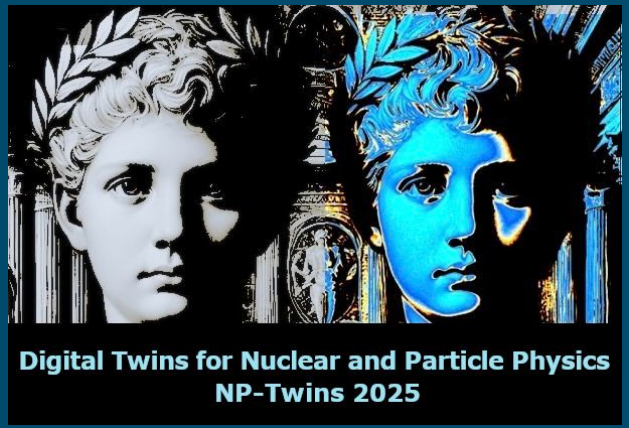
Faculty of Physics
Warsaw University of Technology

Towards more precise correlation studies with machine learning-based particle identification with missing data



Łukasz Graczykowski, Marek Mytkowski
in collaboration with
M. Janik, M. Karwowska, S. Monira, K. Deja, M. Kasak, M. Jakubowska, M. Olędzki

Messina, Italy
6 October 2025

Based on:
EPJ C 84 (2024) 7, 691
JINST 19 (2024) 07, C07013

Goals

- Use **ALICE** and its data as a **unique environment** for **Machine Learning (ML) research**

- Identify **areas** where both ALICE (or HEP in general) and ML communities can **mutually benefit** from each other

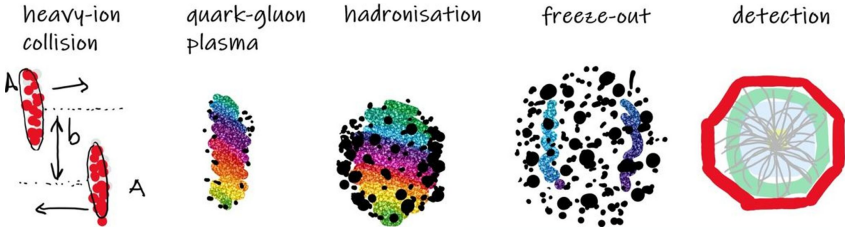
- Our solutions should be **easily applicable to other experiments** with similar capabilities

- **Disclaimer:**
  - I'm a **physicist without a big ML background** – few years ago I started my (human) learning of machine learning :)
  - My task is to **guide and coordinate the work of WUT ML computer scientists** within ALICE
  - The solution may be **complicated** from a physicist perspective, but the balance is to keep the project interesting for ML itself and be useful for us at the same time!

QGP, HI collisions and dedicated experiments

Heavy-Ion collisions are used to create, for a brief moment, a deconfined state of matter - the **Quark-Gluon Plasma (QGP)**

QGP studies require **dedicated experiments**

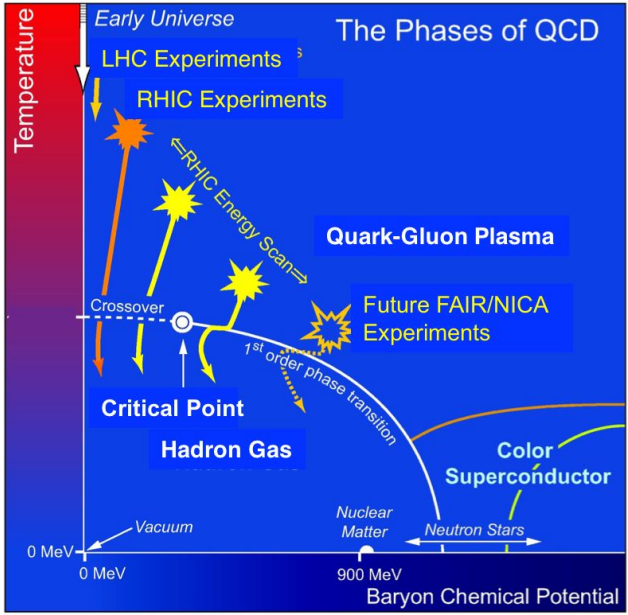


ALICE Collaboration Phys. Rev. C 101, 044907

In operation: ALICE@LHC, STAR@RHIC, NA61@SPS
in future: CBM@FAIR, MPD@NICA

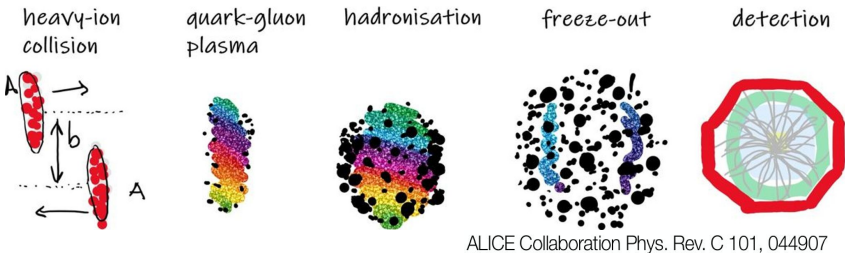
Common feature: Particle Identification (PID)

- QGP is a *bulk* phenomenon (low to intermediate-pT Particles; particle ratios, collective flow, etc.)
- possibility to identify particles in wide momentum range (down to ~100 MeV/c)
- π, K, p, $e^{\pm}$, $\mu^{\pm}$, deuterons, tritons, ^{3}He, ^{4}He strange and charm hadrons



QGP, HI collisions and dedicated experiments

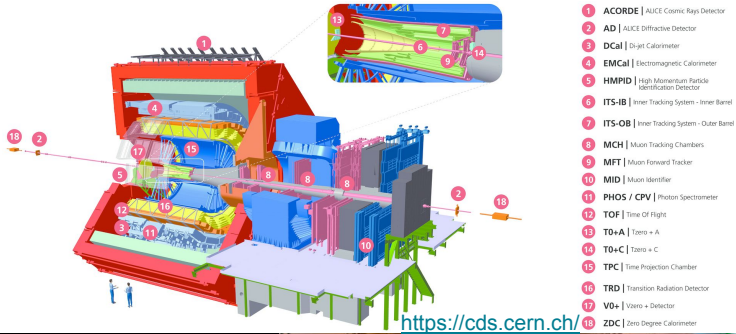
Heavy-Ion collisions are used to create, for a brief moment, a deconfined state of matter - the **Quark-Gluon Plasma (QGP)**



ALICE Collaboration Phys. Rev. C 101, 044907

QGP studies require **dedicated experiments**

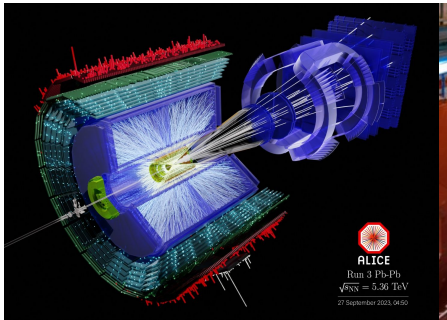
In operation: ALICE@LHC, STAR@RHIC, NA61@SPS
in future: CBM@FAIR, MPD@NICA

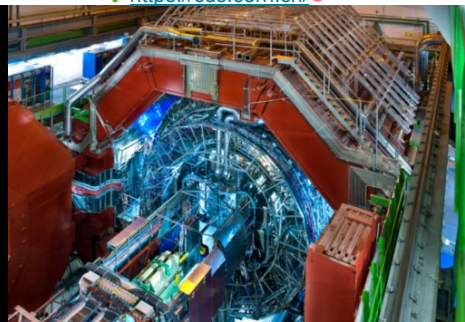


https://cds.cern.ch/

Common feature: Particle Identification (PID)

- QGP is a *bulk* phenomenon (low to intermediate-pT Particles; particle ratios, collective flow, etc.)
- possibility to identify particles in wide momentum range (down to ~100 MeV/c)
- π, K, p, $e^{\pm}$, $\mu^{\pm}$, deuterons, tritons, ^{3}He, ^{4}He strange and charm hadrons

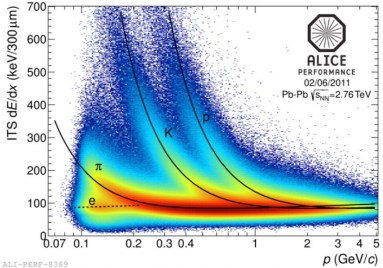


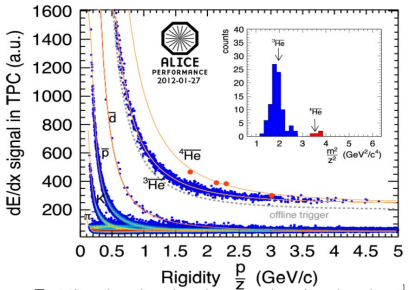


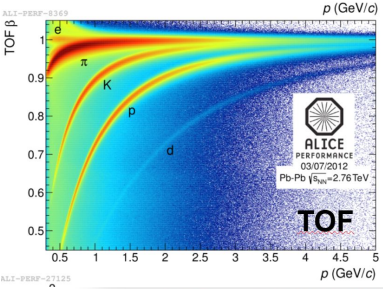
Particle identification (PID)

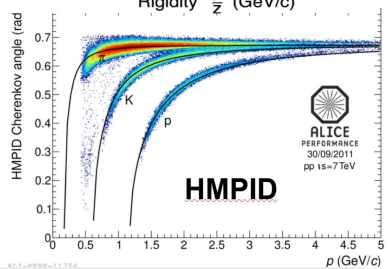
Aim: provide high purity samples of particles of a given type

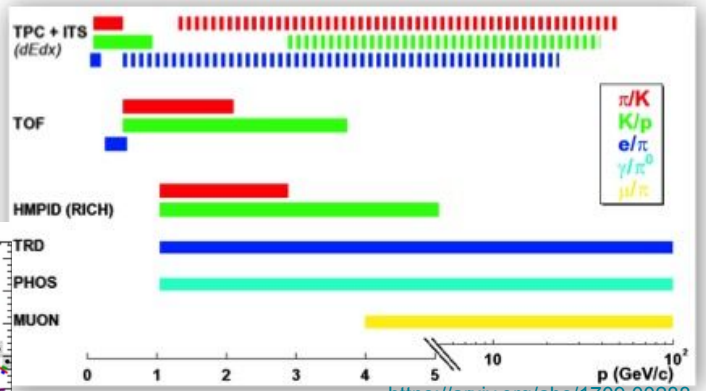
- **an essential step** for many physics analyses, especially **correlations of identified particles**
- we use **ALICE as our R&D environment**
- **PID is a distinguishing feature of ALICE**
  - identification of particles of momenta in a **very wide momentum range**
  - practically **all known PID techniques** employed: dE/dx energy loss, time-of-flight, Cherenkov radiation and transition radiation



