ML4FP 2026 • Atlanta

Transformers for Particle Physics

Notebooks 1, 2, 3 • github.com/ml4fp/2026-gatech

THREE NOTEBOOKS

Outline

1 Self-attention from scratch

Build a working transformer jet classifier piece by piece.

lecture ≈ 30 min • tutorial ≈ 40 min

2 The cost of attention

Quadratic scaling, low-rank structure, Linformer.

lecture ≈ 15 min • tutorial ≈ 20 min

3 The Particle Transformer

Physics priors as an attention bias; interpretability on real ParT weights.

lecture ≈ 20 min • tutorial ≈ 20 min

CONTEXT

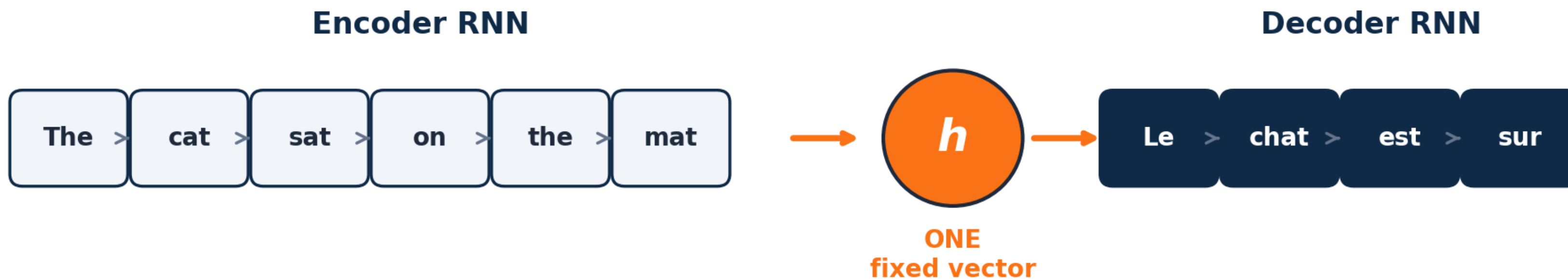
Architectures before the transformer

MLPs	Just stacks of dense layers. Treat each input as a fixed-length vector. no notion of ‘nearby pixels’ or ‘nearby particles’.
CNNs	Convolutions exploit locality and translation symmetry — perfect for images
RNNs / LSTMs	Process sequences one step at a time. Carry a hidden state. Used to be the standard for language. Hard to parallelize, struggle with long contexts.
GNNs	Graphs of nodes + edges. Naturally permutation-equivariant. ParticleNet is in this family.
Transformers	Every element can ‘look at’ every other element directly via attention. Parallelizable. Now the dominant architecture for almost everything.

Today we build the last row from scratch.

PRE-2014

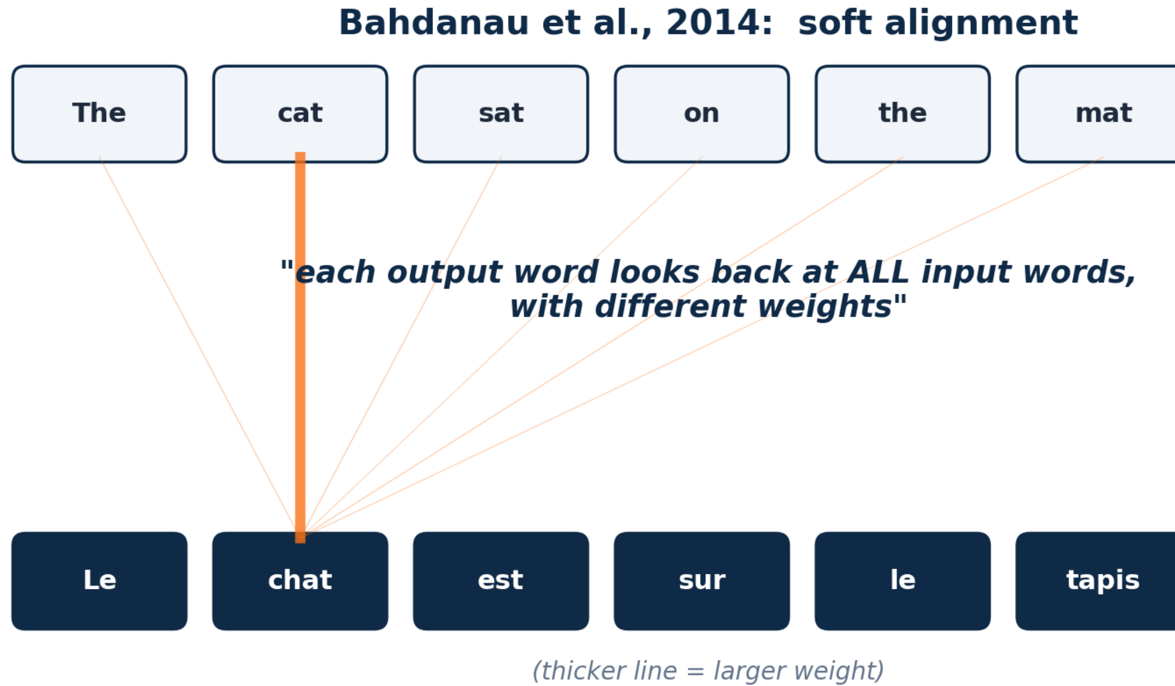
Before Attention Transformers: the RNN!



- Before transformers, the dominant way to handle sequences was the RECURRENT neural network (RNN, LSTM, GRU).
- They worked left-to-right, one element at a time, carrying a single 'hidden state' vector forward.
- For machine translation: an encoder RNN squeezed the whole source sentence into ONE hidden vector, which the decoder unrolled into the target sentence.
- Problem: ONE vector cannot remember a long sentence. Performance fell off a cliff for long inputs.

FIRST ATTENTION MECHANISM

Bahdanau 2014: soft alignment



- Bahdanau, Cho, Bengio (2014): instead of one bottleneck vector, let the decoder LOOK BACK at every encoder state and DECIDE which to focus on.
- Each output position gets a weighted combination of input states — they called this 'soft alignment'. This is the very first attention mechanism.
- Translation quality improved substantially. But the network was still recurrent: each step had to wait for the previous one, so it was slow to train.

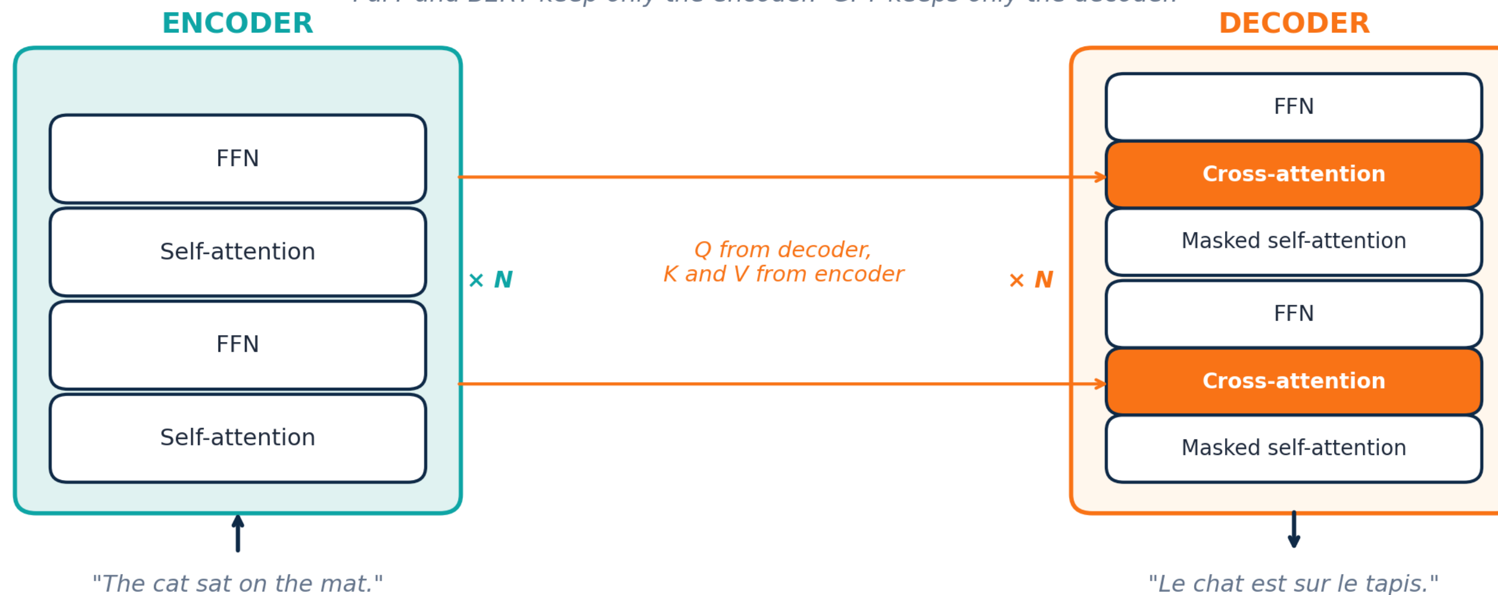
Vaswani 2017

'Attention Is All You Need.' Vaswani et al., NeurIPS 2017.

