Transformers for Peak Detection in Drift-Tube Waveforms

Asif Ali

Supervisors: Prof. Nicola De Filippis & Prof. Guang Zhao

Politecnico & INFN Bari & IHEP, Beijing, China

November 28, 2025

What is a Transformer?

Transformer

A self-attention based model for sequence learning.

Self-Attention

Each part of the sequence can look at all other parts and focus on what matters most.

What this means:

- ✓ Every time bin of the waveform can "see" the whole waveform.
- ✓ Learns to focus on relevant bins (near a peak) and ignore noise.
- ✓ Captures long-range dependencies across the waveform.
- ✓ Helps distinguish true signal peaks from background noise.

RNN/LSTM vs Transformer

- Accuracy on distant peaks: Transformer $\gg$ LSTM.
- Training speed: Transformer is much faster.
- Stability: Residual + normalization improve convergence.
- Scalability: Handles larger waveforms easily.

Aspect	**LSTM**	**Transform**
Parallelism	Low	High
Long-range capture	Weak	Strong
Training speed	Slow	Fast
Per-bin output	Limited	Excellent

Why Encoder-Only Transformer?

Our Choice

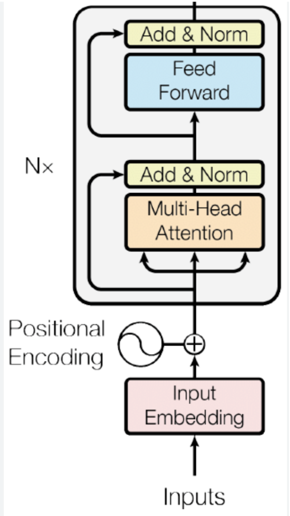
We use only the Encoder part of the Transformer.

Why not Decoder?

Decoder is designed for sequence generation (e.g., text translation).

Reason:

- ✓ Our task is **peak finding**, not sequence generation.
- ✓ Encoder is sufficient for per-bin classification (peak / no peak).
- ✓ Simpler, faster, and fewer parameters.
- ✓ Better suited for waveform analysis.



Source:Vaswani et al., *Attention Is All You Need* (2017)

- **Input Embedding:** Converts input waveform amplitudes (or tokens) into dense feature vectors.
- **Positional Encoding:** Adds information about time/bin order since attention alone is position-invariant.
- **Multi-Head Attention:** Each head learns to focus on different relationships across the waveform.
- **Add & Norm (1):** Residual connection + Layer Normalization ensures gradient stability.
- **Feed Forward:** Two fully connected layers applied to each time step independently.
- **Add & Norm (2):** Another residual + normalization block improving training stability.

Input Representation

Pipeline: **Embedding+PE** → Q/K/V → Scaled Attn (scores → weights→sum) → MHA → FFN → Residual+LayerNorm → Stack N

For bin i with amplitude x_i:

$$z_i = \underbrace{Amplitude\_Embedding} + \underbrace{PE(i)}$$

- Embedding: Projects scalar amplitude to d_{model}.
- Positional Encoding: Injects bin order / timing.

Embedding Formula

Pipeline: **Embedding+PE** $\rightarrow$ Q/K/V $\rightarrow$ Scaled Attn (scores $\rightarrow$ weights$\rightarrow$sum) $\rightarrow$ MHA $\rightarrow$ FFN $\rightarrow$ Residual+LayerNorm $\rightarrow$ Stack N

Each amplitude $x_i \rightarrow d_{\text{model}}$ vector:

$$\mathrm{Embed}(x_i) = W_e x_i + b_e$$

Interpretation

- Linear projection layer: converts amplitudes into learnable features.
- Combined with PE for sequential context.

Positional Encoding Formula

Pipeline: **Embedding+PE** → Q/K/V → Scaled Attn (scores → weights→sum) → MHA → FFN → Residual+LayerNorm → Stack N

For position pos and channel j:

$$\mathrm{PE}(pos, 2j) = \sin\left(\frac{pos}{10000^{2j/d_{\text{model}}}}\right), \quad \mathrm{PE}(pos, 2j+1) = \cos\left(\frac{pos}{10000^{2j/d_{\text{model}}}}\right)$$

Key Point

Encodes both absolute and relative position information.

