Energy Reconstruction for HGCAL using ML Techniques

ML School 2026 — Complete study

Electron & pion energy regression: BDT, DNN, GCN, Transformers

CMS High-Granularity Calorimeter test-beam prototype

Santanu Mahata[†], Shubham Kumar, Sanskar Nanda, Poonam Naik, Manaswini Priyadarshini

[†]speaker

Outline

- Introduction to HGCAL & the problem
- Data pipeline (hits $\rightarrow$ per-layer table)
- Electron reconstruction
  - profile, correlation, features
  - BDT / DNN / GCN
  - resolution & resolution fit
    $\sigma_E = \sqrt{a^2 E + b^2 + c^2 E^2}$
- Pion reconstruction
  - hadronic physics, model spread
- V2 — improved models
  - advanced features
  - resolution-aware (log-E) training
  - Transformers
  - final resolution comparison, all models
- Evaluation methodology

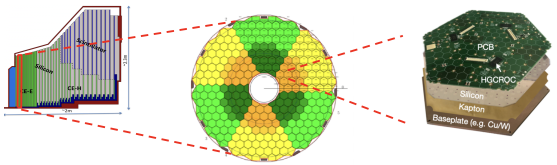
Introduction to HGCAL

- HGCAL = High-Granularity Calorimeter: a Silicon + plastic-scintillator *sampling* calorimeter for the CMS endcap at the HL-LHC.
- Motivation: HL-LHC delivers $\sim$10$\times$ more luminosity $\Rightarrow$ extreme radiation and pileup; the current endcap cannot survive.
- Sampling design: alternating absorber (Pb/Cu/steel) + active Si/scintillator layers.
- 4D imaging: every hit has (E, x, y, z) $\Rightarrow$ hundreds of hits per particle, resolving the *shower shape*.

Geometry

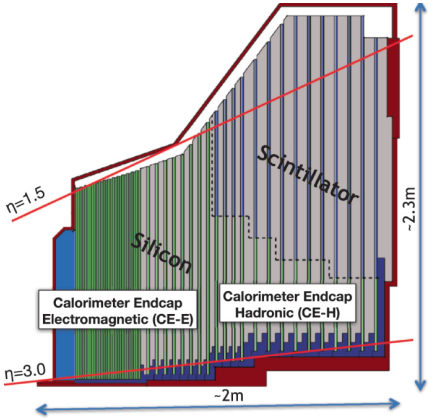
- CE-E (electromagnetic): 28 Si layers
- CE-H (hadronic): $\sim$24 layers Si + scintillator
- + AHCAL behind $\Rightarrow$ 78 layers total for pions

A particle is destroyed into a *shower*, the calorimeter samples the deposited energy. Task: recover the incident energy from the hit pattern.



Endcap cross-section $\rightarrow$ hexagonal-module wheel $\rightarrow$ single module (PCB+HGCROC+Si+Kapton+baseplate).

HGCAL structure — CE-E and CE-H endcap cross-section



Endcap cross-section: CE-E (electromagnetic, Silicon, green) followed by CE-H (hadronic, Silicon + Scintillator, grey), spanning $\eta = 1.5$–3.0, $\sim$2 m across, $\sim$2.3 m deep.

The regression problem — and why physics makes it easy or hard

Task (supervised regression): map one event's set of hits $\{(x, y, z, E)\}$ to the scalar E_{true} (beam energy, known in a test beam). **Headline metric:** $R^2 = 1 - \dfrac{\sum_i (E_{\text{true},i} - \hat{E}_i)^2}{\sum_i (E_{\text{true},i} - \bar{E})^2}$ (fraction of target variance explained by the model; R^2=1 is a perfect predictor, R^2=0 matches always predicting the mean).

Electrons (EM shower)

- bremsstrahlung + pair production
- compact, reproducible, nearly linear
- summed signal ≈ incident energy
- E_{tot} correlates 0.9995 with energy
- 28 layers, 648,277 events

Pions (hadronic shower)

- nuclear interactions, huge fluctuations
- large invisible energy (nuclear binding)
- summed signal is a biased, weak estimator
- E_{tot} corr only 0.69; hit count corr 0.87
- 78 layers, 836,658 events

This EM-vs-hadronic contrast drives every result: electrons saturate; pions have real ML headroom.

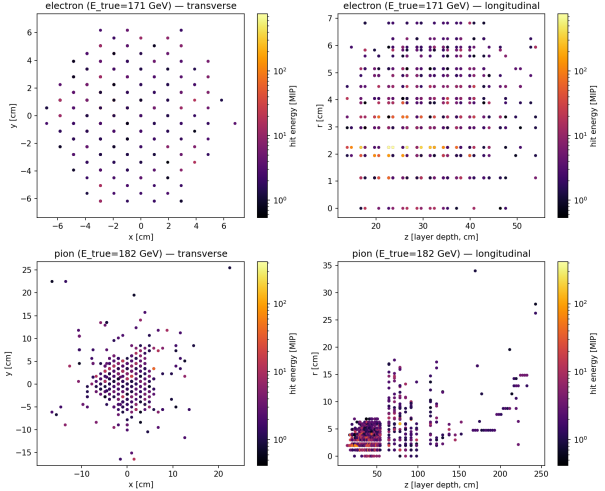
Data pipeline: from raw hits to a per-layer table

Raw HDF5: flat arrays `rechit_energy/x/y/z`, indexed by cumulative `nhits`; label `target` (e) / `true_energy` (π). **Flatten** each event to fixed length: per layer ℓ, the energy sum + energy-weighted mean/std of x, y, r + hit count. Energy-weighted moments:

$$\bar{v}_\ell = \frac{\sum_{h\in\ell} E_h v_h}{\sum_{h\in\ell} E_h}, \qquad \sigma^2_{v,\ell} = \frac{\sum_{h\in\ell} E_h (v_h - \bar{v}_\ell)^2}{\sum_{h\in\ell} E_h}$$

computed for all (event,layer) at once with `np.bincount` ($\sim$100$\times$ faster than a Python loop; validated to 8×10^{-8}). Result: $8\,n_{\text{layers}} + 2$ columns (226 for e, 626 for π); auto-detects 28 vs 78 layers.

