Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

Large Language Models for Particle Physics Experiments

Christian Weber, Elena Zhivun

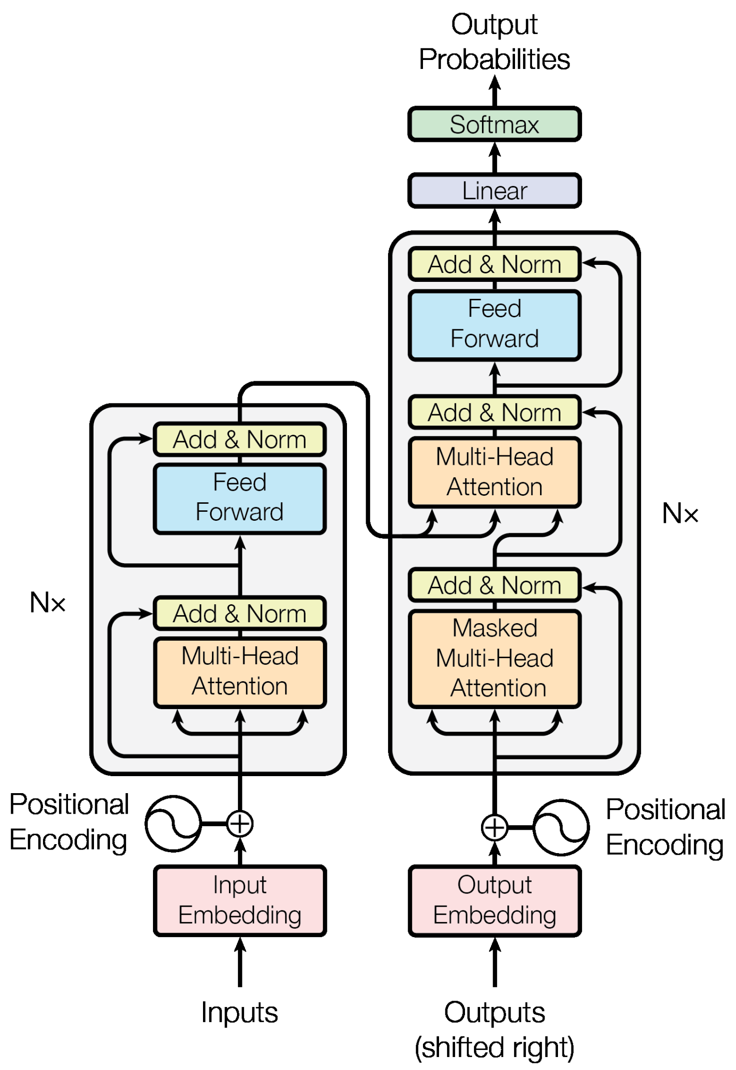
April 13th, 2013

@BrookhavenLab

Large Language Models

- Large Language Models (LLMs) are a type of neural network trained on large amounts of unlabeled text using self-supervised learning.
- 100+ billion parameters in a transformer network – deep learning model for the processing of sequential input data
- Operates on sequences of ‘tokens’:
  - words, characters, or sub-words, depending on implementation
  - represented as vectors in an n-dimensional space
- Model learns representation of language
  - Stored as model parameters
  - Concepts and meaning are represented in the relevant phase space, related concepts are closer aligned therein
- Many different LLMs available: ChatGPT, LaMDA, BLOOM, …

The Transformer – model architecture



Source: Attention Is All You Need

Brookhaven National Laboratory

Already useful in HEP I

Generate algorithm to match reconstructed jets to quarks.

Prompt and response:

ZH

I do have two c++ objects, quark1 and quark2, and a c++ vector called myJets, whose elements are called jet. For each element of myJets and quark1 or quark2 I can calculate a metric deltaR: deltaR(quark1, jet) and deltaR(quark2, jet). How can I find the two jets that minimize the metric deltaR for each, quark1 and quark2.

You can find the two jets in the `myJets` vector that minimize the metric deltaR for each `quark1` and `quark2` by iterating over each element in `myJets` and computing the deltaR for each `quark`. Here is an example code snippet that demonstrates how you can do this:

c++ Copy code

```
#include <iostream>
#include <vector>
```

Code output continued:

c++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

// Define a struct to hold the jet and its corresponding deltaR
struct JetDeltaR {
    double deltaR;
    int jetIndex;
};

