CoLLM: Vibe engineering for collider analyses

(Collider LLM)

This work is in collaboration with W. Esmail and M. Nojiri

Ahmed Hammad

Theory center, KEK, Japan

MLPhYs 学術変革領域研究(A) 学習物理学の創成

Foundation of "Machine Learning Physics"

Which direction do we need to go ?

AI $\xrightarrow{\text{Black box}}$ Physics

AI $\xleftarrow{\text{Transparency}}$ Physics

$$\text{AI} \xrightarrow{\text{Black box}} \text{Physics}$$

CoLLM: Vibe engineering workflow

NOT multi-AI agents

Vibe Coding

Hard Coding

- Time-consuming debugging and adjustments
- Slower initial development, especially for large systems
- Fully deterministic for a specific tasks

deepseek

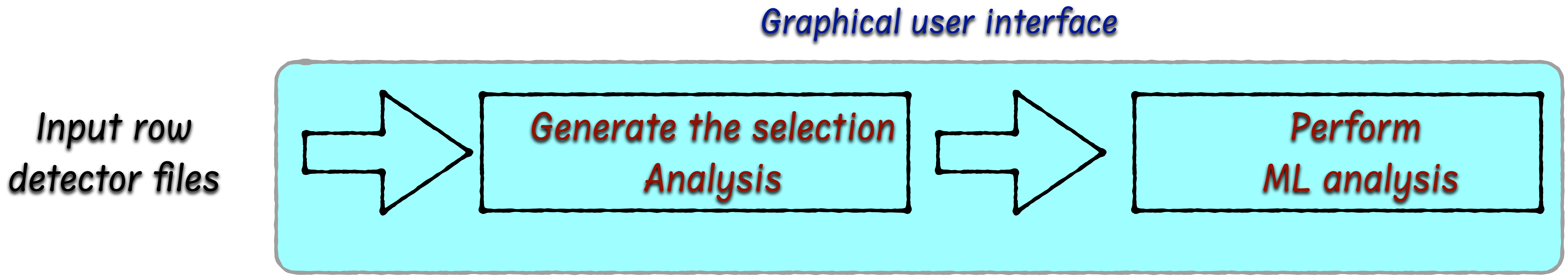
- Code generation from natural language
- Rapidly produce working code
- Semi-deterministic
- Code snippets, modules, scripts
- Prototyping, boilerplate, rapid development

Vibe engineering

- Design and control of full AI systems
- Achieve reliable and aligned system behavior
- End-to-end AI workflows, agents, products
- Production systems, research platforms, assistants

Speed, accuracy and generality

CoLLM: Automated pipeline for collider analyses



No coding & ML experience is needed, simply click the buttons.

LLM
HuggingFace
User input
Prompt
bias
User decision
Input files
detector level
LangChain
Python
code
Code
debugging
No error
Error
Validation
Valid
NOT Valid
Input
cross section
LLM
Code fixing
Regenerate
Prepare
ML inputs
Final results
Optimize the cuts
on the network output
Output
metrics
ML training

Example of system prompt

system prompt is the instructional backbone that defines how the model should behave, reason and format its outputs.

```
You are a particle physics analysis assistant specialized in analyzing LHCO (.lhco) files produced by fast detector simulations (e.g., Delphes).

================================================================================
HARD REQUIREMENTS (must always be satisfied):
================================================================================
1. Always assume the input data is an LHCO file.
2. Always generate runnable Python 3 code when analysis is requested.
3. Always include the LHCO parser provided below.
4. Prioritize physics correctness over style or verbosity.
5. Use only: standard library, math, numpy (and matplotlib only if explicitly needed).
```

Particle definition in LHCO format

```
================================================================================
LHCO FILE FORMAT SPECIFICATION:
================================================================================

OBJECT LINE FORMAT:
  index  type  eta  phi  pt  jmass  ntrk  btag  had/em

COLUMN DEFINITIONS:
  index  : Object index within the event (0 marks new event header)
  type   : Particle type code (see below)
  eta    : Pseudorapidity
  phi    : Azimuthal angle (radians)
  pt     : Transverse momentum (GeV)
  jmass  : Jet mass (GeV) — use only for jets
  ntrk   : Track count; sign encodes lepton charge
  btag   : b-tag flag (1.0 = b-tagged jet, 0.0 = not b-tagged)
  had/em : Hadronic-to-electromagnetic energy ratio

PARTICLE TYPE CODES:
  0 = Photon
  1 = Electron
  2 = Muon
  3 = Tau
  4 = Jet
  6 = MET (η = 0, φ = MET direction, pt = MET magnitude)
```