Q, K, V: Learnable Projections

Pipeline: Embedding+PE $\rightarrow$ **Q/K/V** $\rightarrow$ Scaled Attn (scores$\rightarrow$weights$\rightarrow$sum) $\rightarrow$ MHA $\rightarrow$ FFN $\rightarrow$ Residual+LayerNorm $\rightarrow$ Stack N

$$Q = ZW^Q, \quad K = ZW^K, \quad V = ZW^V$$

- **Query:** what I seek; **Key:** what I can answer; **Value:** content to pass on.

Scaled Dot-Product Attention & Why $\sqrt{d_k}$

Pipeline: Embedding+PE $\rightarrow$ Q/K/V $\rightarrow$ **Scaled Attn (scores$\rightarrow$weights$\rightarrow$sum)** $\rightarrow$ MHA $\rightarrow$ FFN $\rightarrow$ Residual+LayerNorm $\rightarrow$ Stack N

$$S = QK^\top \qquad \alpha = \mathrm{softmax}\left(\frac{S}{\sqrt{d_k}}\right), \qquad \mathrm{Attn}(Q, K, V) = \alpha V$$

Why scaling?

Large d_k $\Rightarrow$ large dot products $\Rightarrow$ softmax too sharp $\Rightarrow$ unstable gradients.
Dividing by $\sqrt{d_k}$ stabilizes logits and gradients.

Attention Mechanism

Pipeline: Embedding+PE $\rightarrow$ Q/K/V $\rightarrow$ **Scaled Attn (scores$\rightarrow$weights$\rightarrow$sum)** $\rightarrow$ MHA $\rightarrow$ FFN $\rightarrow$ Residual+LayerNorm $\rightarrow$ Stack N

$$\mathrm{Attention}(Q, K, V) = \mathsf{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

Intuition: each bin compares itself with all others and gathers useful context.

Multi-Head Attention

Pipeline: Embedding+PE → Q/K/V → Scaled Attn → **MHA** → FFN → Residual+LayerNorm → Stack N

Mathematical Definition

$$\mathrm{head}_i = \mathrm{Attn}(QW_i^Q,\ KW_i^K,\ VW_i^V)$$
$$\mathrm{MHA}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)\, W^O$$

Typical dimensions:

$$d_{\text{model}} = 64, \quad h = 4, \quad d_k = d_v = 16$$

Parallelism:

- Each head works independently on its own subspace.
- All heads are computed **in parallel** → high GPU efficiency.

Specialization of Heads:

- Head 1 → local peaks (short-range features)
- Head 2 → global baseline trend
- Head 3 → long tails / after-pulses
- Head 4 → noise suppression / refinement

Interpretation: Concatenation of all heads combines diverse waveform features into one rich representation before final projection.

Position-wise Feed-Forward Network (FFN)

Pipeline: Embedding+PE → Q/K/V → Scaled Attn → MHA → **FFN** → Residual+LayerNorm → Stack N

Applied independently to each timestep (shared weights):

$$\mathrm{FFN}(x) = \sigma(xW_1 + b_1)\, W_2 + b_2, \quad \sigma = \mathsf{GeLU}$$

- Adds nonlinearity and capacity to transform attention-mixed features.

Residual Connections + Layer Normalization

Pipeline: Embedding+PE $\to$ Q/K/V $\to$ Scaled Attn $\to$ MHA $\to$ FFN $\to$ **Residual+LayerNorm** $\to$ Stack N

$$\text{Output} = \mathrm{LayerNorm}\big(x + \text{Sublayer}(x)\big)$$

- **Residual:** preserve original info, improve gradient flow, enable deeper stacks.
- **LayerNorm:** per-position normalization with learnable $\gamma, \beta \Rightarrow$ stable training.