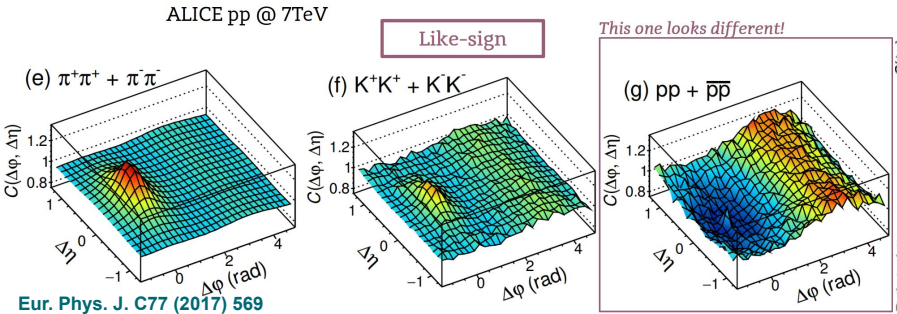




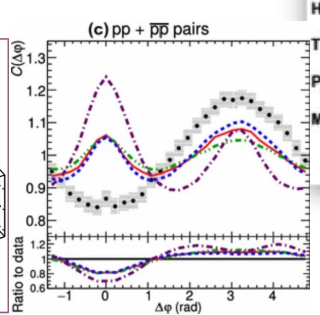




https://arxiv.org/abs/1709.00288



Eur. Phys. J. C77 (2017) 569

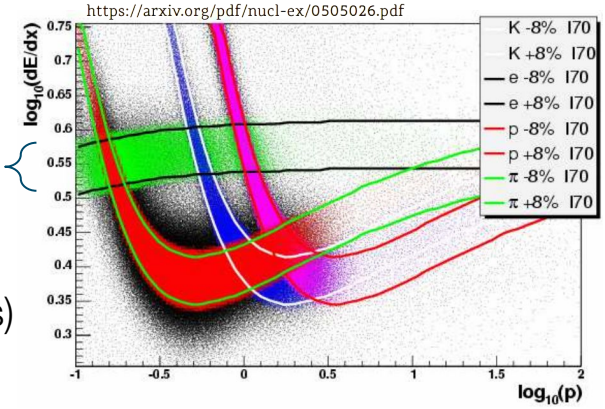


ALICE pp $\sqrt{s}$ = 7 TeV, $|\Delta\eta| < 1.3$
PYTHIA6 Perugia-0
PYTHIA6 Perugia-2011
PYTHIA8 Monash
PHOJET

Present state-of-art

1. **Traditional method:**
   - hand-crafted selections of selected quantities, e.g., **nσ**
   - <u>problems:</u>
     - overlapping signals
     - <u>high purity at the cost of low efficiency</u>
     - time-consuming optimization (where the signals cross)

https://arxiv.org/pdf/nucl-ex/0505026.pdf



- **Metrics**
  - **Purity (*precision*)** and **efficiency (*recall*)** calculated from MC simulated data with full detector response (anchored to the specific data collection period = run)
    - normally measured as a function of transverse momentum p_T

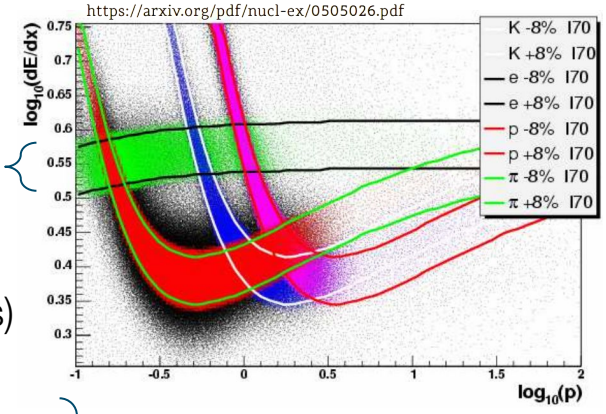
$$\text{Efficiency} = \frac{N_{\text{true positives}}}{N_{\text{true particles}}}$$

$$\text{Purity} = \frac{N_{\text{true positives}}}{N_{\text{true positives}} + N_{\text{false positives}}}$$

Present state-of-art

1. **Traditional method:**
   - hand-crafted selections of selected quantities, e.g., **nσ**
   - problems:
     - overlapping signals
     - high purity at the cost of low efficiency
     - time-consuming optimization (where the signals cross)

https://arxiv.org/pdf/nucl-ex/0505026.pdf



2. **Bayesian method** (ALICE, EPJ Plus 131 (2016) 168)**:**
   - updating probability of an hypothesis with each new evidence
   - priors = best guess of true particle yields per events
   - posteriors ~ purity of a given particle species
   - increased purity, results consistent with the traditional method

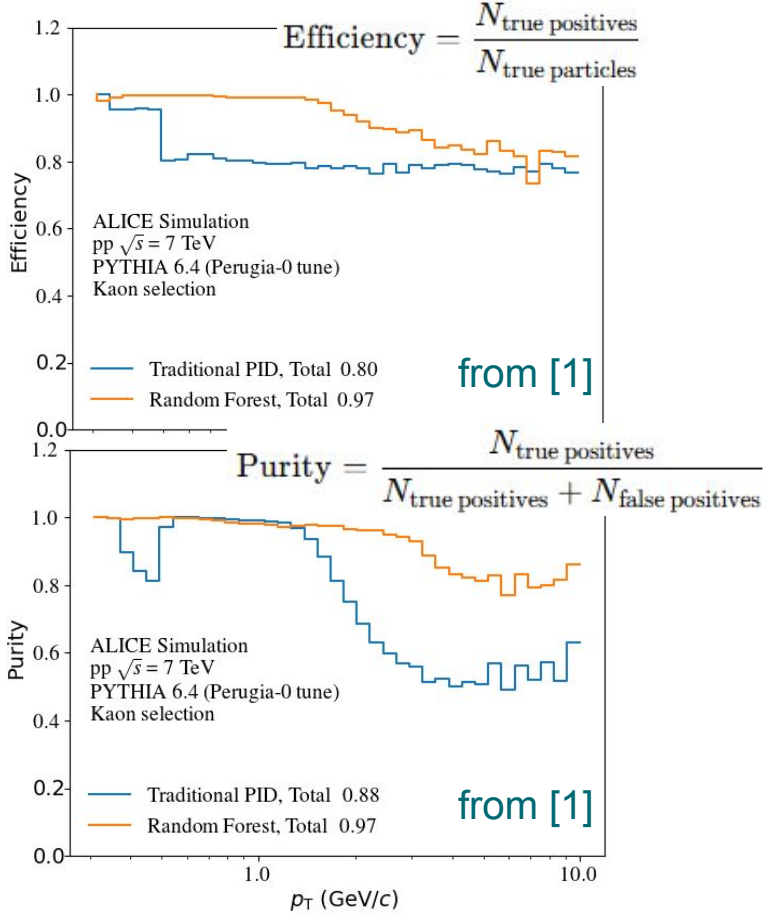
not covered in this talk
yields similar results

Both methods available in O^2 – ALICE Run 3 software

Can we do any better?

Yes!
With ML :)

ML for PID



Advantages of the ML approach to PID:

- **classification** – a ''standard'' ML problem
- can use **more track parameters** as input
- can learn **more complex relationships**
- many software libraries available

Note also **the limitations**:

- depends on **quality of the training data** (MC)
- hard to **quantify uncertainties**
- hard to follow classifier's ''reasoning'' (**black box**)

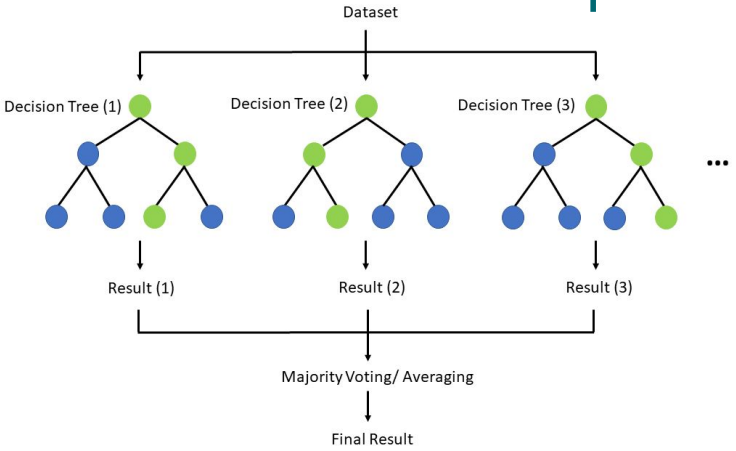
Our **first works** show ML can **greatly improve** purity and efficiency:

1. Random Forest: T. Trzciński, Ł. Graczykowski, M. Glinka, ALICE Collaboration. Using Random Forest classifier for particle identification in the ALICE experiment. Conference on Information Technology, Systems Research and Computational Physics, pp. 3-17. 2018
2. Domain Adaptation: M. Kabus, M. Jakubowska, Ł. Graczykowski, K. Deja, ALICE Collaboration. Using machine learning for particle identification in ALICE. JINST, v. 17, p. C07016. 2022

Proof-of-concept: Random Forest

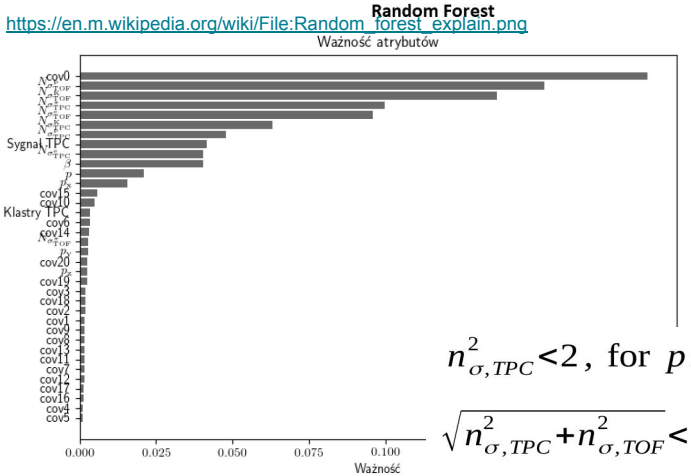
T. Trzciński, Ł. Graczykowski, M. Glinka, Conference on Information Technology, Systems Research and Computational Physics, 3-17. 2018

2018



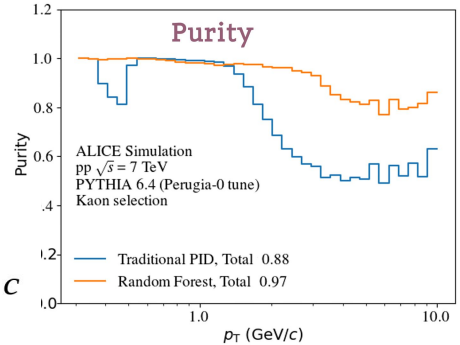
- Preliminary work with ALICE Run 2 data
- First solution - **Random Forest**
- Model works on **high-level track parameters**
- Depends on the **quality of Monte Carlo sample** and **post-processed information** (i.e. nσ calculation)
- Can be used **only for analysis-specific use-case** (concrete dataset and specific particle selection)
  - model has to be **trained by the specific end user**

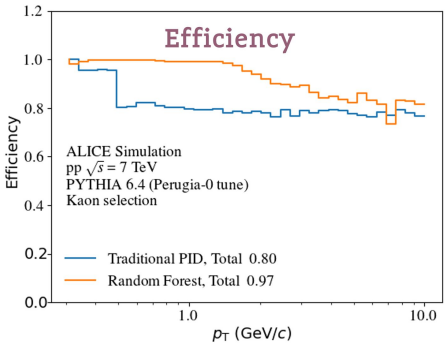
https://en.m.wikipedia.org/wiki/File:Random_forest_explain.png



$$n^2_{\sigma,TPC} < 2, \text{ for } p \leq 0.5 \text{ GeV}/c$$

$$\sqrt{n^2_{\sigma,TPC} + n^2_{\sigma,TOF}} < 2, \text{ for } p > 0.5 \text{ GeV}/c$$