- The paper drops recurrence entirely and uses attention to mix information across the sequence.
- Every position computes its output from every other position in parallel. No sequential bottleneck, which makes the architecture well-suited to GPUs.
- Adopted broadly over the next 5 years: NLP (BERT, GPT), vision (ViT), structural biology (AlphaFold 2), and particle physics.

The Particle Transformer is one specialization of this architecture for particle physics..

Encoder–decoder transformer



- Built for machine translation. The encoder reads the source with self-attention. The decoder generates the target one token at a time.
- Cross-attention is the bridge: queries from the decoder pull keys and values out of the encoder so each output word can look at every source word.
- For non generation tasks, oftentimes we keep only the encoder. Each particle's vector gets updated by thea ttention mechanism!

Classifier (ParT, BERT)

`Input → encoder → one label`

- Encoder-only transformer.
- Reads the whole jet at once, produces one prediction.
- Today: 10 jet classes (QCD, H→bb, top→bqq, ...).
- Loss: cross-entropy against a known jet label.

Generative (GPT, Llama, OmniJet-α)

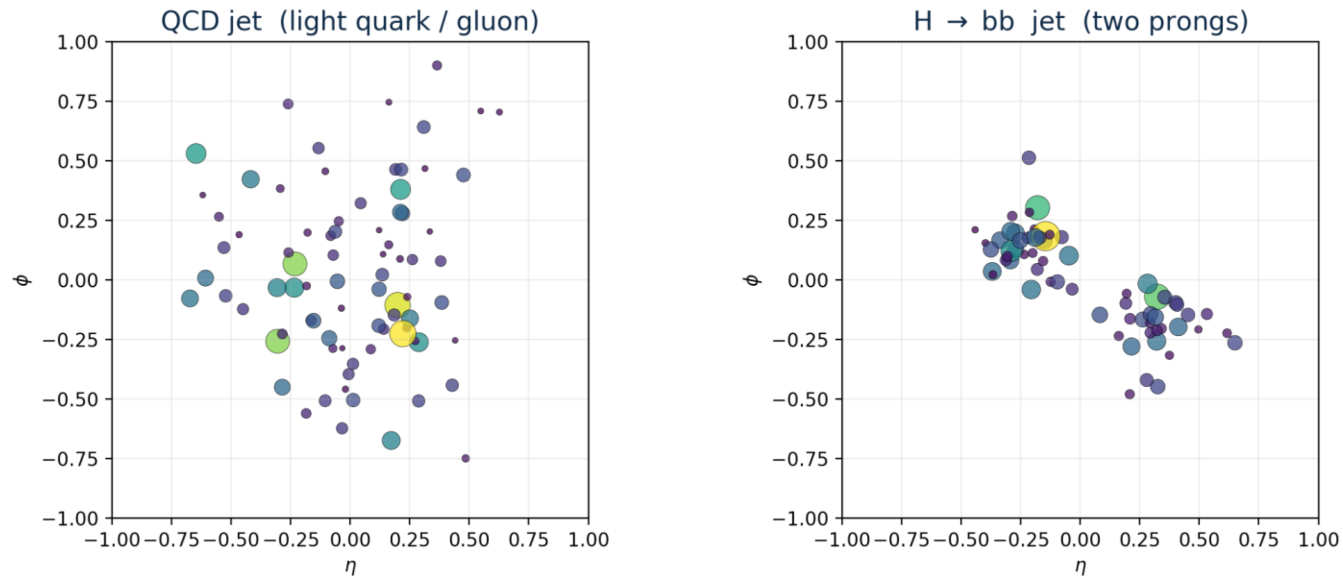
`Input → decoder → next token, repeat`

- Decoder-only transformer (causal mask).
- Produces output one token at a time, sampling from a probability distribution.
- Trained to predict the next word from the previous ones.
- Loss: cross-entropy against the next token.
- Used for text generation.

Our transformer reads the jet and outputs P(class). It does NOT generate new particles.

MOTIVATION

Transformers in particle physics



- Many modern HEP problems are ‘set of things’ problem such as the list of particles in a jet! This is also known as a point cloud.
- Those sets are unordered and have variable length. Transformers handle that directly without any kind of additional processing.
- Transformers currently lead the public benchmarks for jet tagging (JetClass), and are widely used for calorimeter reconstruction, track finding, and event-level tasks.

VOCABULARY

Tokens vs. particles

	Large language model	Particle transformer
Input Unit	Token (≈ word piece)	Particle (or hit, or tower)
Input feature	Word-piece ID + position	(p_T, η, ϕ, energy, ...)
Sequence length	Up to 100k+ tokens	≈ 50–200 particles per jet
Ordering	Order matters — adds positional encoding	Order does NOT matter — no positional encoding
Output	Next-token probabilities	Class probabilities for the jet

PART

1

Self-attention, from scratch

Notebook 1 • building a transformer jet classifier piece by piece

Jets are sets, not sequences

Sequence: order matters

"dog bites man" ≠ "man bites dog"

Set: order doesn't matter

jet = unordered cloud of particles

- ‘Particle #1’ vs ‘particle #2’ is just an arbitrary order we wrote them down in.
- Re-shuffling the particles does NOT change the jet, so it must NOT change the prediction.

↔ **Compare with LLMs:** LLMs DO care about order (‘dog bites man’ ≠ ‘man bites dog’) — they bolt on positional encoding to break the symmetry. For jets we skip that step entirely.

JetClass Dataset!

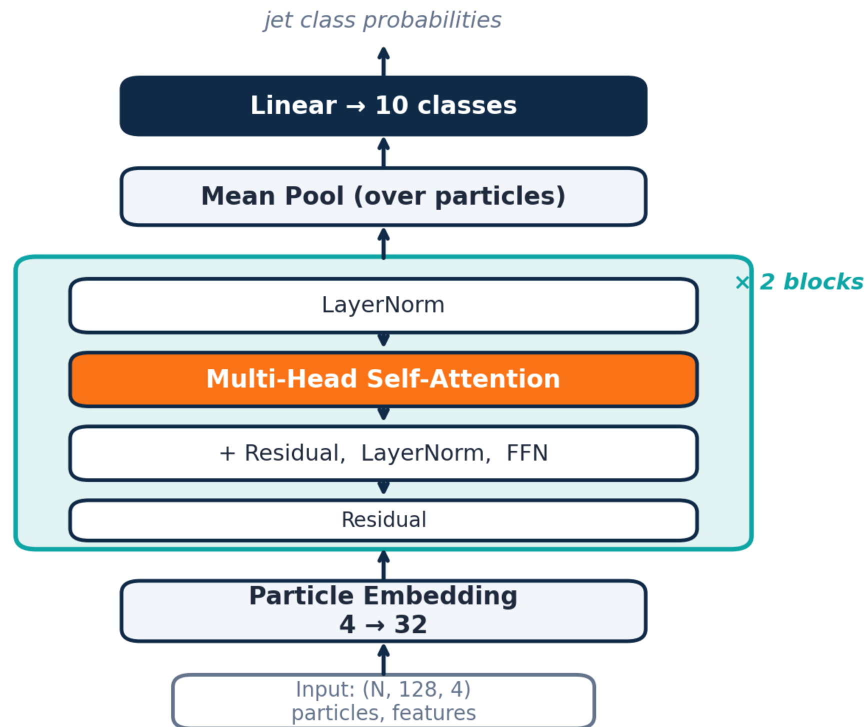
- Each jet: up to 128 particles, zero-padded to a fixed length.
- Each particle: 4 kinematic features (p_T, η, ϕ, energy) + a bunch of additional features
- 10 classes: QCD, H→bb, H→cc, H→gg, H→4q, H→qqℓν, Z→qq, W→qq, top→bqq, top→bℓν.

What we want:

input: `(N, 128, 4)` ← N jets, 128 particles each, 4 features each
mask: `(N, 128)` ← True for real particles, False for padding
output: `(N, 10)` ← class probabilities

NOTEBOOK 1

Full architecture



1. Embed each particle
2. Mix particles via Multi-Head Self-Attention
3. Per-particle feed-forward network
4. Stack 2 of those blocks
5. Mean-pool to one vector per jet
6. Linear layer → 10 class scores

BUILDING BLOCK

Particle embedding

Just a linear layer: 4 features → 32-dimensional vector per particle.

```
z = W x + b
```

Why embed?

Subsequent layers all act on 32-dim vectors. The embedding gives each particle ‘room to grow’ — the model can learn richer per-particle representations than the raw 4 numbers allow. Better representation for the model to learn from! Note that the linear layer only mixes along the feature axis!

↔ **Compare with LLMs:** in an LLM the embedding also produces a better representation of words. In fact anthropic has done interpretability studies mapping representations of concepts in language!

Physics Particle Transformer uses this linear layer with a small 3-layer MLP and adds particle-ID features (charge, hadron/lepton flags).

BUILDING BLOCK

Self-attention

Imagine each particle 'looks around' and asks

"which of the other particles are relevant to me, and what information do they have?"

↔ **Compare with LLMs:** the same mechanism is used in LLMs. Each word asks 'which other words are relevant for predicting / understanding me?'

Attention in sentences!

Read this sentence and decide which word the underlined word depends on:

The **cat** that the dog chased **was** tired.

- 'was' has to agree with the subject. The subject is 'cat', not 'dog' — even though 'dog' is the nearest noun.
- To embed 'was', the model needs to find 'cat' in the sentence and ignore the distractors.
- That 'find and weight the relevant other tokens' operation is exactly self-attention.

