



# AI Assisted Code Development in ATLAS/CERN

Attila Krasznahorkay



# Disclaimer



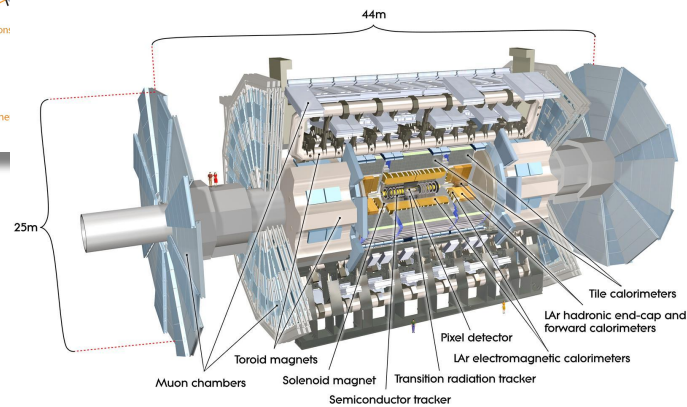
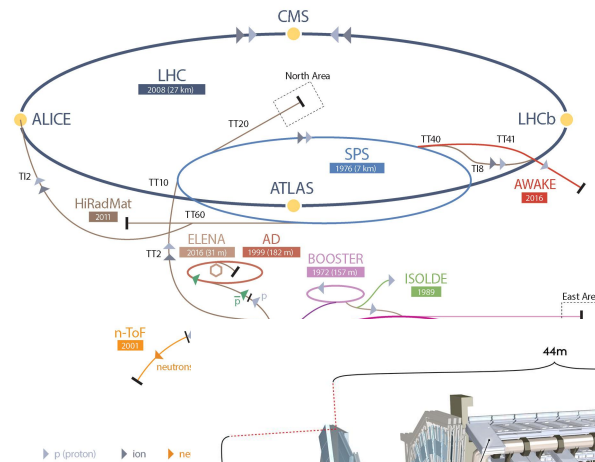
I am **not** speaking “officially” for [ATLAS/CERN/UMass](#). The following are in a large part my personal views/opinions.