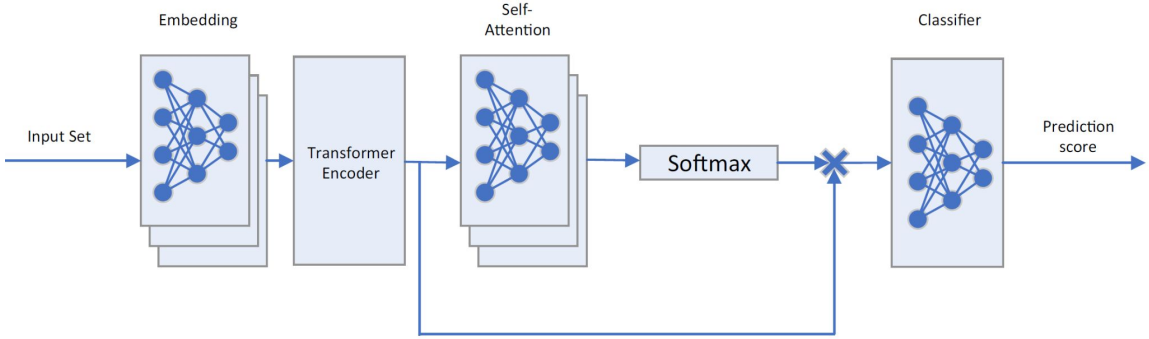


Current solution - our model

- Solution **general enough** to be used for variety of analyses
- **At present our input data has 19 features:** i.e. momentum components, charge sign, DCA_{XY}, DCA_{Z}, TPC number of clusters, detector signals (TPC dE/dx, TOF time, TRD signal), etc.
- **Data might be missing** for a given track from one or more detectors due to, e.g., too small p_T
- In **"standard" ML** approaches dealing with such cases, people use **data imputation** or **case deletion** - however artificially altered data may bias the physics results!
  - **Challenge:** classify particles without making any assumptions about the missing values
- The **proposed model** is much more advanced than the proof-of-concept solution and has **4 steps** (see next slides)
- For **details**, see our **two papers**:
  - EPJ C 84 (2024) 7, 691
  - JINST 19 (2024) 07, C07013

Current solution - our model

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak,
M. Jaik, JINST 19 (2024) 07, C07013



1. **Feature Set Embedding** to encode the input features and their position in input vector
2. **Transformer Encoder** to detect patterns in the input
3. Additional **self-attention network** to pool the encoder output set into a single vector
4. **Classifier** (a simple neural network) to classify a given particle type

Inspired by AMI-Net proposed for medical diagnosis from incomplete data (medical records)

Attention-based Multi-instance Neural Network for Medical Diagnosis from Incomplete and Low Quality Data

Zeyuan Wang[1,3], Josiah Poon[1], Shiding Sun[2], Simon Poon[1*]
[1]School of Computer Science, The University of Sydney, Syndey, Australia
[2]School of Mathematics, Renmin University of China, Beijing, China
[3]Beijing Medicinovo Technology Co.,Ltd., Beijing, China
[1,3]zwan7221@uni.sydeny.edu.au, [1]{josiah.poon, simon.poon}@sydney.edu.au, [2]sunshiding@ruc.edu.cn

2019 International Joint Conference on Neural Networks (IJCNN)

details on slide 27

Neural Network: How to handle missing data?

Raw input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	-	2.7
0.6	35.4	16.7	-
3.5	57.3	-	-

Strategy 1: Mean Imputation

Raw input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	-	2.7
0.6	35.4	16.7	-
3.5	57.3	-	-
	mean	10.3	2.55

Transformed input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	10.3	2.7
0.6	35.4	16.7	2.55
3.5	57.3	10.3	2.55

NEXT: Train **single** neural network. Fill the same mean values for target data.

Strategy 2: Regression Imputation

Raw input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	-	2.7
0.6	35.4	16.7	-
3.5	57.3	-	-

Missing column is treated as predicted response in regression method on the basis of the complete observations.

Transformed input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	13.3	2.7
0.6	35.4	16.7	2.1
3.5	57.3	17.3	2.5

NEXT: Train **single** neural network. Use the same fitted regression model for target data.

Strategy 1 & 2: Drawbacks

- **Imputation bias:** Artificially filling missing values can distort physics results.
- **Information loss:** A missing feature carries meaning; replacing it with an average erases that signal.
- **Limitations of regression:** Standard regression methods may not be precise enough for the complexity of relationships in High Energy Physics data.

Mean imputation:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	10.3	2.7
0.6	35.4	16.7	2.55
3.5	57.3	10.3	2.55

Regression imputation:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	13.3	2.7
0.6	35.4	16.7	2.1
3.5	57.3	17.3	2.5

Strategy 3: Grouping and Ensembling

Raw input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	-	2.7
0.6	35.4	16.7	-
3.5	57.3	-	-

Group observations by missing data combination. In this example 4 groups exists.

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4

p	TPC	TRD
1.6	35.6	2.7

p	TPC
3.5	57.3

p	TPC	TOF
0.6	35.4	16.7

NEXT: Train **separate** neural networks for every missing data combination. Ensemble them to make it technically simpler.

Strategy 3: Grouping Drawbacks

- **Complexity of training:** More models are trained (4 in the example).
- **Less training data:** Each model of the ensemble is trained on merely subset of original training data; some groups are significantly less numerous: less information for model to extract.

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4

p	TPC
3.5	57.3

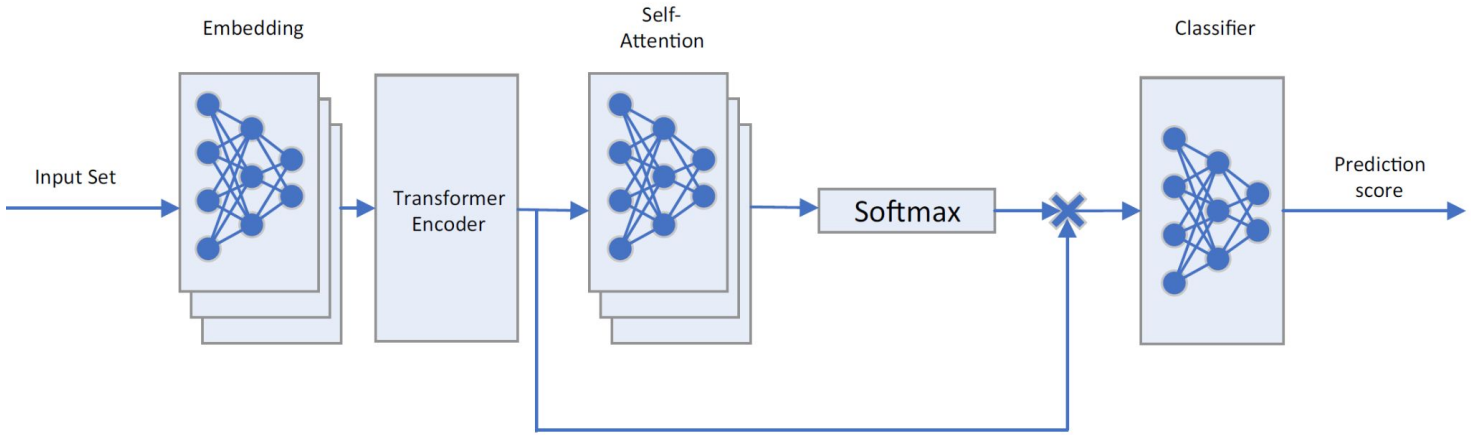
p	TPC	TRD
1.6	35.6	2.7

p	TPC	TOF
0.6	35.4	16.7

Strategy 4: Transformer (our solution)

Why?

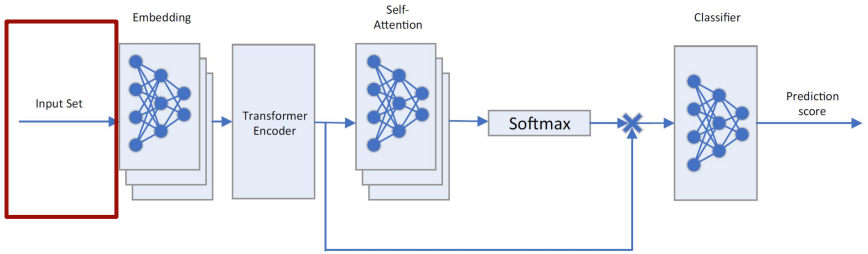
- **Flexible input handling:** Transformer encoder architecture can process data with varying feature sizes.
- **Captures complex relationships:** Learns intricate dependencies between features.
- **Efficient scaling:** Performs well as data size and complexity grow.



Strategy 4: Transformer (our solution)

Raw input:

p	TPC	TOF	TRD
0.1	65.4	3.8	2.4
1.6	35.6	-	2.7
0.6	35.4	16.7	-
3.5	57.3	-	-



Feature-value encoding:

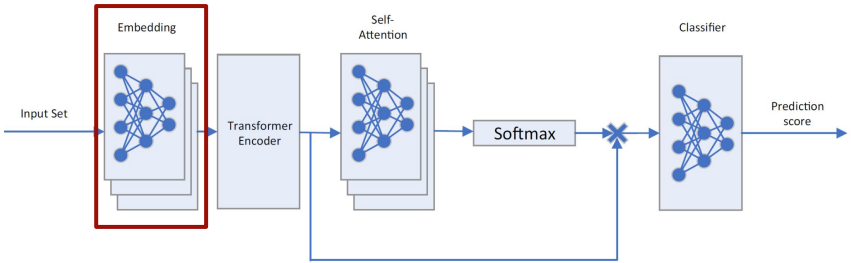
Feature				Value
1	0	0	0	1.6
0	1	0	0	35.6
0	0	0	1	2.7

From every observation vector we encode every feature as separate vector with position in original vector. We **skip** missing data.

Strategy 4: Transformer (our solution)

Feature-value encoding:

Feature				Value
1	0	0	0	1.6
0	1	0	0	35.6
0	0	0	1	2.7



Feature-set embedding:

Embedding Neural Network
NN Layers: [5, 128, 32]

C1	C2	C3	C4	...	C32
0.53	0.25	0.57	0.35		0.16
0.12	0.67	0.47	0.23		0.7
0.92	0.72	0.34	0.86		0.85

Strategy 4: Transformer (our solution)

Feature-value encoding:

Feature				Value
1	0		0	1.6
0	1	…	0	35.6
0	0		1	2.7

In reality we have 19 features for every particle track, which makes encoded vectors of 20 variables.

