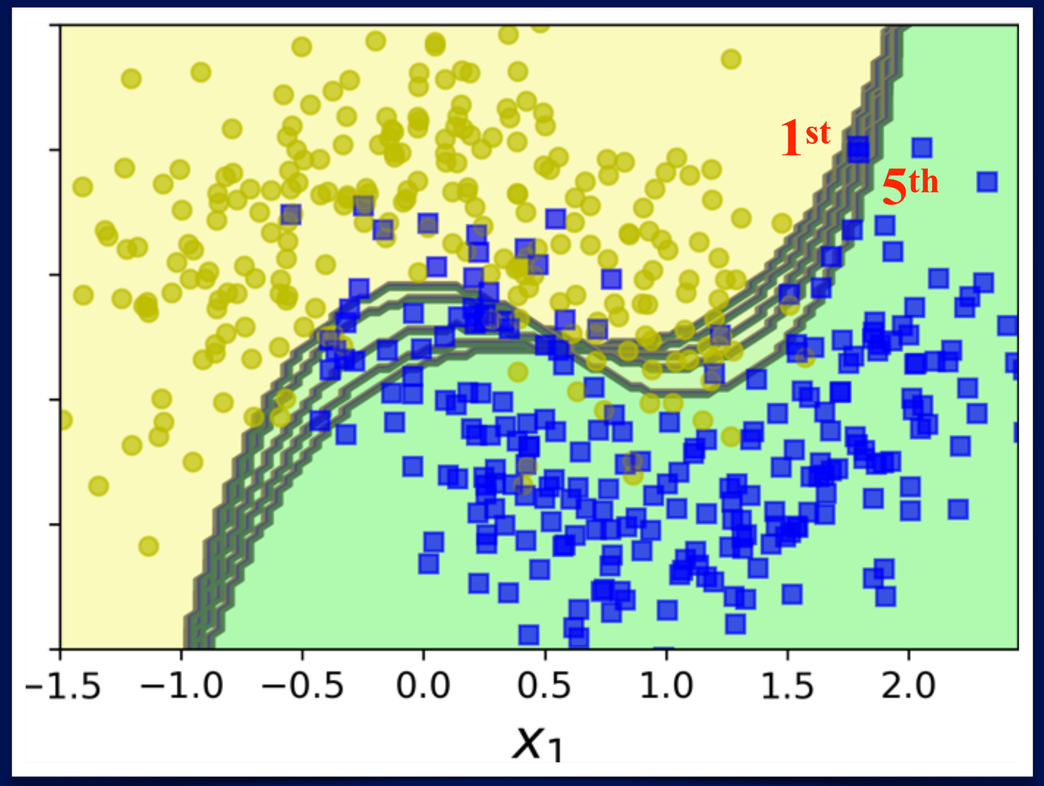
MLPs and BDTs



Manuel Franco Sevilla,
University of Maryland

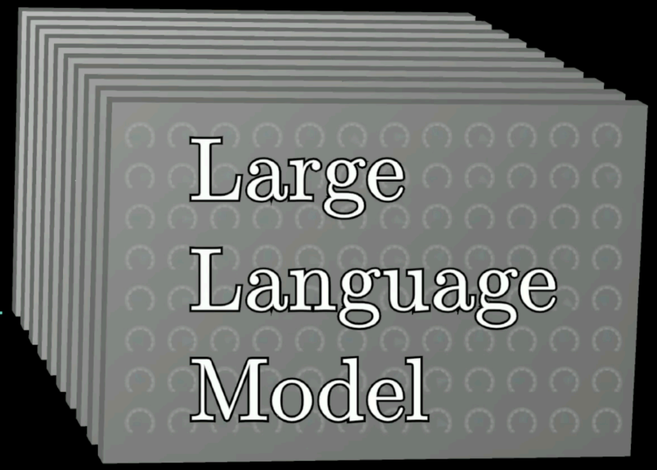
30th July 2026
Quarknet @ UMD
summer workshop

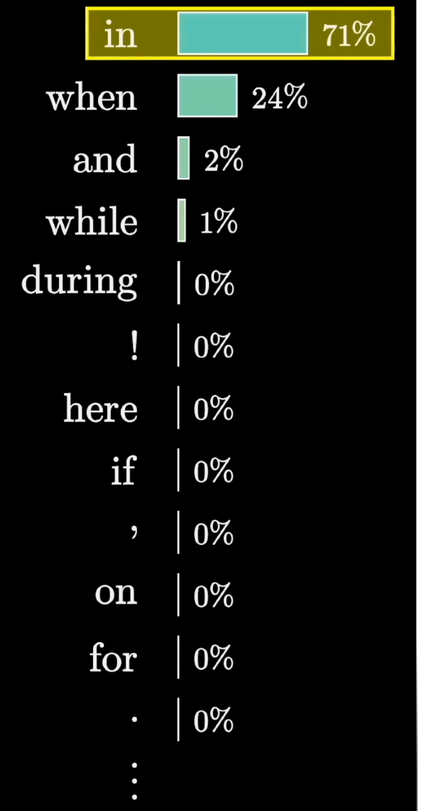
LLMs predict one token at a time

What follows is a conversation between a user and a helpful, very knowledgeable AI assistant.

User: Give me some ideas for what to do when visiting Santiago.

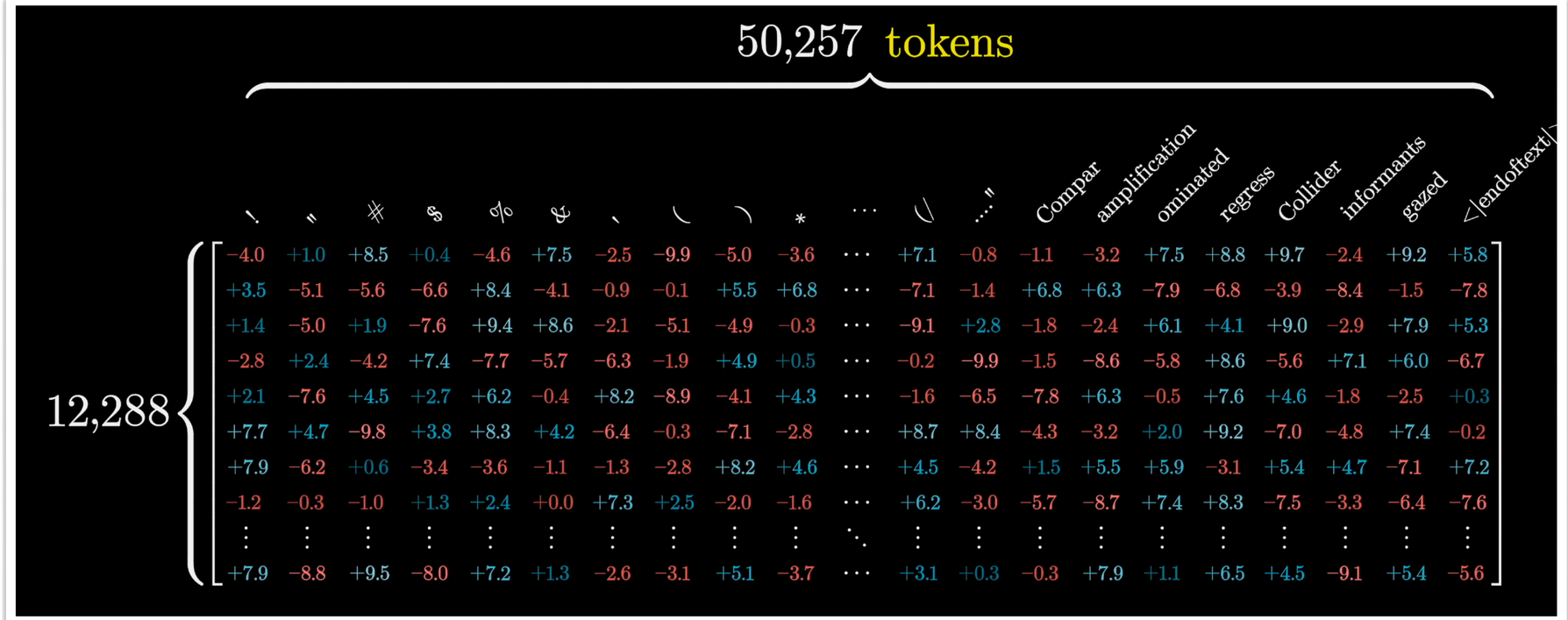
AI Assistant: Sure, there are plenty of things to do in _____