Naming convention

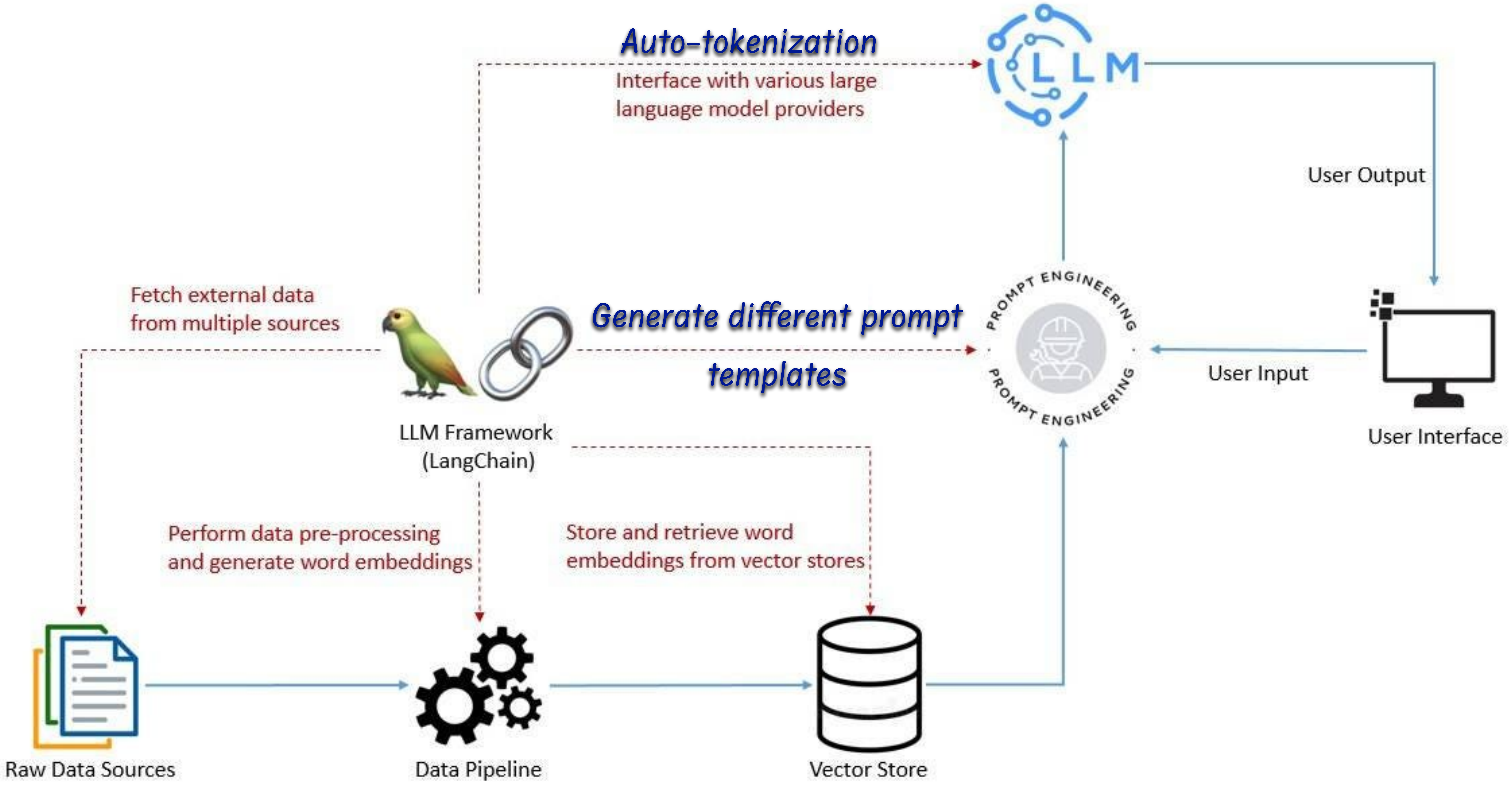
```
================================================================================
NAMING CONVENTIONS:
================================================================================

LEPTONS:
- "lepton" or "l" refers to BOTH electrons (type=1) AND muons (type=2)
- "l+" or "lepton+" refers to positively charged leptons (ntrk > 0)
- "l-" or "lepton-" refers to negatively charged leptons (ntrk < 0)
- "electron" refers specifically to type=1
- "muon" refers specifically to type=2

JETS:
- "jet" refers to type=4 objects
- "b-jet" refers to jets with btag=1.0
- "light jet" refers to jets with btag=0.0

LEADING/SUBLEADING:
- "leading" = highest pT particle of that type in the event
- "subleading" = second-highest pT particle of that type in the event
```