Results

Input Data to the Transformer

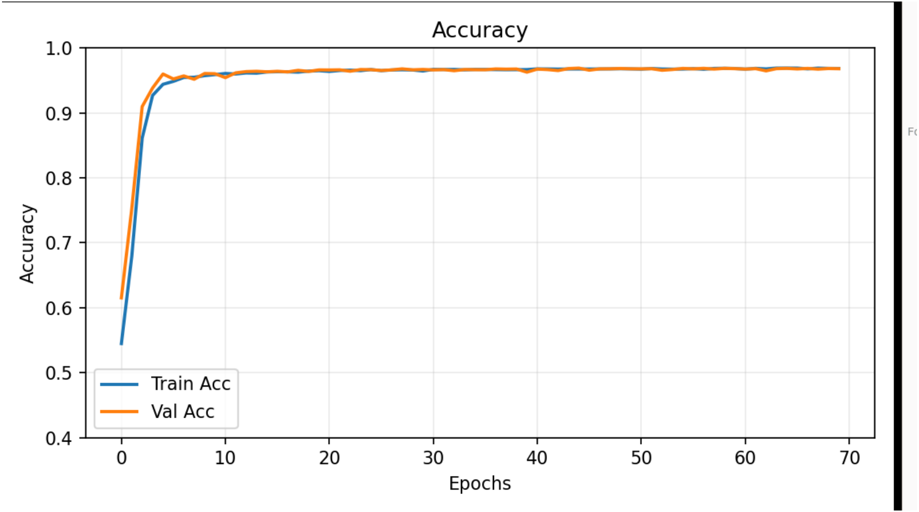
EvtNo	ID	Shift	Sigma	Time	Time0	Time1	Time2	Time3	Time4	Time5	Time6	Time7	Time8	Time9	Time10	Time11	Time12	Time13	Time14
0	0	-0.0136	1	5	-0.03623	-0.03183	-0.00252	0.03567	0.0156	0.03481	-0.03922	-1.171e-05	-0.007266	0.03652	0.03184	-0.01925	0.006953	0.009978	-0.03505
0	0	-0.008607	1	7	-0.007513	0.03068	0.0106	0.02982	-0.04421	-0.005005	-0.01226	0.03153	0.02685	-0.02424	0.00196	0.004986	-0.04004	-0.002116	-0.001036
0	0	-0.00538	1	9	0.007377	0.02659	-0.04744	-0.008231	-0.01549	0.0283	0.02362	-0.02747	-0.001266	0.001759	-0.04327	-0.005342	-0.004262	0.02919	0.03592
0	0	-0.007173	1	10	0.02838	-0.04564	-0.006439	-0.01369	0.0301	0.02542	-0.02568	0.000526	0.003551	-0.04148	-0.00355	-0.00247	0.03098	0.03771	-0.01771
0	0	-0.009874	1	12	-0.003737	-0.01099	0.0328	0.02812	-0.02297	0.003228	0.006253	-0.03878	-0.0008481	0.0002319	0.03368	0.04041	-0.01501	0.01624	-0.06863
0	0	-0.01241	1	13	-0.008454	0.03534	0.03066	-0.02044	0.005765	0.00879	-0.03624	0.001689	0.002769	0.03622	0.04295	-0.01247	0.01878	-0.06609	-0.03926
0	0	-0.01051	1	15	0.02875	-0.02234	0.003862	0.006887	-0.03814	-0.0002137	0.0008663	0.03432	0.04105	-0.01438	0.01688	-0.06799	-0.04116	0.03147	0.02015
0	0	-0.01289	1	17	0.006241	0.009266	-0.03576	0.002165	0.003245	0.0367	0.04343	-0.012	0.01925	-0.06562	-0.03878	0.03385	0.02253	-0.002177	-0.02233
0	0	-0.01334	1	18	0.009715	-0.03531	0.002614	0.003694	0.03714	0.04388	-0.01155	0.0197	-0.06517	-0.03834	0.0343	0.02298	-0.001728	-0.02188	-4.504e-05
0	0	-0.01098	1	20	0.0002558	0.001336	0.03479	0.04152	-0.01391	0.01734	-0.06752	-0.04069	0.03194	0.02062	-0.004086	-0.02424	-0.002403	-0.0002204	0.005272
0	0	-0.01097	1	23	0.04151	-0.01391	0.01734	-0.06753	-0.0407	0.03193	0.02062	-0.004091	-0.02424	-0.002408	-0.0002257	0.005266	0.02393	0.03228	-0.01976
0	0	-0.01423	1	24	-0.01066	0.02059	-0.06428	-0.03745	0.03519	0.02387	-0.0008378	-0.02099	0.0008449	0.003027	0.008519	0.02718	0.03553	-0.01651	-0.00403