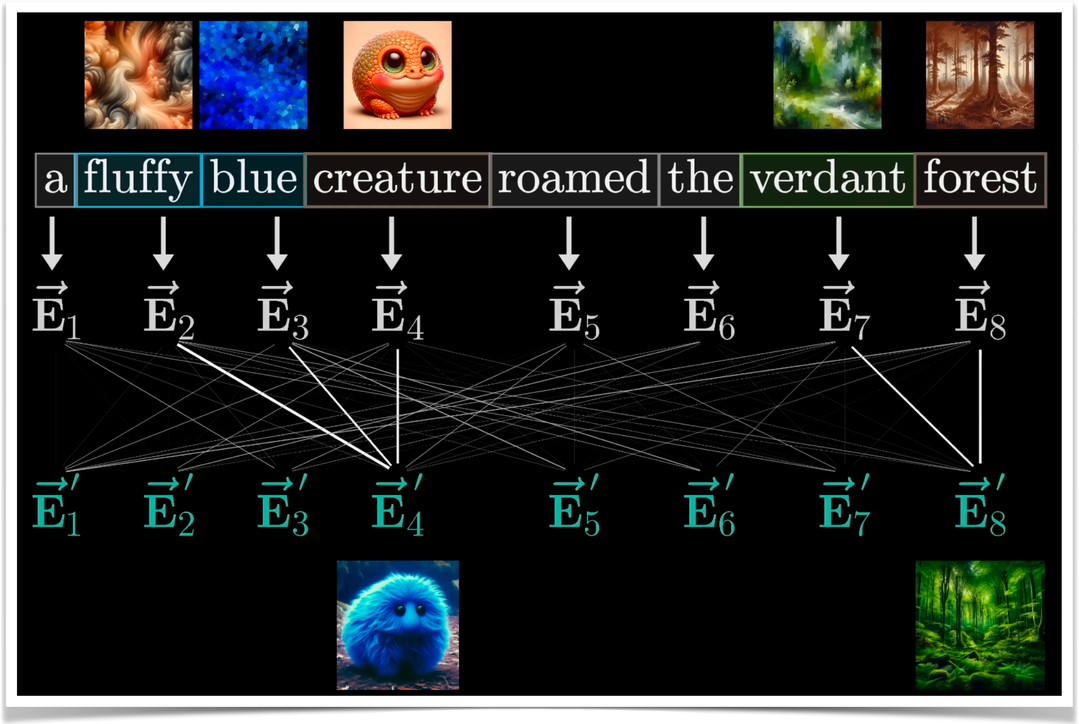


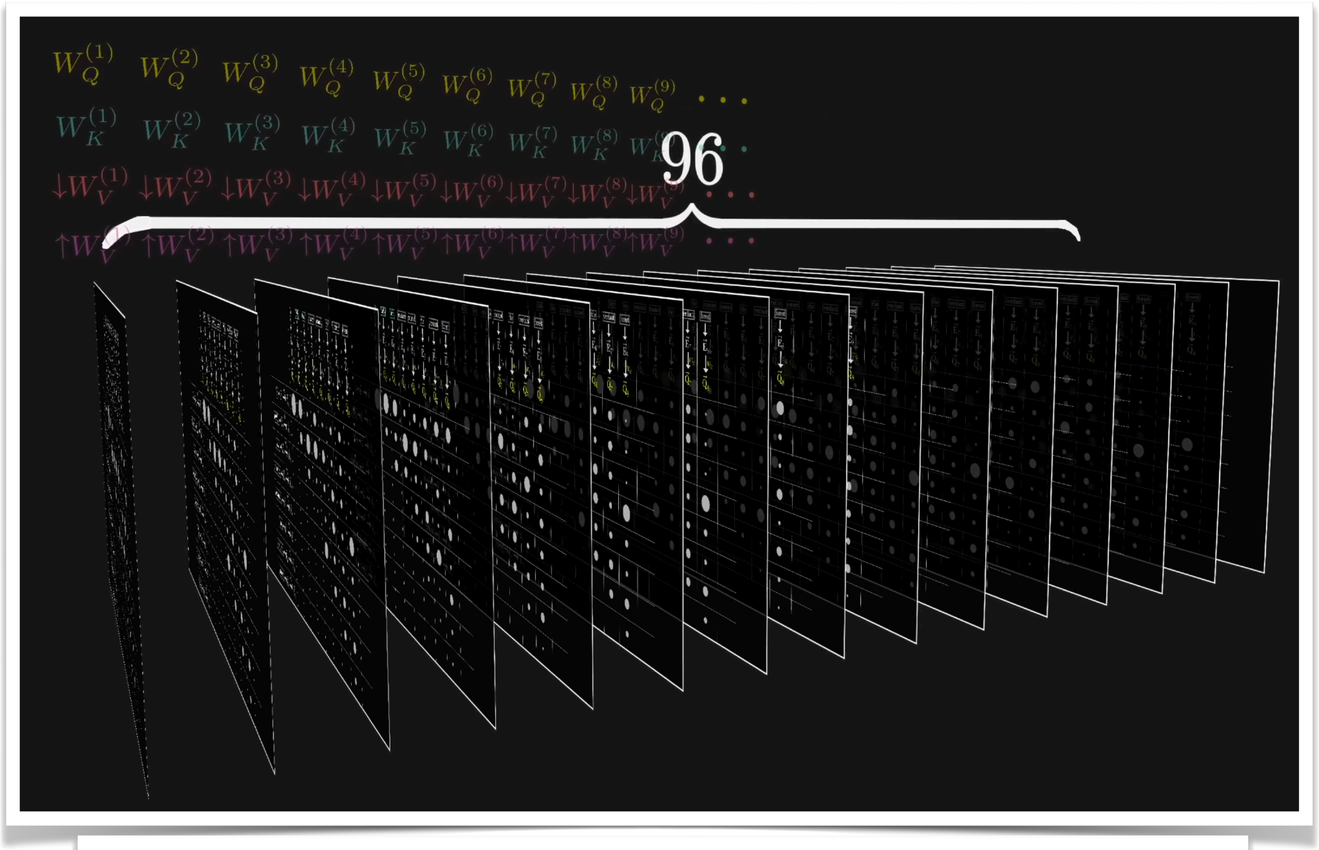
Tokenization and latent space

- **Tokenization** learned **from large text corpus**
  - Characters that often appear together merged in token
- **Each token** has a **unique embedding**
  - **Embedding learned** during **training**

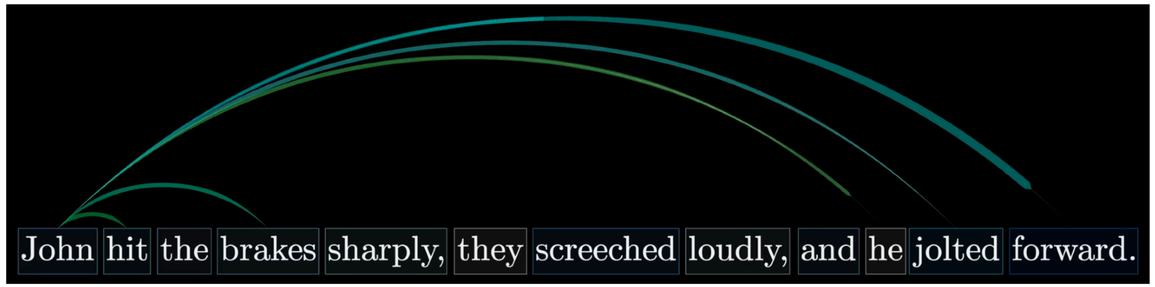


“Attention” brings context



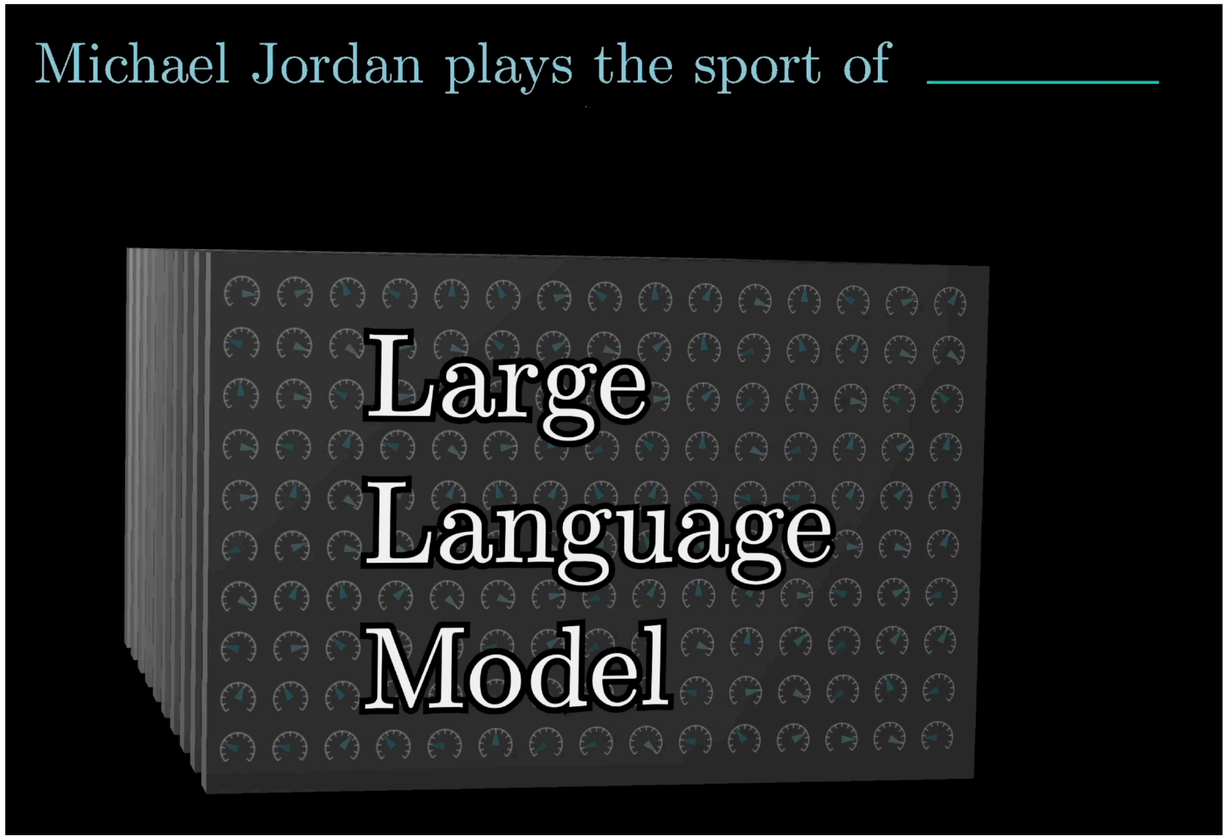


- **Attention** applies **“fluffy blue”** to **“creature”** through **3 matrix multiplications**

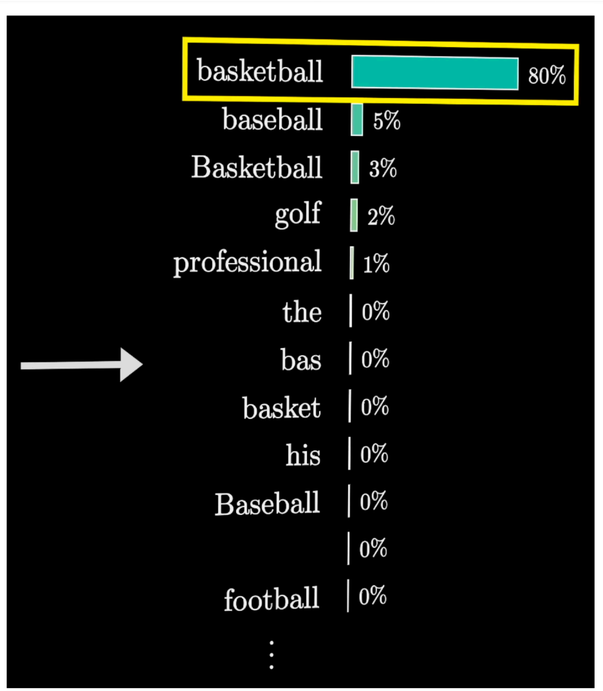


LLMs know facts

- **LLMs** not only **know meaning** of **words**, but **also facts about the world**

