ:(B,L,D)	← $\text{Softplus}(\text{Parameter}+\text{Broadcast}_D(\text{Linear}_1(x)))$
$\overline{A}, \overline{B} : (B, L, D, N)$	← discretize(Δ, A, B)
y	← SSM($\overline{A}$, $\overline{B}$, C)(x) Time-varying: recurrence (scan) only
return y	

dynamics based on content (input) and context (hidden states) respectively." As shown in Table. 2, we first impose two linear transformations to establish that the parameter matrices B and C are conditional upon the current input x_t. This selection mechanism makes the current input not only affects the current latent state h_t, but also influences the current output y_t. Then we impose $Softplus(Parameter + BroadcastD(Linear1(x)))$ to make the time step Δ depend on the current input x_t too. Δ controls the balance between how much to focus or ignore the current input x_t. "a large Δ resets the state h_t and focuses on the current input x_t, while a small Δ persists the state and ignores the current input x_t." Consequently, all the discrete parameter matrix $\overline{A}$ and $\overline{B}$ are conditional upon the current input x_t.

As shown in Fig. 1, this is how selection mechanism works. After we obtain the $\overline{A}$ and $\overline{B}$, we update the latent state of the last timestep h_{t-1} and derive the current timestep output y_t based on the time-varying: recurrence (scan) process.

3.3 Explanation of the Mamba block

The selective SSM layers are independent sequence transformations that can be seamlessly integrated into neural network frameworks. The foundational H3 architecture is the basis for the most well-known SSM architectures. These designs typically consists of a linear attention-inspired block interleaved with a MLP block. In the Mamba [15] block, they streamline the architecture by merging these two key elements into a single, uniformly stacked structure, as depicted in Figure 2. This modification draws inspiration from the gated attention unit (GAU).

The Mamba block involves expanding the model dimension D by a controllable expansion factor E. As shown in Fig. 2, For each block, most of the parameters ($3ED^2$) are in the linear projections ($2ED^2$ for input projections, ED^2 for output projection) while the inner SSM (projections for Δ, B, C and the matrix A) contributes less. The Mamba model architecture is constructed by repeating the Mamba block, interleaved with standard normalization and

residual connections. They always fix to E=2 in their experiments and use two stacks of the block to match the $12D^2$ parameters of a Transformer's interleaved MHA and MLP blocks. Finally, they use the SiLU/Swish activation function.

In our model settings, we also set E=2 and only impose one Mamba block. Now let's delve into the details of the Mamba block. The Mamba architecture, as shown on the right of Fig. 2, starts with an input that is first projected through two linear projection layers, expanding the dimensionality from D to ED. These projections increase the model's capacity for capturing complex features. Following the left linear projection layer, the sequence encounters a causal convolution layer, which processes the data while maintaining the temporal order by only using present and past information to determine the output at each step. Directly above the causal convolutional layer, there is a gating mechanism marked with σ. This represents a nonlinear transformation, typically a SiLU activation function, that modulates the information flow through the network, adding an element of adaptability to the processing. The output of the gating mechanism is then passed to a State Space Model (SSM) block. The SSM is responsible for modeling the temporal dynamics and dependencies within the sequence. Following the right linear projection layer, there is another SiLU activation function. The output of the right linear projection layer, after being processed by the SiLU activation function, is then element-wise multiplied with the output of the State Space Model (SSM) block. The result of this multiplication undergoes one final transformation through an output linear projection layer. This layer projects the data back down from the expanded dimension of ED to the original dimension D, preparing the processed sequence for output or further processing. However, since we only impose one Mamba block, the final projection layer projects the data back down from the expanded dimension of ED to 1, utilizing a sigmoid activation function to yield hard energy fractions for each particle.

3.4 Customized Mamba Model for PileUp Mitigation

4 Model Training and Evaluation

4.1 Tranining process

4.2 Evaluation metrics

4.3 Physics performance

5 Summary

References

[1] G. Soyez, *Pileup mitigation at the LHC: A theorist's view*, *Phys. Rept.* **803** (2019) 1 [1801.09721].

