# Generative machine learning for fast silicon detector simulation

*Tadej* Novak[1]*

[1]Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia

**Abstract.** Simulation of physics processes and detector response is not only a vital part of high energy physics research, but also represents a large fraction of computing cost. Generative machine learning is successfully complementing full (GEANT4-based) simulation as part of fast simulation setups improving the performance compared to classical approaches. A lot of attention has been given to calorimeters being the slowest part of the full simulation, but their simulation speed becomes comparable to that of silicon semiconductor detectors when fast simulation is used. This makes silicon detectors the next candidate for optimisation, especially with the growing number of channels in future detectors. This work explores the use of transformer architectures for fast simulation of silicon tracking detectors. The Open Data Detector is used as a benchmark detector. Physics performance is estimated comparing tracks reconstructed using the ACTS tracking framework obtained from the full simulation and the machine learning-based one.

## 1 Introduction

A large part of the physics programme of the Large Hadron Collider (LHC) relies on accurate simulation of collision events that complement the collected data. Monte Carlo (MC) simulation can be divided into four main steps [1]: generation of physics events and the immediate decays, simulation of the detector and particle interactions, digitisation of the energy and charge deposited on the sensitive regions of the detector into formats comparable with the actual detector readout, and the reconstruction of the results using the same algorithms as for data collected by the experiments. Producing MC samples represents the majority of experiments' CPU usage, e.g. the CMS experiment used 85 % of their CPU for MC in the period between 2009 and 2016, where half of it was spent on the detector simulation [2].

The LHC detector simulation is predominately done using the GEANT4 toolkit [3], which propagates each particle through individual detector volumes and uses Monte Carlo methods to trigger interactions at predefined steps. Although the simulation process can be optimized by tuning thresholds for particle selection and step lengths, it remains computationally intensive. To overcome this challenge, "fast simulation" models have been developed, where the detector response to particles with specified properties is parametrized, offering a more efficient alternative. For example the ATLAS experiment successfully integrates fast calorimeter simulation as part of AtlFast3, the most recent iteration of the ATLAS fast simulation

*e-mail: tadej.novak@cern.ch

setup [4]. To improve physics performance, deep generative models have been studied and already used in production for specific types of particles [5, 6].

By the end of the HL-LHC programme [7], the ATLAS and CMS experiments are expected to have collected up to ten times the amount of data recorded during the first three runs. As a result, many of the cutting-edge physics analyses will start to have their sensitivity limited by the available statistics of the Monte Carlo simulation. A simulated sample larger than the collected data will be required to increase the precision of Standard Model background modelling. To match the expected resource constraints, the goal is to produce up to 90 % of the MC samples using fast simulation methods. The existing setups will have to be extended to cover the whole detector, including silicon tracking detectors. This study explores the use of generative machine learning for the simulation of these detectors.

## 2 Data Representation of Silicon Detector Simulation

For this study, the Open Data Detector (ODD) [8] is used, a generic HL-LHC style tracking detector. It is loosely modelled after the ATLAS Inner Tracker (ITk) [9, 10] and comprises a pixel system, a short- and a long-strip system. In total, it describes 3332 pixel modules and 9714 strip modules. It is reasonably close to a real-world detector in terms of geometric layout, number and type of sensitive elements, and passive material.

The detector is simulated using GEANT4. Each particle interaction, with either sensitive or passive material, is called a hit. One of the following processes may occur at each hit: multiple scattering, energy loss due to ionisation and radiation (bremsstrahlung), photon conversion (electron-positron pair creation) or hadronic interaction. For simplicity only hits occurring in the sensitive detectors are considered. This allows to utilise the layered structure of the detector and describe the simulated detector response to an individual particle as a sequence of discrete hits. Simulated hit positions are illustrated in Fig. 1.