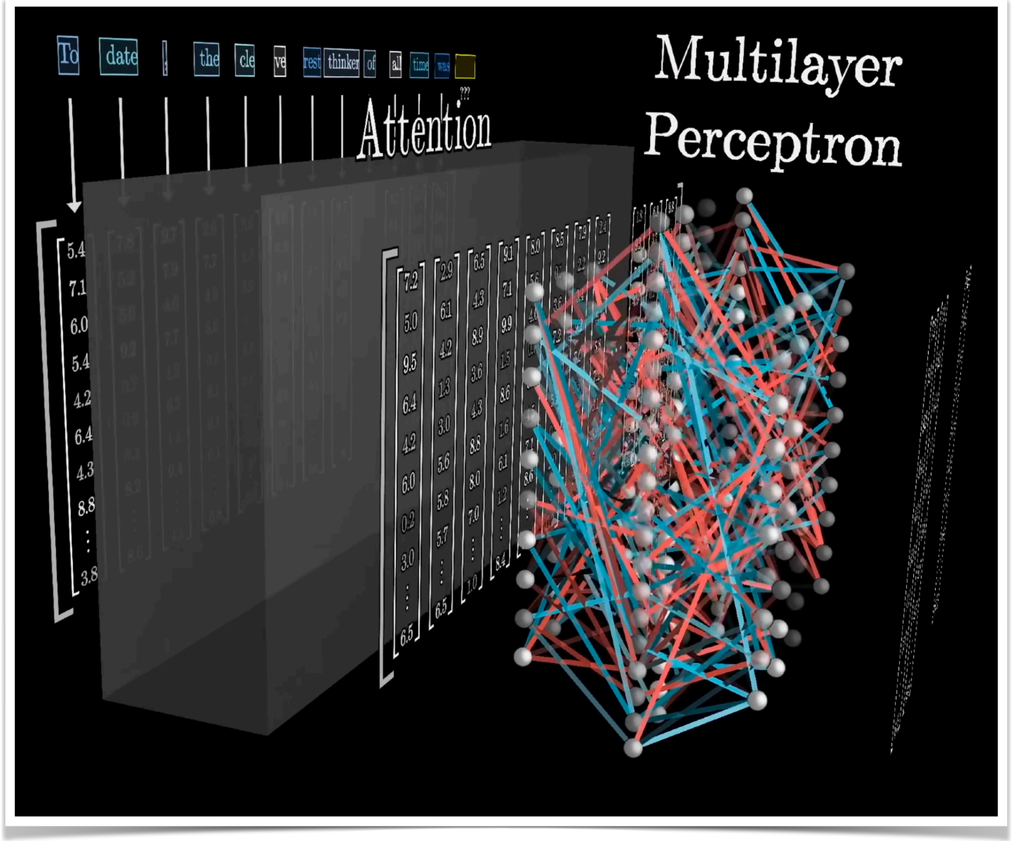


Facts like that **Michael Jordan plays basketball**, some **baseball**, some times **golf**, or that he was **born** in **1963** in **NYC**

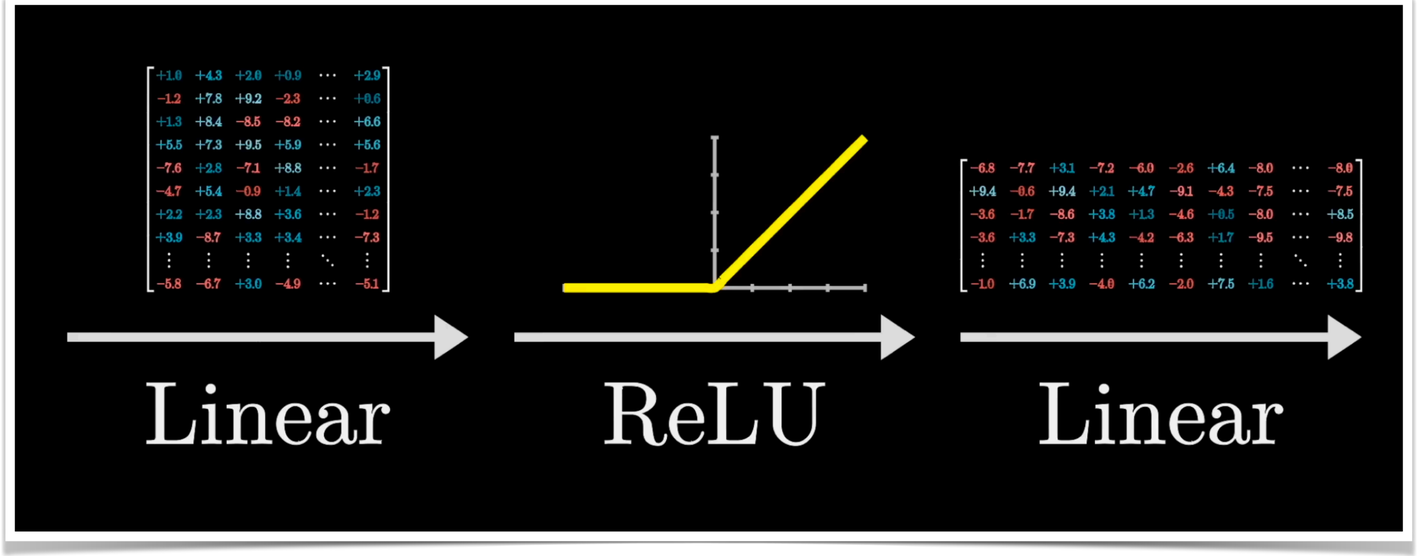


Multi-Layer Perceptrons (MLPs)

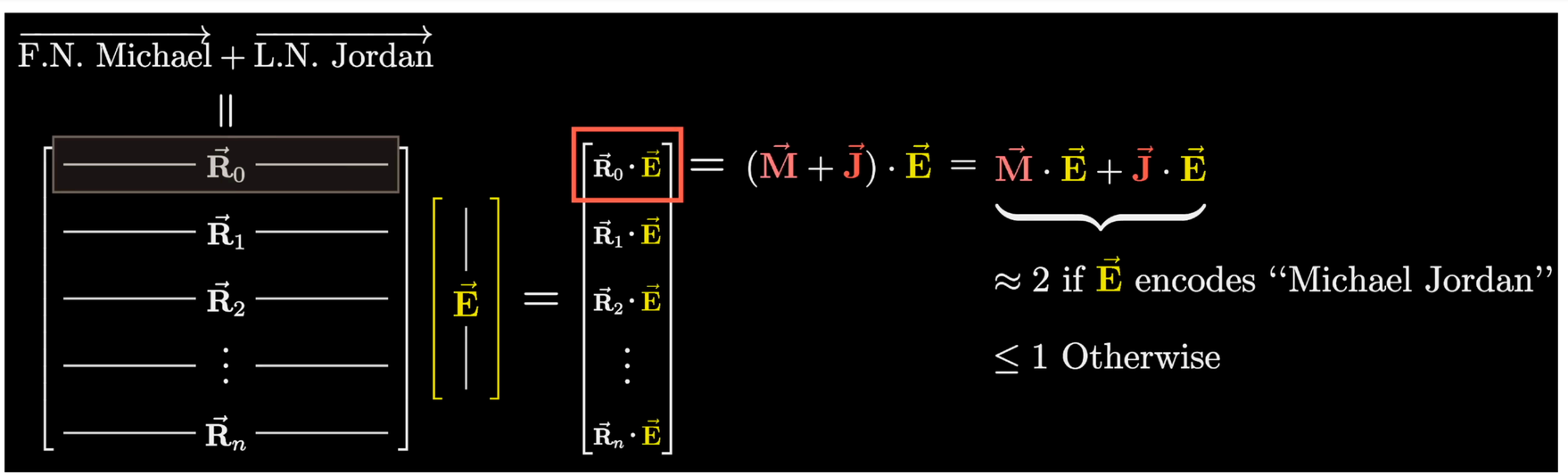
- It is believed that **facts live** in the **weights** of the **MLPs**
  - ➡ **2/3** of **all LLM weights** in **MLPs**



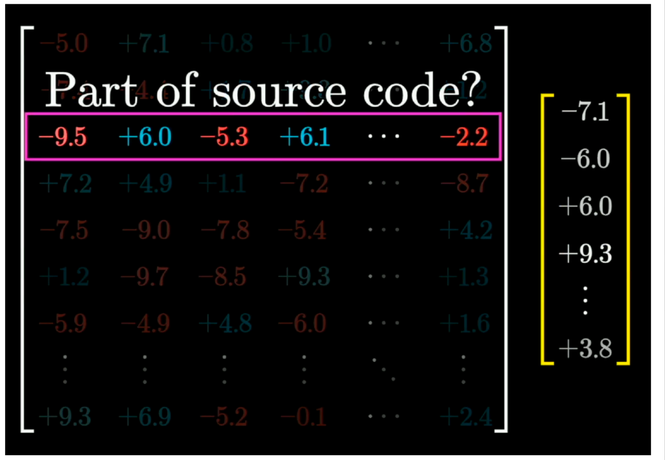
MLPs are **neural networks** with **two linear transformations** and a **non-linear neuron activation function** (ReLU) in between