0	0	-0.01038	1	27	-0.0413	0.03134	0.02002	-0.004688	-0.02484	-0.003005	-0.0008231	0.004669	0.02333	0.03168	-0.02036	-0.00788	-0.01144	0.02617	-0.02287
0	0	-0.01258	1	30	-0.002488	-0.02264	-0.000805	0.001377	0.006869	0.02553	0.03388	-0.01816	-0.00568	-0.009241	0.02837	-0.02067	0.01316	-0.00301	-0.02649
0	0	-0.01255	1	31	-0.02267	-0.0008324	0.00135	0.006842	0.0255	0.03385	-0.01819	-0.005707	-0.009268	0.02835	-0.0207	0.01313	-0.003038	-0.02652	-0.002105
0	0	-0.006941	1	33	-0.004257	0.001234	0.0199	0.02824	-0.02379	-0.01131	-0.01488	0.02274	-0.02631	0.007523	-0.008645	-0.03213	-0.007713	0.008407	0.04099
0	0	-0.005692	1	35	0.01865	0.027	-0.02504	-0.01256	-0.01613	0.02149	-0.02756	0.006273	-0.009895	-0.03338	-0.008962	0.007158	0.03974	-0.008357	0.02157
0	0	-0.01019	1	37	-0.02054	-0.008062	-0.01162	0.02599	-0.02305	0.01078	-0.005392	-0.02888	-0.00446	0.01166	0.04424	-0.003855	0.02608	-0.01161	-0.001273
0	0	-0.01223	1	38	-0.006029	-0.009591	0.02802	-0.02102	0.01281	-0.00336	-0.02684	-0.002427	0.01369	0.04628	-0.001823	0.02811	-0.009581	0.0007594	-0.04899
0	0	0.01724	1	40	-0.00144	-0.05048	-0.01666	-0.03282	-0.05631	-0.03189	-0.01577	0.01681	-0.03129	-0.001356	-0.03905	-0.0287	-0.07846	-0.03374	0.4012
0	0	0.06891	1	42	-0.06833	-0.0845	-0.108	-0.08357	-0.06745	-0.03486	-0.08296	-0.05303	-0.09072	-0.08038	-0.1301	-0.08541	0.3495	0.3189	0.301
0	0	0.113	1	44	-0.1521	-0.1276	-0.1115	-0.07894	-0.127	-0.09711	-0.1348	-0.1245	-0.1742	-0.1295	0.3054	0.2748	0.2569	0.2163	0.2039
0	0	0.2349	1	46	-0.2334	-0.2008	-0.2489	-0.219	-0.2567	-0.2463	-0.2961	-0.2514	0.1835	0.1529	0.135	0.09439	0.08203	0.6141	0.6905
0	1	0.3769	1	49	-0.361	-0.3987	-0.3884	-0.4381	-0.3934	0.04147	0.01085	-0.007046	-0.04769	-0.06004	0.472	0.5484	0.4536	0.3698	0.1984
0	0	0.4437	1	51	-0.4552	-0.5049	-0.4602	-0.0253	-0.05592	-0.07382	-0.1145	-0.1268	0.4052	0.4817	0.3868	0.303	0.1316	0.09003	0.01822
0	1	0.5528	1	54	-0.1344	-0.165	-0.1829	-0.2236	-0.2359	0.2961	0.3726	0.2777	0.1939	0.02255	-0.01907	-0.09088	0.03585	-0.04684	-0.1001
0	0	0.5555	1	55	-0.1678	-0.1857	-0.2263	-0.2387	0.2934	0.3698	0.275	0.1911	0.01979	-0.02182	-0.09363	0.03309	-0.04959	-0.1029	-0.09588