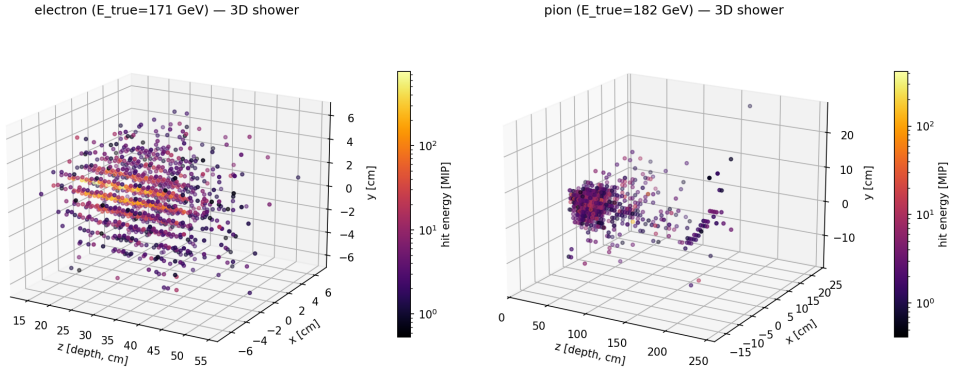
Event display — one shower of each



median-energy electron (171 GeV) and pion (182 GeV) event; transverse (x-y) and longitudinal (z vs r) hit maps, colour = hit energy (log scale).

Electron stays within $r \lesssim 6\,\text{cm}$, $z \lesssim 55\,\text{cm}$ (compact EM shower); pion spreads to $r \sim 25\,\text{cm}$, z up to 250 cm — the deep, diffuse hadronic shower seen on the pion profile slide.

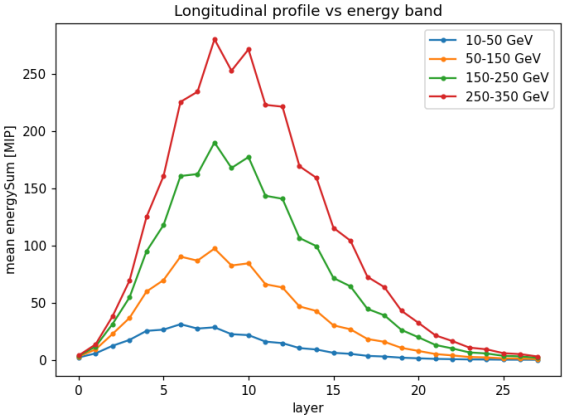
Event display — the same two showers in 3D



same two events as the previous slide, (x, y, z) **hit cloud**, **shared camera angle**. The electron shower is a short, dense stub; the pion shower is a sparse, wandering trail that reaches nearly 5× deeper — the 2D projections on the previous slide show this same contrast from two angles, this is the whole thing at once.

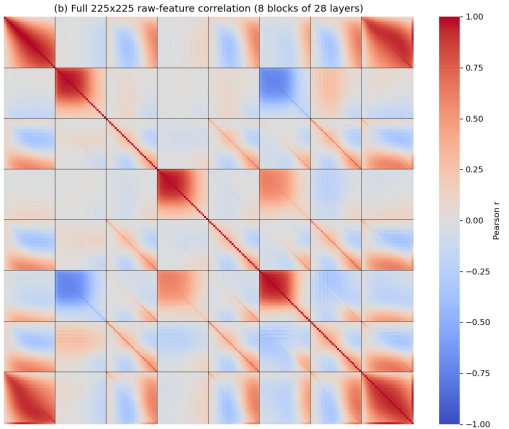
Electron energy reconstruction

Longitudinal shower profile (electron)



- Mean energy per layer, by energy band.
- Rises → peaks (shower max, layer ∼8) → falls — textbook EM development.
- Amplitude scales with energy; shower max drifts slightly deeper (log scaling).
- This curve *is* the physics we summarise with the engineered features.

Feature correlation — why we engineer features



- Full 225×225 correlation (8 families $\times$ 28 layers).
- Bright near-diagonal blocks = adjacent layers are almost identical $\Rightarrow$ massive redundancy.
- Effective degrees of freedom $\ll 225$.
- Feature importance (MI): $\sim$17 raw features carry almost no energy information.

$\Rightarrow$ Compress to a few physics moments.

Feature engineering — 11 physics moments

Longitudinal (from E_ℓ, layer index ℓ):

$$\text{depth} = \frac{1}{E_{\text{tot}}} \sum_\ell E_\ell \ell \qquad \text{(centre of gravity)}$$

$$\text{length} = \sqrt{\frac{1}{E_{\text{tot}}} \sum_\ell E_\ell (\ell - \text{depth})^2} \qquad \text{(spread)}$$

$\text{frac}_{f,m,b}$ = energy in front/mid/back third

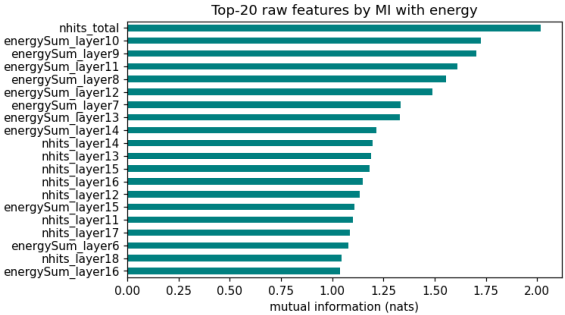
Lateral: energy-weighted mean r (radius) and r-width. Plus E_{tot}, peak-layer energy/index, n_{hits}.

Result (electron):

- 11 engineered features BEAT all 225 raw: R^2 0.99912 vs 0.99768; MAE 2.14 vs 3.54 GeV.
- Less noise $\Rightarrow$ less overfitting.
- E_{tot} alone corr 0.9995 $\Rightarrow$ near-linear EM calorimetry.

V2 adds skewness/kurtosis, shower start/extent, EE/FH/BH section energies, log-transforms (24 features).

Feature importance (electron)



- Ranked by mutual information: non-linear dependence of each feature with the energy (k-NN estimator, nats).
- n_{hits} and mid-shower layer energies dominate — exactly where the profile peaks.
- Complements correlation: this ranks *information*, correlation flags *redundancy*.

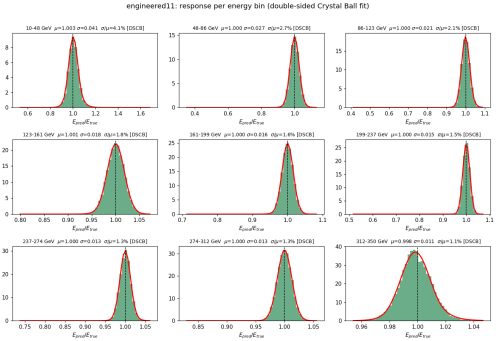
BDT — Boosted Decision Trees

Mechanics. Additive ensemble of shallow trees, $F_M(x) = \sum_m \nu\, h_m(x)$; each new tree h_m fits the negative gradient (residuals) of the loss so far. ν = learning rate.

- sklearn `GradientBoostingRegressor` (the repo model).
- Knobs: # trees, ν, depth/leaves, subsampling.

Result.

- Electron test $R^2 = 0.9991$ (tuned).
- Pion: sklearn GBR $R^2 = 0.954$.



per-bin E_{pred}/E_{true} with double-sided Crystal Ball fits (electron BDT).