Each hit can be described with 7 features. Firstly, particle type and charge are encoded in a discrete particle ID. Each sensitive detector module also has its own geometry ID. No additional readout segmenting is performed, meaning each detector is treated as a continuous block of silicon. The hit position can be described either in terms of global or local coordinates. To ensure hits occur on the actual sensitive detector regions and not in the vacuum,

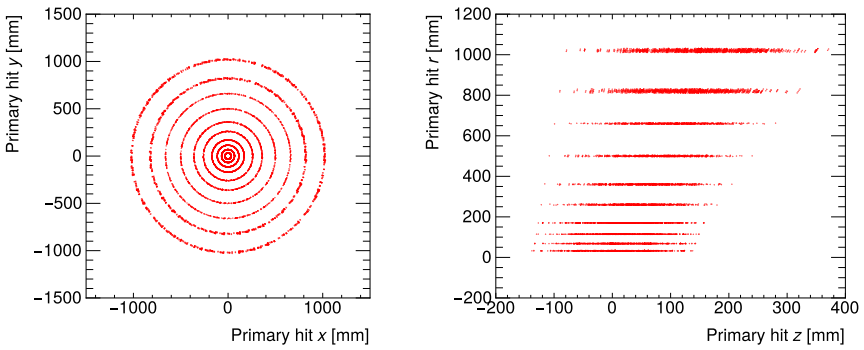


**Figure 1.** Projections of simulated hit coordinates in the $x$-$y$ plane (left) and the $z$-$r$ plane (right) using the full simulation setup. Single muon sample with muons and anti-muons of $p_{\mathrm{T}}$ between 70 and 90 GeV and $\eta$ between 0.05 and 0.25 is used.

each hit position is projected onto a corresponding tracking surface. To define it, each module is first rotated to align its thinnest side with the $z$-axis. In the $x$-$y$ plane the detector is transformed into a rectangular representation. As $z \ll x(y)$, the thickness of the detector is not included in the surface projection and the final transformation encodes the position in a 2D space yielding two local continuous coordinates. Finally the particle momentum after the hit is described with three additional continuous features.

To describe a collision event, each particle is assigned its own sequence of hits starting from the interaction point. An additional start hit is defined using a virtual module with the initial momentum and the beamspot position. The sequence ends once the particle leaves the tracking detector, also described with an additional virtual hit.

For the following results a single muon sample is used, with muons and anti-muons of transverse momentum $p_{\mathrm{T}}$ between 70 and 90 GeV and pseudorapidity $\eta$ between 0.05 and 0.25. Their production position is set to the center of the detector and smeared with a standard deviation of 50 mm in the $z$-direction, aligned with the beam. Secondary particles originating from material interactions with the detector are not considered.

## 3 The Transformer Model and the Training Setup

A transformer [11] is a deep neural network architecture used for sequence modelling and transformation, most commonly for natural language processing tasks. First, text or other discrete information is split into tokens and assigned a sequential numerical representation. As part of the training, every token gets assigned a vector in the embedding space. The model then consists of two main components, the encoder and the decoder. Both consist of multiple layers, each containing the self-attention part and the feed-forward neural network. The former determines the relative importance of each sequence element relative to the others and the latter acts as the traditional fully-connected neural network layer. In transformers, multiple attention layers run in parallel, a mechanism known as multi-head attention. Since the model contains no recurrence or convolution, a sinusoidal positional encoding is applied to add sequence order information at embedding level.

When simulating the silicon detector response, the goal of the model is to predict the next element of the sequence, where both input and output work on the same data representation. A decoder-only architecture can be used in this case. As an optimisation, all sub-sequences are modelled at the same time using masking, where all sequence elements after the current position are not taken into account in the self-attention mechanism.

Every feature is tokenised separately in its own dictionary. Continuous features are additionally discretised beforehand by rounding to two decimal places. Features are then packed together in a tuple representing a hit. Batching particle sequences this results in a 3D data


**Figure 2.** Data representation diagram, where dark blue rectangles represent features and light blue ones hits in the event sequence.

representation, illustrated in Fig. 2. Each feature is also assigned a separate embedding representation. The embeddings are then summed together and a common positional encoding is applied. The input to the multi-head attention layers is a single vector in a combined embedding space. The output of the transformer is passed through a final fully-connected layer, where the output vector dimension amounts to the sum of sizes of all tokenisation dictionaries. When segmented to individual dimensions, the resulting vectors can be interpreted as probabilities of a token on that specific position. A cross-entropy loss is computed for each feature individually along the sequence and averaged per batch. Finally losses for all features are summed and used as the training optimisation metric.

A PyTorch [12] implementation of the transformer model is used. The model follows the original implementation with `GELU` [13] used as the activation function. The parameters of the model are specified in detail in Table 1 with the final model size of 30.4 million parameters.