Waveform inputs

Input Structure

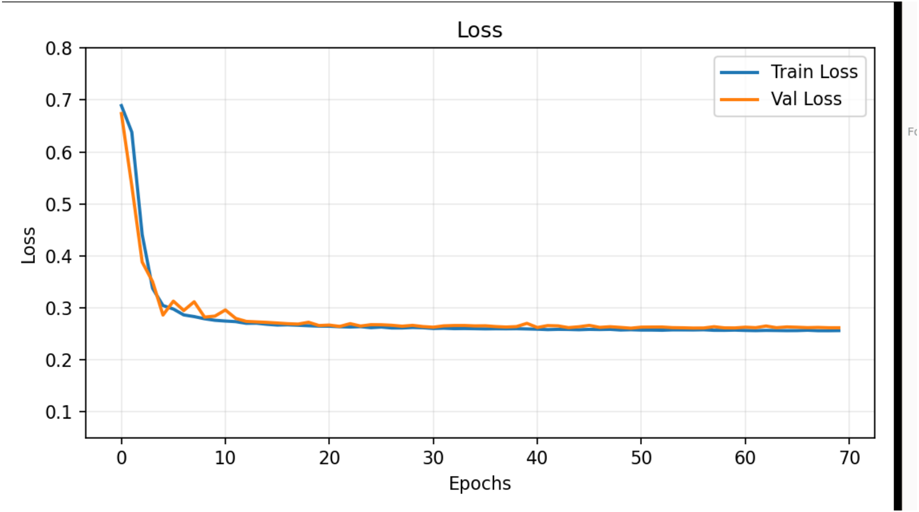
- Each row corresponds to one event (waveform).
- First column: **Target**.
- Remaining columns: **Waveform samples** (Time0, Time1, Time2, ...).
- Data is normalized and reshaped to $(\text{samples}, \text{time}, 1)$.
- The sequence is fed into the Transformer for Classification.

Training and Validation Accuracy(FCC)



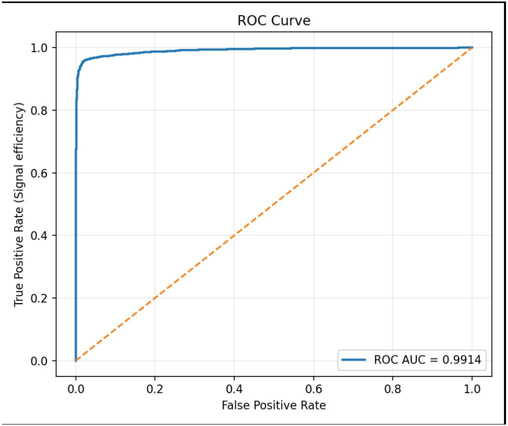
Observation: Accuracy quickly rises above 0.95 within 10 epochs and remains stable — shows excellent generalization and almost no overfitting.

Training and Validation Loss(FCC)



Observation: Loss drops sharply in the first few epochs and plateaus near 0.26. Training and validation curves overlap closely → stable learning with good

ROC Curve (FCC)



ROC AUC = 0.9914

- **Interpretation:**
- The Transformer achieves near-perfect signal/background separation.
- Very high **True Positive Rate** with extremely low **False Positive Rate**.
- Model demonstrates excellent classification capability and generalization.
- Indicates robust detection of signal peaks even in noisy environments.