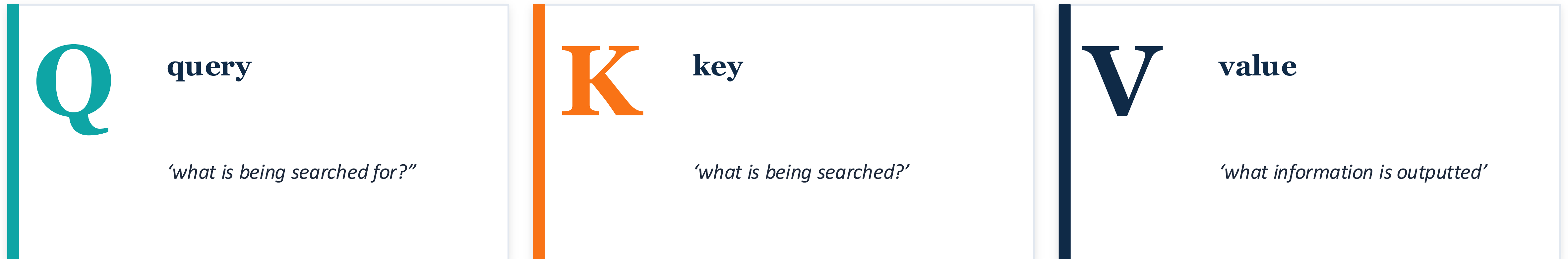
THREE PROJECTIONS OF THE INPUT

Q, K, V

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

X = stack of all 128 embedded particles (128 × 32 (embedding size))

W_Q, W_K, W_V are learned weight matrices — three different ‘lenses’ on the same particle.



↔ **Compare with LLMs:** same in GPT — every token has its own Q, K, V projections.

DESIGN

Why three different projections

If Q = K = V (just $x \cdot x^T$)

→ *each particle mostly attends to itself*

0.86	0.08	0.00	0.04	0.02
0.01	0.84	0.04	0.05	0.06
0.05	0.07	0.78	0.10	0.00
0.07	0.05	0.06	0.79	0.02
0.08	0.09	0.03	0.06	0.74

With learned W_Q, W_K, W_V

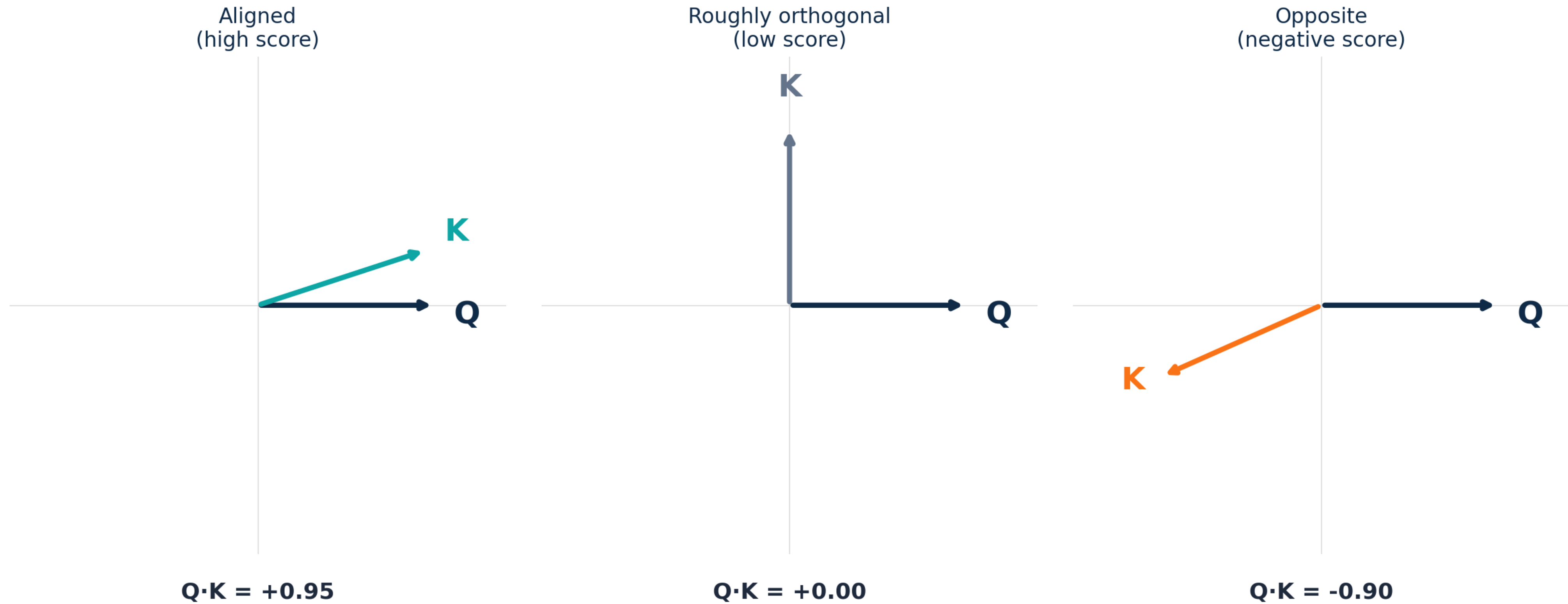
→ *Q asks one question, K answers a different one*

0.04	0.26	0.15	0.24	0.32
0.28	0.26	0.06	0.15	0.26
0.29	0.35	0.17	0.05	0.14
0.34	0.10	0.18	0.35	0.03
0.19	0.29	0.08	0.17	0.27

- If Q = K = V = x, the score matrix is $x \cdot x^T$ — a symmetric self-similarity matrix. Each particle ends up mostly attending to itself. Useless.
- Three separate matrices W_Q, W_K, W_V give the model three independent degrees of freedom — Q learns what to look for, K learns what to advertise, V learns what to pass on.
- In trained models, these three projections end up looking very different from each other.

GEOMETRY

Dot product as similarity



$Q \cdot K = |Q|\ |K| \cos\theta$. Same direction → big positive. Orthogonal → zero. Opposite → negative.

So scoring all (Q_i, K_j) pairs is just asking 'how aligned is each query with each key in feature space?'

FOUR STEPS

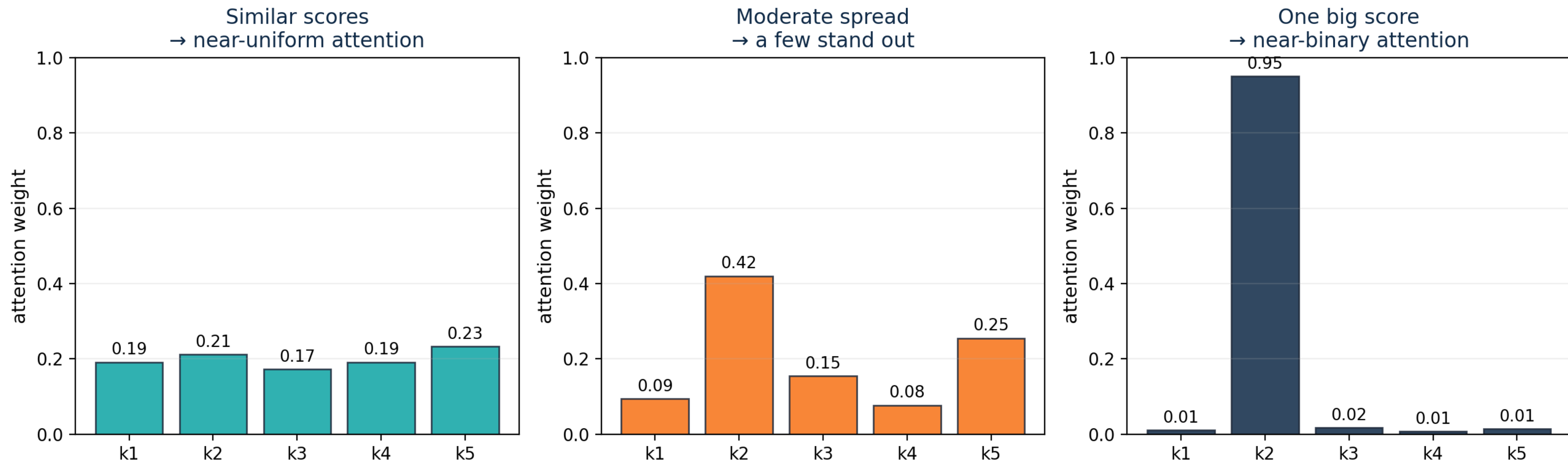
The attention formula

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

1. $QK^\top$ all-pairs dot products — every query meets every key. Big number = ‘these two are compatible’.
2. $\div \sqrt{d_k}$ rescale so the scores don't blow up as the vectors grow longer.
3. `softmax` turn each row into a probability distribution over the other particles (rows sum to 1).
4. $\times V$ weighted sum of values — each particle becomes a blend of the others, with the blend chosen by the softmax.

Softmax behavior

$$\mathrm{softmax}(x)_i = \exp(x_i) / \Sigma_j \exp(x_j)$$



Worked example: 3 toy particles, d_k = 2

Imagine we have only 3 particles and 2-dimensional embeddings.