# ATLAS at the LHC

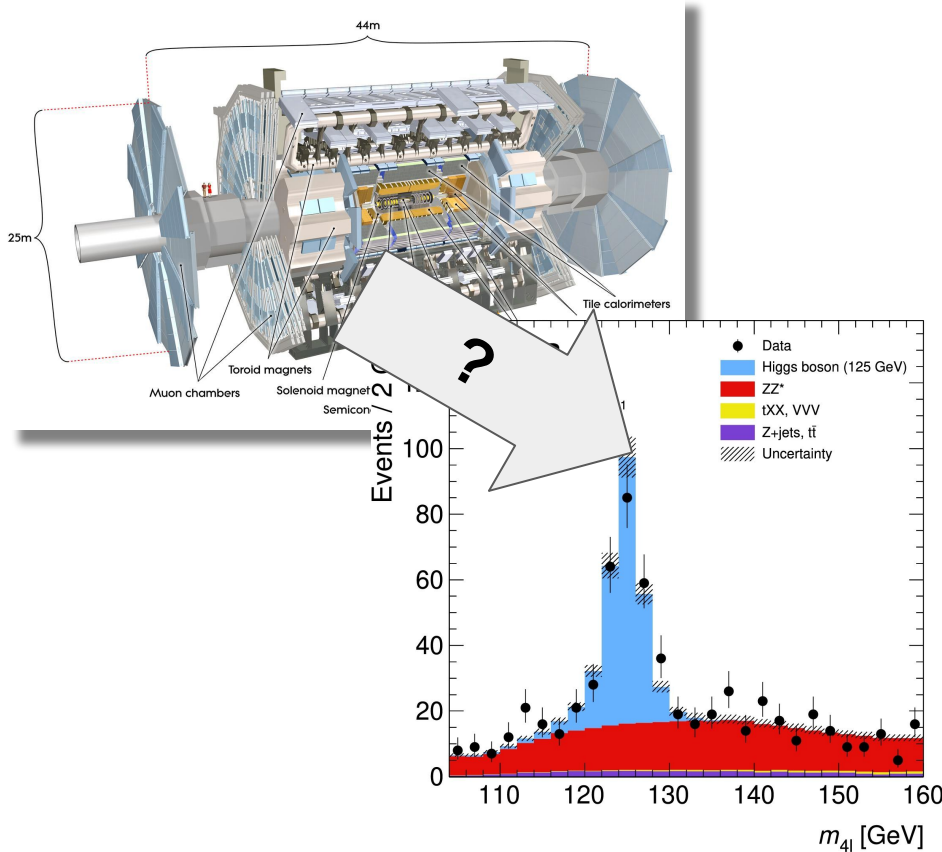


- The Large Hadron Collider is the flagship accelerator in High Energy Physics “at the energy frontier”
- ATLAS is one of the 2 general purpose experiments at the LHC
  - It was designed to be able to study as wide variety of physics as possible
- ATLAS is probably the largest collaboration in physics, consisting of >6000 physicists/engineers/students

CERN's Accelerator Complex



# Data Processing in ATLAS



- In our current data taking we get  $\sim 1\text{B}$  p-p collisions per second
  - We select about 2 kHz of events for permanent storage using a complicated system of custom hardware and specialized software
- The recorded “raw” data is processed “offline” by reconstructing physics objects (electrons, muons, etc.)
  - These are written into smaller, analysis ready files on LHC’s worldwide computing grid
- Analyzers process petabytes of that data according to their own physics analysis

# atlas/athena



- The offline software of ATLAS is publicly available under:  
<https://gitlab.cern.ch/atlas/athena>
  - It holds all ATLAS specific code needed to simulate, reconstruct and (in part) analyze ATLAS's data
- It is a mono-repo holding millions of lines of code in a lot of different languages
  - In a way that allows us to build [different projects](#) out of the same repository, building just a specific (sub-)collection of code into the different projects

The screenshot shows the GitLab repository page for atlas/athena. The page displays a list of folders and files, a merge commit, and a table of code statistics.

language	files	code	comment	blank	total
C++	37,992	3,337,059	896,042	923,535	5,156,636
Python	6,034	706,150	118,562	148,146	972,858
XML	903	218,869	14,227	32,871	265,967
CMake	2,044	47,566	0	13,147	60,713
C	235	38,612	7,455	9,359	55,426
Markdown	276	16,982	101	6,273	23,356
JSON	66	13,063	12	181	13,256
CUDA C++	53	10,837	1,221	3,458	15,516
YAML	85	10,538	650	515	11,703
HTML	23	4,233	203	545	4,981

# ML/AI in ATLAS/HEP Software



The screenshot shows a code editor interface. On the left, a file browser displays a directory structure for 'athena'. The selected file is 'README.md', which contains the following text:

```
Flavor Tagging Inference
```

This package contains the trivial code that FTAG uses to run the GNN. The isolation of this and base other projects off.

This code is meant to be identical between the main and 24.0 branches.

It is meant as "stand-alone" code: it should be usable in Athena, AthAnalysis, and Analysis

```
Package Overview
```

There are several user-level tools here:

- `GNN`: Low-level implementation of the GNN taggers. Allows lower-level manipulation. I
- `GNNTool`: ASG interface to `GNN`.
- `MultiFoldGNN`, `MultiFoldGNNTool`: Like `GNN`, but initializes with a number of network determined by the `jetFoldHash`.
- `FoldDecorateAlg`: adds a hash (`jetFoldHash`) to each jet based on some jet and ev fold (in the `MultiFoldGNN`) is random.

```
Other Files
```

There are also several tools that you probably don't have to touch:

- `CustomGetterUtils`: Models rely on some information that isn't stored in accessors. These are defined in `CustomGetterUtils`.