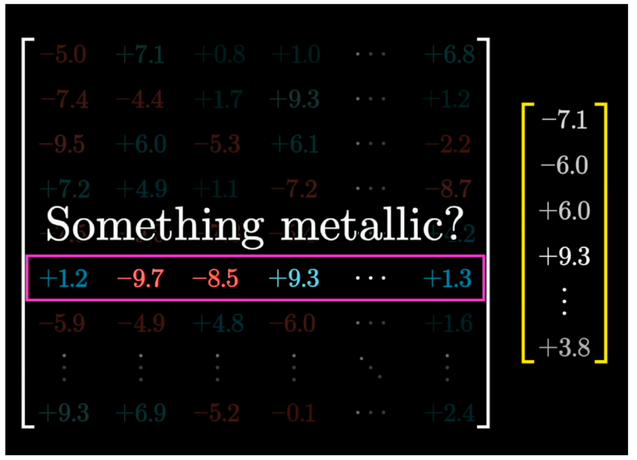


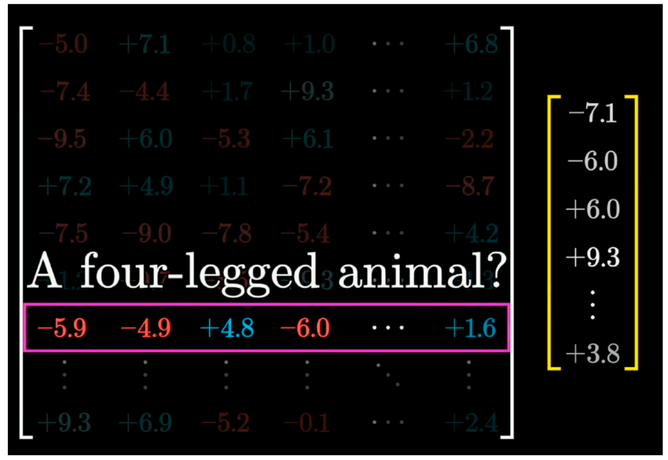
Asking questions



- **First linear transformation** seems to **ask different questions**

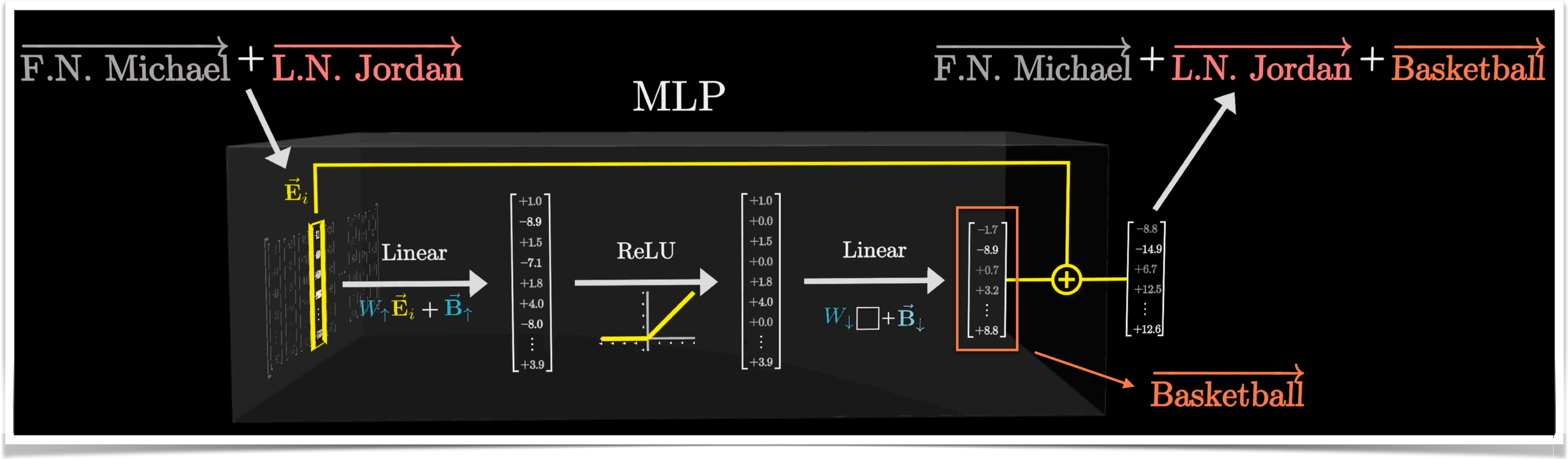




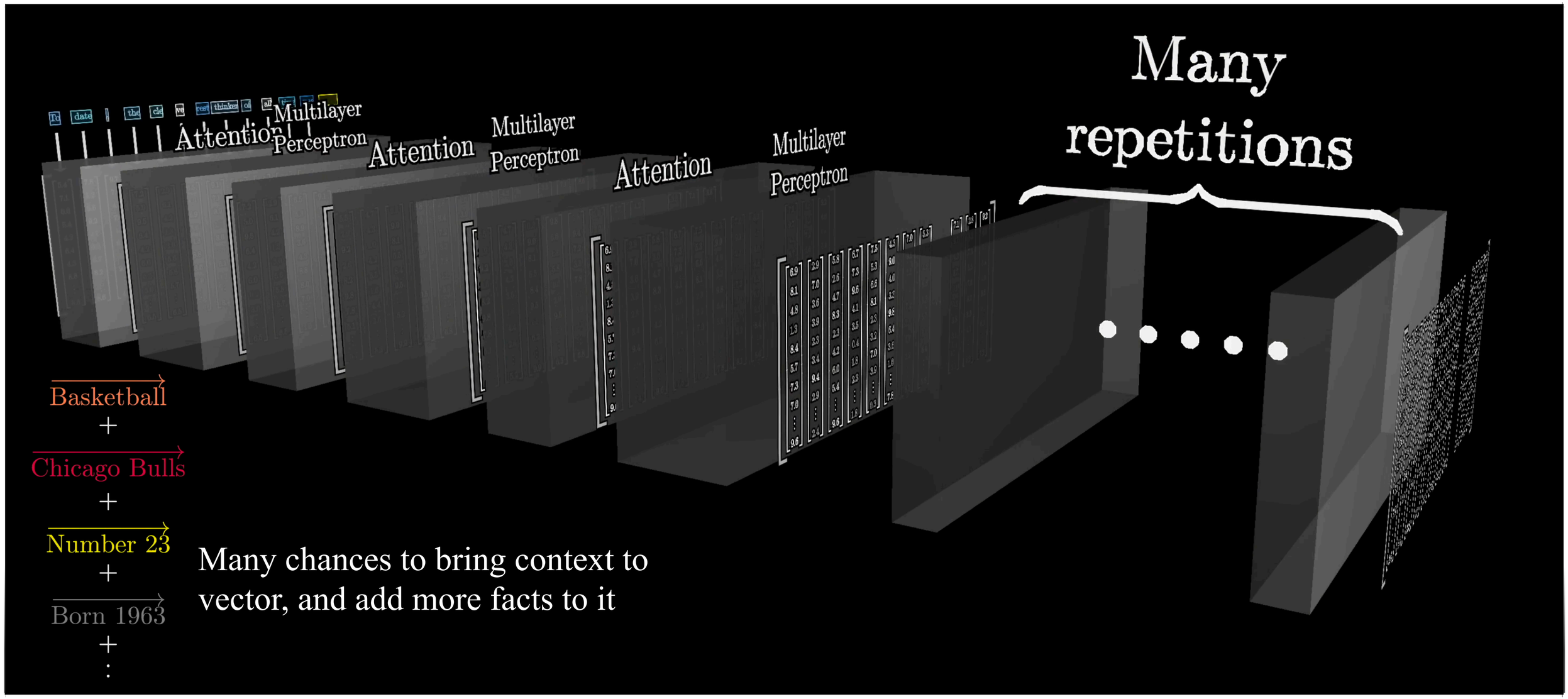


Updating meaning

- **If** the **answer** is **positive**, the **ReLU lets it go through**
- **Second linear transformation** calculates how to **update vector**
  - ➡ In this cases, it **adds "Basketball"** to the **vector** that **already** contained **"Michael Jordan"**

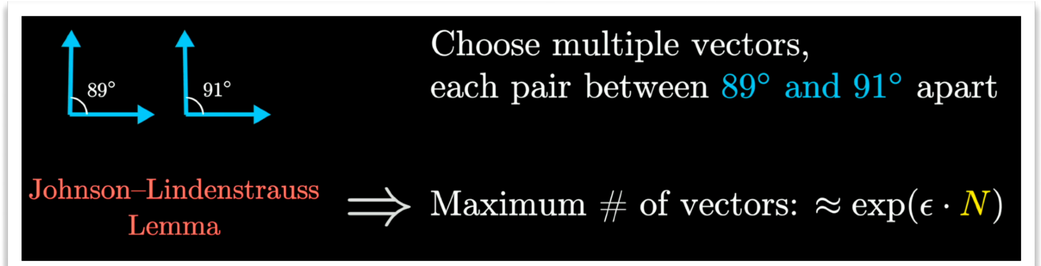


LLMs have many Attention+MLP layers



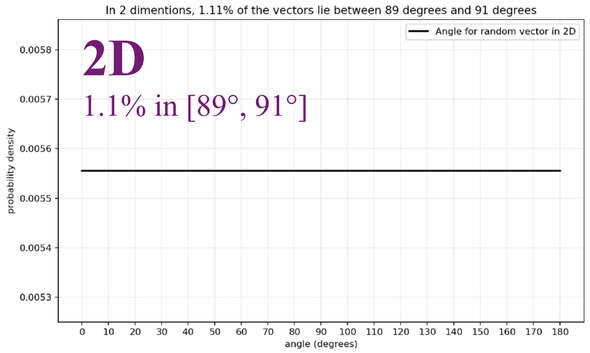
How can vectors store so much information?