Embedded particles X (3 × 2):

```
X = [[1, 0],
     [0, 1],
     [1, 1]]
```

Pretend the W matrices are simple:

```
W_Q = I        (identity)
W_K = I        (identity)
W_V = [[1, 2],
       [3, 4]]
```

So we get:

Q (3 × 2)

	q1	q2
p1	1.00	0.00
p2	0.00	1.00
p3	1.00	1.00

K (3 × 2)

	k1	k2
p1	1.00	0.00
p2	0.00	1.00
p3	1.00	1.00

V (3 × 2)

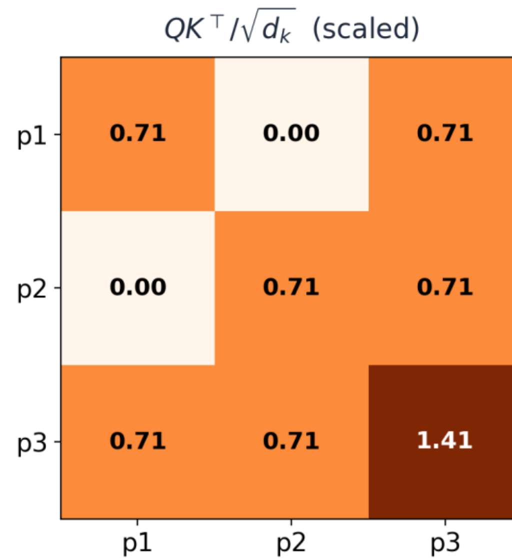
	v1	v2
p1	1.00	2.00
p2	3.00	4.00
p3	4.00	6.00

Step 1+2: compute scores, then scale

Take all-pairs dot products $Q \cdot K^T$:

Row i = 'how compatible is particle i with each particle j?'

Now divide by √d_k = √2 ≈ 1.41:

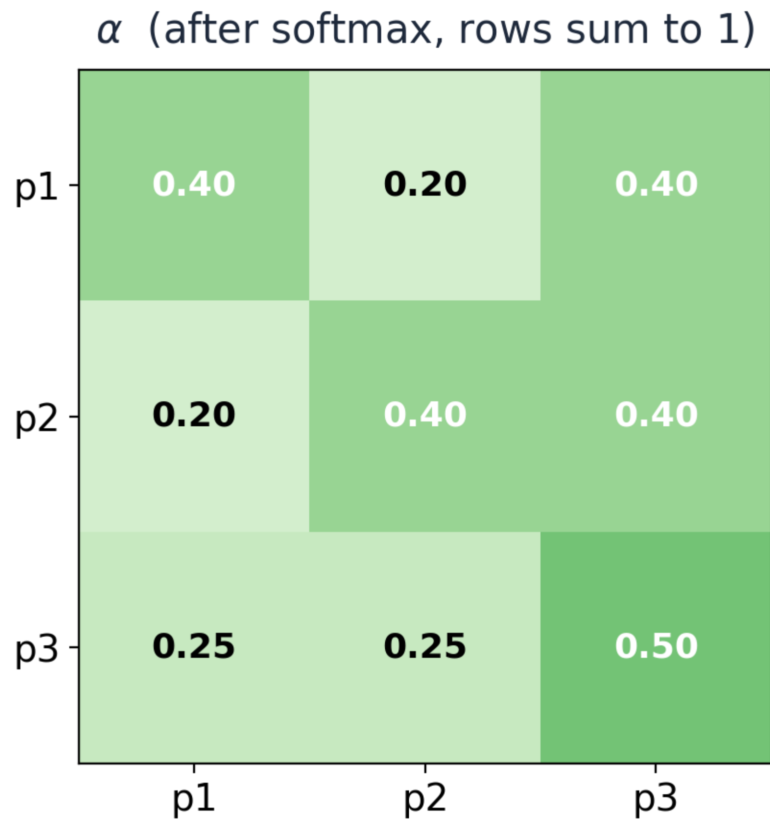


Why scale?

Without it, dot products grow as d_k grows, pushing softmax into very peaky regions and killing gradients.

Step 3: softmax → attention weights

Take softmax along each row: exponentiate, then divide by the row sum.



Row 1 (particle 1):

$\alpha_1 = (0.40, 0.20, 0.40)$
Particle 1 attends equally to itself and to particle 3, half as much to particle 2.

Two properties of every row:

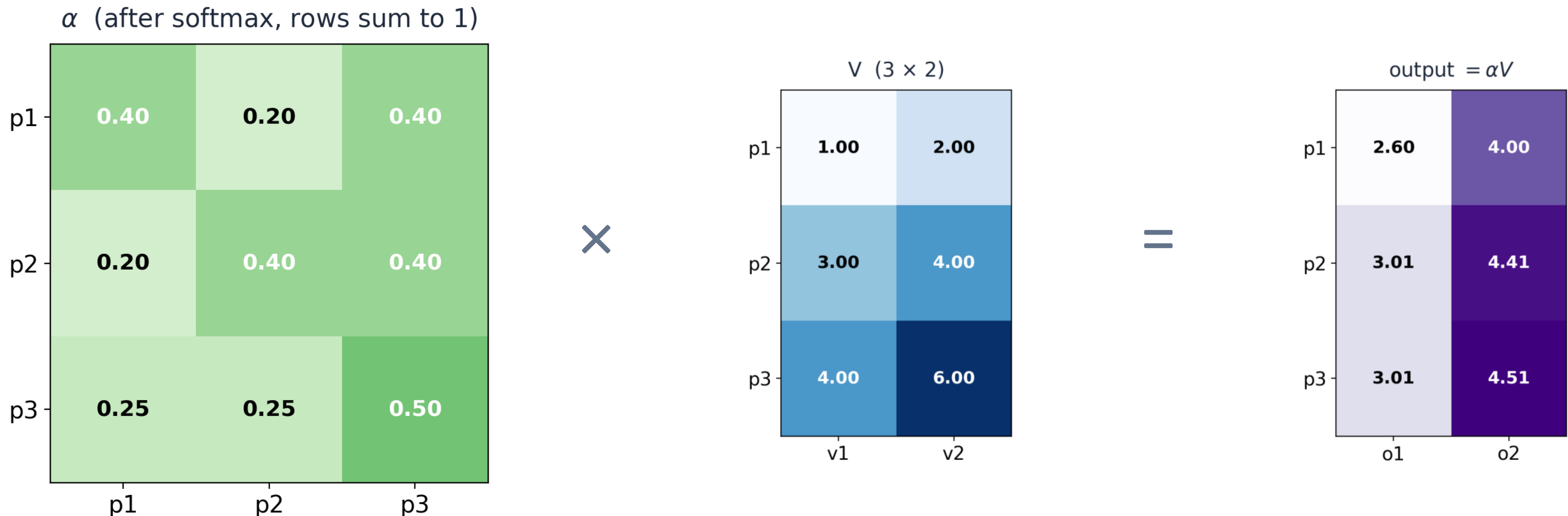
- All entries ≥ 0
- Entries sum to exactly 1 → *this is a property of softmax!*

↔ **Compare with LLMs:** in GPT, masking forces these rows to attend only to earlier tokens. For jets we keep the full matrix.

Step 4: multiply by V → output

This is fed back into the next layer as a new representation of the input! Original input is added to this as "residual"

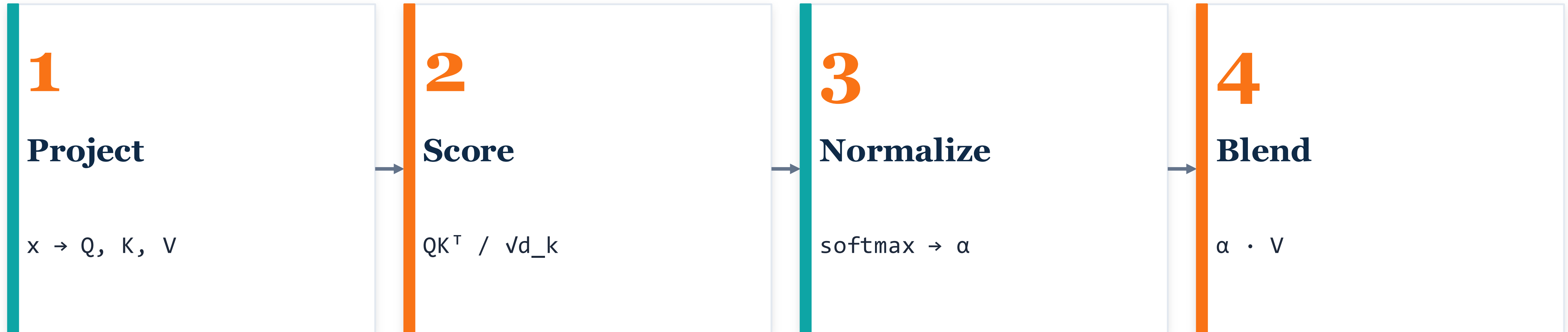
out = α · V each new particle = weighted blend of all V vectors.



Read row 1 of the output: $out_1 = 0.40\cdot(1,2) + 0.20\cdot(3,4) + 0.40\cdot(4,6) = (2.60,\ 4.00)$ ✓

FOUR STEPS

Self-attention recap



Three properties to remember

- **It is parallel:** every output is computed independently from a matrix product — fits a GPU beautifully.
- **It is global:** every particle can attend to every other in a single layer — no propagation needed.
- **It is permutation-equivariant:** shuffle the input rows, you get the same outputs in a shuffled order.