On the right, the C++ code defines the `AthInfer` namespace and the `TritonTool` class:

```
namespace AthInfer {
class TritonTool : public extends<AthAlgTool, IATHInferenceTool> {
public:
// Constructor
TritonTool(const std::string& type, const std::string& name,
const IInterface* parent);
// Destructor
virtual ~TritonTool();
// @name Function(s) inherited from @c AthAlgTool
// @f
// Initialize the tool
virtual StatusCode initialize() override;
// @f
// @name Function(s) inherited from @c IATHInferenceTool
// @f
// Run inference with multiple inputs and multiple outputs
virtual StatusCode inference(InputDataMap& inputData,
OutputDataMap& outputData) const override final;
// Print the tool's properties and configuration
virtual void print() const override;
// @f
private:
// @name Tool properties
// @f
StringProperty m_modelName{this, "modelName", "", "Model name"};
IntegerProperty m_port{this, "port", 8081, "Port ID for Triton server"};
StringProperty m_modelVersion{this, "modelVersion", "",
"Model version, empty for latest"};
FloatProperty m_clientTimeout{
this, "clientTimeout", 0,
"Client timeout in milliseconds, 0 for no timeout"};
StringProperty m_url{this, "url", "", "Triton URL"};
BooleanProperty m_useSSL{this, "useSSL", false,
"Use SSL for Triton server connection"};
// @f
// Implementation details for the tool
struct Impl;
// Pointer to the implementation details
std::unique_ptr<Impl> m_impl;
}; // class TritonTool
} // namespace AthInfer
```

- HEP has used ML/AI since a long time
  - Mostly for classification tasks in the past, and then some regression as well at the LHC
- The last years saw a big boom in studies in this area
  - “Final analyses” rarely don’t use ML classification for something
  - Most reconstruction areas use some ML steps. Some of them mainly rely on ML. (Tau and b-jet reconstruction for instance.)
- No HEP-wide agreement on inference infrastructure
  - [ONNXRuntime](#) is quite popular, but far from the only solution
- Using transformers/LLMs is quite new though!

# Develop / Build / Test



- The millions of lines of ATLAS offline software are built on top of ~200 “external” packages
  - Which are built for us in an organized way in the [LCG software stack](#)
- So in order to configure the (CMake) build of any of our code, we need to be in a very specific runtime environment
  - Most users can only build the code in a terminal as a result
  - Exposing an LLM to our code, such that it would know where to find headers, is not trivial 🤔

## LCG Info: Release 109a ATLAS\_1

The image displays a collage of screenshots related to the ATLAS\_1 software stack. At the top, there is a 'Release Info' page for '109a\_ATLAS\_1'. Below it, a document titled 'Using CMake with Athena' provides instructions on how to build the software. The bottom portion of the image shows a terminal window where the 'cmake' and 'make' commands are being executed. The terminal output shows the configuration of the Athena software stack, including the detection of various compilers and the setting of environment variables like 'ATHENA\_ROOT' and 'ATHENA\_LIBS'.

- [VSCode](#) became a very successful editor/IDE in ATLAS in the last years as well
  - Giving a significant edge to LLMs that it would directly support. CoPilot being the prime example.
  - Having access to a free tier of CoPilot also helps a lot!
- As CERN is recognized by GitHub/Microsoft as a valid organization for [GitHub Education](#), many people at CERN do use CoPilot Pro for free
  - Including me
- I myself have actively used it for a few years now. But still at a novice level.
  - Getting code suggestions as I type can sometimes be extremely helpful, and sometimes be ridiculously annoying. Had to turn it off for some periods because of it.
- “Chat mode” I only tried seriously once myself
  - I asked CoPilot with ChatGPT backing to translate [a CUDA based shared library](#) into a HIP one. It did not go well. 😞
  - I was hoping that the model would recognize the code organization more thoroughly. Realizing that it really just needed to do some search-and-replace operations. Didn't happen quite that way...