// Define a function to calculate the deltaR between two objects
double calculateDeltaR(double eta1, double phi1, double eta2, doubl
    double dEta = eta1 - eta2;
    double dPhi = abs(phi1 - phi2);
    if (dPhi > M_PI) {
        dPhi = 2 * M_PI - dPhi;
    }
    return sqrt(dEta * dEta + dPhi * dPhi);
}

int main() {
    // Define the quarks and jets
    double quark1Eta = 1.0;
    double quark1Phi = 2.0;
    double quark2Eta = -1.0;
    double quark2Phi = 3.0;
```

Already useful in HEP II

Explain what 'RECAST' is
(analysis reinterpretation tool)

ZH What is RECAST in the context of the ATLAS experiment

In the context of the ATLAS experiment at CERN, RECAST (REanalysis of Cms and ATlas data) is a tool developed to facilitate the reinterpretation of experimental results. It allows researchers to quickly and easily test new theoretical models against existing ATLAS data, without the need for access to the original data or Monte Carlo simulations.

RECAST works by providing a standardized interface for submitting requests to the ATLAS collaboration to reanalyze existing data using different theoretical models. The tool then compares the results of the original analysis with those obtained using the new models, allowing researchers to quickly assess the compatibility of their theories with the data.

Find reference for RECAST

ZH Point me to a reference for RECAST

Sure, here is a reference to a paper describing RECAST:

"RECAST: Extending the Impact of Existing Analyses" by B. Nachman et al., Eur. Phys. J. C 74 (2014) 3076.

The paper describes the development and implementation of RECAST in the context of the ATLAS experiment at the Large Hadron Collider (LHC), and provides examples of how the tool can be used to reinterpret existing ATLAS data in the search for new physics.

Already useful in HEP III

Process ROOT TTree data format in Python

ZH

Write me a Python code that processes a TTree with two Branches, 'Branch1' and 'Branch2

Sure, here is an example Python code that uses ROOT to process a TTree with two branches, 'Branch1' and 'Branch2':

python Copy code

```python
import ROOT

# Open the input file
file = ROOT.TFile.Open("input.root")

# Get the input tree
tree = file.Get("tree")

# Define variables to hold the branch values
branch1_val = ROOT.Double()
branch2_val = ROOT.Int()

# Set the branch addresses
tree.SetBranchAddress("Branch1", branch1_val)
tree.SetBranchAddress("Branch2", branch2_val)
```

Output continued:

```python
# Loop over the tree entries
for i in range(tree.GetEntries()):

    # Get the current entry
    tree.GetEntry(i)

    # Do something with the branch values
    print("Entry", i, "- Branch1:", branch1_val, "Branch2:", branch2_

# Close the input file
file.Close()
```

This code opens an input file, retrieves a TTree named "tree", and sets the branch addresses for "Branch1" and "Branch2" using the `SetBranchAddress()` function.

The code then loops over the entries in the tree using a for loop, and gets the values of the "Branch1" and "Branch2" branches for each entry using the `GetEntry()` function. The values are then printed to the console using the Python `print()` function.

Finally, the code closes the input file using the `Close()` function.

Note that this is just a basic example, and you would likely want to perform some actual analysis or processing on the branch values instead of just printing them to the console. Additionally, you may need to import additional ROOT libraries or modules depending on your specific needs.

Already useful in HEP IV

Generate ATLAS analysis template

ZH

Write me a template for an AthAnalysis based ATLAS analysis

Sure! Here's a basic template for an AthAnalysis-based ATLAS analysis:

c++ Copy code

```
#include "xAODRootAccess/Init.h"
#include "xAODRootAccess/TEvent.h"
#include "xAODRootAccess/TStore.h"
#include "xAODEventInfo/EventInfo.h"
#include "xAODJet/JetContainer.h"

#include <EventLoop/Job.h>
#include <EventLoop/StatusCode.h>
#include <EventLoop/Worker.h>

class MyAnalysis : public EL::AnaAlgorithm {
public:
  MyAnalysis() {}
```

```
private:
  xAOD::TEvent* m_event; //!
  xAOD::TStore* m_store; //!

  // Add any other member variables or functions as needed
};

EL::StatusCode MyAnalysis::initialize()
{
  // Initialize xAOD access
  xAOD::Init("MyAnalysis").ignore();

  // Get the input TEvent object
  m_event = wk()->xaodEvent();
  m_store = wk()->xaodStore();

  return EL::StatusCode::SUCCESS;
}

EL::StatusCode MyAnalysis::execute()
{
  // Get the event information
  const xAOD::EventInfo* evtInfo = 0;
  m_event->retrieve(evtInfo, "EventInfo").ignore();

  // Get the jets from the event
  const xAOD::JetContainer* jets = 0;
  m_event->retrieve(jets, "AntiKt4EMTopoJets").ignore();