BUILDING BLOCK

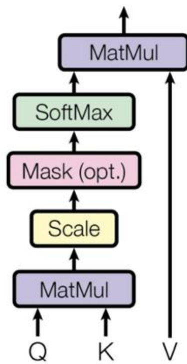
Masking

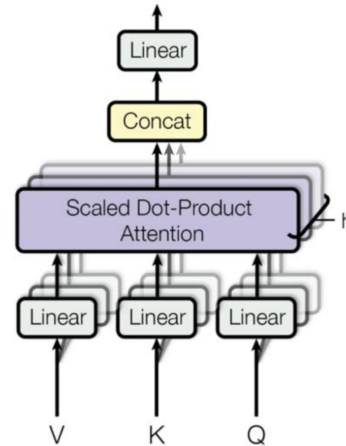
- Most jets have < 128 particles. We pad with zeros to make a fixed-size tensor.
- Real particles must not attend to those padded zeros, since they are meaningless.
- Trick: set the scores at padded positions to $-\infty$ before the softmax.
- After softmax, $e^{(-\infty)} = 0$, so the padded positions get zero attention weight.
- Transformers can take variable sequence lengths this way!

```
scores = scores.masked_fill(~mask, float('-inf'))
```

↔ **Compare with LLMs:** GPT-style decoders use the same trick for a different reason: set future tokens to $-\infty$ so the model can only look back, not ahead.

Multi-head attention



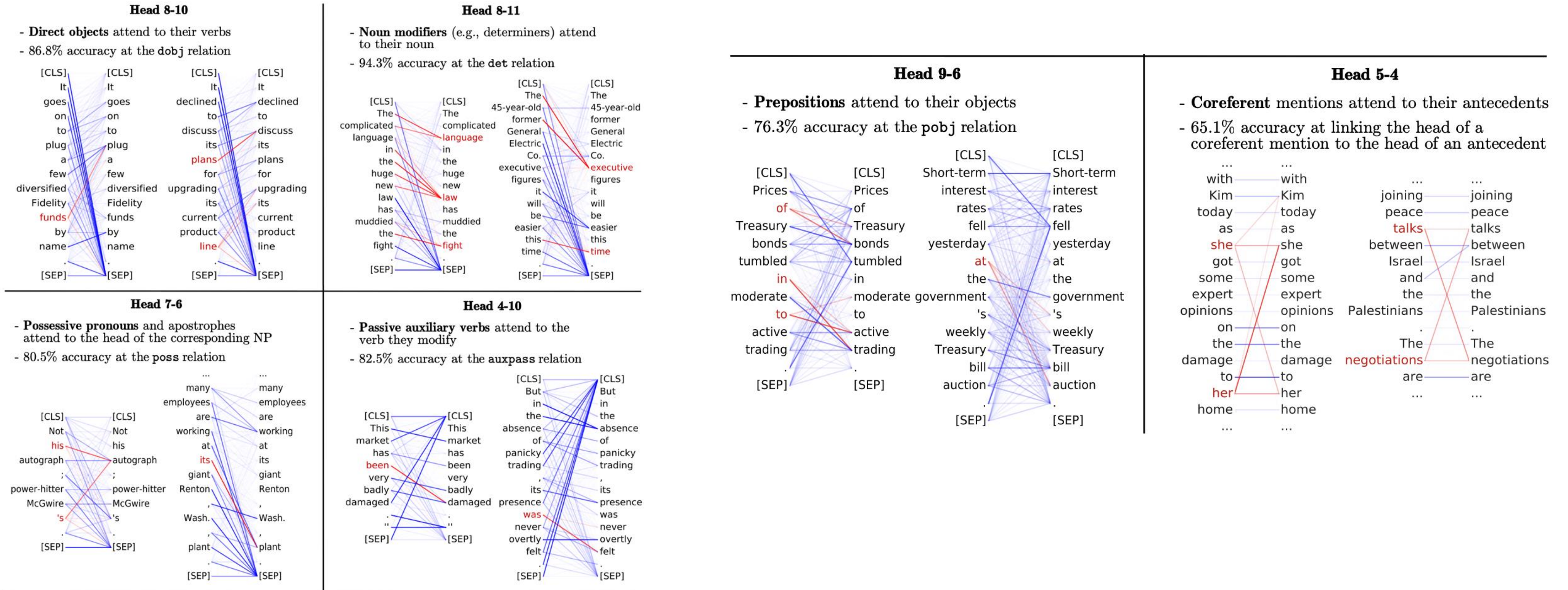


- A single head learns one notion of ‘compatible particles’. Run h heads in parallel on the same input. Each head has its own Q, K and V projections and learns its own pattern.
- Splits the d embeddings into h equal slices (e.g. h heads × d/h dims), run attention on each slice, concatenate, apply one final linear embedding
- Splits the data along the feature embedding dimension! Each head can look at a different section of data

Why use multiple heads?

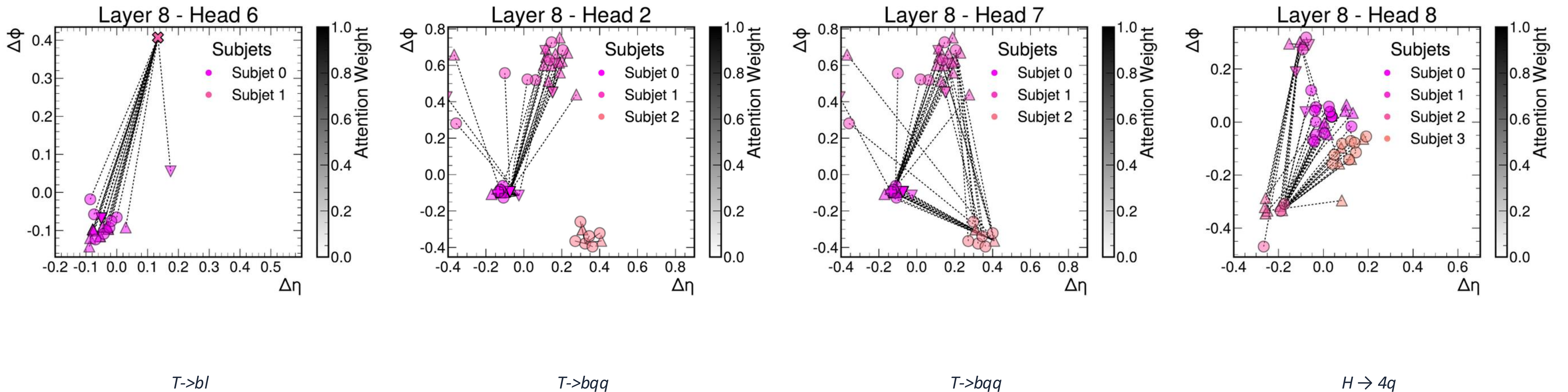
One attention map can only express one ‘way to compare two particles’. Different heads can pick up different relationships

BERT attention heads



Clark et al. (2019) probed every head in pre-trained BERT. Heads pick up specific linguistic patterns. No one told the model these categories — it found them on its own!

ParT attention heads



T->bl *T->bqq* *T->bqq* *H → 4q*

Same idea as BERT heads. If language heads learn syntax (direct object, preposition object), particle heads seem to learn some **substructure**: leptons, single subjets, multi-prong groups. Notebook 3 runs ParT_full and reproduces these patterns from the pre-trained weights.

PUTTING IT TOGETHER

A transformer block

```
x → LayerNorm → Multi-Head Attention → +residual → LayerNorm → FFN → +residual → out
```

- Inputs and outputs have the same shape: (128 particles, 32 dims). So we can stack as many as we want.
- Each block lets the particles share a bit more information.

BUILDING BLOCK

Transformer Block

```
x → LayerNorm → Multi-Head Attention → +residual → LayerNorm → FFN → +residual → out
```

Feed-forward network

Two linear layers (d → 4d → d) applied independently to each particle. Adds non-linear capacity AFTER attention has mixed information across particles.

LayerNorm

Re-center each particle's feature vector to zero mean and unit variance. Keeps the numbers in a healthy range so training is stable.

Residual connections

output = input + sublayer(input). The skip path gives gradients a direct route through deep networks. Adds original representation onto the attention representation.

↔ **Compare with LLMs:** the same three components appear in every modern LLM: FFN, LayerNorm, residual.

REFERENCE

Function definitions

softmax

turns a vector of real numbers into a probability distribution (positive, sums to 1).

$$\mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

GELU

nonlinear activation inside the feed-forward network. Smooth version of ReLU; multiplies the input by the standard-normal CDF Φ.

$$\mathrm{GELU}(x) = x\,\Phi(x), \qquad \Phi = \text{CDF of } N(0,1)$$

LayerNorm

re-centers each particle's feature vector to zero mean and unit variance, then re-scales with learnable γ, β. Keeps activations in a healthy range.

$$\mathrm{LayerNorm}(x)_i = \gamma_i \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta_i$$

Residual connection

```
y = x + f(x)
```

the output of every sub-layer adds onto its input, so gradients can flow through entire deep neural netowrk.

PUTTING IT TOGETHER

Pooling: particles → one jet vector

- After the blocks: shape is still (128 particles, 32 dims). We need ONE 10-class prediction per jet.
- Pooling: collapse the 128 particle vectors into a single 32-d vector.
- We use a MASKED MEAN — average only over real particles, not the padded zeros.
- Other methods of pooling including CLS token attention! Anything that is able to pool information together into the correct shape!

```
jet_vec = (x · mask).sum(dim=1) / mask.sum(dim=1)
logits  = Linear(32 → 10)(jet_vec)
```

Hands-on: Notebook 1

- 4 small exercises!.
- Solutions are shown right after each exercise — try it yourself first!!!!!!
- Plot attention weights at the end and look for structure.