DNN, GCN and GNN

DNN — MLP on the 11 features: $h^{(k)} = \phi(W^{(k)} h^{(k-1)} + b^{(k)})$, ReLU/GELU, linear energy output. V2: BatchNorm, dropout, cosine LR, trained on $\log E$.

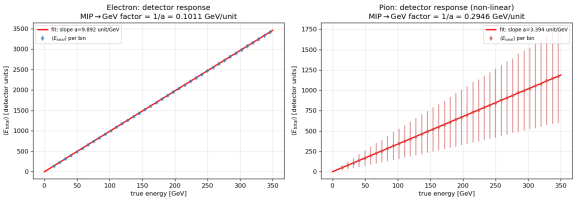
GCN / GNN — each event = a graph, one node per layer (features [energySum,rMean,rStd,nhits]); edges between *successive* layers (GCN) or *all* layers (GNN).

Message passing (`GCNConv`):

$$h_i^{(k+1)} = \phi\Big(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(k)} h_j^{(k)} \Big)$$

3 conv layers → global mean-pool → MLP → energy. **Electron**: BDT ≈ DNN ≈ GCN ≈ GNN, all $R^2 \approx 0.999$ — tied.

Response calibration & the resolution definition



Nominal (no ML): calibrate raw E_{tot} to GeV via the detector response — the slope a of $\langle E_{\text{tot}} \rangle$ vs E_{true}; factor $= 1/a$.

- electron $a = 9.89$, factor 0.101 (linear).
- pion $a = 3.39$, factor 0.295 (huge fluctuations, non-linear).

Error bars = per-bin std. dev. of the raw signal across events (shower-to-shower spread, not a fit uncertainty) — see backup for details. **Resolution** = DSCB core σ/μ (width / fitted peak), not σ/E — matters when biased (π: $\mu \approx 0.65$).

Double-sided Crystal Ball fit

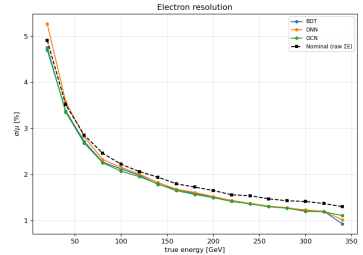
In each true-energy bin we fit the response ratio $r = E_{\text{pred}}/E_{\text{true}}$ with a DSCB: Gaussian core + power-law tails on both sides ($t = (r-\mu)/\sigma$):

$$f(t) = \begin{cases} A_L(B_L - t)^{-n_L}, & t \le -\alpha_L \\ e^{-t^2/2}, & -\alpha_L < t < \alpha_R \\ A_R(B_R + t)^{-n_R}, & t \ge \alpha_R \end{cases}$$

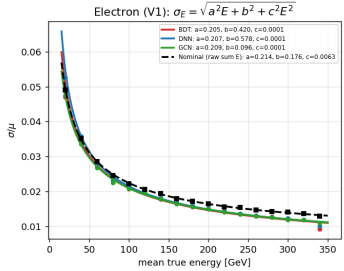
A, B fixed by continuity of f, f'. Calorimeter response has non-Gaussian tails (leakage; hadronic fluctuations) — the DSCB core σ is not inflated by them, unlike a plain Gaussian. The reported resolution is σ/μ.

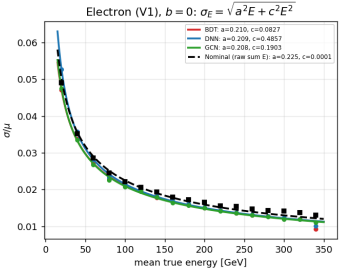
Electron resolution — and the resolution fit

$$\sigma_E = \sqrt{a^2 E + b^2 + c^2 E^2}$$



models vs nominal (σ/μ)



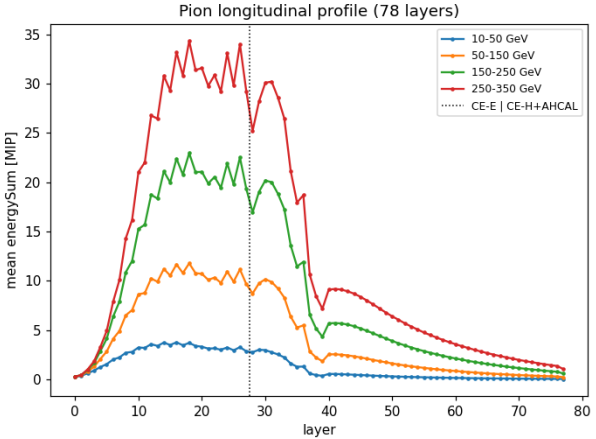


full fit (left) and $b=0$ fit (right); a=stoch, b=noise, c=const.

All four models overlap and barely beat the nominal floor. EM energy is nearly linear in E_{tot}, so ML adds only a $\sim$1% correction — no headroom, no winner. Resolution $\sim$5% @20 GeV $\rightarrow$ $\sim$1% @340 GeV.

Pion energy reconstruction

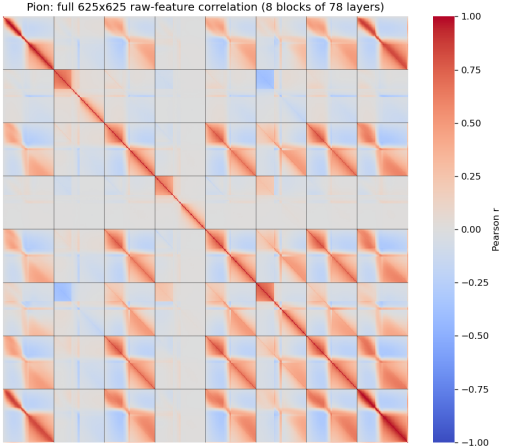
Pion longitudinal profile — deep, two-component



- 78 layers (CE-E | CE-H+AHCAL).
- Hadronic shower penetrates deep; a first EM-like peak then a broad hadronic tail.
- Shower depth ∼29 layers (vs ∼9 for e).
- Large event-to-event fluctuations (the wiggles).

Pion physics — the raw sum is a weak estimator

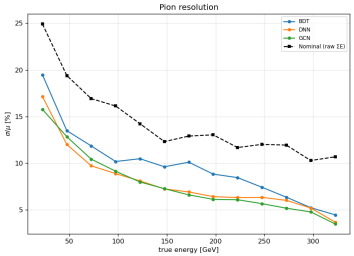
- E_{tot} correlates only 0.69 with energy (vs 0.9995 for e).
- n_{hits} correlates 0.87 — hit count beats summed energy!
- Hadronic showers lose a large *invisible* fraction to nuclear binding $\Rightarrow$ the measured signal under- and mis-estimates event-by-event.
- Single global MIP$\rightarrow$GeV factor leaves a non-linear bias ($\mu : 0.64 \rightarrow 0.73$).