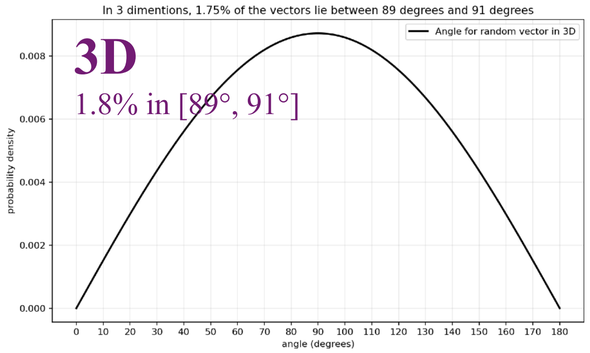
- **Latent space** in **GPT-3** has **12,288 dimensions**
  - ➡ Does that mean that **each vector** in latent space **can store at most 12,288 meanings?**
    - ✦ **Clearly not**, as vectors can contain **all words in English** as well **facts from Wikipedia** and **much more**
- **Meanings** defined **along quasi-orthogonal directions**
  - ➡ Johnson-Lindenstrauss lemma: **quasi-orthogonal vectors increase exponentially with dimension**

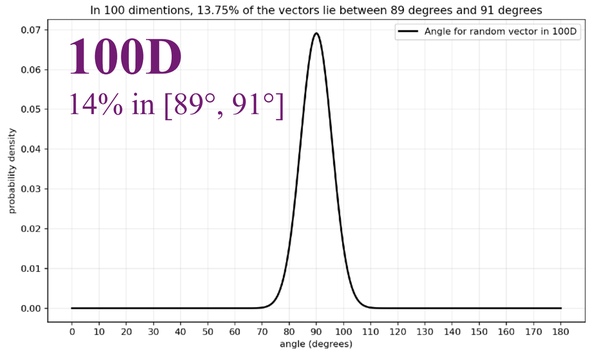


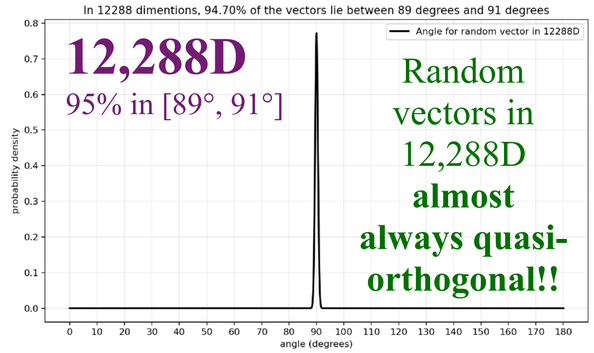
Angle between **fixed vector** and **random vector** in **N dimensions**

Can this be one **source** of **hallucinations?**

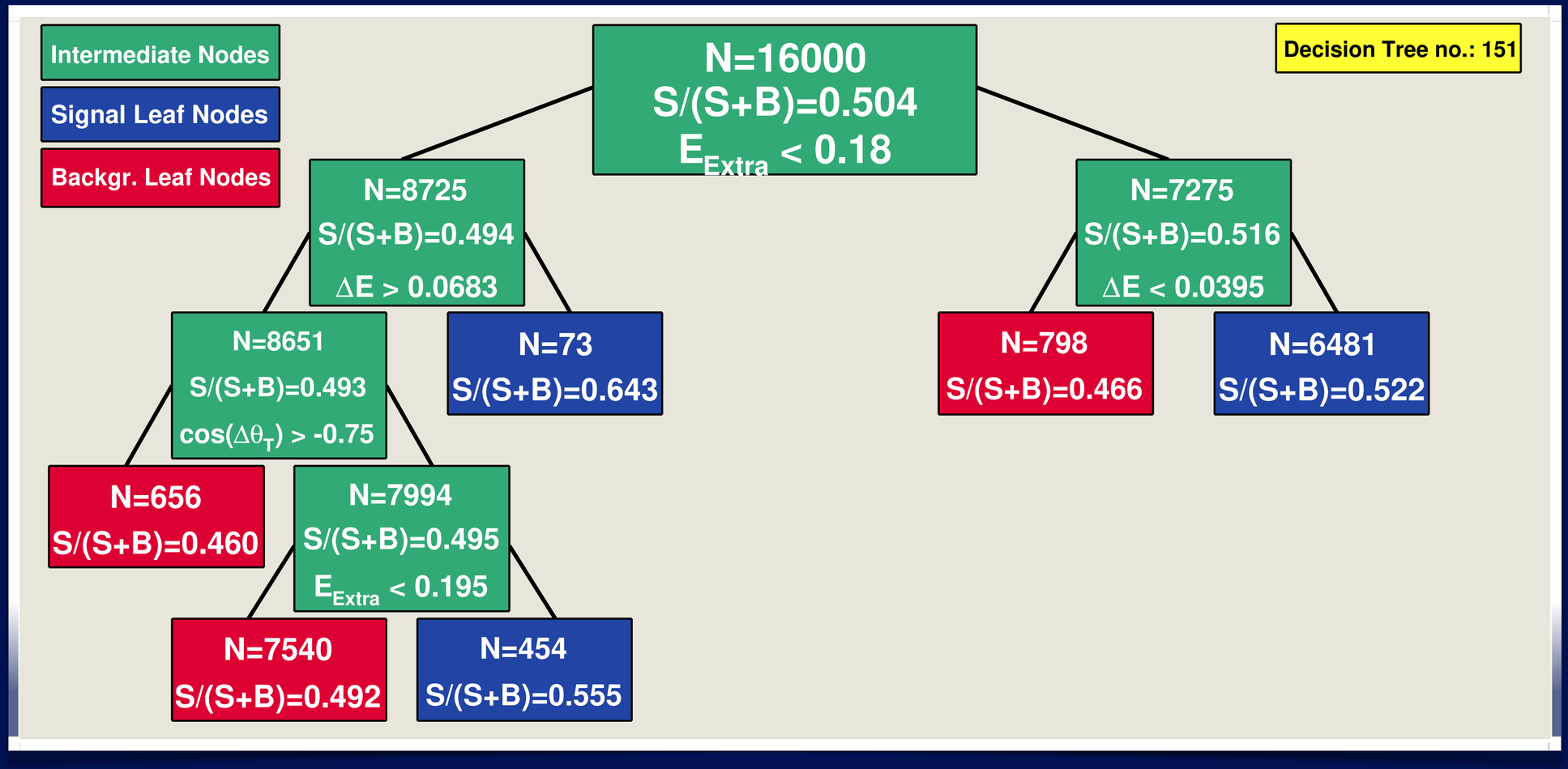




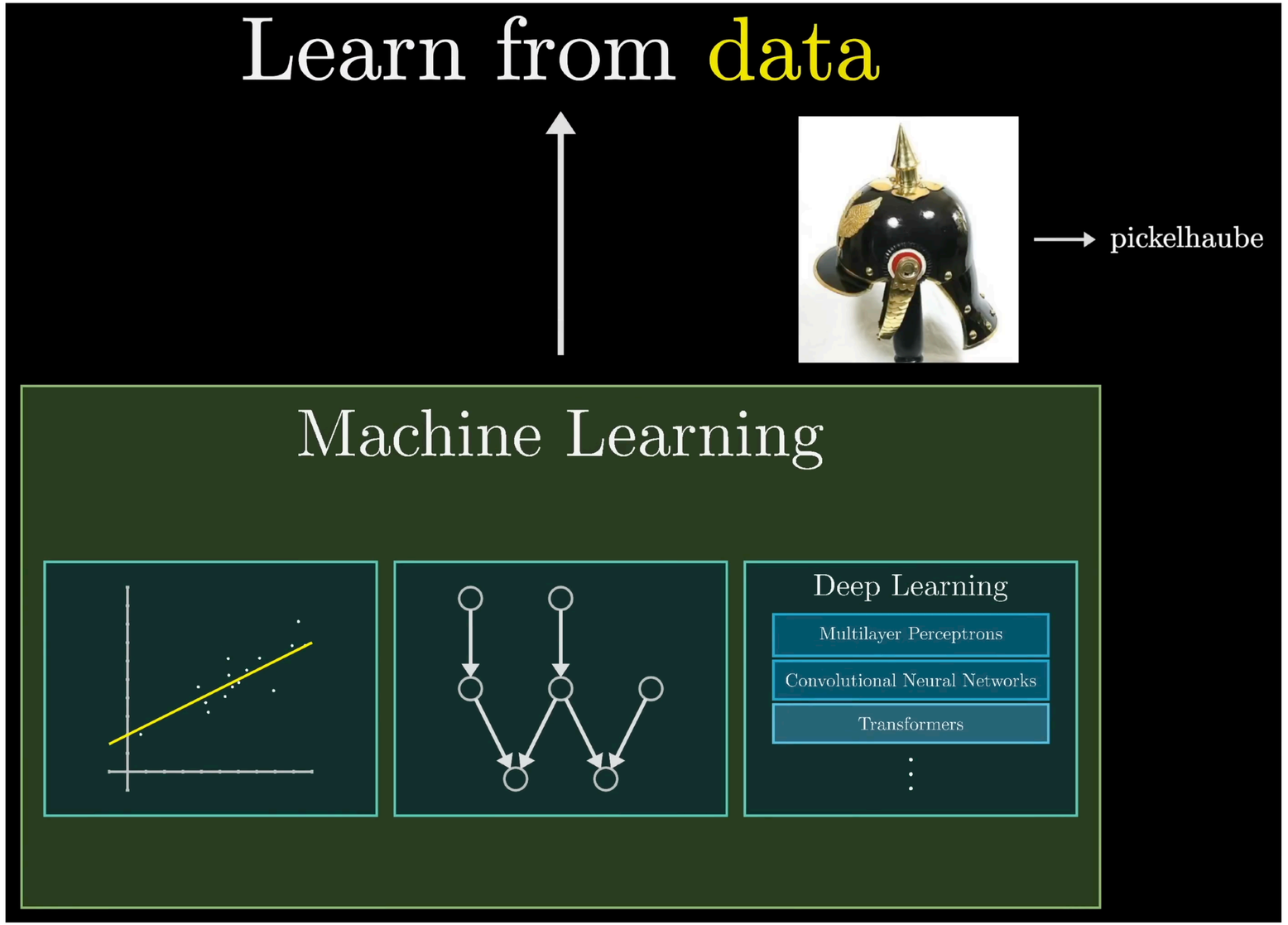




BDTs


Intermediate Nodes
Signal Leaf Nodes
Backgr. Leaf Nodes
Decision Tree no.: 151
N=16000
S/(S+B)=0.504
E_{Extra} < 0.18
N=8725
S/(S+B)=0.494
ΔE > 0.0683
N=7275
S/(S+B)=0.516
ΔE < 0.0395
N=8651
S/(S+B)=0.493
cos(Δθ$_T$) > -0.75
N=73
S/(S+B)=0.643
N=798
S/(S+B)=0.466
N=6481
S/(S+B)=0.522
N=656
S/(S+B)=0.460
N=7994
S/(S+B)=0.495
E_{Extra} < 0.195
N=7540
S/(S+B)=0.492
N=454
S/(S+B)=0.555