**Table 1.** Transformer model parameters used.

| Model parameter | Value |
|---|---|
| input dimension | 256 |
| layers | 3 |
| heads | 4 |
| feed-forward dimension | 1024 |
| activation function | `GELU` |
| dropout | 0.1 |

Training is performed using the `AdamW` optimiser [14], an enhanced version of `Adam`, where weight decay is applied during the parameter update, leading to more consistent regularisation and better generalisation. Additionally gradient clipping is enabled to prevent exploding gradients affecting the result too significantly. This is achieved by limiting the maximum value of the gradient norm to 5.0. During the training the learning rate is varied using the cosine annealing with warm restarts. In an epoch, the learning rate follows one period of the cosine function.

The training was performed over 15000 epochs on the Vega HPC [15] using 4 NVIDIA A100 Tensor Core GPUs over 6 days. 320000 input events were split in a $2 : 1 : 1$ ratio between training, validation and test samples. Full training setup details are outlined in Table 2.

**Table 2.** Model training setup parameters used.

| Training parameter | Value |
|---|---|
| events | 320000 |
| epochs | 15000 |
| optimiser | `AdamW` |
| base learning rate | 0.001 |
| weight decay | 0.01 |
| gradient clipping | 5.0 |
| batch size | 2400 |

## 4 Results

Once the transformer model is trained, the inference is performed on all of the input dataset, including the training dataset. The inference is done in an iterative way. For each particle the starting position and momentum are encoded in a virtual hit and a single-element sequence is defined. Particle type is inherently part of the hit description and conditions the model. At each iteration step $N$ the next element of the sequence is chosen. For each of the 7 hit features the most probable token is taken. While this does not directly take correlations between features into account, the attention mechanism does see the hit as a whole. As no sampling is performed, the most probable values of all features can be taken. The resulting element gets appended to the sequence and the sequence length changes to $N + 1$. If the last element is a virtual stop hit, i.e. the particle has left the detector, the iteration is stopped. Otherwise the sequence gets passed to the next iteration step. The inference rate on a single A100 GPU is about 8 s per 10000 tracks. On the other hand, running on a CPU is at the moment an order of magnitude slower compared to `Geant4`, but no optimisation has been done.

The described model is not a generative model yet, as the same input particle would get the same generated hits. To introduce randomness, a random integer between 1 and 10000 is assigned to each sequence during training as an additional feature to add a stochastic element to the problem. At inference stage, a random integer is sampled for each particle.

Alternatively, the resulting multinomial probability could be sampled. However, this is not possible in the 3D representation, as direct correlations between features are lost. While the representation could be flattened, this would increase the sequence length by 7 and increase computational complexity significantly. Therefore, this approach was not attempted at this stage.

The generated hit sequences are compared with the hits produced in the full simulation. Fig. 3 shows the comparison of number of hits simulated by either `Geant4` or the neural network. Looking at the inclusive distribution the majority of events have number of hits modelled well. Still the tail of the distribution indicates that the neural network predicts more hits that there should be. The absolute difference in hit number is exponentially falling but still a significant number of events show a difference of several hits. Note that random numbers are involved in the simulation so exact reproduction is not expected.

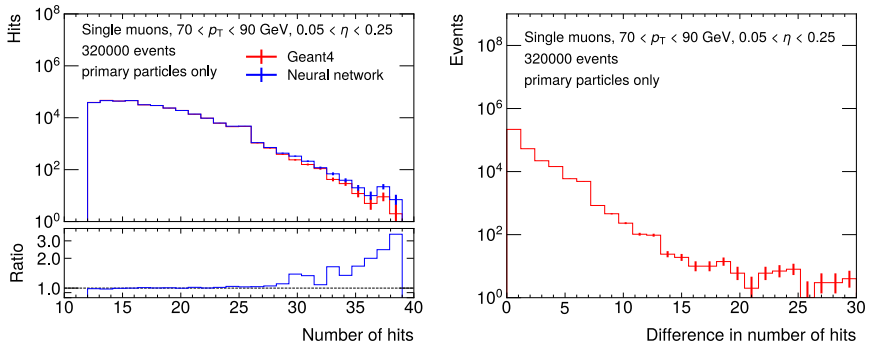


**Figure 3.** Comparison of number of hits simulated using GEANT4 and the neural network, the total number of hits (left) and the difference in the number of hits between both types of simulation (right). The ratio is shown on a logarithmic scale to better highlight deviations for high number of hits.