- In the CERN software group I know 2 people that actively use it
- **Person A is a novice user**
  - They “expose” a terminal utility to the athena source tree, and ask the model to do things for them
    - No teaching of Claude about how the code is built, or anything like that
  - They mostly generate code for “standard tasks”, writing ATLAS offline software
  - They also use Claude for non-work tasks, including organizing some household tasks
- **Person B is an advanced user**
  - They let Claude build and test their code, while asking it to develop different things for them
  - They mostly generate code for “extracurricular tasks”. Writing code that they otherwise would just not have written.
  - Not sure about non-work usage in their case 🙄
- **I’m not actually sure which one of them scares/worries me more**
  - The novice user generates more standard code, but I worry that their knowledge of our codebase is eroding
  - The advanced user showed some amazing [Blender](#) renders just yesterday of our future detector that he could’ve only done using Claude. It’s not mission-critical, but was made with code that none of us now know.

# Access Democratization



- People in ATLAS make use of LLMs for code writing all on their own at the moment
  - People resident at CERN and at many other institutes / universities can take advantage of GitHub Education. But many members of ATLAS can not do this.
  - Many people I know pay personally for their private CoPilot/Claude subscriptions.
- This already created a separation of haves and have-nots in the experiment 😞
- At many different organizational levels, including at the CERN management level, this has been a topic since a while
  - Ideas are floating around about training and hosting a HEP specific LLM at CERN that would be made available to all CERN users
  - It is not a simple thing to sort out though. I myself am also not sure exactly what role CERN should play in training/managing LLMs like this. 🤔

- Many people already extensively use LLMs in their code development → Merge/Pull requests actively happen already with LLM written/designed code
- There is no fixed guidance **yet** on how to handle this in ATLAS
  - The general sense is that we should only ever push code into Athena that a developer could have written on their own as well, given enough time.
  - But this is not a black or white question. Especially for junior developers. 😞

## Athena AI Coding guidelines 2.0

There is a nice summary of the best practices, privacy and legal considerations [here](#), which should be considered before using AI coding assistants.

1. Contributors should thoroughly understand and test what they check in and ensure it complies with ATLAS coding guidelines. (AI-generated or not.)
  - Use assistants as an accelerator rather than a replacement for coding expertise. ("Could I have written this myself, given enough time.")
2. Contributors should, if they consider it to be relevant:
  - mention that AI was using in commit messages
  - add the "AI-assisted" label to the merge request if the amount of LLM-generated code is significant
  - add details of the use of the LLM in the merge request description (e.g. the natural language prompt used)
3. Contributors should, if/when available, use CERN provided LLMs rather than commercial offerings.
4. Code review must be done by a human. However, AI tools may be used in addition to a full human review cycle.

# Personal Feelings / Outlook



- I'm mostly worried about average developers forgetting how to write code by themselves
  - I.e. what happens if all of a sudden we can't use AI anymore
- I see a very strong parallel with the US Navy's use of GPS...
  - That officers, after some time where this wasn't done, are taught about celestial navigation once again. So that losing GPS would not make all of the Navy stop.
- I have not seen LLMs do high-level code design yet
  - *This may just be because of my limited outlook on this*
  - Coming up with good overall designs at this point is still up to humans
  - What happens when nobody is left with enough low-level knowledge to make such high-level designs? 🤔

# Summary



- AI assistance is by all means improving our software development in HEP
  - Whenever I work offline these days, with no CoPilot help, I notice right away
- I worry mostly about young people, not about seasoned developers
  - Though I also worry that I may be like the people arguing in the past that only assembly programming was reasonable programming
- Most people didn't deeply understand the code that they were writing, even in the past. So maybe AI help is still a net positive...?



<http://umass.edu>