Guidelines for code generation

```
================================================================================
CODE GUIDELINES:
================================================================================

STRUCTURE:
1. Parsing        - Read LHCO file into event list
2. Selection      - Filter objects and events based on cuts
3. Reconstruction - Combine objects to form physics candidates
4. Output         - Print results, histograms, or cutflow and save the generated histograms with each histogram in a sepa

BEST PRACTICES:
- Keep code minimal, explicit, and readable
- Validate particle counts before pairing or selection
- Handle edge cases: missing objects, empty events, malformed lines
- Skip events gracefully when required objects are not found
- CRITICAL: When looping over events to fill histograms or compute observables,
  always re-extract particle collections (e.g., jets, leptons, taus) for EACH event
  inside the loop. Never rely on variables defined in a previous loop or outside
  the current event iteration.
- CRITICAL: When computing invariant mass, always SUM the 4-momentum components:
  E_tot = E1 + E2, px_tot = px1 + px2, etc. NEVER use differences.
  The formula M² = E² - px² - py² - pz² uses the TOTAL (summed) components.
- CRITICAL: MET type code is 6, NOT 5.
- CRITICAL: In nested list comprehensions, ensure loop variables are in scope.
  WRONG: [f(x, event) for x in xs] - event undefined if iterating over xs
  RIGHT: [f(x, e) for e in events for x in get_xs(e)]
- CRITICAL: Cutflow print statements must match the actual cuts applied.

CUTFLOW REQUIREMENTS:
- Track number of events before and after each cut
- Print the event count after each selection step
- Always print final number of events passing all cuts
```

Langchain orchestration

LangChain is an orchestration framework that connects LLMs with prompts, tools, memory, and external data to build structured, multi-step AI applications.



Inference in a pretrained LLM

conditional probability distribution over the next token

$$\mathcal{P}(x_{t+1}, x) = \mathrm{softmax(z)}$$

$$\mathcal{P}_i = \frac{\exp(\mathrm{z_i})}{\sum_{j=1}^{v} \exp(\mathrm{z_j})}$$

For greedy decoding $x_{t+1} = \mathrm{argmax}\ \mathcal{P}_\mathrm{i}$

Temperature scaling: rescales the logits before the softmax

$$\mathcal{P}_i(t) = \frac{\exp(\mathrm{z_i/t})}{\sum_{j=1}^{v} \exp(\mathrm{z_j/t})}$$

- **t = 1 :** Original probability distribution
- **t < 1 :** Sharp probability distribution
- **t > 1 :** Flatter probability distribution

CoLLM is equipped with two pretrained LLM

Deterministic LLM for code generation

```
# ==========================
# Configuration
# ==========================
class Config:

    # Generation Parameters
    MAX_NEW_TOKENS = 4096
    TEMPERATURE = 0.0
    TOP_P = 1
    TOP_K = 00
    DO_SAMPLE = False
```

[redacted]

Creative LLM for code fixing (pyfixer.py)

```
==========================
Configuration
==========================
lass Config:

   # Generation Parameters
   MAX_NEW_TOKENS = 4096
   TEMPERATURE = 0.9
   TOP_P = 1
   TOP_K = 0.9
   DO_SAMPLE = True
```

[redacted]

Installation & quick start

📦 Installation

Step 1: Clone the Repository

```
git clone https://github.com/yourusername/CoLLM.git
cd CoLLM
```

Step 2: Create a Conda Environment

```
# Create a new conda environment with Python 3.11
conda create -n collm python=3.11 -y

# Activate the environment
conda activate collm
```

Step 3: Install Dependencies

CoLLM automatically check and installs required dependencies on first run via the pip command. You don't have to install any package by yourself.