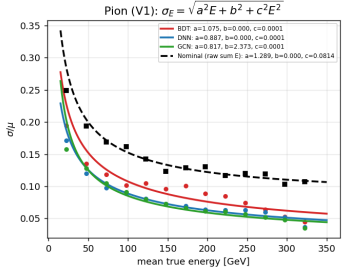


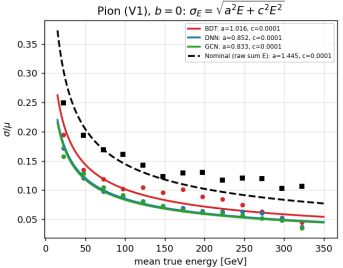
full 625×625 raw-feature correlation (pion).

$\Rightarrow$ large ML headroom over the nominal — and model choice now matters.

Pion model comparison — models separate







full fit (left) and b=0 fit (right).

- GCN best (V1); DNN close, BDT behind (numbers on the left plot).
- Models open a big gap over the nominal (dashed) — unlike electrons.

V2 — Improved models & Transformers

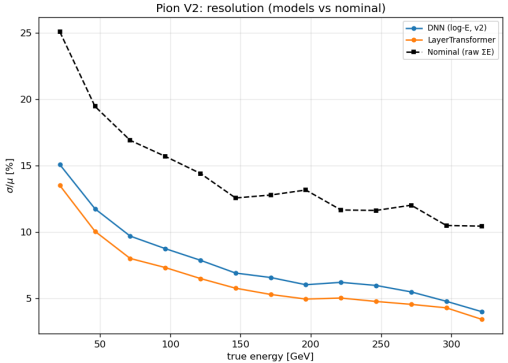
V2 improvements

1. Advanced features (24): skewness/kurtosis, shower start/extent, EE/FH/BH section energies, peak concentration, log-transforms. *Reconstructed only* (no MC-truth leakage).
2. Resolution-aware DNN: train on $\log E$ so the loss is a relative-error objective:

$$\mathcal{L} = \frac{1}{N} \sum_i \left(\log \frac{E_{\mathbf{pred}\,i}}{E_{\mathbf{true}\,i}} \right)^2$$

aligned with σ/μ.

3. Transformers (next slide).
4. Post-hoc calibration $c(E_{\mathbf{pred}}) = \langle E_{\mathbf{true}}/E_{\mathbf{pred}} \rangle$ to flatten the bias.



pion V2 σ/μ: LayerTransformer lowest.

Transformers — attention architectures

LayerTransformer (best on pions)

- Tokens = the calorimeter layers (+ learned positional encoding).
- Self-attention softmax$(QK^\top/\sqrt{d})V$: every layer attends to every other — captures long-range front/back correlations a fixed graph misses.
- 3 encoder layers, 4 heads, GELU, mean-pool $\rightarrow$ energy.

HitTransformer (Set-Transformer)

- Tokens = the raw hits (padded/masked point cloud).
- Attention over real hits only; masked mean-pool.
- Most expressive, but noisier for single-particle regression.

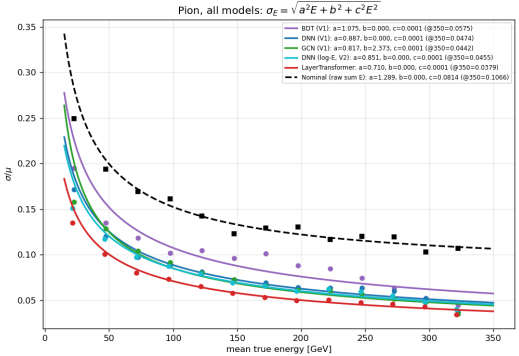
Pion result: LayerTransformer best, HitTransformer noisier — numbers on the next slide.

V2 result — LayerTransformer wins on pions

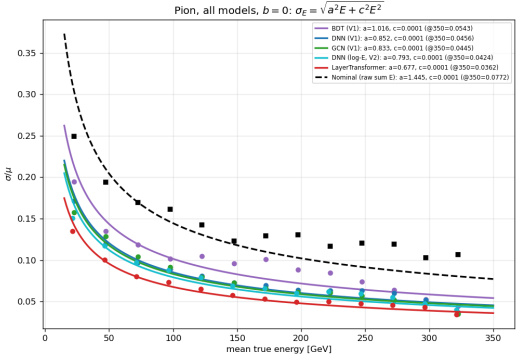
Model	R^2	σ/μ @22 GeV	σ/μ @172 GeV
LayerTransformer	0.978	13.5%	5.3%
DNN (log-E)	0.968	–	–
HitTransformer	0.930	–	–
BDT (V1, sklearn)	0.954	19.5%	10.1%
V1 best (GCN)	0.970	15.8%	6.6%
Nominal (raw ΣE)	–	25.1%	12.8%

Attention over layers beats the fixed-graph GCN for hadronic showers. Electrons stay saturated at $\approx$0.999 (no headroom).

Resolution fits, all models — $\sigma_E = \sqrt{a^2E + b^2 + c^2E^2}$ (and b=0)



full: a=**stoch**, b=**noise**, c=**const**



b=0: **stochastic** ⊕ **constant**

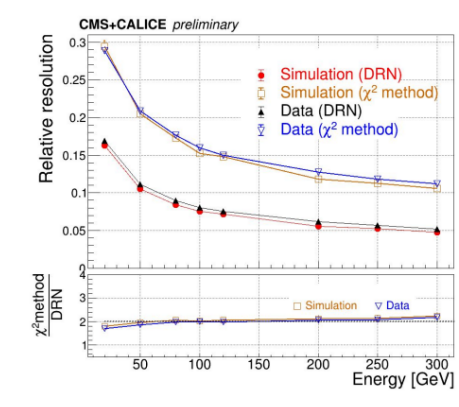
The pion resolution is stochastic-dominated: every model curve fits with a alone ($b, c \to 0$). Only the nominal (raw ΣE) carries a constant term $c \approx 0.081$ — an $\sim$8% irreducible floor that all ML models remove. LayerTransformer has the smallest stochastic term ($a = 0.710$, vs 0.817–1.075 for V1 BDT/DNN/GCN). Fit on $\sigma_E = \sigma_{/\mu} \cdot E$; plotted as $\sigma/\mu = \sigma_E/E$.

Thank you

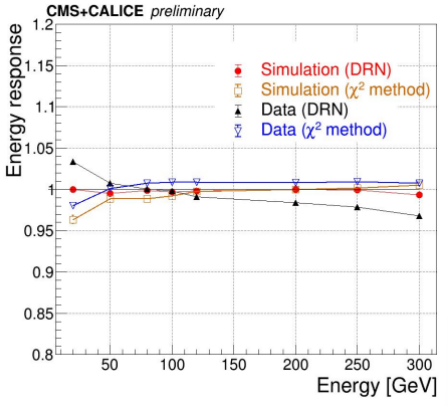
Backup: per-bin DSCB grids, V2 electron plots, hit-level results, response fits for all models.