Feature-set embedding:

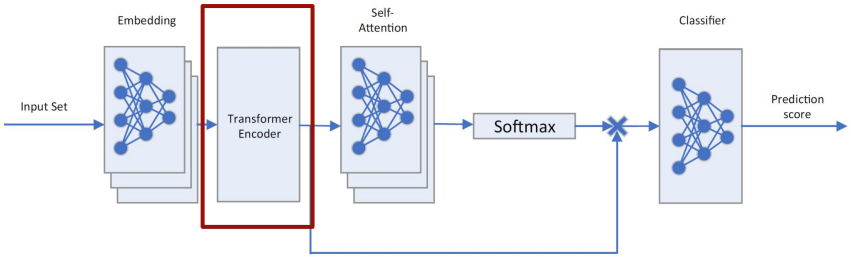
Embedding Neural Network
NN Layers: [20, 128, 32]

C1	C2	C3	C4	…	C32
0.53	0.25	0.57	0.35		0.16
0.12	0.67	0.47	0.23		0.7
0.92	0.72	0.34	0.86		0.85

Strategy 4: Transformer (our solution)

Feature-set embedding:

C1	C2	C3	C4	…	C32
0.53	0.25	0.57	0.35		0.16
0.12	0.67	0.47	0.23		0.7
0.92	0.72	0.34	0.86		0.85



Transformer Encoder
2 blocks, 2 heads
(multi-head attention)

C1	C2	C3	C4	…	C32
0.23	0.54	0.87	0.45		0.36
0.54	0.43	0.12	0.73		0.57
0.54	0.54	0.63	0.16		0.65

Transformer Encoder layer **captures complex relationships** between features. Dimensionality of data is the same.

Strategy 4: Transformer (our solution)

After Transformer Encoder:

Self-attention pooling NN
NN Layers: [32, 64, 32]

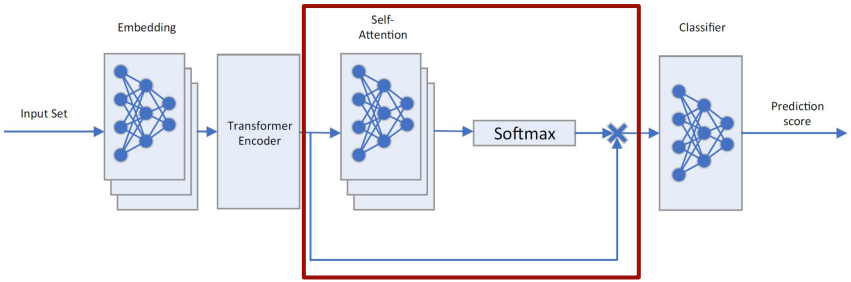
[3,32] real numbers

Softmax

[3,32] weights

C1	C2	C3	C4	...	C32
0.23	0.54	0.87	0.45		0.36
0.54	0.43	0.12	0.73		0.57
0.54	0.54	0.63	0.16		0.65

C1	C2	C3	C4	...	C32
0.05	0.03	0.08	0.1		0.01
0.03	0.08	0.06	0.01		0.1
0.02	0.04	0.07	0.05		0.03



Neural Networks learns to **weight columns** for given vector. Then **softmax** is applied to transform real numbers to weights.

Strategy 4: Transformer (our solution)

With weights calculated:

C1	C2	…	C32
0.23	0.54	…	0.36
0.54	0.43	…	0.57
0.54	0.54	…	0.65

✖

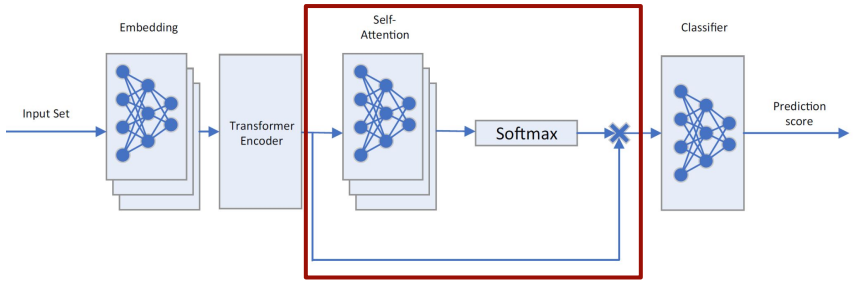
C1	C2	…	C32
0.05	0.03	…	0.01
0.03	0.08	…	0.1
0.02	0.04	…	0.03

=

C1	C2	…	C32
0.01	0.02	…	0
0.02	0.03	…	0.06
0.01	0.02	…	0.02

Sum:

C1	C2	…	C32
0.04	0.07	0	0.08



Perform **element-wise multiplication** of rows of data and weights matrices. Take a **sum over columns** to obtain single vector.

Strategy 4: Transformer (our solution)

Pooled vector:

C1	C2	…	C32
0.04	0.07	0	0.08

Classifier
NN Layers: [32, 64, 1]

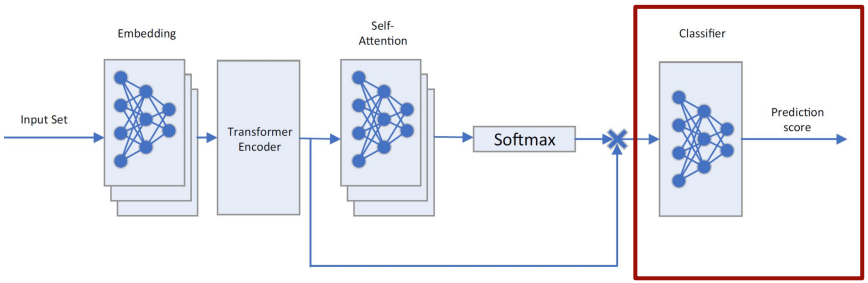
Prediction (certainty)

0.67 ≥ 0.5

Is it kaon?

True → YES

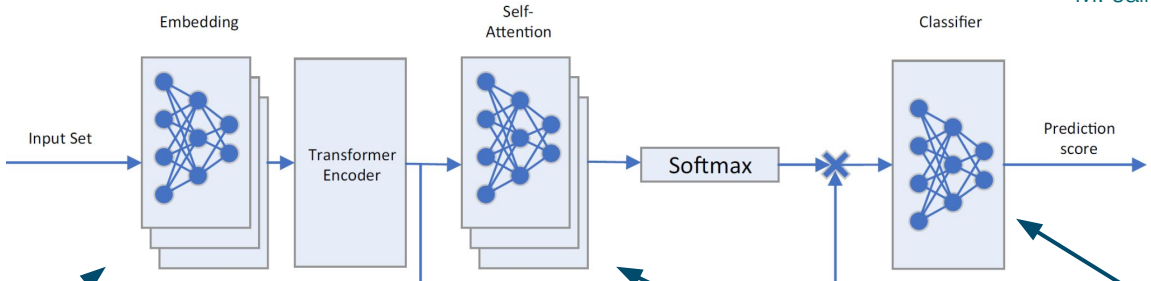
False → NO



User-specified, flexible threshold to make **final binary prediction**.

Details of the architecture

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak,
M. Jaik, JINST 19 (2024) 07, C07013



This model is applied **separately** for pions, kaons, protons

"certainty"

2 heads
in each head
2 layers (att.+NN)

Embedding			Transformer encoder Encoder layer Multi-head attention		Neural network			Self-attention			Classifier		
Input	Hidden	Output	Dimension	Heads	Input	Hidden	Output	Input	Hidden	Output	Input	Hidden	Output
20	128	32	32	2	32	128	32	32	64	32	32	64	1

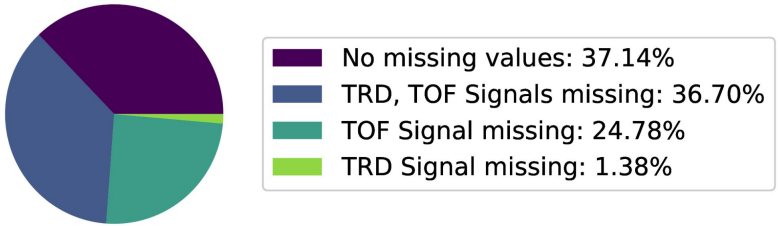
- **dropout** value 0.1 at the output of embedding and each Encoder layer (to limit overfitting)
- **activation function** (between neural network layers): *ReLU (Rectified Linear Unit)*
- **loss function** that is minimized is *binary cross entropy* (for *one vs all* approach)
  - to minimize differences between *predicted* and *true* values (labels from MC truth data)

Test setup

M. Kasak, K. Deja. M. Karwowska, M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak, M. Jaik, JINST 19 (2024) 07, C07013

- **Dataset:** Run 2 general-purpose MC (Pythia 8) pp at $\sqrt{s}$ = 13 TeV with full detector simulation with GEANT 4 (both MC truth and reconstructed data are used)
  - TPC signal is always required
- **Standard nσ method:**
  $|n_{\sigma, TPC}| < 3$ for $p_T < 0.5$ GeV/c, $\sqrt{(n_{\sigma, TPC}^2 + n_{\sigma, TOF}^2)} < 3$ for $p_T \geq 0.5$ GeV/c
- **Dataset details:**
  - no. tracks: ~2.7 million
  - 30% - test dataset
  - from the 70% of the rest:
    - 70% training
    - 30% validation

Missing data distribution



Results – pions, kaons, protons

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak,
M. Jaik, JINST 19 (2024) 07, C07013

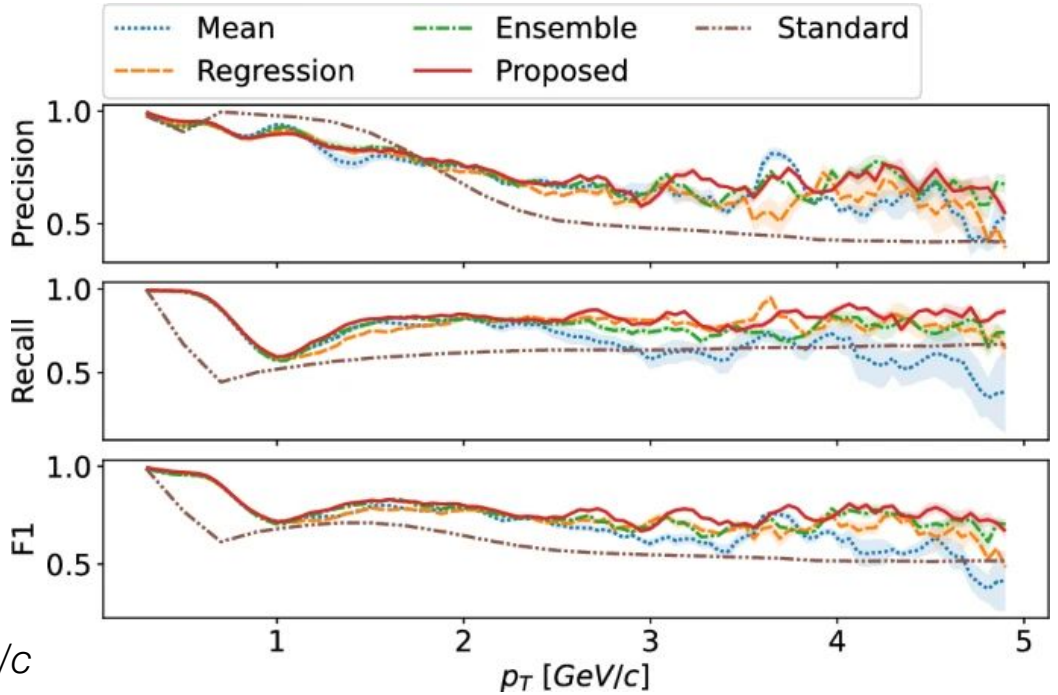
F_1 = (purity x efficiency) / (purity + efficiency)

FSE + attention with **very good scores** of **F_1, purity (precision)** and **efficiency (recall)**