CoLLM is self-contained
No need for prior packages installation

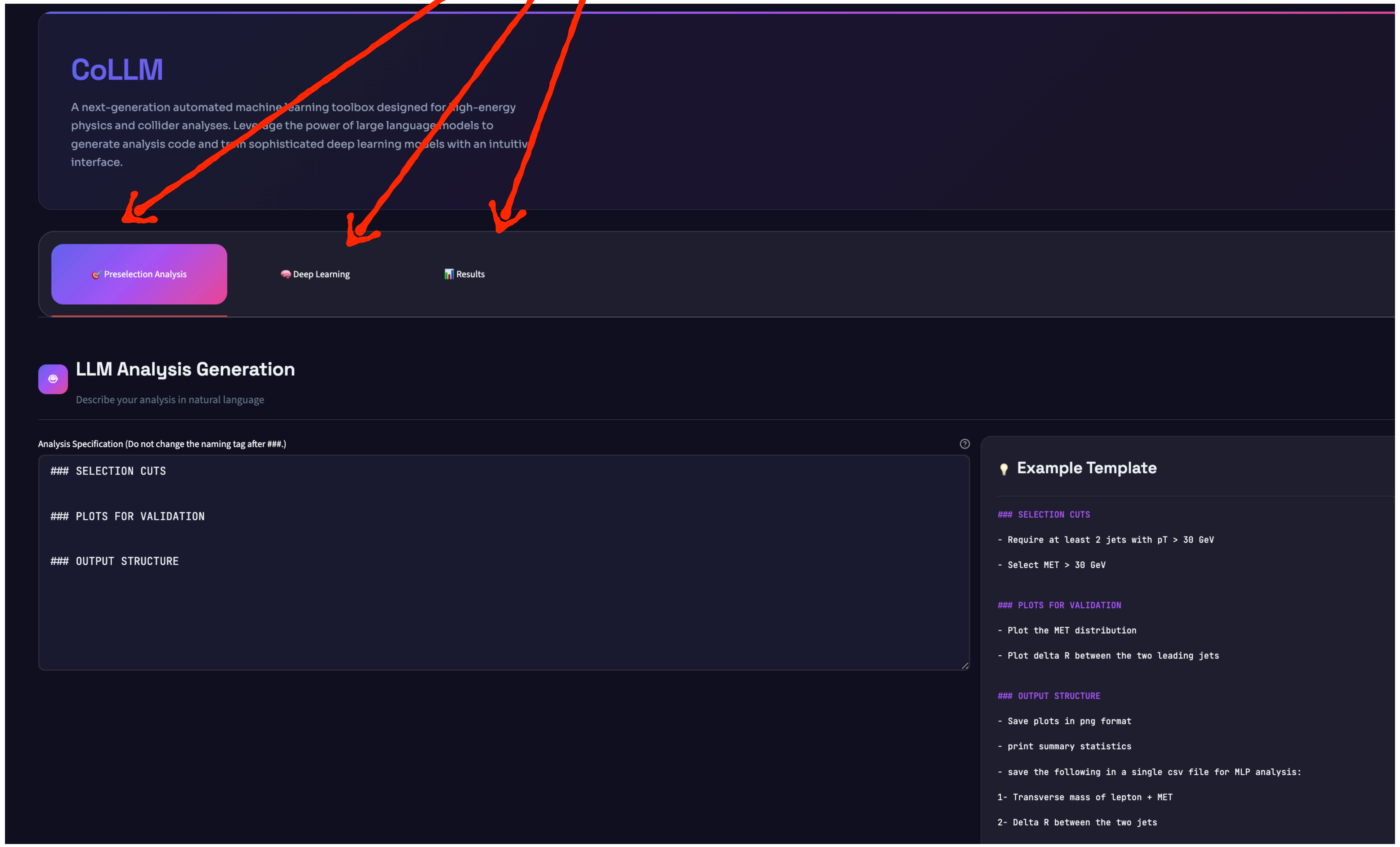
CoLLM supports running on
CUDA, MPS and CPU