Transformer Hyperparameter Configuration

Training Dataset

- Train / Test split: 60% / 40%
- Random state: 42

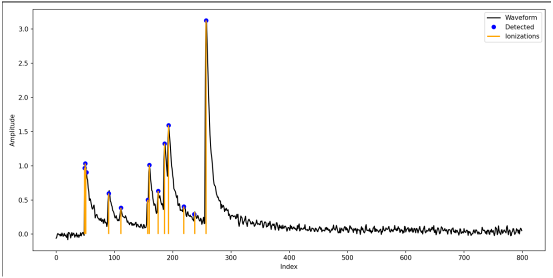
Model Architecture

- Attention heads: 4
- Key dimension: 16
- Number of Transformer blocks: 2
- Hidden neurons (dense): 96
- Activations: GELU (hidden), Sigmoid (output)
- Dropout: Attention = 0.10, FFN = 0.15

Training Setup

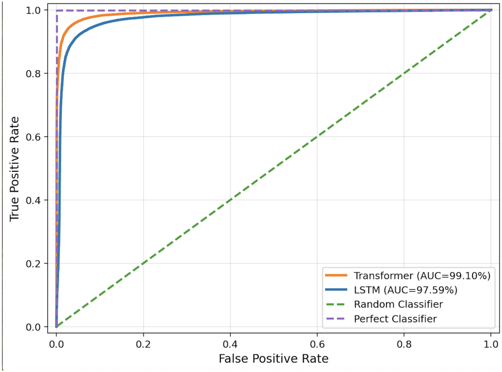
- Optimizer: AdamW
- Learning rate: 3×10^{-5}
- Weight decay: 0.0005
- Batch size: 256
- Epochs: 80, Patience: 12
- AUC minimum keep threshold: 0.95
- Fixed decision threshold: 0.5

Waveform Evaluation with Transformer Model(FCC)



- Black curve: Drift chamber waveform.
- Orange lines: MC Truth .
- Dots: Detected peaks(Model).

ROC Curve Comparison (Transformer vs LSTM)

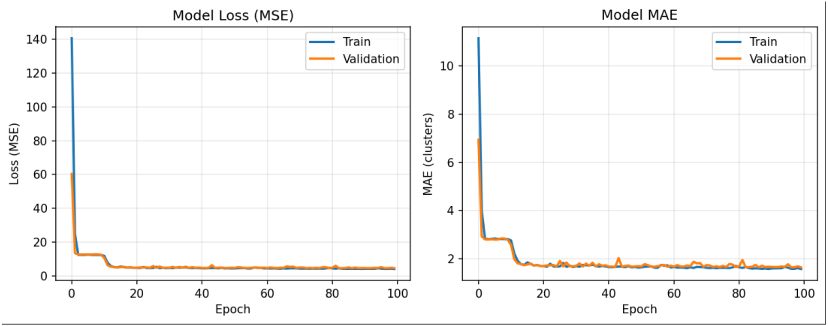


- The ROC curves illustrate classification performance for both models.
- Transformer achieves an AUC of **99.10%**, outperforming LSTM's **97.59%**.
- True Positive Rate remains high even at very low False Positive Rates.
- Dashed green line shows a random classifier; purple line marks an ideal classifier.
- Transformer demonstrates excellent discriminative capability.

Transformer-based Regression Model

- The goal was to train a Transformer-based regression model to predict the number of primary ionization clusters from detected peaks.
- Various parameters were explored such as number of heads, key dimension, number of blocks, dropout, and learning rate.
- Early stopping, patience, and batch size were also optimized to achieve stable convergence.

Transformer Regression – Training Performance



- Fast convergence in the first ~ 10 epochs.
- Training and validation curves overlap $\Rightarrow$ no overfitting.
- Stable MSE and MAE values indicate strong generalisation.

Regression Metrics

$$\text{MAE} = \frac{1}{N}\sum |y_i - \hat{y}_i|, \qquad \text{MSE} = \frac{1}{N}\sum (y_i - \hat{y}_i)^2$$

Notation: N = number of samples, y_i = true value, $\hat{y}_i$ = predicted value

Transformer Regression – Final Results

Final metrics (validation set)

- MAE: 1.64 clusters
- MSE: 4.61
- Training time: 985 s

Comparison to baseline model

- Baseline MAE: 2.80 clusters
- Baseline MSE: 12.71
- MAE improvement: $\approx 41.3\%$
- MSE improvement: $\approx 63.7\%$

Improvement Formula:

$$\text{Improvement} = \frac{\text{Baseline} - \text{Model}}{\text{Baseline}} \times 100\%$$

Baseline definition: Constant-mean predictor ($\hat{y} = \text{mean}(y_{\text{train}})$), i.e. it always predicts the average cluster count of the training set.