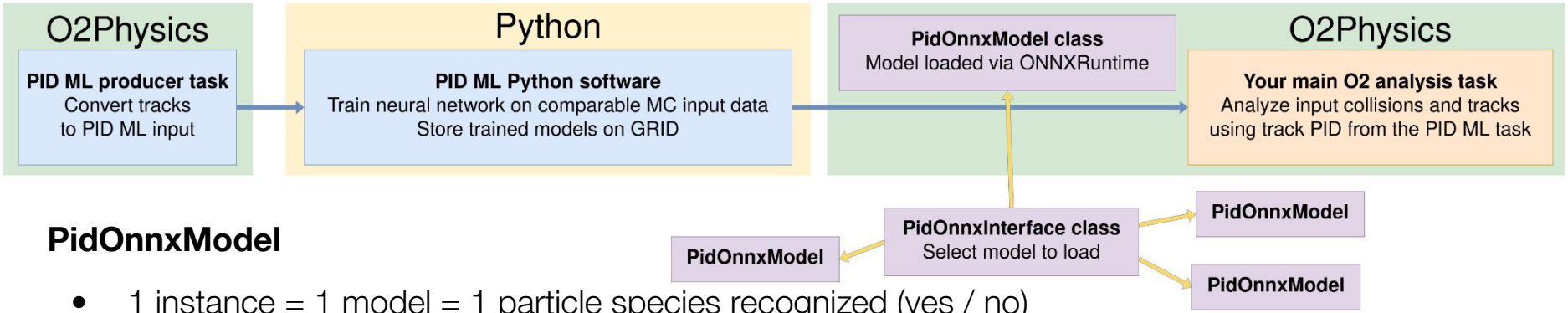
Proposed model (FSE+Attention)
compared to **other approaches:**

- **imputation:**
  artificial bias in data
  - mean
  - regression
- **NN ensemble** (4 networks)**:**
  potentially large complexity
  less data for training
- **standard:**
  nσ method
  $|n_{\sigma,\,TPC}| < 3$ for $p_T < 0.5$ GeV/c
  $\sqrt{(n_{\sigma,\,TPC}^2 + n_{\sigma,\,TOF}^2)} < 3$ for $p_T \geq 0.5$ GeV/c

kaon selection



Integration with O^2: user interface



PidOnnxModel

- 1 instance = 1 model = 1 particle species recognized (yes / no)
- **convenient interface** clearly separated from the rest of analysis
- using all capabilities of **Python ML libraries** for training
- ONNX file format and **ONNXRuntime** software used for inference in O^2 C++ environment
- models **stored in CCDB** (experiment's database) for each run and available to access in data analysis code by users (via a "helper task")

PidOnnxInterface

- **automatically select most suitable model** for user needs or manual mode
- as **little additional knowledge** from the analyser as possible (*"change 1 line in the code"*)

https://onnx.ai/

Conclusions

R&D phase of the ML PID (almost) **finished!**

FSE+Attention model works well for the three basic identified hadron species (pions, kaons, protons)

Lots of work done, but still more ahead!

Plans for future:

- tests with Run 3 data with new O^2 analysis framework (*ongoing*)
- automation of model training and regular training of models for new Run 3 datasets (*implementation*)
- extending the model with domain adaptation (*still to do*)
- advertise PID ML among ALICE analyzers (*to do when fully implemented*) and outside ALICE

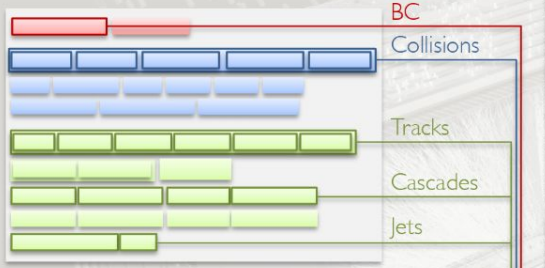
The work has been carried out by an interdisciplinary team from 4 faculties of WUT:

- *Physics:* Ł. Graczykowski (*general idea, coordination, evaluation*), M. Janik (*evaluation*), M. Karwowska (*implementation*), S. Monira (*tests of implemented model*)
- *Electronics and Information Technology:* Kamil Deja, Miłosz Kasak (*ML R&D*)
- *Electrical Engineering:* Monika Jakubowska (*coordination, evaluation*)
- *Mathematics and Computer Science:* Marek Mytkowski, Mateusz Olędzki (*implementation*)


Thank you!

In a nutshell: the general analysis task structure

Data Model



Your analysis task!

struct yourAwesomeTask — Basic task definition

- **produces**<something> name; — Declares tables that may be created by this task
- **Partition** declarations — Declares partitions: new tables based on selection criteria
- **Filter** declarations — Declares filters: selection criteria
- **Output object** declarations — Declares output objects: Histograms or HistogramRegistry
- **Configurable** declarations — Declares configurables: values that can be set by user
- **init()** — Set up before processing data
- **process(●●●●●)** — Subscribe (connect to input) and process data
- **defineDataProcessing()** — Information for task -> DPL processor conversion

Examples:

- loop over tracks in each event
- loop over cascades in each event
- Correlate cascades with jets
  → do **physics**!

●●●●● = tells the framework which tables the user is interested in and which to merge / relate to one another

Very theoretical → now we will go practical! Let's run and customize our own task

Crash course: how do you run something?

- Each analysis task is an executable ⟶ this means you can run them in the command line!

```
o2-analysistutorial-histograms --aod-file AO2D.root | o2-analysis-track-propagation | o2-analysis-timestamp
```

Example task	Input file	Helper task Propagates tracks to PV	Helper task Provides timestamps

- All tasks have to be provided separated with a 'pipe' character ("|")
- `--aod-file` can receive an AO2D file or you can use `--aod-file @listoffiles.txt` with a list of files!
- Typically, many helper tasks are required: we will introduce you to this in the hands-on!
- This is, among other things, a consequence of the AO2D content
  - not all table information is available in the AO2D: minimalistic!
  - Some tables and columns are generated on-the-fly to minimize data storage: a strict necessity in Run 3!

- General event (centrality/multiplicity percentile) and track properties (PID values) have to be calculated!
- And beyond that: tracks are stored at their 'innermost update' in the AO2D (TracksIU)
  - Tracks to be propagated to the primary vertices by the track propagation task
  - We'll also show you this later…

Run 2 results

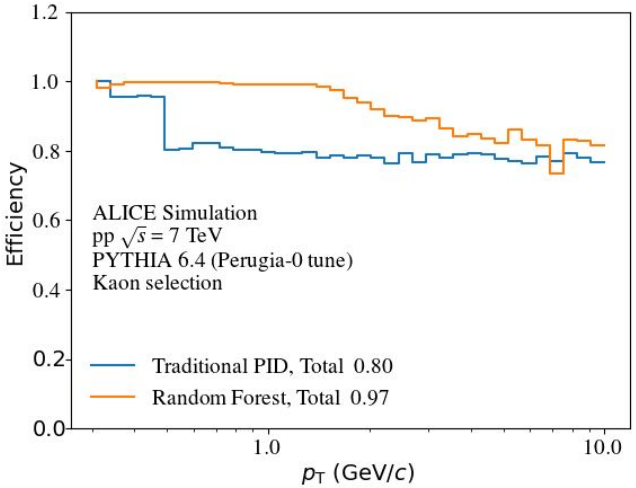
- pp at 7 TeV, Pythia 6 Perugia-0
- kaons vs other particles

Traditional PID:

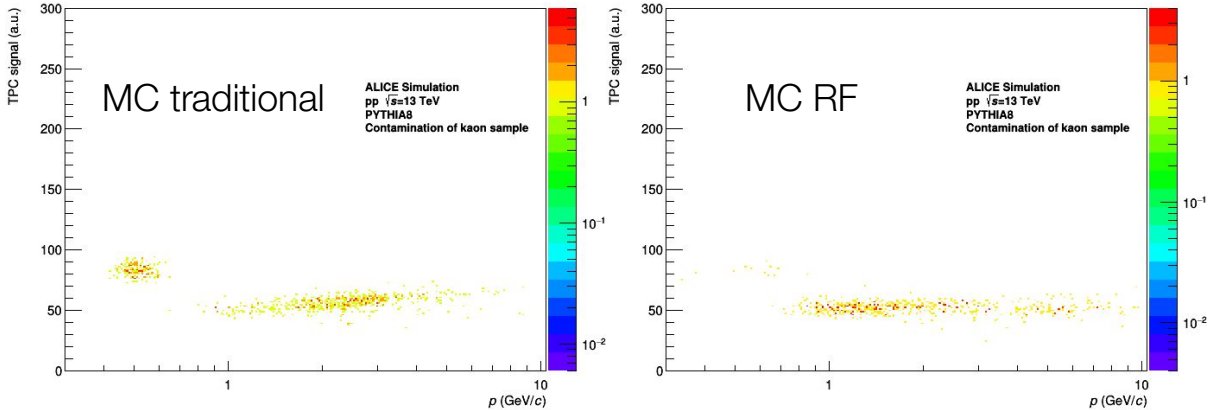
$$n^2_{\sigma,\mathrm{TPC}} \qquad p_\mathrm{T} \leq 0.5\mathrm{GeV/c}$$

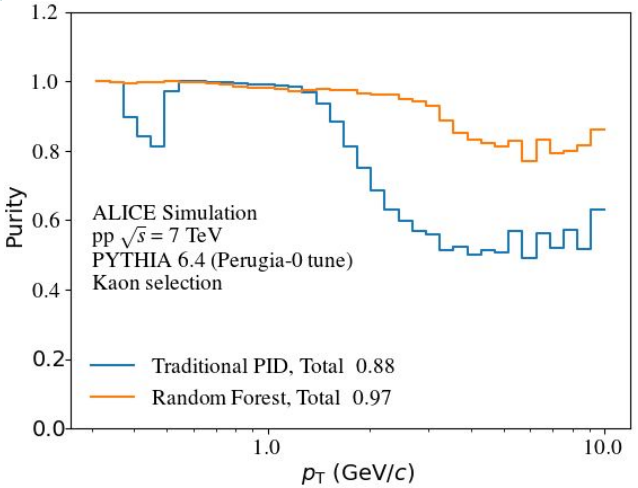
$$\sqrt{n^2_{\sigma,\mathrm{TPC}} + n^2_{\sigma,\mathrm{TOF}}} \qquad p_\mathrm{T} > 0.5\mathrm{GeV/c}$$

much higher efficiency and **purity** with Random Forest



Contamination of kaon samples





Example: FSE with one-hot encoding

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, Ł. Graczykowski
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, Ł. Graczykowski, K. Deja,
M. Kasak, JINST 19 (2024) 07, C07013

Table 1: Preprocessing of data samples into feature set values – example.

(a) 3 data samples with 5 attributes with different amount of missing values.

id	momentum	TOF	TPC	TRD	ITS
1	0.1		3		5
2	7	70	24	13	88
3		78			

(b) First particle

key					value
1	0	0	0	0	0.1
0	0	1	0	0	3
0	0	0	0	1	5

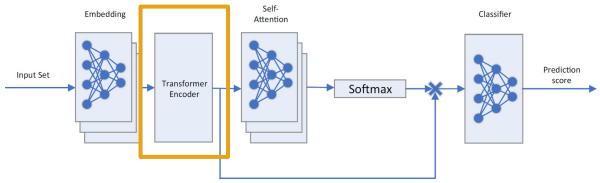
(c) Second particle.

key					value
1	0	0	0	0	7
0	1	0	0	0	70
0	0	1	0	0	24
0	0	0	1	0	13
0	0	0	0	1	88

(d) Third particle.