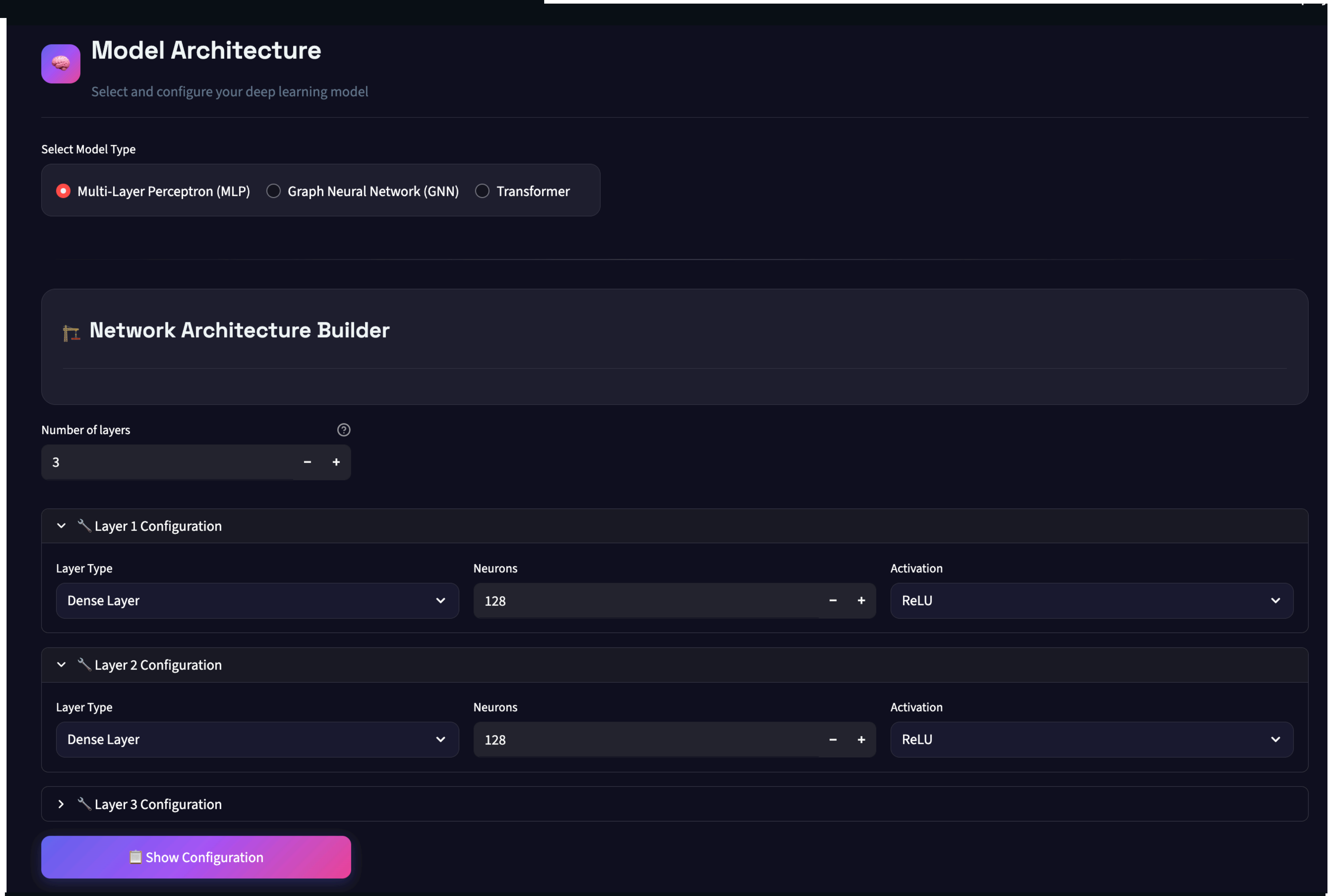
```
Checking requirements...
INFO: numpy already installed
WARNING: Installing matplotlib...
INFO: Successfully installed matplotlib
INFO: tqdm already installed
INFO: yaml already installed
WARNING: Installing langchain...
INFO: Successfully installed langchain
WARNING: Installing langchain-huggingface...
INFO: Successfully installed langchain-huggingface
INFO: transformers already installed
INFO: huggingface_hub already installed
INFO: accelerate already installed
INFO: pydantic already installed
WARNING: Installing streamlit...
INFO: Successfully installed streamlit
INFO: PyTorch 2.9.1+cu128 installed
INFO: CUDA not available, using CPU
Starting CoLLM GUI...

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://130.87.250.19:8501
```