Benchmark — external reference results

External benchmark — CMS+CALICE preliminary (DRN vs χ^2)



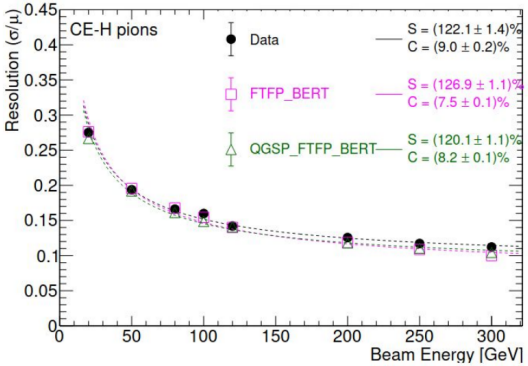
Relative resolution vs energy.



Energy response vs energy.

External CMS+CALICE preliminary result (not produced by this project), shown for context: Deep-Regression-Network (DRN) vs a χ^2-method reconstruction, Simulation and Data. An outside reference point for calorimeter-ML performance, not a direct comparison to this project's electron/pion datasets or models.

External benchmark — CE-H pion resolution, data vs Geant4



CE-H pion resolution σ/μ vs beam energy: Data vs two Geant4 physics-list simulations (FTFP_BERT, QGSP_FTFP_BERT), with stochastic (S) and constant (C) fit terms. External CMS HGCAL test-beam result (not produced by this project), shown for context on published pion-resolution benchmarks and typical stochastic/constant fit terms in the same σ/μ convention used throughout this deck.

Backup

Architectures, code, and ML methodology in full detail

Backup — Data flattening (hits → per-layer table)

Variable-length hit sets → fixed per-layer features via one vectorised grouped sum (`src/flatten_hgcal.py`). ∼100× faster than a per-event Python loop.

```
grp = event_local * n_layers + layer # unique (event,layer) bucket per hit
sw = np.bincount(grp, weights=E) # sum of energies (denominator)
esum = np.bincount(grp, weights=E) # energy sum per layer
def wstats(v): # energy-weighted mean & std of x / y / r
    swv = np.bincount(grp, weights=E*v)
    swv2 = np.bincount(grp, weights=E*v*v)
    mean = swv/sw ; var = swv2/sw - mean*mean
    return np.where(sw>0,mean,0), np.sqrt(np.where(sw>0,np.maximum(var,0),0))
# layers auto-detected (28 e / 78 pi); label key 'target' vs 'true_energy'
```

Validated to 8×10^{-8} vs `statsmodels.DescrStatsW`.

Backup — Feature engineering (11 moments)

`src/features.py` — compress 225 raw columns to physics moments of the shower.

```
E = df[energySum_cols].values ; layers = np.arange(n)
Etot = E.sum(1)
depth = (E*layers).sum(1)/Etot # longitudinal centre of gravity
length = np.sqrt((E*(layers-depth[:,None])**2).sum(1)/Etot) # spread
frac_front = E[:,:n//3].sum(1)/Etot # + mid, back thirds
lat_mean = (E*rMean).sum(1)/Etot # energy-weighted transverse radius
lat_spread = (E*rStd ).sum(1)/Etot
# + E_max_layer, layer_of_max, nhits_total -> 11 features
```

V2 (24 feats): skewness/kurtosis, reconstructed shower start/extent, EE/FH/BH section energies, `e_top5_frac`, `rstd_at_max`, log-transforms. *No MC-truth keys* (avoid leakage).

Backup — V2's 13 additional features (24 total)

`src/features_v2.py` — adds higher-order shower-shape detail to the 11 V1 features, all from the reconstructed per-layer profile (E_ℓ, r_ℓ^{std}).

```
skew = sum(E*(l-depth)^3)/Etot / sd^3 # long_skew: longitudinal asymmetry
kurt = sum(E*(l-depth)^4)/Etot / sd^4 # long_kurt: longitudinal peakedness
thresh = 0.02 * E.max() # 2% of peak-layer energy
shower_start, shower_end = first/last layer with E > thresh # + n_active_layers
frac_EE = E[:28].sum()/Etot # EE / FH / BH = section energies
frac_FH, frac_BH = E[28:mid].sum()/Etot, E[mid:].sum()/Etot # (pion only, n>28)
e_top2_frac, e_top5_frac = top-2 / top-5 layer energies, sum / Etot # peak concentration
rstd_at_max = rStd[layer_of_max] # transverse width AT the shower peak
log_Etotal, log_nhits = log1p(E_total), log1p(nhits_total) # compress wide dynamic range
```

All 13 are functions of already-available reconstructed quantities (no new raw inputs) — `shower_start/end` use a 2% reconstructed-peak threshold, *not* the MC-truth `shower_start_layer` key (which exists in the pion file but is never read).

Backup — BDT: gradient boosting

Additive ensemble $F_M(x) = \sum_m \nu\, h_m(x)$; tree h_m fits the negative gradient (residuals) of the loss so far. ν = learning rate (shrinkage).

```
GradientBoostingRegressor(n_estimators=400, learning_rate=0.05, max_depth=3, subsample=0.9)
```

Split criterion (regression): variance reduction; boosting shrinkage ν trades bias/variance. Results on the main BDT slide.

Backup — DNN: V1 (Keras, 11 feats) and V2 (log-E, PyTorch)

V1 — tuned width=64, depth=3 (`g2_best_params.json`):

```
m = Sequential([Input((11,))])
for _ in range(3):
    m.add(Dense(64, activation="relu"))
m.add(Dense(1))
m.compile(Adam(1e-3), loss="mse")
```

V2 — 24 feats, trained on $\log E$:

```
net = Sequential(Linear(24,256),
  BatchNorm1d(256), ReLU(), Dropout(.1),
  Linear(256,256), BatchNorm1d(256),
  ReLU(), Dropout(.1), Linear(256,1))
loss = F.mse_loss(net(x), log(y_true))
E_pred = exp(net(x)) # back to GeV
```

V2 adds BatchNorm + dropout + cosine LR schedule and trains on $\log E$ so the loss is a *relative*-error objective: $\mathcal{L} = \frac{1}{N}\sum(\log E_{\mathrm{pred}\,i} - \log E_{\mathrm{true}\,i})^2 = \frac{1}{N}\sum(\log E_{\mathrm{pred}\,i}/E_{\mathrm{true}\,i})^2 \Rightarrow$ penalises the ratio that enters σ/μ, unlike V1's plain MSE.