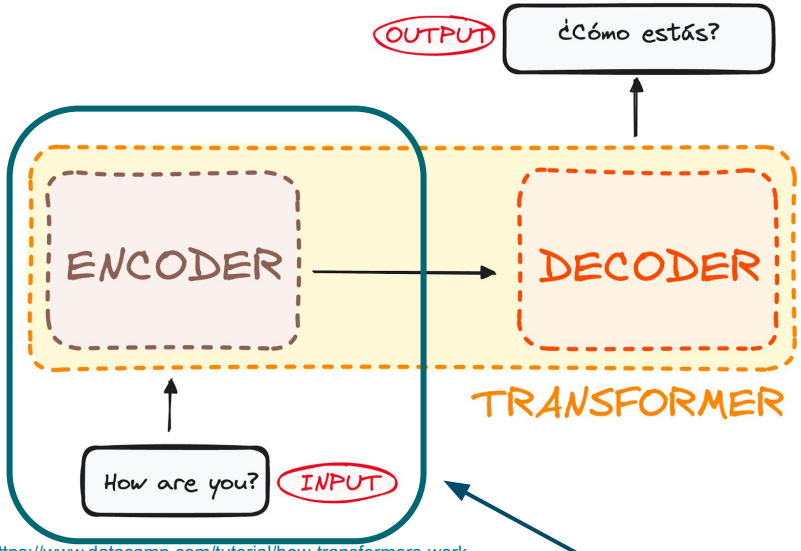
key					value
0	1	0	0	0	78

Step 2: Transformer Encoder



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com



https://www.datacamp.com/tutorial/how-transformers-work

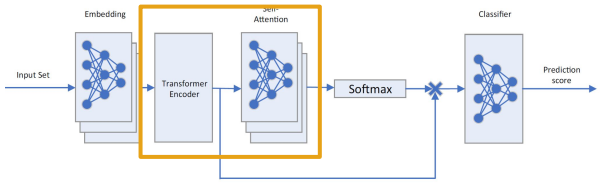
- Idea from original **Transformer** architecture proposed by Google (NIPS 2017 article)
- Developed for **transforming input data** into a **contextualized representation** on the output
- Transformer currently serves as **basis for the Natural Language Processing** tools (such as **ChatGPT**)
- In our case, **vectors from Embedding are processed by the Encoder only**
  - we do not need Decoder in our use-case

Steps 2 and 3: self-attention

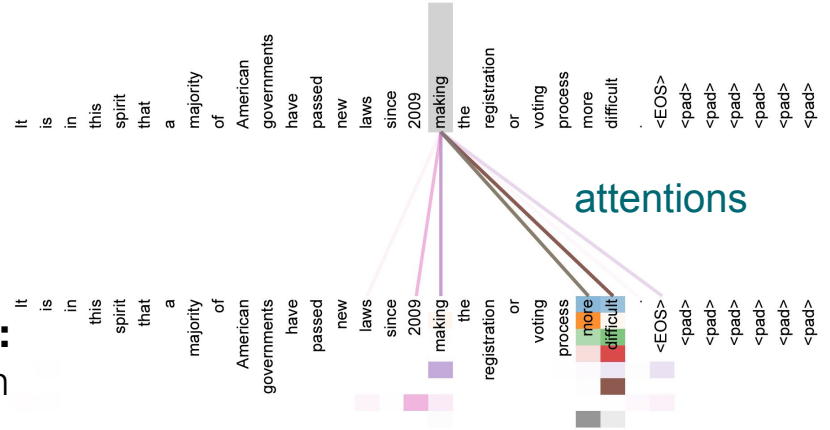
- **Attention** and **self-attention** are mechanisms used to help model focus on relevant parts of the input data
  - **self-attention** focuses on **relationships within the same input sequence**

- ***Example:*** "The cat sat on the mat"
  - when processing the word "cat," it considers other words (i.e. "the" or "mat") to understand their contribution to the meaning of "cat" (in the context of the entire sentence)

- Usage of **self-attention in Transformer architecture:**
  - in **single-head attention**, a single set of attention scores is used to focus on a particular part of the input sequence → limited ability to capture different relationships
  - **multi-headed attention** uses multiple attention heads, where each head focuses on different parts of the input simultaneously



We use **self-attention twice**:
- in **Transformer Encoder**
- before **Classifier**



colors = attentions from different heads

NIPS 2017 article

Results

F_1 = 2 x (purity x efficiency) / (purity + efficiency)
best model, **2nd best model**

ML outperforms the standard way

FSE + attention with **very good scores** of F_1

No flaws of other methods:

- imputation:
  artificial bias in data
- case deletion:
  no ability to analyze samples with missing detector signals
- NN ensemble:
  potentially large complexity

	π	p	K	$\bar{\pi}$	$\bar{p}$	$\bar{K}$
standard	87.87 ± 0.87	74.61 ± 1.88	73.17 ± 1.57	87.66 ± 0.87	69.12 ± 1.93	69.44 ± 1.60
NN ensemble	98.45 ± 0.04	95.42 ± 0.12	86.74 ± 0.16	98.27 ± 0.42	94.60 ± 0.10	84.91 ± 0.48
mean	98.40 ± 0.01	95.54 ± 0.06	86.36 ± 0.34	98.34 ± 0.01	94.75 ± 0.20	84.67 ± 0.38
attention + FSE	98.50 ± 0.02	95.79 ± 0.07	87.44 ± 0.14	98.44 ± 0.02	94.89 ± 0.14	86.00 ± 0.13
regression	98.40 ± 0.04	95.49 ± 0.15	86.22 ± 0.46	98.36 ± 0.03	94.57 ± 0.13	85.01 ± 0.13

	π, only complete data	p, only complete data	K, only complete data	$\bar{\pi}$, only complete data	$\bar{p}$, only complete data	$\bar{K}$, only complete data
case deletion	99.37 ± 0.01	99.43 ± 0.16	96.95 ± 0.06	99.37 ± 0.01	99.13 ± 0.26	96.33 ± 0.11
NN ensemble	99.38 ± 0.01	99.46 ± 0.13	97.23 ± 0.10	99.34 ± 0.18	99.33 ± 0.10	96.87 ± 0.09
mean	99.27 ± 0.04	99.47 ± 0.08	96.08 ± 0.36	99.27 ± 0.04	99.20 ± 0.27	95.45 ± 0.33
attention + FSE	99.36 ± 0.01	99.48 ± 0.02	97.04 ± 0.17	99.37 ± 0.03	99.44 ± 0.08	96.91 ± 0.11
regression	99.25 ± 0.07	99.37 ± 0.07	95.62 ± 0.39	99.28 ± 0.02	99.10 ± 0.13	95.11 ± 0.58

Example: FSE with one-hot encoding

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, Ł. Graczykowski
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, Ł. Graczykowski, K. Deja,
M. Kasak, JINST 19 (2024) 07, C07013

Table 1: Preprocessing of data samples into feature set values – example.

(a) 3 data samples with 5 attributes with different amount of missing values.

id	momentum	TOF	TPC	TRD	ITS
1	0.1		3		5
2	7	70	24	13	88
3		78			

(b) First particle

key					value
1	0	0	0	0	0.1
0	0	1	0	0	3
0	0	0	0	1	5

(c) Second particle.

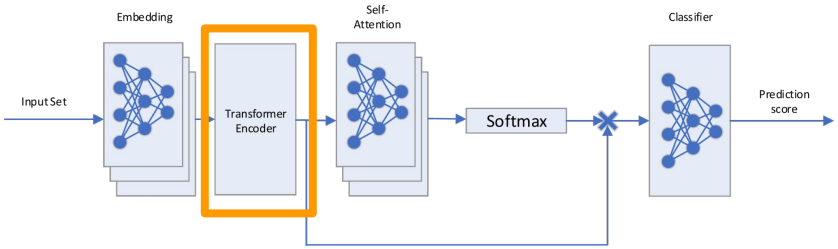
key					value
1	0	0	0	0	7
0	1	0	0	0	70
0	0	1	0	0	24
0	0	0	1	0	13
0	0	0	0	1	88

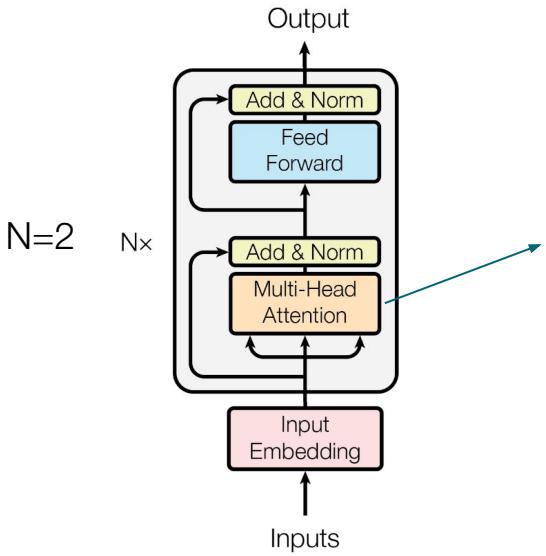
(d) Third particle.

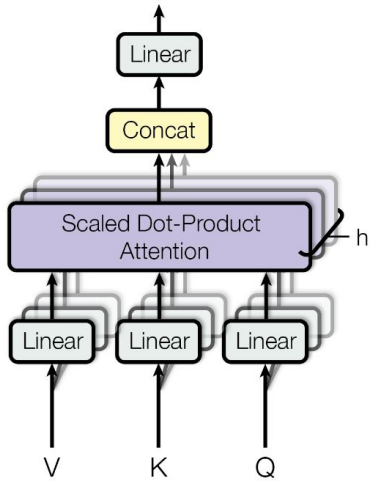
key					value
0	1	0	0	0	78

The attention continued

2. Transformer Encoder







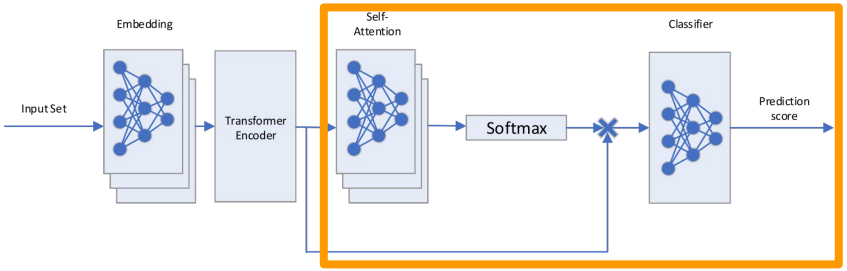
- adjusted original Transformer Encoder
- attention without convolutions and recurrence
- finding self-correlations in an instance set of vectors
- example: a specific detector signal could be used if and only if the momentum is in a specific range

modified diagram from the article

$$Q, K, V \in \mathbf{R}^{n \times d_k}$$

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

Pooling and classification



Classifier: a simple neural network
expects a single vector as an input

Solution: self-attention to pool the variable-size vector set from Transformer Encoder

$$\{v_1, v_2, ..., v_n\}, \; v_i \in \mathbf{R}^{d_{model}}$$

$$e_i = NN(v_i) \qquad \forall i \in [1, n]$$ self-attention values

$$\alpha'_j = softmax(e'_j) \qquad \forall j \in [1, d_{model}]$$ self-attention weights

$$o_j = \sum_{k=1}^{n} \alpha_{kj} v_{kj} \qquad \forall j \in [1, d_{model}]$$ pooled output vector