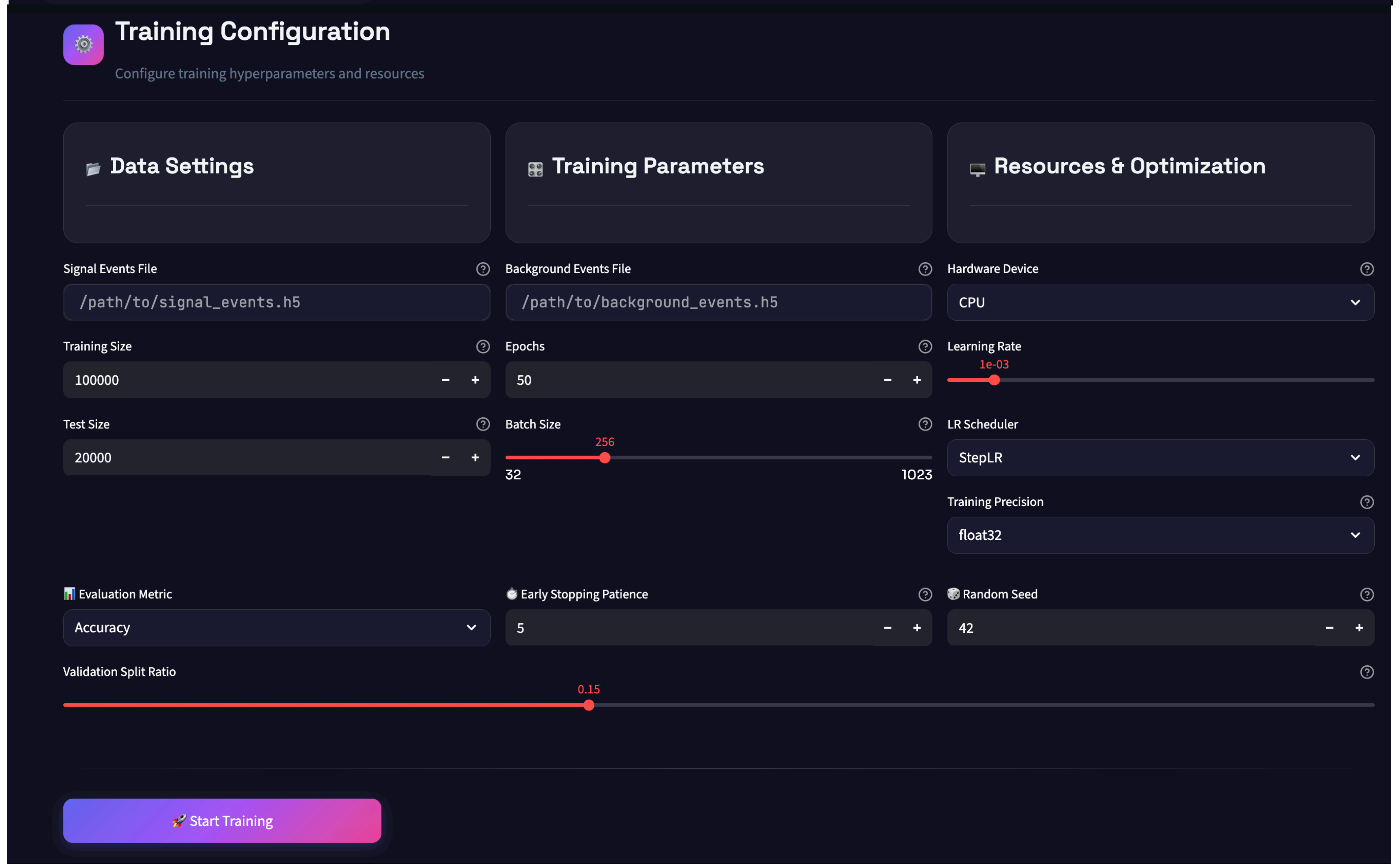
Graphical User interface

```
./run.sh --run_GUI
```