Goal during the tutorial:

Be able to write the attention formula from scratch and read an attention-weight plot.

PART

2

What does attention actually cost?

Notebook 2 • Attention is $O(n^2)$

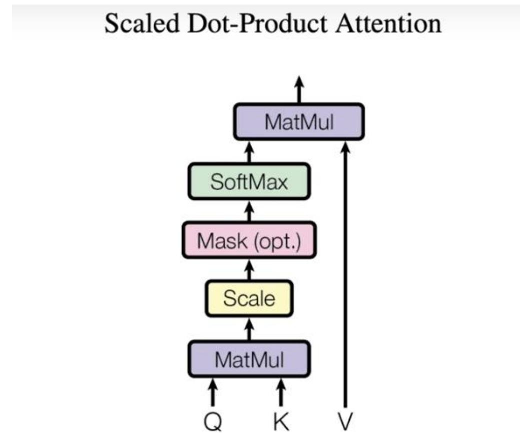
THE PROBLEM

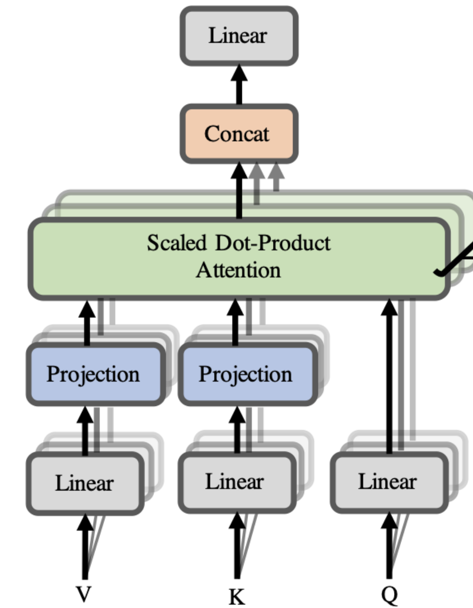
Attention is quadratic

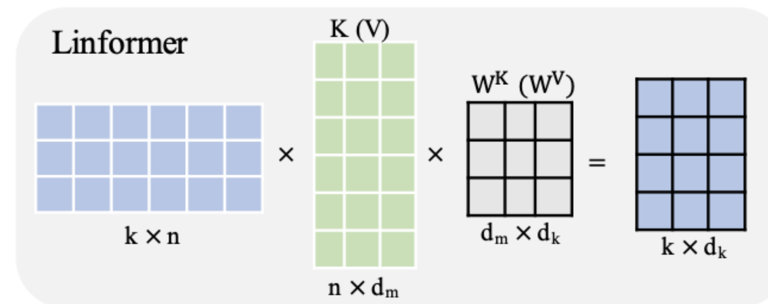
- Attention computes an $n \times n$ matrix (every particle vs every other particle).
- Double $n \rightarrow 4\times$ the cost. Triple $n \rightarrow 9\times$ the cost.
- Very compute heavy for long sequences!

Setting	n	n^2 entries	Verdict
Jets	128	16,384	Fine on GPUs
LLM context (GPT-4)	128,000	≈ 16 billion	Needs serious engineering

Linformer







Linformer is one of several tricks that can be used to keep attention affordable.

TWO EXTRA PROJECTION MATRICES

Linformer

$E, F \in \mathbb{R}^{k \times n}$ are learnable, fixed-shape. We pick k = 32 << n = 128.

```
Standard attention:   scores = Q · Kᵀ                       ← n × n
Linformer:            K' = E · K   (k × d)              ← compress first
                      V' = F · V   (k × d)
                      scores = Q · K'ᵀ                    ← n × k
```

	Standard	Linformer (k=32, n=128)
Attn matrix size	n × n = 16,384	n × k = 4,096
Complexity	$O(n^2)$	$O(n\ k)$
Memory	$O(n^2)$ (unless with flash attention)	$O(n\ k)$
Speedup at n=128	—	≈ 4×

More speedup the bigger n is than k!

Efficient attention methods

Method	Time	Memory	Idea
Standard	$O(n^2)$	$O(n^2)$	Full attention matrix
Flash Attention	$O(n^2)$	$O(n)$	IO-aware tiling — EXACT ATTENTION, with special GPU implementation
Linformer	$O(n\ k)$	$O(n\ k)$	Project K, V to k dimensions
Performer	$O(n)$	$O(n)$	Approximate softmax with random features
Sparse attention	$O(n\sqrt{n})$	$O(n\sqrt{n})$	Only attend to selected pairs

↔ **Compare with LLMs:** Flash Attention is what makes 1-million-token LLMs possible. Whenever you see Anthropic, OpenAI, or Google publish a long-context model, an efficient-attention method is doing the heavy lifting underneath. Linear memory growth with sequence length!!!

EXACT ATTENTION, HARDWARE-AWARE

FlashAttention

- FlashAttention computes the exact same $\mathrm{softmax}(QK^T/\sqrt{d})V$
- Where the $n \times n$ matrix lives matters. Standard PyTorch writes it to slow GPU HBM memory, then reads it back. Memory traffic dominates the runtime,
- FlashAttention tiles Q, K, V into blocks, keeps the partial softmax inside fast on-chip SRAM, and never materializes the full $n \times n$ matrix anywhere.
- Result: identical output, 2–10× faster, and dramatically lower memory use.

↔ **Compare with LLMs:** this is what makes 200K-token Claude and 1M-token Gemini possible. The model is the same, the attention kernel is FlashAttention 2/3.

In PyTorch: torch.nn.functional.scaled_dot_product_attention picks the FlashAttention kernel on modern GPUs.

Performer and sparse attention

Performer

- Rewrites softmax(QK^T) as a kernel $K(Q, K) \approx \phi(Q) \cdot \phi(K)^T$ with random feature maps.
- Instead of computing $(Q K^T) V$, you compute $Q (K^T V)$.
- $(K^T V)$ is only $d \times d$ — never $n \times n$.
- Linear in n, in both time and memory. Pays an accuracy cost for a big speedup at large n.

Sparse / local / windowed attention

- Decide a priori that most pairs shouldn't attend to each other. Only compute scores for the kept pairs.
- Local windows (Longformer): each token attends to $\pm w$ neighbors.
- Strided / block patterns (Sparse Transformer, BigBird).
- For instance, one might do in physics: 'only attend within $\Delta R < 0.4$' is a natural choice for collimated jets.

WHERE N ≈ 100 STILL HURTS

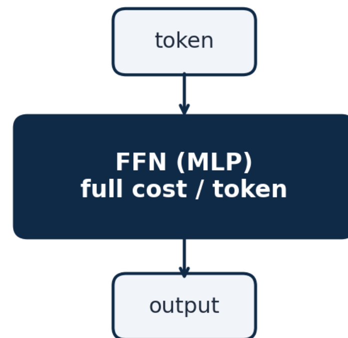
Attention on Trigger!

- L1 trigger: collisions arrive at 40 MHz. The hardware has to decide whether to keep the event in a few microseconds, on FPGA.
- At those budgets, $O(n^2)$ attention for $n \approx 100$ is already painful — not just because of memory, but because of arithmetic depth and pipeline length on the chip.
- Recent work pushes linear-attention transformers (Linformer-style with physics-aware partitioning) and patch/hierarchical attention to make attention more efficient.

Mixture of Experts

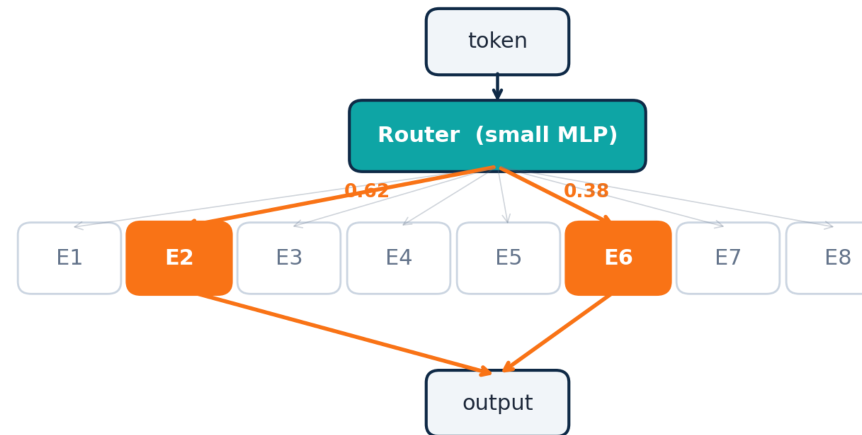
Standard FFN

every token goes through one big MLP



Mixture-of-Experts FFN

router picks top-k of N experts; only those run



weighted sum of E2 and E6 outputs (top-k=2 of N=8)

- Replaces the FFN with N ‘expert’ MLPs and a small router. For each token, the router picks the top-k experts and the token only passes through those.
- Parameter count scales with N, compute per token with k. Trades wider capacity against constant inference cost.
- Experts can specialize on specific topics!

↔ **Compare with LLMs:** MoE is keeps same compute level with more parameters. Many production LLMs use both: FlashAttention for attention, MoE for the FFN.