Classifier score: logistic function $f(x) = \frac{1}{1+e^{-x}}$, range (0, 1)
"certainty" that a given particle belongs to the given type

Architecture of tested neural networks

Attention + FSE

- embedding layers: 19 – 128 – 32 neurons
- Transformer Encoder:
  - Multi-Head Attention: dimension 32, 2 heads
  - neural network layers: 32 – 128 – 32 neurons
  - 2 layers of Multi-Head Attention + neural network
- Self-Attention layers: 32 – 64 – 32 neurons
- classifier layers: 32 – 64 – 1 neurons
- dropout 0.1 at the output of embedding and each Transformer Encoder layer
- ReLU activation between neural network layers
- classifier loss function: binary cross entropy

Imputations, case deletion, and NN ensemble

- 3 hidden layers of sizes 64, 32, 16 with Leaky ReLU activation
- dropout 0.1 after each activation layer
- input size:
  - imputations and case deletion: 19 as all missing features are imputed
  - ensemble: 4 networks with input sizes 19, 17, 17, 15

Simple network implementation

PyTorch

- linear layers with ReLU, sigmoid at the end
- simple: dropout after each linear layer

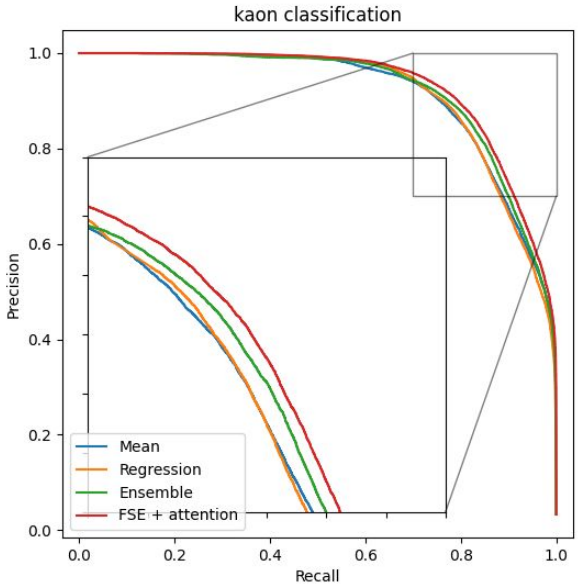
Parameters:

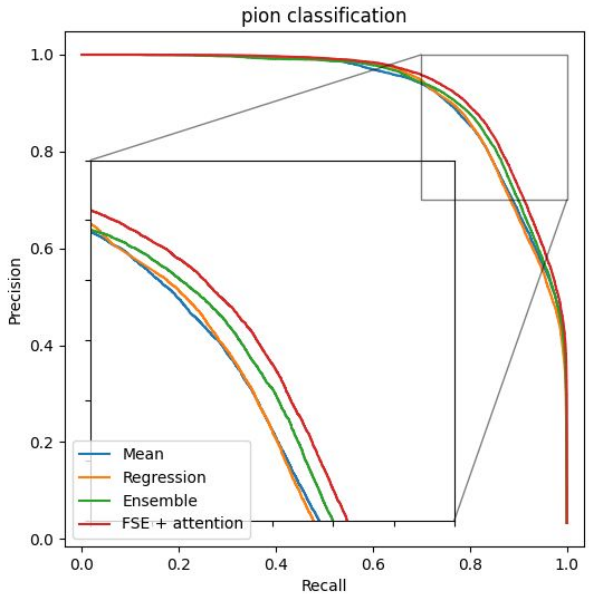
- optimizer: Adam
- output layer: 1 node (yes / no for a given particle)
- loss function: binary cross entropy
- scheduler: exponential with rate 0.98
- learning rate: 0.0005
- batch size: 64
- epochs: 30

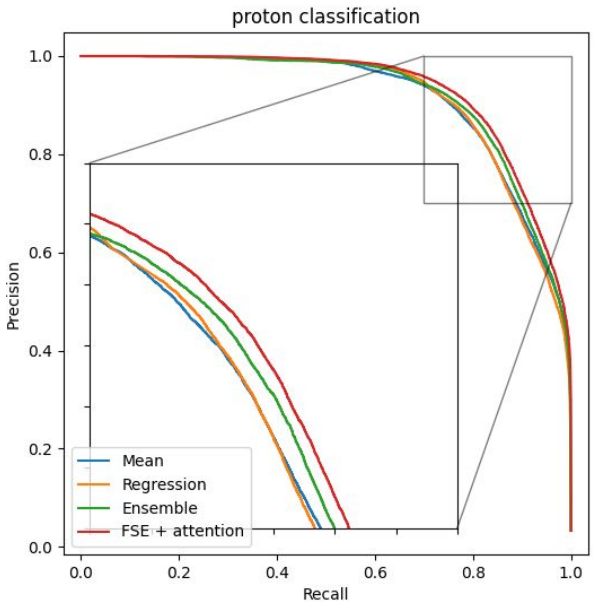
Sample ROC curves

FSE+attention achieves **best results.**

Little variation between particle species.







More to go: domain adaptation

M. Kabus, M. Jakubowska, Ł. Graczykowski, K. Deja
JINST 17, C07016. 2022

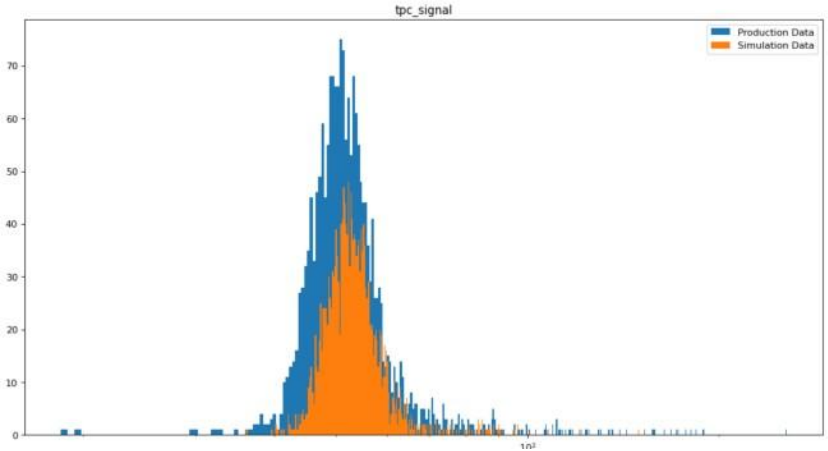
- **Monte Carlo never ideally matches the experimental data** (both physics and detector response simulation)

- **Problem:** transferring the knowledge from a **labeled source domain (MC data)** to **unlabeled target domain (experimental data)**, when both domains have different distributions of attributes

- How can we transfer the knowledge from training to inference?

Standard PID example: **"tune on data"**

- get parametrization from data → real data
- generate a random detector signal → MC data
- equivalent distributions of real and MC samples – the differences are statistical fluctuations
- does not include correlations between attributes



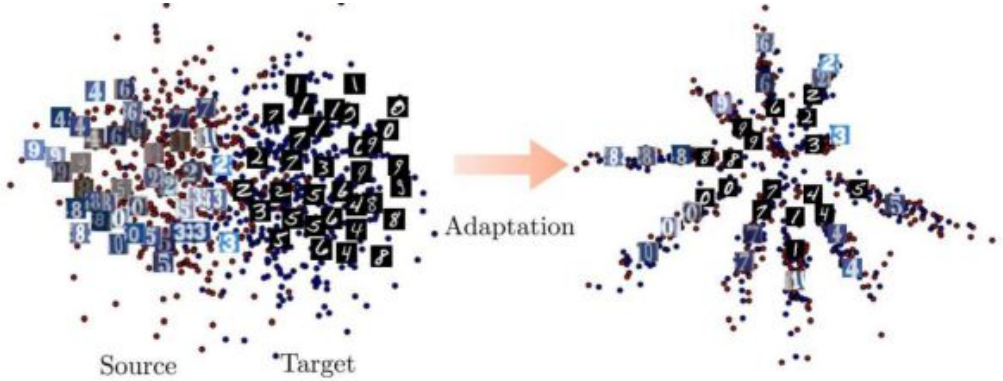
Machine learning:

- actually **learn** the difference between data domains
- translate both data to a single common hyperspace

More to go: domain adaptation

(a) MNIST

(b) SVHN



More to go: domain adaptation

M. Kabus, M. Jakubowska,
Ł. Graczykowski, K. Deja
JINST 17, C07016. 2022

Feature mapping: input → domain invariant features

Particle classifier: recognize particles based on domain invariant latent space

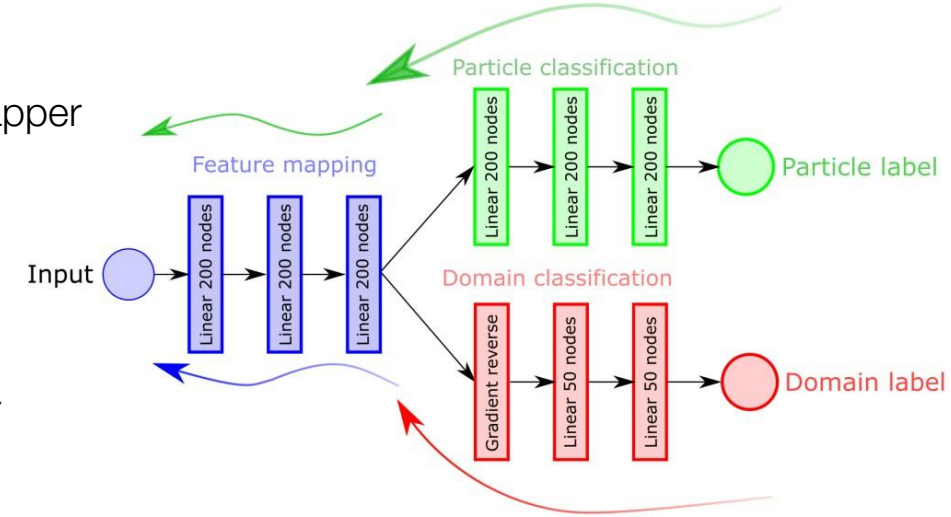
Domain classifier: recognize MC vs real samples

Training more complicated:

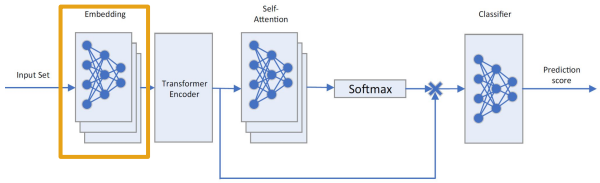
1. Train the domain classifier independently.
2. Freeze the domain classifier.
3. Train jointly particle classifier and feature mapper **adversarially** to the domain classifier.
4. Weights of the feature mapper:
   gradient from particle classifier
   + reversed gradient from domain classifier