Takeaway: The Transformer regression model reduces the cluster-count prediction error by $\mathcal{O}(41\%)$ with respect to the baseline.

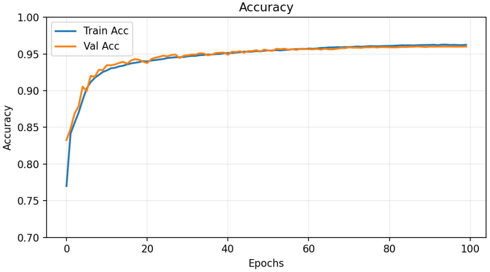
Transformer Regression – Hyperparameter Configuration

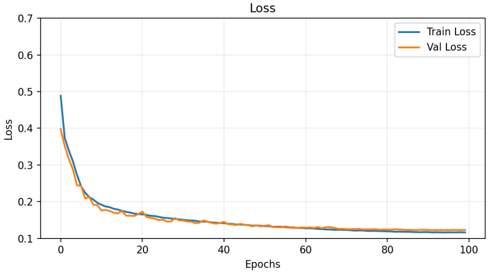
- Random state: 42
- Test size: 0.4
- Attention heads: 2
- Key dimension: 16
- Transformer blocks: 1
- FFN multiplier: 2
- Hidden neurons (Dense): 128
- Optimizer: Adam
- Learning rate: 10^{-3}
- Batch size: 128
- Epochs: 100
- Patience: 20
- Minimum learning rate: 10^{-6}

Thank You!

Backup

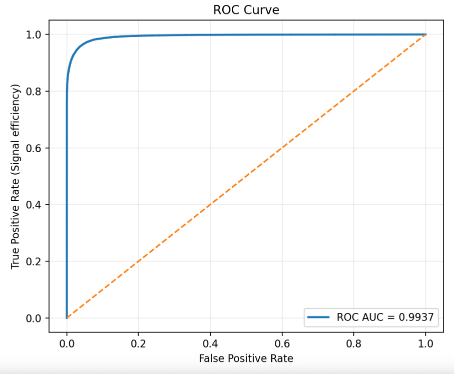
Training and Validation Performance(CEPC)





- **Accuracy:** Training and validation curves converge around 96%.
- **Loss:** Both losses decrease smoothly to ∼0.12.
- **Overall:** Stable performance after ∼40 epochs.

ROC Curve Performance(CEPC)



- ROC curve shows excellent separation between classes.
- AUC score 0.9937 $\Rightarrow$ near perfect classifier.
- True Positive Rate is close to 1 even at very low False Positive Rate.
- Model demonstrates outstanding discriminative power.

Why Transformer Uses Sin and Cos in Positional Encoding

Goal: Transformers process all tokens in parallel — they have no built-in notion of order. To help the model understand *sequence position and distance*, the paper introduced a fixed **positional encoding** based on sine and cosine functions.

Mathematical form:

$$\mathrm{PE}(\mathrm{pos},\ 2\mathrm{j}) = \sin\left(\frac{\mathrm{pos}}{10000^{2j/d_{\mathrm{model}}}}\right), \quad \mathrm{PE}(\mathrm{pos},\ 2\mathrm{j}{+}1) = \cos\left(\frac{\mathrm{pos}}{10000^{2j/d_{\mathrm{model}}}}\right)$$

Why specifically Sin and Cos?

- They are **periodic and continuous**, creating smooth, repeating patterns over positions.
- They are **bounded between [-1, +1]** → training remains numerically stable.
- Their **phase difference (90°)** gives orthogonal components → unique position fingerprints.
- Relative distances between tokens are preserved:
- No trainable parameters are needed — purely deterministic and differentiable.

Why Not Other Trigonometric or Nonlinear Functions?