Hands-on: Notebook 2

- 1 exercise: fill in the three Linformer lines ($E \cdot K$, $F \cdot V$, $Q \cdot K'^T$).
- See how fast linformer is compared to transformer
- Is there any loss in performance?

PART

3

The Particle Transformer

Notebook 3 • putting physics into the architecture

MOTIVATION FOR PART

The plain transformer's blind spot

- Vanilla attention sees only learned embeddings of single particles. It has to discover from scratch:
- There is no physics embedded inside the model
- We already know there are some pairwise features that can be calculated between particles. Why not just directly TELL the model?

The Particle Transformer's modification

Add Lorentz-invariant pairwise physics features to the attention computation as an additive bias on top of the model's learned scores.

ADD A BIAS U_IJ

ParT attention formula

$$\Delta = \sqrt{(y_a - y_b)^2 + (\phi_a - \phi_b)^2},$$
$$k_T = \min(p_{T,a}, p_{T,b})\Delta,$$
$$z = \min(p_{T,a}, p_{T,b})/(p_{T,a} + p_{T,b}),$$
$$m^2 = (E_a + E_b)^2 - \|\mathbf{p}_a + \mathbf{p}_b\|^2,$$

Plain transformer:

α_{ij} = softmax(Q_i · K_j / √d_k)

Particle Transformer:

$$\alpha_{ij} = \mathrm{softmax}\left(\frac{Q_i \cdot K_j}{\sqrt{d_k}} + U_{ij}\right)$$

U_ij is computed by a small MLP from FOUR pairwise physics features:

$$U_{ij} = \mathrm{MLP}\left(\Delta R_{ij},\ k_{T,ij},\ z_{ij},\ m_{ij}^2\right)$$

MECHANICS

Pair embedding → attention bias

- For each pair (i, j) we have four numbers: (ΔR, k_T, z, m²).
- Feed them into a small MLP that outputs ONE bias per attention head.
- Add that bias to the raw QK scores inside the softmax.
- Result: attention is informed by physics from the start — no need to learn it from scratch.

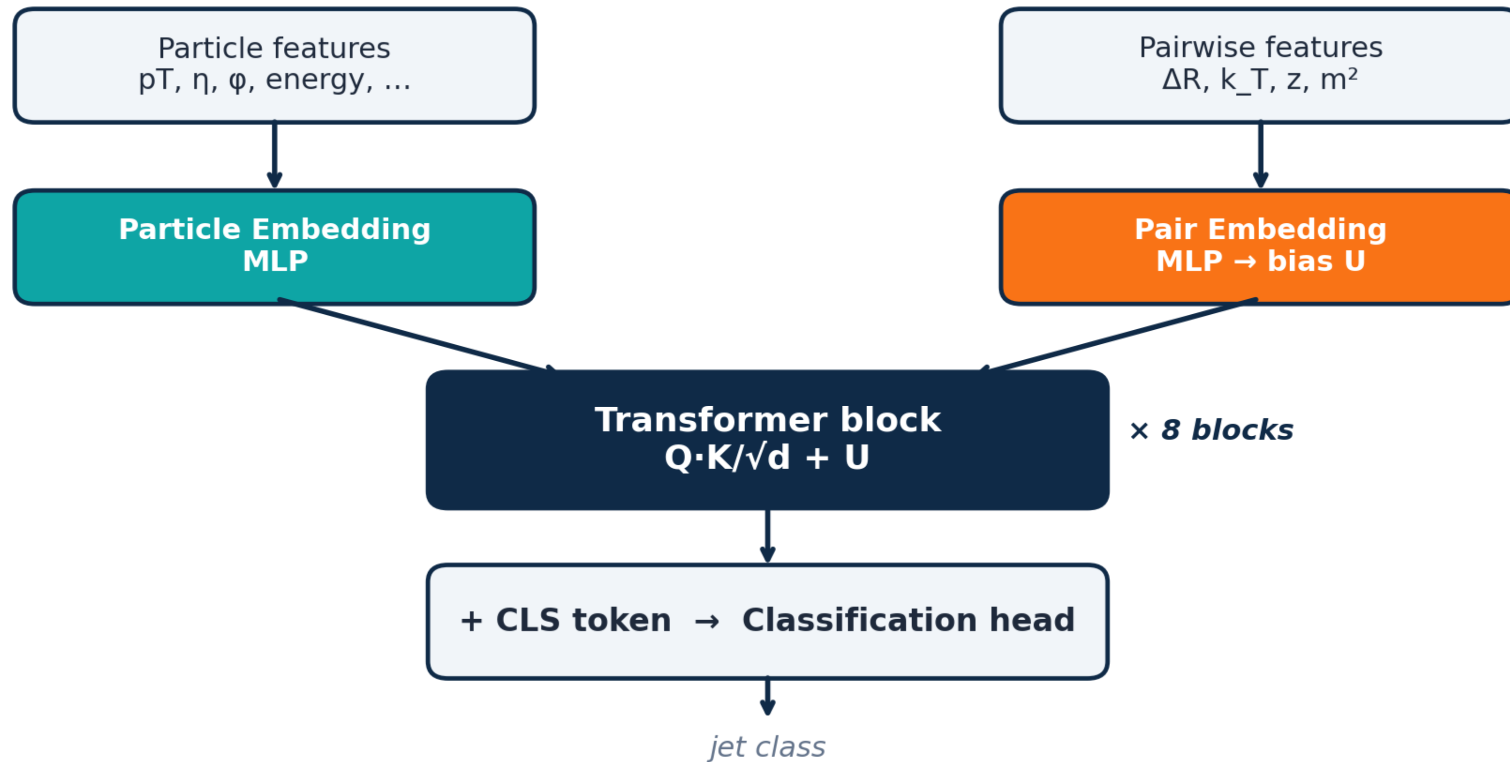
$$\begin{aligned}
\Delta &= \sqrt{(y_a - y_b)^2 + (\phi_a - \phi_b)^2}, \\
k_{\mathrm{T}} &= \min(p_{\mathrm{T},a}, p_{\mathrm{T},b})\Delta, \\
z &= \min(p_{\mathrm{T},a}, p_{\mathrm{T},b})/(p_{\mathrm{T},a} + p_{\mathrm{T},b}), \\
m^2 &= (E_a + E_b)^2 - \|\mathbf{p}_a + \mathbf{p}_b\|^2,
\end{aligned}$$

Three properties of this design

- **ADDITIVE —** the model can still learn whatever it wants from QK; U is just a head start.
- **PERMUTATION-EQUIVARIANT —** U_ij depends on particle kinematics, not order.
- **LORENTZ-INVARIANT —** the four features are Lorentz-invariant by construction.

ARCHITECTURE

The full ParT pipeline



Same skeleton as Notebook 1, with two additions: pairwise bias U and a CLS token.

Mean pool vs CLS token

Mean pool (Notebook 1)

- Average all particle vectors.
- Treats every particle equally in the final summary.
- Permutation-invariant by construction.
- Simple — works very well for small models.

Class token (CLS) — ParT

- Add ONE learnable ‘virtual particle’ to the sequence in the last 2 layers.
- Attention lets it pool selectively (not uniformly).
- Final CLS representation → classifier head.

↔ **Compare with LLMs:** the CLS token is borrowed from BERT — it's the canonical ‘sentence summary’ slot in encoder-only LLMs.

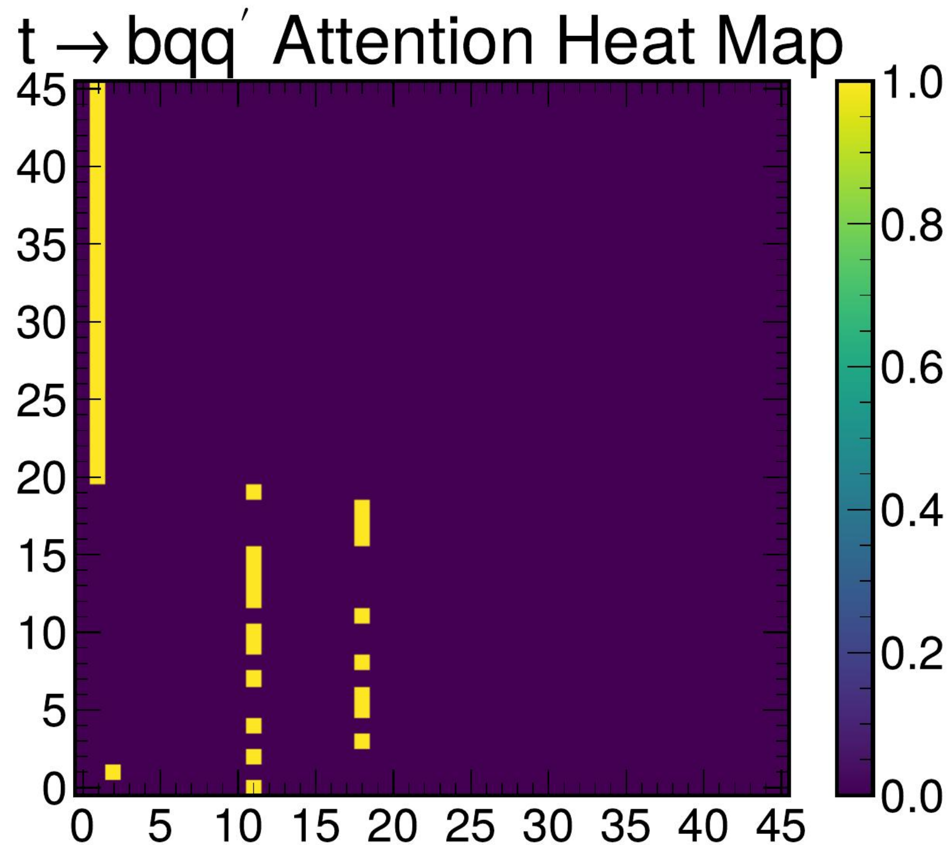
PARAMETER COUNTS

Model sizes

Model	Parameters
Our toy model (NB 1, NB 2)	≈ 30,000 parameters
ParT_kin (kinematics only)	≈ 2.1 million
ParT_full (kinematics + particle ID)	≈ 2.2 million
GPT-2 small	≈ 124 million
GPT-4 (rumored)	≈ 1+ trillion

Same architecture at all scales. Today's tutorial builds the smallest version end-to-end.