Application time similar to a standard classifier

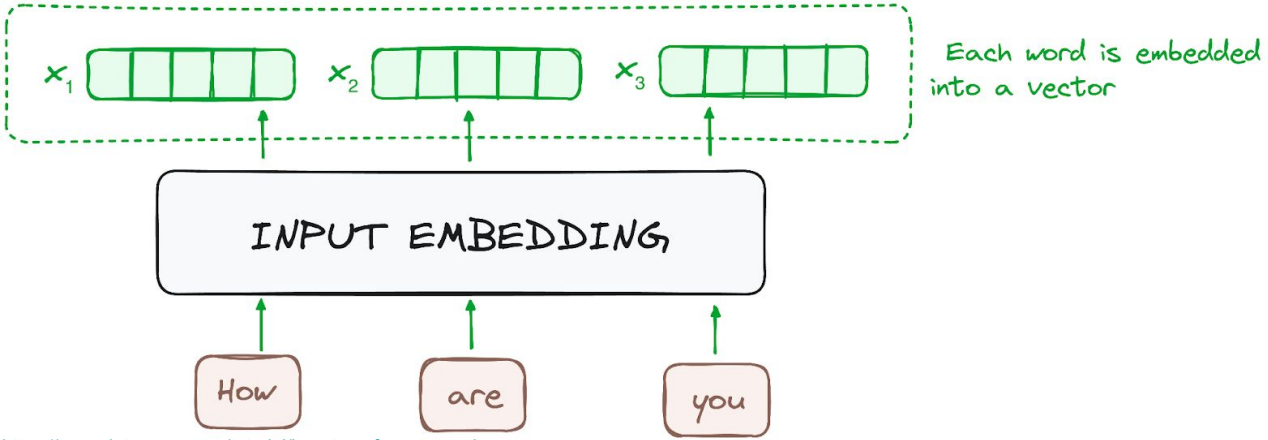
Our current solution still misses this step



Step 1: Embedding

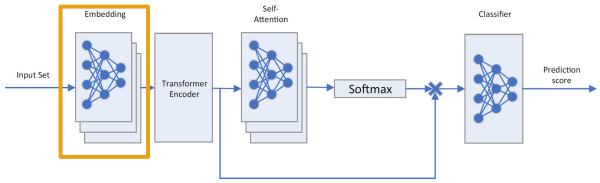


- **Embedding** is a technique to **handle complex data**
- It works by **converting high-dimensional data** (i.e. sequences of words, documents, images, etc.), **into lower-dimensional** and **abstract vector representation (embedding space)**
- It allows for capturing **meaningful relationships between data entities** (words, etc.)



https://www.datacamp.com/tutorial/how-transformers-work

Step 1: Feature Set Embedding



Missing data challenge:
classify without making any assumptions about the missing values

Feature Set Embedding (NIPS 2010 article):

- first, create (feature,value) pairs; no value → no pair
  - no need to model missing data (i.e. imputation)
- pairs in embedding space: similar features are close to each other
- pairs are then combined (by NN with a single hidden layer) into vectors (embeddings)

Feature Set Embedding for Incomplete Data

David Grangier
NEC Labs America
Princeton, NJ
dgrangier@nec-labs.com

Iain Melvin
NEC Labs America
Princeton, NJ
iain@nec-labs.com

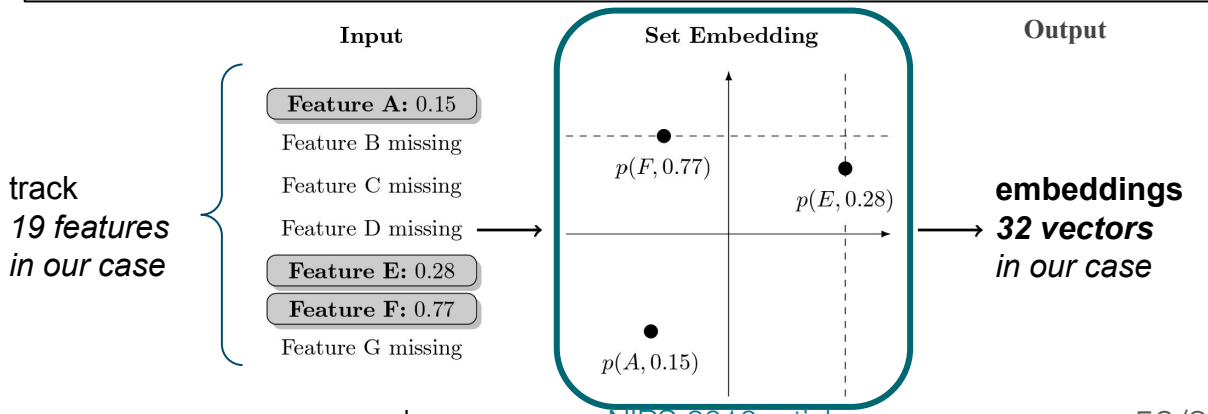
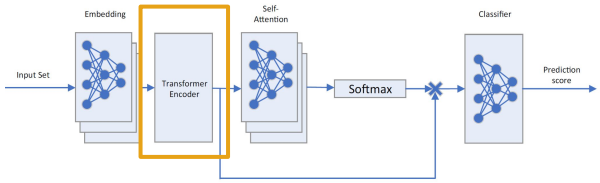
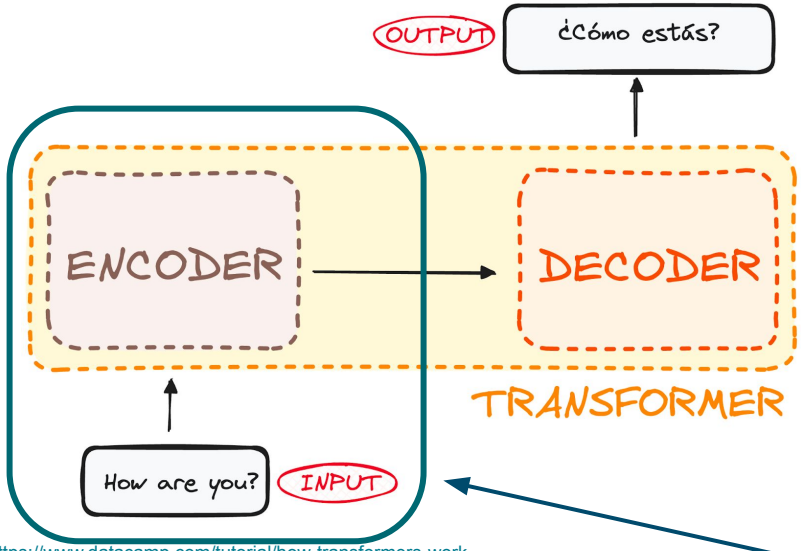


Image source: NIPS 2010 article

Step 2: Transformer Encoder





https://www.datacamp.com/tutorial/how-transformers-work

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

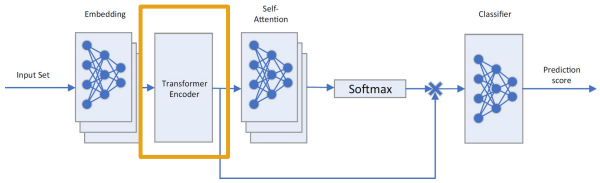
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

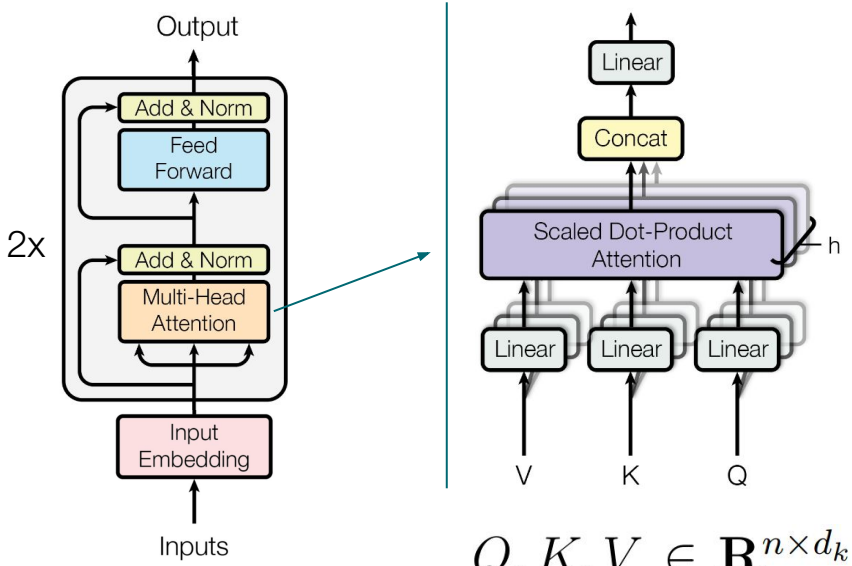
Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

- Idea from original **Transformer** architecture (NIPS 2017 article)
- In our case, **vectors from Embedding are processed by the Encoder only**
  - it finds relations between available features regardless of the amount of missing values

Step 2: Transformer Encoder





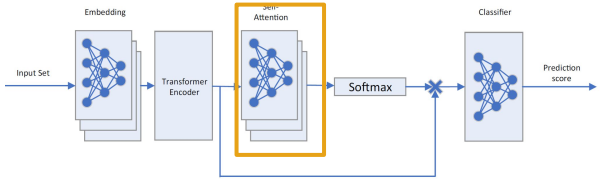
modified diagram from the Transformer article

- Encoder processes 32 embedding vectors to connect different features each vector represents
  - we use **2-head attention** (to find more complex relationships)
  - each head has **2 layers**: **attention** (for to the whole set of vectors) **+ dense NN** (applied to each vector separately)
  - example: a specific detector signal could be used if and only if the momentum is in a specific range

$$Q, K, V \in \mathbf{R}^{n \times d_k}$$

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

Step 3: self-attention pooling



- The final **classifier** requires a **single output vector**, while we have 32 vectors (processed embeddings) at the output of the Encoder
  - **Solution:** another **self-attention network** (single layer) is used to pool the final vector

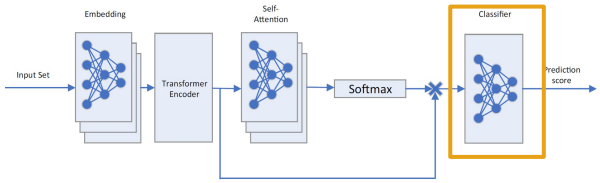
$$\{v_1, v_2, ..., v_n\},\ v_i \in \mathbf{R}^{d_{model}}$$ processed embeddings

$$e_i = NN(v_i) \qquad \forall i \in [1, n]$$ self-attention values

$$\alpha'_j = softmax(e'_j) \qquad \forall j \in [1, d_{model}]$$ self-attention weights

$$o_j = \sum_{k=1}^{n} \alpha_{kj} v_{kj} \qquad \forall j \in [1, d_{model}]$$ **pooled output vector** components

Step 4: classification



- Single **output vector** from the **self-attention network** is propagated to the **classifier**

- **Classifier** is represented by **one simple neural network** (one hidden layer) **per particle** type (**one vs all** approach)
  - the same architecture is used **separately** for pions, kaons, protons

- **Classifier score:** logistic function $f(x) = \frac{1}{1+e^{-x}}$ in **range (0, 1)** represents **"certainty"** that a given particle belongs to the given particle type
  - users can still **balance the efficiency and purity** by setting their own **threshold on the “certainty” value**