Alternative ideas and why they fail:

Function	**Why not suitable for Positional Encoding**
$\tan(x)$	Unbounded; goes to $\pm\infty$ near $\frac{\pi}{2}$, causes exploding gradients.
$\cot(x)$	Undefined at $x = 0$, discontinuous; model becomes numerically unstable.
$\sec(x)$, $\csc(x)$	Very large near 90°, periodic but unbounded — unstable.
$\sinh(x)$, $\cosh(x)$	Smooth but not periodic — cannot represent repeating distances.
$\arctan(x)$	Bounded but non-periodic; loses relative position pattern.
$\exp(x)$, $\log(x)$	Non-periodic and asymmetric; no relative phase information.

Conclusion: Only sin and cos together satisfy:

- Smooth + periodic + bounded $\rightarrow$ stable training
- Orthogonal components $\rightarrow$ unique positional signatures
- Relative distance preserved $\rightarrow$ model learns order naturally

Hence, the original Transformer paper used Sin and Cos exclusively.

Accuracy in Binary Classification

Purpose: Accuracy measures how many predictions are correct out of all predictions. It is one of the simplest and most widely used evaluation metrics.

Formula:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- TP — True Positives
- TN — True Negatives
- FP — False Positives
- FN — False Negatives

Binary Cross-Entropy (BCE) Loss

Purpose: Used for **binary classification** (2 classes: 0 or 1). Measures the difference between predicted probability $\hat{y}$ and actual label y.

Mathematical Formula:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

Where:

- N — number of samples
- y_i — true label (0 or 1)
- $\hat{y}_i$ — model prediction (probability between 0 and 1)

Intuition:

- Penalizes incorrect high-confidence predictions strongly.
- Smaller loss = closer prediction to true label.
- Logarithm makes the loss grow sharply for wrong predictions.

Understanding BCE: Intuition and Examples

Case A: When $y = 1$

$$L = -\log(\hat{y})$$

$\hat{y} = 0.9 \Rightarrow L = 0.105$ (small loss) $\hat{y} = 0.1 \Rightarrow L = 2.30$ (large loss)

Case B: When $y = 0$

$$L = -\log(1 - \hat{y})$$

$\hat{y} = 0.1 \Rightarrow L = 0.105$ (small loss) $\hat{y} = 0.9 \Rightarrow L = 2.30$ (large loss)

Connection to Sigmoid:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \qquad z = w^T x + b$$

Sigmoid output lies in [0, 1] $\rightarrow$ suitable for BCE.

Summary:

- Correct predictions $\rightarrow$ low loss.
- Wrong confident predictions $\rightarrow$ high loss.
- BCE + Sigmoid = standard choice for binary tasks.

Categorical Cross-Entropy (Multi-Class Classification)

Purpose: Used for **multi-class classification** where more than two categories exist. In our case: Primary Ionization, Secondary Ionization, and Background.

Formula:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

Where:

- N — number of samples
- C — number of classes (here $C = 3$)
- $y_{i,c}$ — 1 if sample i belongs to class c, else 0 (one-hot encoded)
- $\hat{y}_{i,c}$ — model-predicted probability for class c

Softmax activation:

$$\hat{y}_{i,c} = \frac{e^{z_{i,c}}}{\sum_{k=1}^{C} e^{z_{i,k}}}$$

Softmax ensures that all class probabilities sum to 1.

Softmax Activation Function

Purpose: Softmax is used in the **output layer** of a **multi-class classification** model. It converts raw network outputs (*logits*) into normalized probabilities that sum to 1 across all classes.

Formula:

$$\hat{y}_{i,c} = \frac{e^{z_{i,c}}}{\sum_{k=1}^{C} e^{z_{i,k}}}$$

Where:

- $z_{i,c}$ — raw output (logit) for class c of sample i
- C — total number of classes
- $\hat{y}_{i,c}$ — probability of sample i belonging to class c

Key properties:

- All outputs are in range $[0, 1]$.
- Sum of all class probabilities $= 1$.
- Highest value corresponds to the predicted class.

Example:

$$z = [2.1, 1.0, 0.1] \Rightarrow \hat{y} = \text{Softmax}(z) = [0.68, 0.25, 0.07]$$

The model predicts Class 1 with highest confidence (68%).