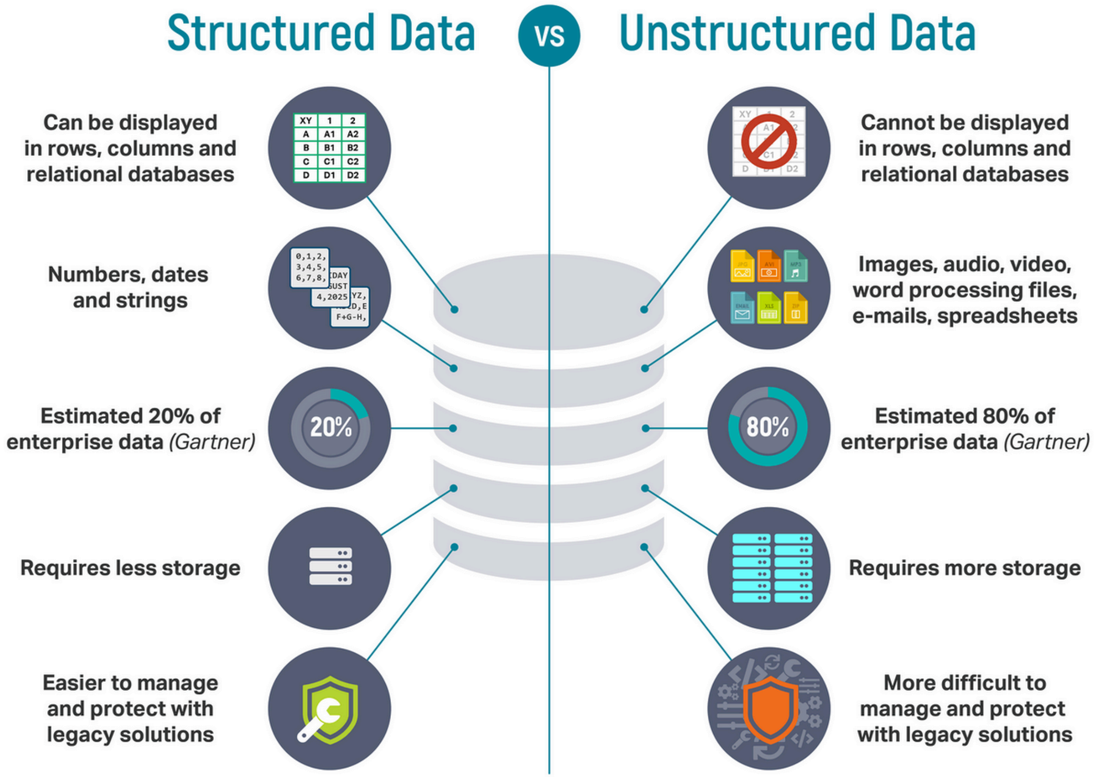
Machine learning



- **Many different techniques**, all sharing
  - ➡ **Tunable parameters**
  - ➡ **Learned** from **data**
- Neural networks, Support Vector Machines, Boosted Decision Trees, etc

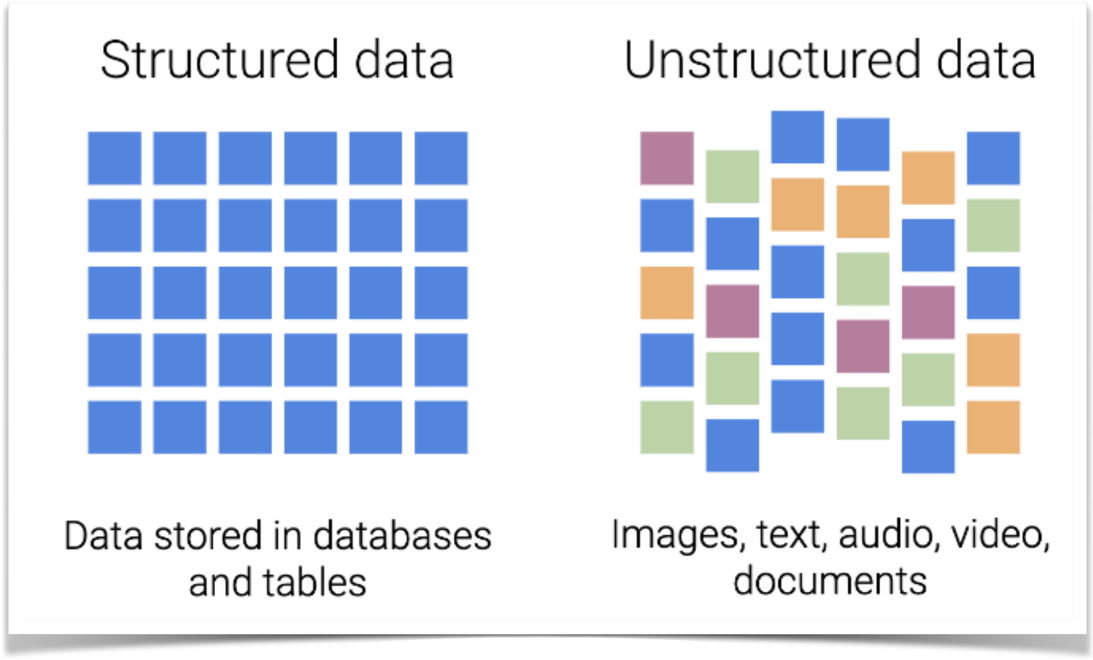
Structured vs unstructured data

- In **HEP**, why do we hear **more about BDTs** than **NNs**?
  - **We have structured data!**
    - Rows → events, columns → branches

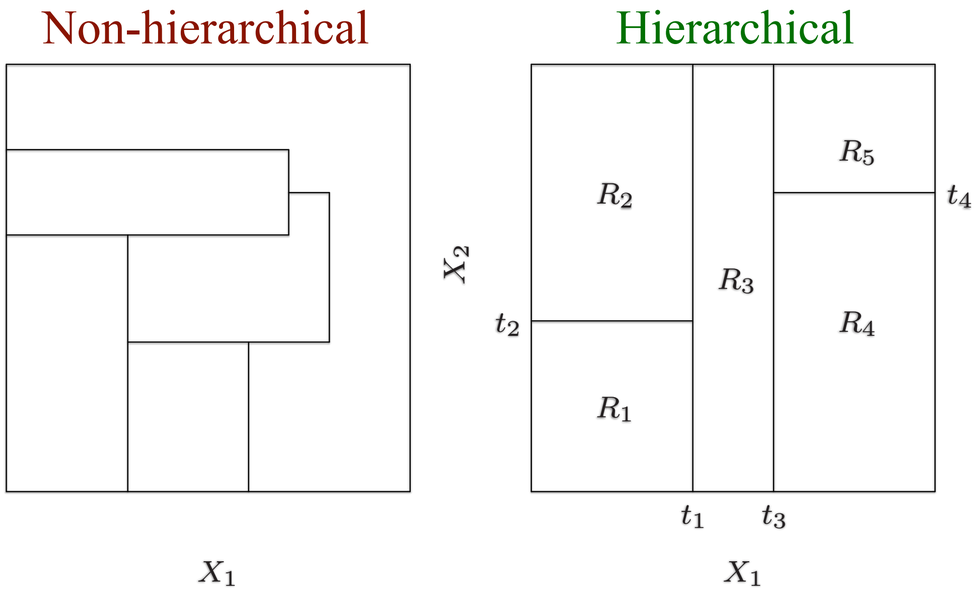


BDTs tend to perform best with **structured data** (eg, TTree)

NNs tend to perform best with **unstructured data** (eg, high res. image of CALO deposits