Backup — GCN: graph construction + message passing

Node = layer ([energySum,rMean,rStd,nhits], standardised); edges = successive layers (GCN).

```
edge_index = sequential_edges(n) # chain graph (or fully_connected_edges(n) = GNN)
class GCN(nn.Module):
    def __init__(s,h=64):
        s.c1=GCNConv(4,h); s.c2=GCNConv(h,h); s.c3=GCNConv(h,h)
        s.lin=Sequential(Linear(h,h),ReLU(),Linear(h,1))
    def forward(s,x,ei,b):
        x=relu(s.c1(x,ei)); x=relu(s.c2(x,ei)); x=relu(s.c3(x,ei))
        return s.lin(global_mean_pool(x,b)) # message pass -> pool -> energy
# Adam(1e-3), MSE, 40 epochs, batch 512
```

$$h_i^{(k+1)} = \phi\Big(\sum_{j\in\mathcal{N}(i)\cup\{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(k)} h_j^{(k)}\Big)$$

Backup — LayerTransformer: tokens, positions, attention

Tokenisation. One token per calorimeter layer (28 e / 78 π), features [energySum, r_{mean}, r_{std}, n_{hits}], standardised, linearly projected to d_{model}=64.
Positional encoding. *Learned* (not sinusoidal): a free parameter pos $\in \mathbb{R}^{1\times 80\times 64}$, init $\mathcal{N}(0, 0.02^2)$, added to the token embedding — lets the model discover its own notion of "depth ordering" from data rather than assuming a fixed frequency basis. **Multi-head attention** (4 heads, each d_k=16): each head computes its own Q_i, K_i, V_i via separate linear projections,

$$\text{head}_i = \text{softmax}\Big(\frac{Q_i K_i^\top}{\sqrt{d_k}}\Big) V_i$$

$$\text{MHA} = \text{Concat}(\text{head}_{1..4})\, W^O$$

so different heads can specialise (e.g. one head on adjacent-layer correlation, another on front/back).

```
embed = Linear(4, 64)
pos = Parameter(randn(1,80,64)*0.02)
enc = TransformerEncoderLayer(
  64, nhead=4, dim_feedforward=128,
  dropout=0.1, activation="gelu")
tr = TransformerEncoder(enc, num_layers=3)
head = Sequential(LayerNorm(64), Linear(64,64),
                  GELU(), Linear(64,1))
def forward(x): # x: (B,L,4)
    h = tr(embed(x)+pos[:,:L])
    return head(h.mean(1))
```

3 stacked encoder layers (attention → residual → LayerNorm → FFN → residual → LayerNorm each); mean-pool over the L output tokens; small MLP head → scalar energy. Best on pions: $R^2 = 0.978$.

Backup — HitTransformer: Set-Transformer over raw hits

Tokenisation. One token per raw hit (*not* per layer): features $[x, y, z,\ e/100,\ \log(1{+}e)]$ (standardised coords). Kept to the top-128 hits by energy per event (electron showers have ∼800 hits; padding/truncating to a fixed size is what makes batching possible). **No positional encoding** — unlike layers, hits have no natural order; adding one would falsely tell the model position matters. This is what makes it a *set* transformer. **Padding mask.** A boolean mask (True = padding) is passed to `src_key_padding_mask`: attention scores to padded positions are set to $-\infty$ before the softmax, so real hits never attend to (or are pooled from) padding.

```
embed = Linear(5, 64)
enc = TransformerEncoderLayer(
  64, nhead=4, dim_feedforward=128,
  dropout=0.1, activation="gelu")
tr = TransformerEncoder(enc, num_layers=3)
def forward(x, pad_mask): # x:(B,128,5)
    h = tr(embed(x),
      src_key_padding_mask=pad_mask)
    keep = (~pad_mask).float()[...,None]
    h = (h*keep).sum(1)/keep.sum(1).clamp(1)
    return head(h) # masked mean-pool
```

Masked mean-pool over real hits only ⇒ permutation-invariant (same output regardless of hit order) and length-invariant (works for any real hit count ≤128). Most expressive architecture here but noisiest for single-particle regression (pion $R^2 = 0.93$) — attends over 128^2 hit pairs with less inductive bias than the fixed layer sequence, so it has more to learn from a comparatively small dataset. Advantage expected in dense multi-particle events (jets), where fine hit-level structure carries information a per-layer summary discards.

Backup — Tunable parameters per model

Model	Parameters
DNN (V1, 11 feats)	9,153
GCN (V1)	12,865
GNN (V1, same net)	12,865
BDT (V1, tuned)	43,398
DNN (log-E, V2)	73,473
HitTransformer	105,153
LayerTransformer	110,209

How. Instantiate each model with its *real*, *tuned* hyperparameters (from the notebooks/HPO results, not re-derived) and sum `p.numel()` over `model.parameters()` in PyTorch — the exact learnable-scalar count, not a hand estimate.

```
sum(p.numel() for p in
    model.parameters())
```

BDT is different. A tree ensemble's size is *data-dependent* — sklearn grows each tree until `min_samples_leaf`/`max_depth` stops it, so node count can't be read off the hyperparameters alone. We fit the real tuned config (684 trees, depth ≤ 5) on 10k electron events and read `tree_.node_count` / `tree_.n_leaves` off every fitted tree; "parameters" = 2 numbers per internal split (feature, threshold) + 1 per leaf (predicted value), summed over all 684 trees. GCN/GNN share one architecture (only edge connectivity differs) $\Rightarrow$ identical count. LayerTransformer and BDT are the two largest — and also the two best-performing pion models.

Backup — Input features per model (I): tabular models

Electron and pion use *identical feature names* — the flattening code auto-detects 28 (e) vs 78 (π) calorimeter layers and compresses either one down to the same named columns (`src/features.py`, `features_v2.py`).

BDT (V1), DNN (V1) — 11 engineered features (from the 28/78-layer profile):

```
E_total, E_max_layer, layer_of_max, shower_depth, shower_length,
frac_front, frac_mid, frac_back, lateral_mean_r, lateral_spread_r, nhits_total
```

DNN (log-E, V2) — 24 features = the 11 above +13 more:

```
long_skew, long_kurt, shower_start, shower_end, n_active_layers,
frac_EE, frac_FH, frac_BH, e_top2_frac, e_top5_frac, rstd_at_max,
log_Etotal, log_nhits
```

All quantities are reconstructed (energy-weighted moments of the per-layer profile) — none use MC-truth keys (`shower_start_layer`, `energyLost*`), so there is no label leakage.

Backup — Input features per model (II): graphs & point clouds

GCN, GNN (V1), LayerTransformer — 4 features per node/token, one node per layer:

```
energySum, rMean, rStd, nhits # per calorimeter layer
```

28 nodes/tokens for electron, 78 for pion (graph edges: successive-layer for GCN/LayerTransformer, fully-connected for GNN — connectivity differs, node features don't).