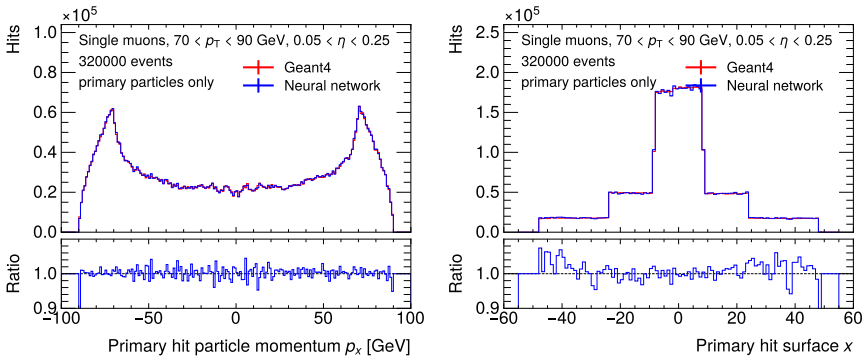


**Figure 4.** Comparisons of inclusive distributions of the primary hit particle momentum $p_x$ (left) and the primary hit surface coordinate $x$ (right) for GEANT4 and the neural network.

Individual hit features are then compared separately. As seen on Fig. 4, particle momenta at each hit are well modelled with small bin-by-bin variations. Additionally, the surface coordinates are also well modelled, including non-continuities that are present due to three detector types. Fluctuations are reaching 10 % at the edges of the distribution. These may indicate that, while the inclusive distribution agrees well between the two types of simulation, coordinates may be incorrectly assigned and then average out. Distributions of hits at individual detector layers show similar agreements and no significant deviation of the neural network prediction compared to the full simulation is observed.

Fig. 5 shows the global hit coordinates after the transformation back from the simulated surface coordinates. Hit radius is modelled exceptionally well indicating that the layer order is properly described by the model. Larger fluctuations are observed for the coordinates in Cartesian coordinate system, especially in the tails of the $z$-coordinate. As smearing in $z$ is applied during the particle generation stage, this distribution is expected to roughly follow the corresponding Gaussian distribution, taking into account that only sensitive detectors can be hit. This suggests that the training data should be prepared more uniformly, as the model tends to learn the low-statistics tails of the distributions less effectively than the high-statistics bulk. Overall the complex shape of the $x$ coordinate is still very well modelled.

The generative nature of the model was also evaluated. A selection of 800 events was simulated with different seeds, meaning different random numbers were assigned. The sur-

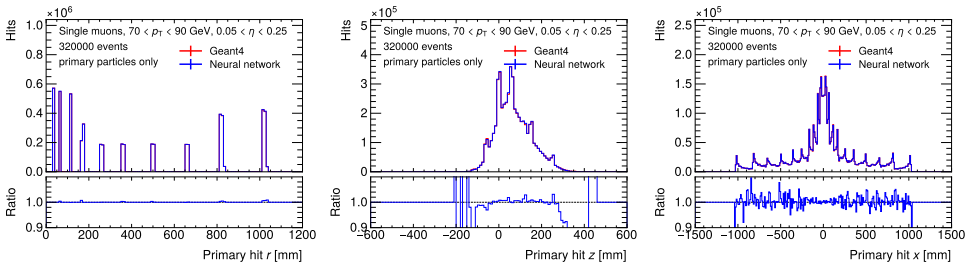


**Figure 5.** Comparisons of inclusive distributions of the primary hit radius $r$ (left), the primary hit coordinate $z$ (middle) and the primary hit coordinate $x$ (right) for GEANT4 and the neural network.

face coordinate distributions show smearing around the expected delta functions when the same seed is used each time, but the overall agreement between the full and fast simulations is preserved.

To evaluate the tracking performance on hits simulated using the transformers a default ODD tracking setup using the ACTS framework [16] was used. At the the first stage of the tracking, the seeding, where initial track candidates are identified from clusters of hits, the average efficiency is 82 % for the machine learning setup compared to 99 % for full simulation. The efficiency is defined as the fraction of generated particles matched to at least one reconstructed seed.