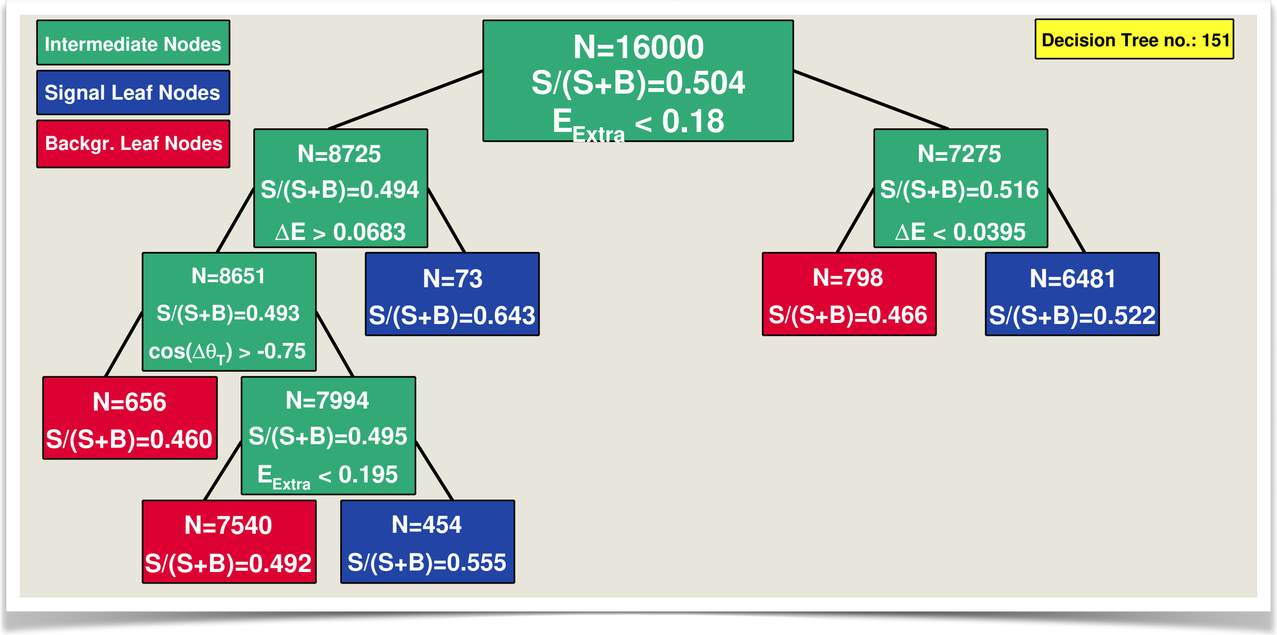


Single decision tree



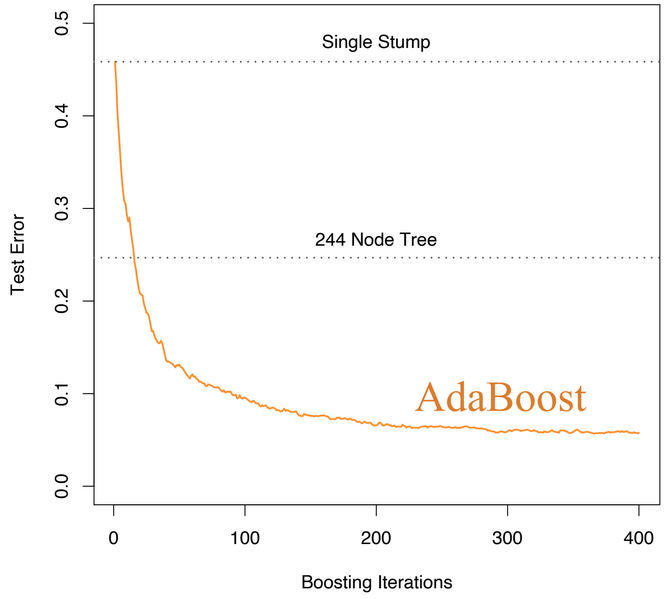
- Applies hierarchical binary cuts
- Very **simple to train**
  - Scan **each variable** for **cut** that **maximizes goal** (eg, purity)
  - **Repeat** for each subsample until reached **max depth**
  - Prune cuts that provide sufficient separation

- **Robust** against **useless inputs**
- Can do **classification**
  - Eg, signal/background
- Cand do **regression**
  - Eg, average of events in leaf

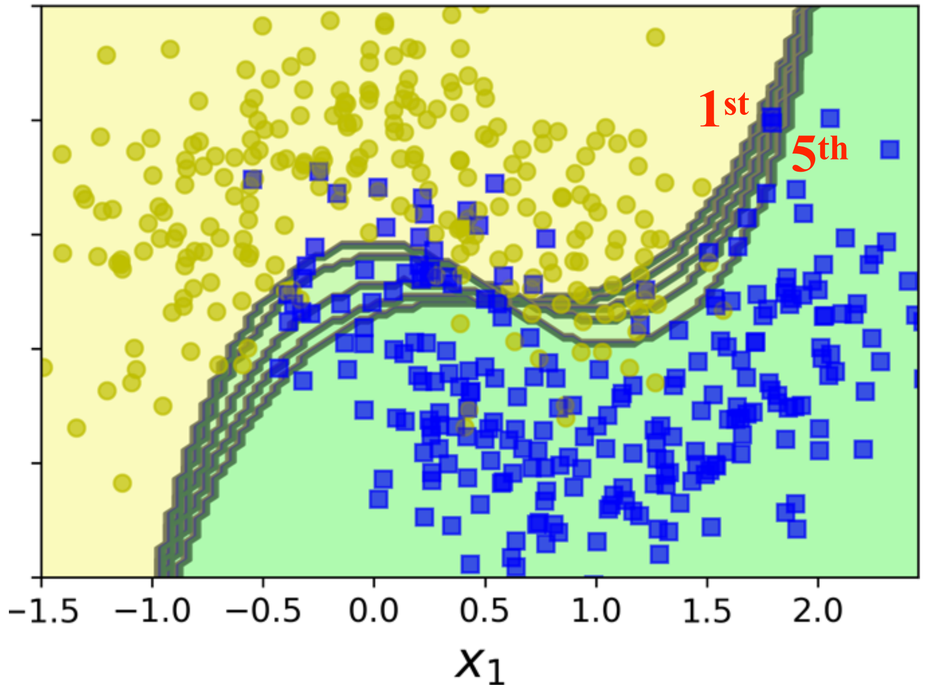


Forest of trees (BDT)

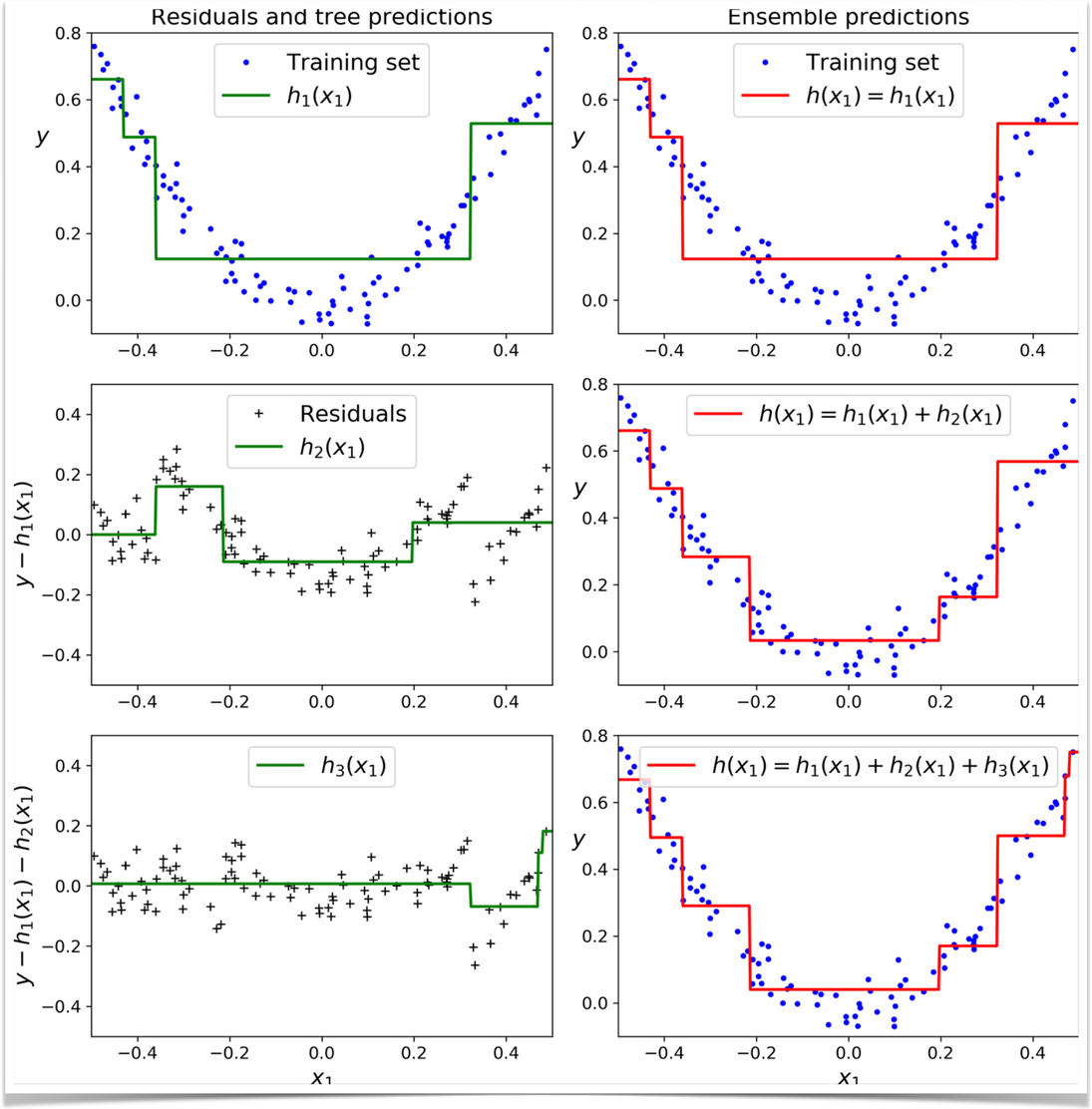
- Single trees are pretty weak classifiers
- Invented in 1997, AdaBoost uses **ensembles of weak classifiers** to make a **strong one**
  - *One of the most powerful learning ideas introduced in the last twenty-four years* (ESL)
- **Trains individual classifiers iteratively**
  - At each step, it increases the weight of the misclassified



The forest significantly outperforms a very deep (244 nodes) tree



Gradient boosted trees and XGBoost



- **GBTs** also an **ensemble of trees**
  - Trees **trained iteratively on residuals**

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\dots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

to **minimize** the **objective function** (eg, mean squares)

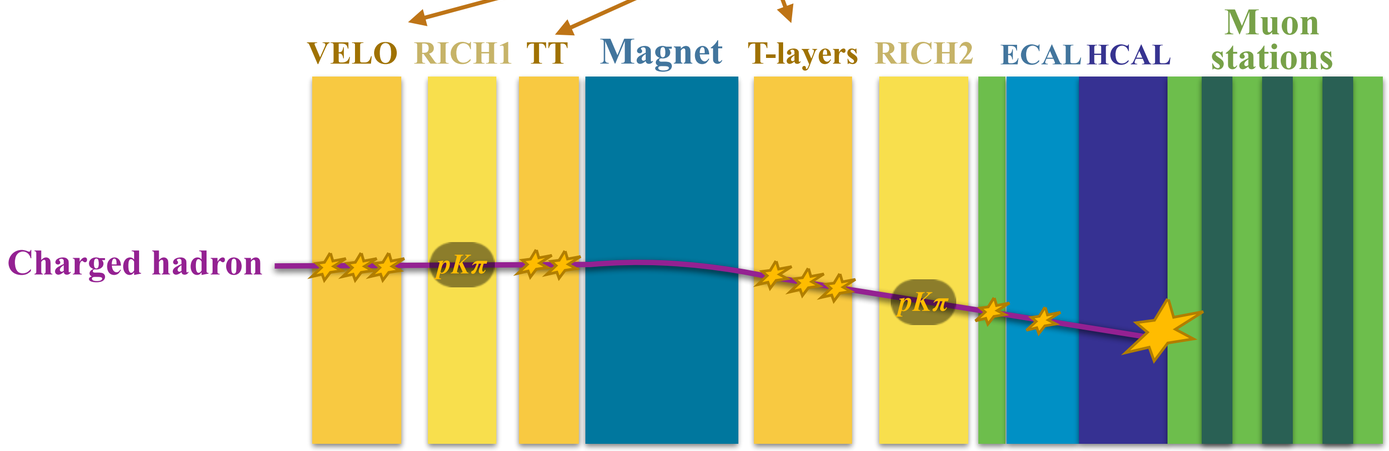
$$\text{obj}^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i)$$