HitTransformer — 5 features per raw hit:

```
x_std, y_std, z_std, e/100, log1p(e) # per calorimeter hit
```

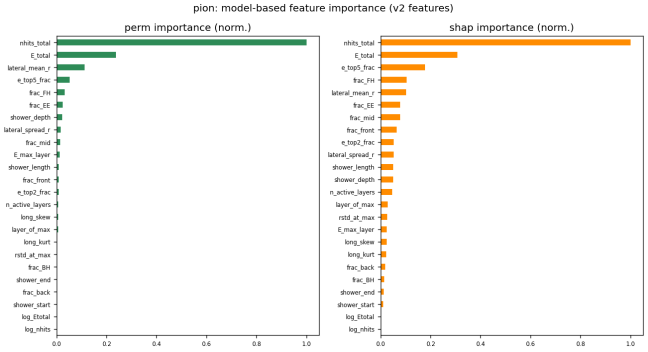
x, y, z standardised to training-set mean/std; hit energy kept un-centred (scaled by $1/100$) so pooling recovers total deposited energy. Point-cloud size: top 128 hits by energy (padded/masked; electron showers have $\sim$800 raw hits, pion showers even more).

Backup — Feature importance (2 methods)

```
# 1) PERMUTATION: drop in test R^2 when a feature's values are shuffled (model-agnostic)
perm = permutation_importance(model, X_test, y_test, n_repeats=5)
# 2) SHAP: game-theoretic per-prediction attributions; mean|SHAP| = global importance
sv = shap.TreeExplainer(model).shap_values(X_test)
```

Findings. Pion top-5 (SHAP): nhits_total > E_total > e_top5_frac > frac_FH > lateral_mean_r. Electron: E_total > nhits > lateral radius/width. Univariate MI ranks *information*; these rank what the *model uses*.

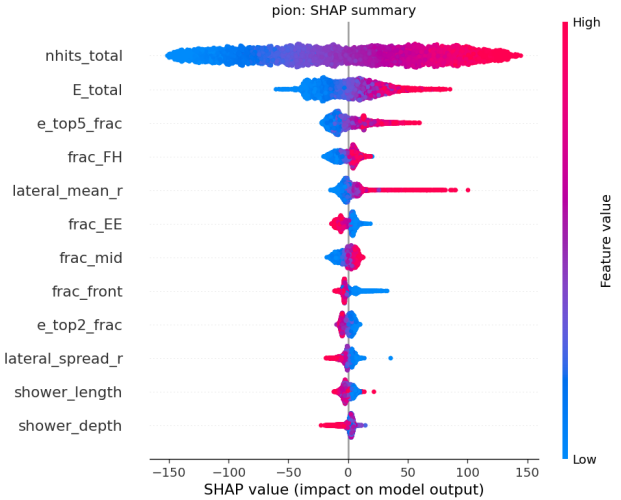
Backup — Model-based feature importance, plot (pion)



- Permutation: test-R^2 drop when a feature is shuffled (honest, model-agnostic).
- SHAP: game-theoretic per-prediction attributions.

Unlike the single-feature (univariate) MI ranking, these show what the model actually uses. (Pion shown; electron analogous.)

Backup — SHAP summary (pion) — and the key finding



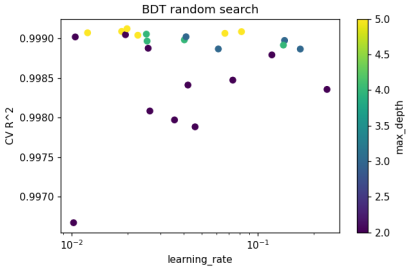
SHAP top-5 (pion):

1. n_{hits} (beats E_{tot}!)
2. E_{total}
3. e_top5_frac (v2 feature)
4. frac_FH (v2 feature)
5. lateral_mean_r

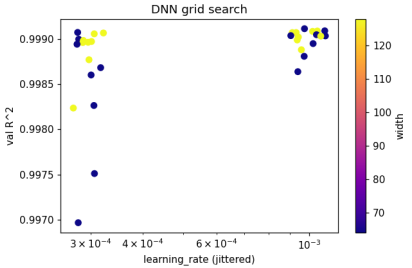
⇒ hit count is the single most important pion feature (matches corr 0.87 vs 0.69), and two v2 advanced features enter the top-5 — validating the feature engineering.

(Electron top-5: E_{tot}, n_{hits}, lateral radius/width, back fraction.)

Backup — V1 hyper-parameter search: BDT (random), DNN (grid)



25 candidates × 3-fold CV = 75 fits.



Grid over 5 knobs, $2^5 = 32$ configs.

BDT space:
n_estimators[200,800]
lr logU[0.01,0.3]
max_depth[2,6)
min_split[2,40)
min_leaf[1,40)
subsample[.6,1]
max_features[.5,1]

DNN space:
width{64,128}
depth{2,3}
lr{1e-3,3e-4}
batch{256,512}
dropout{0,.1}

Both on a 50k-event sub-sample (15% test set held out, never touched by search). BDT: `RandomizedSearchCV(n_iter=25, cv=3)`. DNN: exhaustive grid, each config trained with early stopping (patience 8) on an 80/20 train/val split. BDT sits on a flat plateau for lr$\gtrsim 0.03$ (robust). DNN clusters near its best for lr$\in [3\times10^{-4}, 10^{-3}]$, largely width-independent — 11 low-dimensional inputs leave little to overfit. Both winners re-fit on the full search set reach $R^2 \approx 0.9991$ on the untouched test set.

Backup — Hyper-parameter optimisation (Optuna TPE)

Bayesian TPE sampler; held-out test never touched; single train/val split per trial (avoids CPU oversubscription). Winners re-fit on full data.

```
def dnn_obj(trial): # 20 trials
    h = trial.suggest_categorical("h",[128,256,384]); depth = trial.suggest_int("depth",2,4)
    drop = trial.suggest_float("drop",0.0,0.3); lr = trial.suggest_float("lr",3e-4,3e-3,log=True)
    net = make_dnn(h,depth,drop).to(dev) # trained on log E; val R^2 = objective
    ...
    return r2_score(y_va, np.exp(net(X_va).detach()))
study = optuna.create_study(direction="maximize", sampler=TPESampler(seed=42))
study.optimize(dnn_obj, n_trials=20)
# LayerTransformer: 12 trials (d_model/layers/heads/lr), same pattern
```

Result: HPO mainly *confirms* configs; gains came from architecture + features.

Backup — Hyper-parameter optimisation results (V2/V3, Optuna)

- log-E DNN — 20 trials over width/depth/dropout/learning rate.
- LayerTransformer — 12 trials over d_{model}/layers/heads/learning rate.

Discipline: held-out test never touched; winners re-fit. **Tuned results**: pion log-E DNN val-$R^2 = 0.962$, LayerTransformer val-$R^2 = 0.972$ (already near-optimal, confirmed best). HPO mainly *confirms* the configs — the gains came from architecture (Transformer) and features, not fine-tuning. (A first run hit a CPU-oversubscription hang from nested parallelism — fixed by a single validation split instead of `cross_val_score(n_jobs=-1)`.)