Seed candidates are built by taking three hits in different layers. They have to satisfy several criteria, including their compatibility with a helix trajectory associated with a particle with the estimated momentum. In the case of the neural network the hit displacement for many seed candidates is too large, going above the threshold derived from the maximum allowed multiple-scattering effect. While this is a parameter of the tracking that can be tuned, the threshold would have to be relaxed significantly to reach comparable efficiencies.

Rounding has no effect on the seeding efficiency when applied on the full simulation hits. This confirms that the chosen precision on the input data is sufficient. Large bin-by-bin fluctuations observed in Figs. 4 and 5 are considered the cause of the efficiency drop. A more elaborate loss function, designed to constrain multiple scattering and ensure proper correlations between features to avoid unphysical hits, will need to be implemented in future iterations.

## 5 Conclusions

A novel method to simulate silicon tracking detectors using deep neural networks, namely the transformer model, was developed. Good hit-level physics performance is achieved using a single-muon dataset. Using the generated data as tracking input, the efficiency is 17 % lower than the full simulation.

Several improvements can be considered in the future. Firstly, a proper generative transformer setup could be implemented ensuring full correlations between the hits in the sequence as well as between the individual features. Alternative tokenisation procedures can be implemented to increase the learning rate of the model. Finally, stricter loss functions with physics constraints could be evaluated to reduce the fluctuations in the output and speed-up the training.

## Acknowledgements

This publication is supported by the European Union's Horizon Europe research and innovation programme under the Marie Skłodowska-Curie Postdoctoral Fellowship Programme, SMASH, co-funded under the grant agreement No. 101081355. The SMASH project is co-funded by the Republic of Slovenia and the European Union from the European Regional Development Fund.

## References

[1] ATLAS Collaboration, The ATLAS Simulation Infrastructure, Eur. Phys. J. C **70**, 823 (2010), 1005.4568. 10.1140/epjc/s10052-010-1429-9

[2] J. Apostolakis et al. (HEP Software Foundation), HEP Software Foundation Community White Paper Working Group - Detector Simulation (2018), 1803.04165.

[3] S. Agostinelli et al. (GEANT4), GEANT4 - A Simulation Toolkit, Nucl. Instrum. Meth. A **506**, 250 (2003). 10.1016/S0168-9002(03)01368-8
[4] ATLAS Collaboration, AtlFast3: The Next Generation of Fast Simulation in ATLAS, Comput. Softw. Big Sci. **6**, 7 (2022), 2109.02551. 10.1007/s41781-021-00079-7
[5] ATLAS Collaboration, Deep Generative Models for Fast Photon Shower Simulation in ATLAS, Comput. Softw. Big Sci. **8**, 7 (2024), 2210.06204. 10.1007/s41781-023-00106-9
[6] ATLAS Collaboration, Fast simulation of the ATLAS calorimeter system with Generative Adversarial Networks, ATL-SOFT-PUB-2020-006 (2020), https://cds.cern.ch/record/2746032
[7] I. Zurbano Fernandez et al., High-Luminosity Large Hadron Collider (HL-LHC): Technical design report, **10/2020** (2020). 10.23731/CYRM-2020-0010
[8] P. Gessinger-Befurt, A. Salzburger, J. Niermann, The Open Data Detector Tracking System, J. Phys. Conf. Ser. **2438**, 012110 (2023). 10.1088/1742-6596/2438/1/012110
[9] ATLAS Collaboration, ATLAS Inner Tracker Pixel Detector: Technical Design Report (2017), https://cds.cern.ch/record/2285585
[10] ATLAS Collaboration, ATLAS Inner Tracker Strip Detector: Technical Design Report (2017), https://cds.cern.ch/record/2257755
[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention Is All You Need, in *31st International Conference on Neural Information Processing Systems* (2017), 1706.03762
[12] A. Paszke et al., PyTorch: An Imperative Style, High-Performance Deep Learning Library (2019), 1912.01703.
[13] D. Hendrycks, K. Gimpel, Gaussian Error Linear Units (GELUs) (2016), 1606.08415.
[14] I. Loshchilov, F. Hutter, Decoupled Weight Decay Regularization, in *The Seventh International Conference on Learning Representations* (2017), 1711.05101
[15] IZUM, Vega, https://izum.si/en/vega-en/, [Online; accessed 22. Feb 2025]
[16] X. Ai et al., A Common Tracking Software Project, Comput. Softw. Big Sci. **6**, 8 (2022), 2106.13593. 10.1007/s41781-021-00078-8