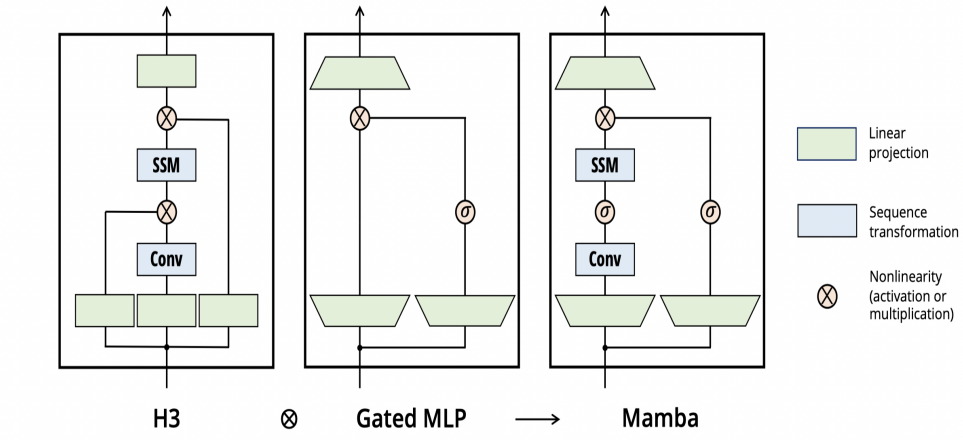

Figure 2: Illustration of Mamba block. Taken from Mamba Paper.

[2] D. Bertolini, P. Harris, M. Low and N. Tran, *Pileup Per Particle Identification*, *JHEP* **10** (2014) 059 [1407.6013].

[3] P.T. Komiske, E.M. Metodiev, B. Nachman and M.D. Schwartz, *Pileup Mitigation with Machine Learning (PUMML)*, *JHEP* **12** (2017) 051 [1707.08600].

[4] J. Arjona Martínez, O. Cerri, M. Pierini, M. Spiropulu and J.-R. Vlimant, *Pileup mitigation at the Large Hadron Collider with graph neural networks*, *Eur. Phys. J. Plus* **134** (2019) 333 [1810.07988].

[5] T. Li, S. Liu, Y. Feng, G. Paspalaki, N.V. Tran, M. Liu et al., *Semi-supervised graph neural networks for pileup noise removal*, *Eur. Phys. J. C* **83** (2023) 99 [2203.15823].

[6] M. Cacciari, G.P. Salam and G. Soyez, *SoftKiller, a particle-level pileup removal method*, *Eur. Phys. J. C* **75** (2015) 59 [1407.0408].

[7] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer and T. Stelzer, *MadGraph 5 : Going Beyond*, *JHEP* **06** (2011) 128 [1106.0522].

[8] DELPHES 3 collaboration, *DELPHES 3, A modular framework for fast simulation of a generic collider experiment*, *JHEP* **02** (2014) 057 [1307.6346].

[9] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten et al., *An introduction to PYTHIA 8.2*, *Comput. Phys. Commun.* **191** (2015) 159 [1410.3012].

[10] B. Maier, S.M. Narayanan, G. de Castro, M. Goncharov, C. Paus and M. Schott, *Pile-up mitigation using attention*, *Mach. Learn. Sci. Tech.* **3** (2022) 025012 [2107.02779].

[11] A. Sirunyan, A. Tumasyan, W. Adam, E. Asilar, T. Bergauer, J. Brandstetter et al., *Particle-flow reconstruction and global event description with the cms detector*, *Journal of Instrumentation* **12** (2017) P10003.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al., *Pytorch: An imperative style, high-performance deep learning library*, *Advances in neural information processing systems* **32** (2019) .

[13] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv preprint arXiv:1412.6980* (2014) .

[14] D.Y. Fu, T. Dao, K.K. Saab, A.W. Thomas, A. Rudra and C. Ré, *Hungry hungry hippos: Towards language modeling with state space models*, *arXiv preprint arXiv:2212.14052* (2022) .

[15] A. Gu and T. Dao, *Mamba: Linear-time sequence modeling with selective state spaces*, 2023.