  - AdaBoost is particular case with exponential objective

- **XGBoost** is **best implementation** of GBTs
  - Won 2016 Higgs Machine Learning Challenge
  - *When in doubt, use XGBoost*

For instance, modeling of LHCb trigger

- Our **Monte Carlo samples** are **huge**
  - To speed up the processing, they **simulate only** the **trackers**



- One of our **triggers relies** on the **calorimeter** to trigger on **hadrons**
  - **How can we emulate its effect??**
- We use **XGBoost**

Implementation

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
```

Import ML packages

Extremely simple implementation in python

```
TRAIN_BRANCHES = [
    'd0_PT',
    'd0_P',
    'k_L0Calo_HCAL_realET',
    'pi_L0Calo_HCAL_realET',
    'rdiff_k_pi'
]
```

Read TTree branches

```
init_frame = RDataFrame(args.tree, args.input)
input_vars = np.array(list(init_frame.AsNumpy(columns=TRAIN_BRANCHES).values())).T
dfs, output_br_names = process_directives(REGRESSION_BRANCHES, init_frame)
regression_var = dfs[-1].AsNumpy(columns=['d0_et_diff'])
```

```
if args.classifier == 'bdt':
    rng = np.random.RandomState(1)
    classif = AdaBoostRegressor(DecisionTreeRegressor(max_depth=args.max_depth),
                                n_estimators=args.ntrees, random_state=rng)
elif args.classifier == 'xgb':
    classif = xgb.XGBRegressor(n_estimators=args.ntrees, max_depth=args.max_depth)
classif.fit(cl_input_vars, regression_var['d0_et_diff'])
```

Fit BDT or XGB to data

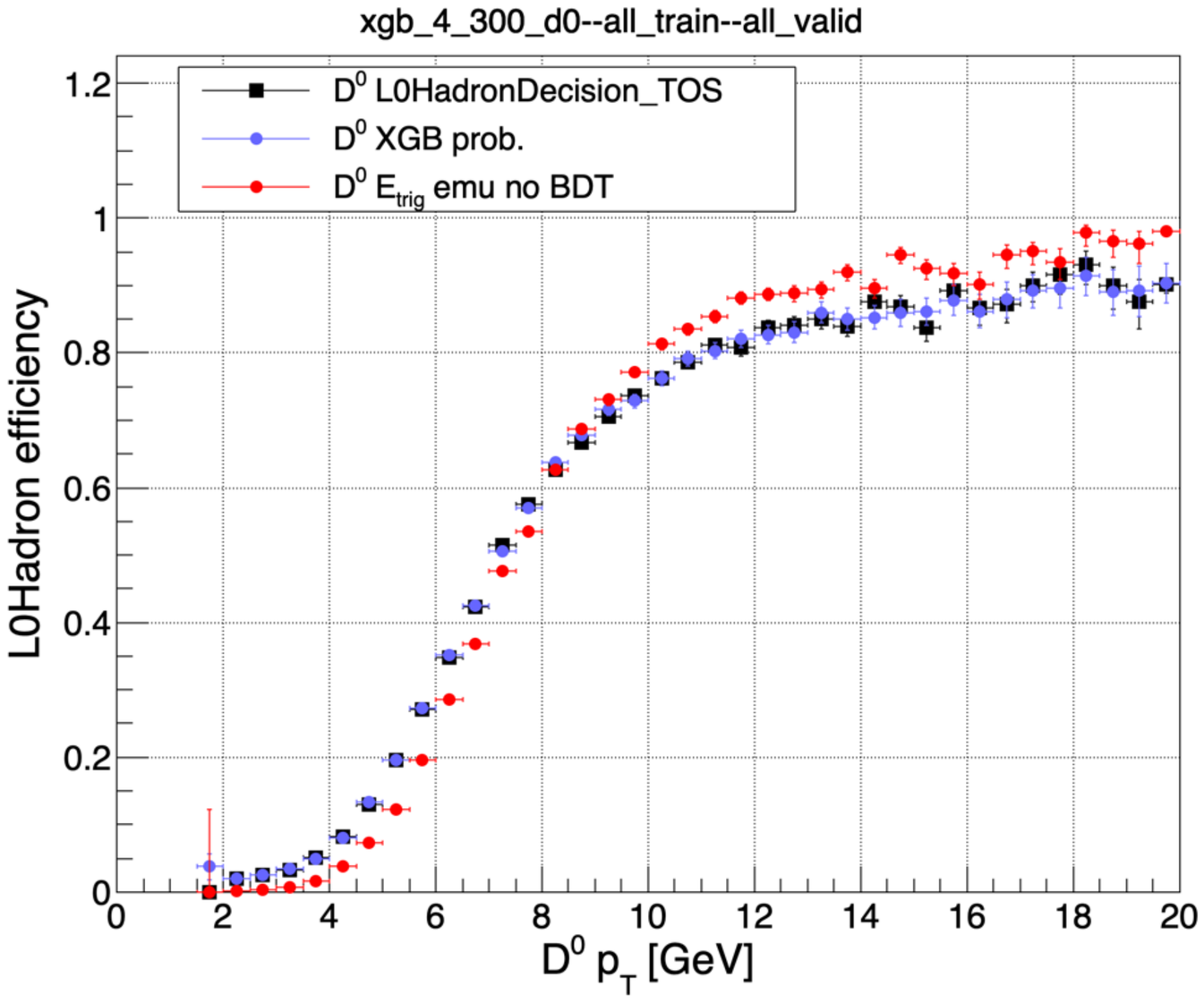
Save classifier

```
with open(args.output, 'wb') as f: pickle.dump(classif, f)
```

Read classifier and calculate output on new TTree

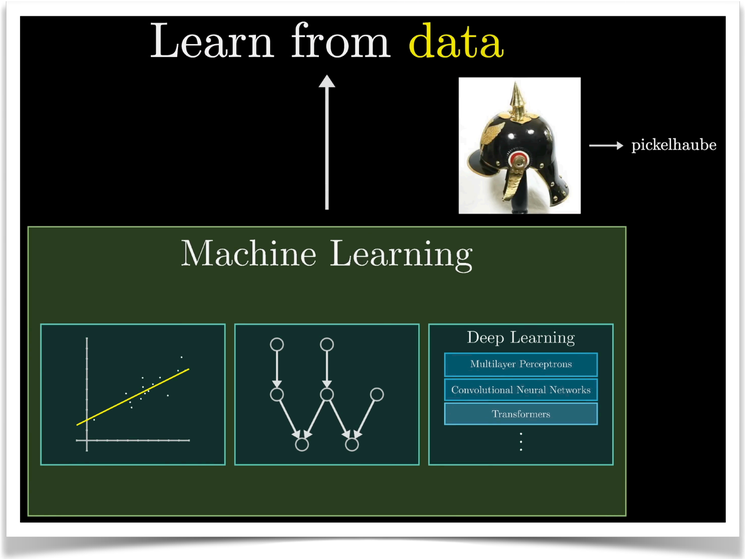
```
classif = pickle.load(open(load_file(args.load_cl), 'rb'))
d0_et_corr = classif.predict(input_vars)
output_tree['d0_et_corr'] = d0_et_corr
```

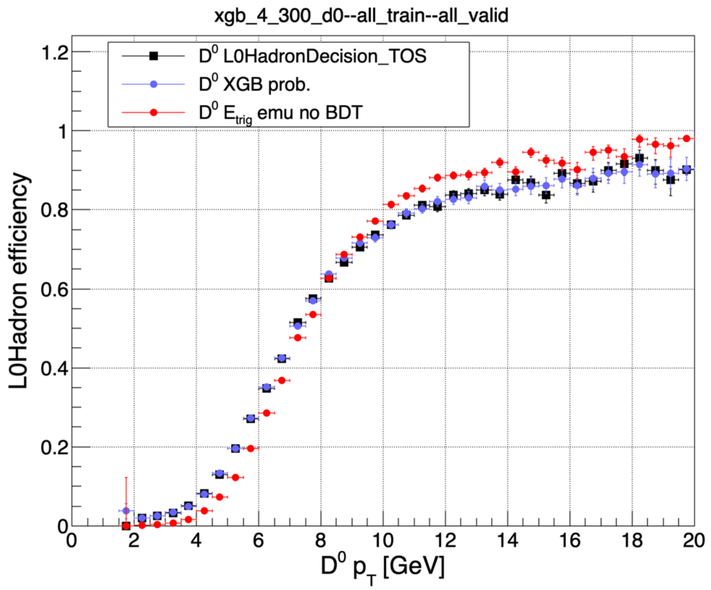
Beautiful emulation



- ~ XGB (blue) **follows** the **fully simulated curve (black)** beautifully

Conclusions





- **Machine learning** is a **powerful tool** that learns from **large amounts of data**
- **Different techniques** useful for **different problems**
- **NNs** tend to perform **best** with **unstructured data**
  - ➡ Eg, high res. image of CALO deposits
- **BDTs** tend to perform **best** with **structured data**
  - ➡ Eg, our ntuples stored in TTrees
- **BDTs** have some **nice features**
  - ➡ **Robust against useless inputs**
  - ➡ Can do **classification** (eg, signal vs background)
  - ➡ Cand do **regression**