Sparse attention



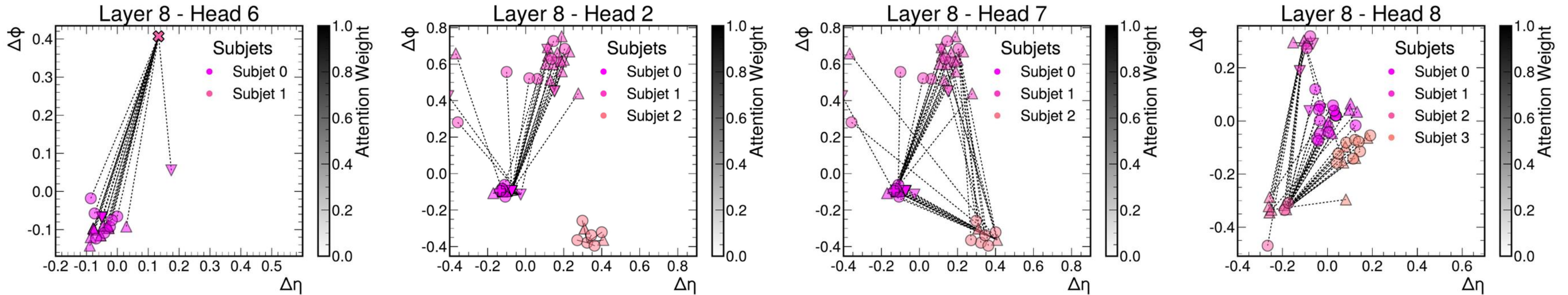
Finding 1

Attention becomes nearly binary.

- The softmax could spread weight across all pairs.
- Trained ParT concentrates each row of α on roughly one partner. The other entries are close to zero.
- Nothing in the architecture forces this. The model learns it from the training loss.
- In the notebook, we read the attention from the pre-trained ParT_full weights and see exactly this.

Substructure!

- The attention edges concentrate within prongs — particles from the same subjet, not across the jet.
- H → 4q jet: edges cluster around the four prongs.
- Top → bqq jet: three-prong structure shows up.
- Top → bℓν jet: the lepton and the b-quark prong stand out.



CONNECTION

ParT vs. traditional observables

ParT discovers …	Traditional observable	What it measures
Binary attention clusters	N-subjettiness τ_N	How N-pronged is the jet?
Pairwise momentum correlations	Energy correlation functions e_N	N-point momentum correlations
Subjet attention groups	Jet mass / soft-drop mass	Mass of the (groomed) jet
Hard-splitting identification	Splitting scales $\sqrt{d_{12}}$	Scale of the hardest splitting

ParT computes related quantities inside attention, then combines them across heads and layers. There is no need to design each observable by hand.

10-CLASS ACCURACY

Particle Transformer benchmark

Model	Accuracy	Approach
PFN (Deep Sets)	77.2 %	Set function, no particle interactions
P-CNN	80.9 %	Jet-image convolutions
ParticleNet	84.4 %	Graph network with edge convolutions
Plain transformer	84.9 %	Standard self-attention (your NB1 model, scaled up)
Particle Transformer	**86.1 %**	**Self-attention + physics interaction bias**

The performance gap from plain transformer to ParT is from the pairwise physics features alone. Further more performant models such as LGATR is based upon the particle transformer backbone.

Hands-on: Notebook 3

- Exercise 1: compute pairwise ΔR for a jet
- Exercise 2: add the interaction bias to attention scores (one line).
- Plot ΔR, k_T, z, m^2 matrices for a QCD jet next to an H→bb jet. The substructure is visible directly.
- Inspect the pre-trained ParT_full weights — embedding dim, heads, pair-embedding shape, CLS token.
- Run ParT_full on H→4q, top→bqq, top→bℓv. Plot attention plot.

Goal: leave the notebook able to write the ParT modification on a whiteboard and explain what each pairwise feature does.

DEFINITION

Foundation models

A foundation model is a single model trained on a large amount of data (oftentimes unlabeled), then adapted to many downstream tasks.

- Trained once (expensive), used many times. The downstream user fine-tunes on a small labeled sample, or swaps the head, or freezes the encoder.
- The base model architecture is almost always a transformer.
- Can also be multimodal!

Examples in language: BERT, GPT, T5, Llama, Claude. Examples in vision: ViT, CLIP, SAM. Biology: AlphaFold, ESM.

Foundation models in language

BERT-style (masked language modeling)

Devlin et al., 2018

Mask a percentage of input tokens. Predict the missing tokens from the surrounding ones.

```
"The cat ___ on the mat"  →  "sat"
```

- Encoder-only transformer.
- Good for classification, retrieval, embeddings.
- Fine-tune by adding a small task-specific head.

GPT-style (next-token prediction)

Radford et al., 2018+

Predict the next token from the previous ones. Repeat the whole way through every sentence in the corpus.

```
"The cat sat on the"  →  "mat"
```

- Decoder-only transformer (causal mask).
- Good for generation, completion, chat.
- Modern LLMs (GPT-4, Claude, Gemini) all in this family.

Foundation models for jets

Replace tokens with particles. Replace word-piece IDs with (pT, η, φ, ...). Run the same self-supervised objectives on millions of unlabeled jets.

OmniJet-α (Birk, Hallin, Kasieczka, 2024)

arXiv:2403.05618

GPT-style. Tokenize each particle, train a decoder-only transformer to predict the next particle on the JetClass corpus (~100M jets). Then fine-tune the same backbone for tagging.

Masked Particle Modeling (Heinrich, Lavoie, Plehn et al., 2024)

arXiv:2401.13537

BERT-style. Mask a percentage of the particles in a jet, predict their kinematics from the rest. Fine-tune for top tagging, anomaly detection.

JetCLR (Dillon, Kasieczka, Olischlager et al., 2021)

arXiv:2108.04253

Contrastive pre-training. Augment each jet (rotations, soft drops), train the encoder so two views of the same jet land near each other in embedding space.

SUMMARY

Recap

1

Attention is a weighted average

$\mathrm{softmax}(QK^T/\sqrt{d}) \cdot V$. Each particle decides which other particles to listen to, then blends their value vectors accordingly.

2

Jets are sets!

The model then treats jets as unordered sets

3

Physics features can be added as an attention bias

ParT adds Lorentz-invariant pairwise quantities (ΔR, k_T, z, m^2) as a bias term in the attention scores. The jet-tagging accuracy goes up.

Where to next

If you want to run ParT for real	github.com/jet-universe/particle_transformer (training scripts, pretrained weights, weaver-core)
If you want to interpret transformer attention	Wang et al. 2024 • github.com/aaronw5/Interpreting-Particle-Transformers
If you want to play with HEP foundation models	OmniJet-α • Masked Particle Modeling • JetCLR (all open-source on GitHub)
If you want to learn the LLM side	Karpathy's ‘Let's build GPT’ video • the original ‘Attention Is All You Need’ (arXiv:1706.03762)

FINAL SLIDE

References & thanks

- Vaswani et al., 'Attention Is All You Need'. NeurIPS 2017. arXiv:1706.03762
- Clark, Khandelwal, Levy, Manning, 'What Does BERT Look At?' BlackboxNLP 2019. arXiv:1906.04341
- Qu, Li, Qian, 'Particle Transformer for Jet Tagging'. ICML 2022. arXiv:2202.03772
- Wang et al., 'Interpreting Transformers for Jet Tagging'. NeurIPS ML4PS 2024. arXiv:2412.03673
- Wang et al., 'Linformer: Self-Attention with Linear Complexity'. 2020. arXiv:2006.04768
- Dao et al., 'FlashAttention: Fast and Memory-Efficient Exact Attention'. 2022. arXiv:2205.14135
- Choromanski et al., 'Performer: Rethinking Attention with Performers'. 2020. arXiv:2009.14794
- Birk, Hallin, Kasieczka, 'OmniJet-α: A Foundation Model for Jets'. 2024. arXiv:2403.05618
- Heinrich, Krause, Plehn et al., 'Masked Particle Modeling on Sets'. 2024. arXiv:2401.13537
- Qu, Gouskos, 'ParticleNet: Jet Tagging via Particle Clouds'. 2020. arXiv:1902.08570
- Devlin et al., 'BERT'. 2019. arXiv:1810.04805

Thanks to the ML4FP 2026 organizers, the JetClass authors, and you for sticking with it.

Questions?