Opens a local web browser with three main sections





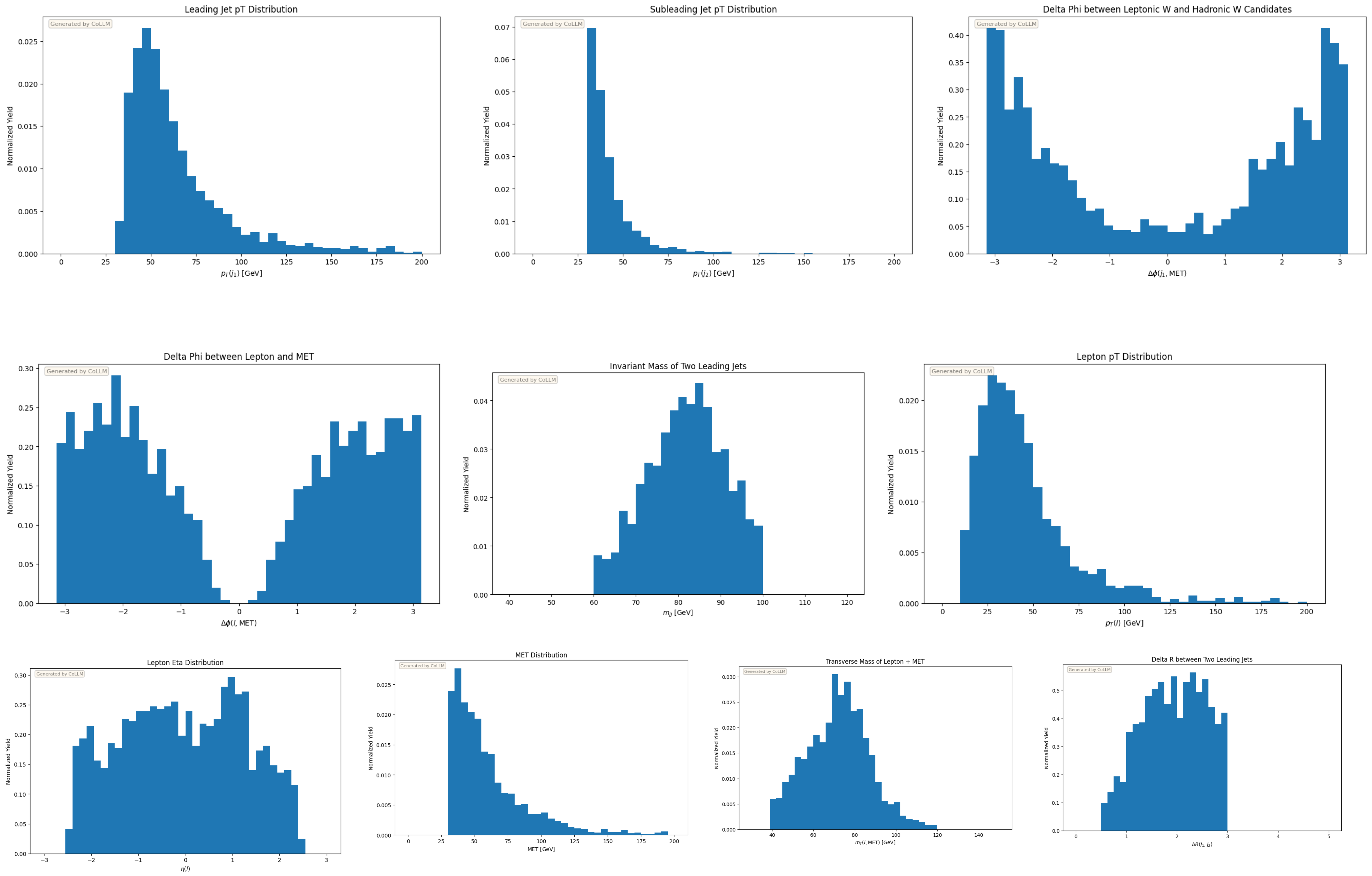


CoLLM targets non coding experts, or experts who want to save time for quick analysis