  // Loop over the jets and do something
  for (const xAOD::Jet* jet : *jets) {
    double pt = jet->pt();
    double eta = jet->eta();
    double phi = jet->phi();
```

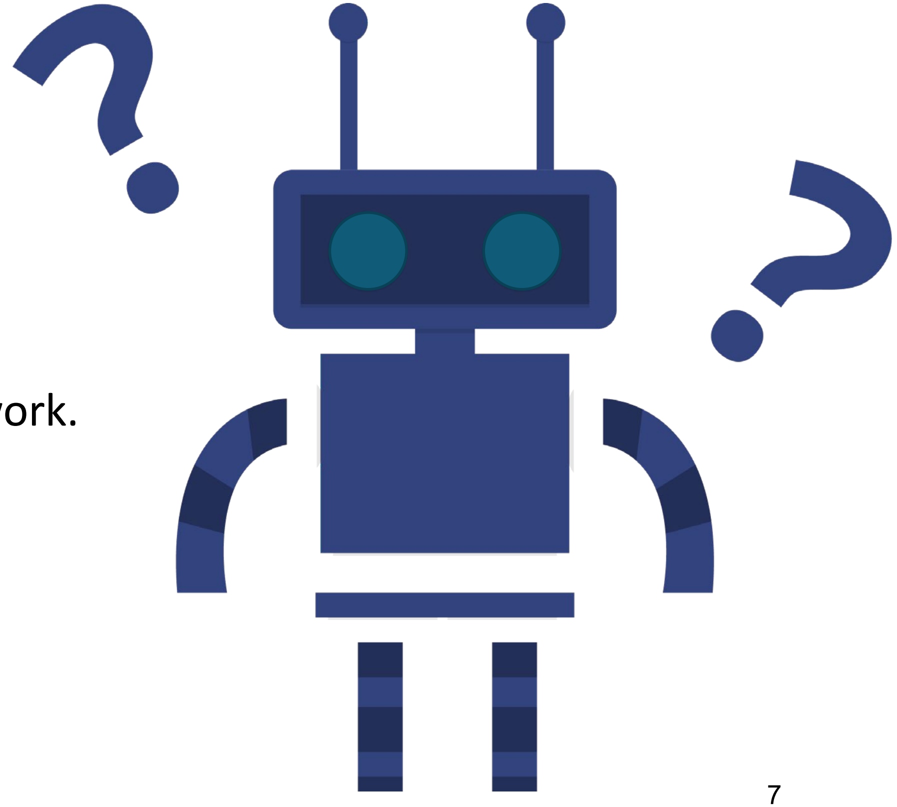
What we would like: ChatHEP

Large Language Models like ChatGPT can already provide utility for work in High Energy Physics. Particularly for less experienced collaborators.

We think however that LLMs utility in HEP can be improved.

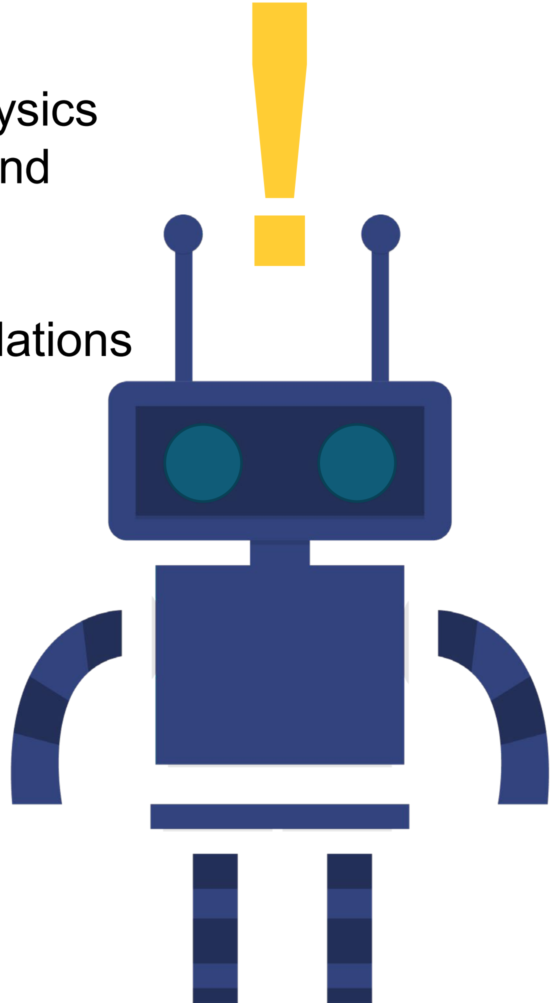
What we would like them to be able to do

- Inform us about the current jet recommendations for ATLAS analyses
- Generate Monte Carlo Event Generator configuration files for specific Beyond Standard Model physics process
- Tell me how electrons are calibrated in my very specific analysis framework.
- Give me an example of running my exotic active learning tool
- Expertise in VHDL
- Reading and understanding electronic components datasheets



Conclusion

- We expect LLMs to become a useful and common tool to help in computing and navigate information.
- We should make sure that we can take advantage of LLMs for particle physics efforts - by having a platform for training language models on our codes and other information.
- We propose to the P5 to include a LLM training platform in its recommendations that either leverages our existing computational infrastructure or is done in cooperation with the private sector.
- Expect the cost training such a model to be around 10 million USD, depending on model size and particularities of the training.



Thank you!