Backup — Double-sided Crystal Ball & σ/μ

Per energy bin, fit $r = E_{\text{pred}}/E_{\text{true}}$ with a Gaussian core + power-law tails ($t = (r-\mu)/\sigma$): $f(t) = e^{-t^2/2}$ (core), $A(B \mp t)^{-n}$ (tails), A, B from continuity.

```
popt = curve_fit(dscb, x_hist, counts, p0=[mu0,sd0,1.5,3,1.5,3,peak], bounds=...)
mu, sigma = popt[0], abs(popt[1]) # core response (mu) and width (sigma)
resolution = sigma/mu # NOT sigma/E_true -> matters when biased (pion mu~0.65)
# fall back to Gaussian if the CB fit fails to converge
```

Tails: EM leakage / hadronic fluctuations $\Rightarrow$ DSCB core σ not inflated by them.

Backup — How the response-plot error bars were computed

Both panels of the response-calibration plot bin events by true energy and show the mean raw signal $\langle E_{\text{total}} \rangle$ per bin. The error bar is the population standard deviation of E_{total} within that bin — i.e. how much the raw calorimeter signal actually varies event-to-event at fixed true energy — *not* a statistical uncertainty on the mean ($\sigma/\sqrt{N}$) and *not* a fit error.

```
be = np.linspace(y.min(), y.max(), 40); bc = 0.5*(be[:-1]+be[1:]) # 40 true-energy bins
mean_sig = np.array([E_total[(y>=lo)&(y<hi)].mean() for lo,hi in zip(be[:-1],be[1:])])
std_sig = np.array([E_total[(y>=lo)&(y<hi)].std() for lo,hi in zip(be[:-1],be[1:])]) # error bar
plt.errorbar(bc, mean_sig, yerr=std_sig, fmt='o', ...)
```

Why pion error bars are so much larger. Electron showers are compact and reproducible (EM shower ⇒ narrow spread of E_{total} at fixed E_{true}, small error bars). Pion (hadronic) showers have large, physical shower-to-shower fluctuations — variable nuclear-binding energy loss, variable shower start depth — so E_{total} varies far more at the same true energy: this is exactly the physics captured in the pion resolution being stochastic-dominated ($a \approx 1$ vs electron's $a \approx 0.2$ on the fit-parameters backup slides), not a computational artefact.

Backup — Nominal calibration & resolution fit

```
# MIP->GeV from the detector RESPONSE (no ML): scipy fit of <E_total> vs E_true (through origin)
popt, _ = curve_fit(lambda E,a: a*E, E_true, E_total); a = popt[0]; factor = 1/a # e:0.101, pi:0.295
E_reco = E_total/a
# resolution fit (absolute): sigma_E = sqrt(a^2 E + b^2 + c^2 E^2) a=stoch,b=noise,c=const
# bounds: a,b >= 0; c >= 1e-4 (kept off exact zero -- see next slide for why)
popt, pcov = curve_fit(f_abs, E, sigma_over_mu*E, bounds=([0,0,1e-4],[inf,inf,inf]))
perr = np.sqrt(np.diag(pcov)) # 1-sigma parameter uncertainties
```

Fit on absolute $\sigma_E = \sigma_{/\mu} \cdot E$; plotted as $\sigma/\mu = \sigma_E/E$. Pion is stochastic-dominated ($b, c \to 0$); only the nominal has a constant term $c \approx 0.081$.

Backup — Resolution fit parameters (I): full a, b, c

Dataset	Model	a	b	c
Electron	BDT	0.205±0.020	0.420±0.522	0.0001†
Electron	DNN	0.207±0.013	0.578±0.251	0.0001†
Electron	GCN	0.209±0.004	≈0†	0.0001†
Electron	Nominal	0.214±0.004	0.176±0.240	0.006±0.000
Pion	BDT	1.075±0.172	≈0†	0.0001†
Pion	DNN	0.887±0.098	≈0†	0.0001†
Pion	GCN	0.817±0.166	2.373±3.095	0.0001†
Pion	Nominal	1.289±0.099	≈0†	0.081±0.007
Pion	DNN (log-E, V2)	0.851±0.065	≈0†	0.0001†
Pion	LayerTransformer	0.710±0.043	≈0†	0.0001†

† Parameter sits *at* its fit bound ($a, b \geq 0$; $c \geq 10^{-4}$). At an active bound the covariance-based error from `curve_fit` is not meaningful (can come out as ≈0 or absurdly large, e.g. $\sigma_c \sim 10^4$) — this is a standard artefact of bounded least squares, not a real uncertainty, so no ± is quoted for those entries.

Backup — Resolution fit parameters (II): b=0

Dataset	Model	a	c
Electron	BDT	0.210±0.002	0.083±0.215
Electron	DNN	0.209±0.002	0.486±0.029
Electron	GCN	0.208±0.001	0.190±0.045
Electron	Nominal	0.225±0.003	0.0001†
Pion	BDT	1.016±0.065	0.0001†
Pion	DNN	0.852±0.030	0.0001†
Pion	GCN	0.833±0.036	0.0001†
Pion	Nominal	1.445±0.123	0.0001†
Pion	DNN (log-E, V2)	0.793±0.038	0.0001†
Pion	LayerTransformer	0.677±0.021	0.0001†

† = boundary-pinned, error not meaningful (see previous slide). Forcing b=0 pushes essentially all of the b=0 fit's freedom onto a (stochastic term); every pion model's c collapses to the floor, consistent with the full fit already finding $c \approx 0$ for every ML model.

Backup — Why the fully-connected GNN was dropped

- The GNN (all-layers-connected `GCNConv`) gave pion $R^2 = 0.82$ — *worse than the nominal* at low energy, and its resolution fit is anomalously large (steep, noise-like b).
- Cause: fully connecting all 78 layers lets every layer average over every other, which *over-smooths* the localised longitudinal structure a hadronic shower carries. The successive-layer **GCN** keeps locality and wins ($R^2 = 0.970$).
- Lesson: graph *connectivity is a hyper-parameter* — too much connectivity destroys information. Attention (LayerTransformer) instead *learns* which layers to combine.
- Excluded from the main comparison for clarity; retained in the JSON results.

Backup — Metrics, splits, data & compute

Metrics

- R^2 (variance explained, defined on the regression-problem slide)
- MAE, RMSE (GeV); resolution σ/μ (per bin, DSCB)

Discipline

- fixed seed; 80/20 train/test; HPO on a held-out val, test never touched
- same split reused by every model (fair comparison)

Data

- electron 648,277 events (28 layers); pion 836,658 (78)
- full data for headline results; documented sub-samples only where exact-GB / hit-level cost forced it

Compute

- RTX 4070 (8 GB) for graph/transformer nets; CPU for trees