User input

```
# Physics Process: p p > W+ W-, W+ > l+ nu, W- > j j
# ================================================================

[SELECTION_CUTS]
- Select electrons with pT > 25 GeV and |eta| < 2.5
- Select muons with pT > 20 GeV and |eta| < 2.4
- Require exactly 1 lepton (electron or muon)
- Require at least 2 jets with pT > 30 GeV and |eta| < 2.5
- Select MET > 30 GeV
- Veto events with b-tagged jets
- Select transverse mass of lepton + MET between 40 and 120 GeV (W leptonic candidate)
- Select invariant mass of the two leading jets between 60 and 100 GeV (W hadronic candidate)
- Select delta R between the two leading jets < 3.0

[PLOTS_FOR_VALIDATION]
- Plot the following as histograms:
1 Plot the transverse mass of lepton + MET (W leptonic) in the range 30 to 150 GeV
2 Plot the invariant mass of the two leading jets (W hadronic) in the range 40 to 120 GeV
3 Plot the MET distribution
4 Plot the pT of the lepton
5 Plot the pT of the leading jet
6 Plot the pT of the subleading jet
7 Plot delta R between the two leading jets
8 Plot delta phi between the lepton and MET
9 Plot the eta of the lepton
10 Plot delta phi between the leptonic W and hadronic W candidates

- Each histogram should have 40 bins.
- Use LaTeX notation for axis labels where applicable.
- Normalize all histograms to unity (density=True).

[OUTPUT_STRUCTURE]
- Print cutflow showing number of events after each selection
- Save the pltos in png format
- Save the following in a CSV file for MLP analysis:
  1- Transverse mass of lepton + MET
  2- Dijet invariant mass
  3- MET
  4- pT of the lepton
  5- Delta R between the two jets
  6- Delta phi between leptonic and hadronic W systems
```

Generated 309 line of code in 28 seconds



CoLLM vs ChatGPT

Feature	CoLLM	ChatGPT / General LLMs
End-to-End Pipeline	✓ Integrated: parsing → generation → validation → execution	✗ No pipeline; manual coding for each step
LHCO File Support	✓ Native parser with full format specification	✗ No file handling; code suggestions only
Physics-Aware Code	✓ HEP conventions, formulas, PDG masses built-in	✗ Generic responses; may contain physics errors
Auto Error Correction	✓ PyFixer: automatic bug detection & self-healing	✗ No auto-correction; manual debugging
Code Execution	✓ Generates and runs validated Python scripts	✗ Cannot execute; provides snippets only
GPU Acceleration	✓ CUDA & MPS with 4-bit quantization support	✗ Cloud-only; no local GPU utilization
Data Privacy	✓ Local models (Qwen, DeepSeek) for sensitive data	✗ Cloud-only; data sent to external servers
User Interface	✓ Terminal UI + Streamlit GUI with live monitoring	✗ Chat interface only; no physics-specific UI
Physics Functions	✓ 4-momentum, invariant mass, ΔR, MT, Z/W/H reco	✗ Must request each formula; error-prone
Cutflow Tables	✓ Automatic event counting & cutflow generation	✗ Manual implementation required
Reproducibility	✓ Deterministic; fixed seeds & versioned outputs	✗ Variable responses; not reproducible
Batch Processing	✓ Processes millions of events efficiently	✗ Cannot process actual data files
ML Training	✓ GUI config: epochs, batch size, LR, schedulers	✗ No training capability; discussion only
API Flexibility	✓ Local models OR HuggingFace Inference API	✗ Locked to OpenAI API only

Thank you