CompF5: End User Analysis

Recommendations & Highlights

Amy Roberts (UC Denver, CDMS)
Peter Onyisi (UT Austin, ATLAS)
Gavin S. Davies (U. Mississippi, NOvA/DUNE/EMPHATIC)

https://snowmass21.org/computational/analysis

Are these covered?

Some topics have perhaps slipped through the cracks between topical groups - not strictly end-user analysis but strong connections

- High-level software “frameworks” - LArSoft, Gaudi, cmssw, Athena, ...
- Software environments for build + deployment
  - Build systems
  - Diverse environments (ability to try new compilers, libraries, ...) vs controlled environments (simpler builds, fewer unexpected interactions). Everything in containers?
  - Continuous integration
- Workflow management

Analysis Software Ecosystems

Finding: There are two primary ecosystems developing for analysis in the next decade: one based on the ROOT libraries and one based on a collection of Python libraries. These projects have similar goals and scope but are organized differently and have different philosophies regarding industry tool use.

Recommendation: development of both ecosystems should be supported. The friendly competition between the two has already resulted in significant improvements for users. Maintaining interoperability between the two (e.g. in data formats) should be a priority.

Analysis Software Languages

Finding: The main general purpose languages used in HEP today are C++ and Python. Projects such as PyROOT have enabled interoperability of these languages at a level not replicated by other options in HEP.

Recommendation: these two languages both have important roles to play in their respective niches and are expected to remain dominant in the near future. Other languages are generally unfamiliar to the community, have weak interoperability with existing libraries, and impose a maintenance burden if used without careful planning. Outside of bottom-up projects in other languages, or overwhelming domain-specific needs (e.g. JavaScript for visualization in web browsers), C++ and Python should be recommended.

Analysis Software Languages (2)

Additional note: there is a reasonable concern about the efficiency of using interpreted languages in core analysis kernels, both in terms of computing resource use and consequent environmental impact. We recommend that the Python ecosystem community demonstrate that this issue has been addressed, and also ensure that just-in-time compiler technology (e.g. numba) is easy to integrate into user code.

Domain-Specific Languages

Finding: A number of domain-specific languages have been proposed to address problems in various spheres, from specifying operations in columnar analysis, to specifying likelihood construction from histograms, to describing analysis at a very high level. Additionally, certain uses of Python (e.g. when scripting ML libraries, or working with RDataFrame) can be viewed as a DSL as they require library-specific knowledge.

Recommendation: “High-level” DSLs that attempt to describe analysis via high-level objects are unlikely to generalize well between experiments as often the “primitives” are too different, but can be useful in certain situations. The lower-level DSLs can be extremely useful where relevant, however effort should be taken to avoid unnecessary duplication of scope as this imposes burdens on users similar to using multiple general purpose languages.

Data Formats

Finding: The ROOT container file format is ubiquitous in HEP, and comes with a serialization scheme tightly linked to the ROOT libraries. Industry and non-HEP tools typically use other formats (e.g. Apache Parquet for the ROOT TTree), and efforts are ongoing to enable their use. In addition the ROOT team foresees an evolution of the TTree to the more-optimized RNtuple. Multiple independent implementations of ROOT I/O are available. As ROOT is specialized to the HEP use case, it supports features missing from other formats.

Recommendation: The ROOT file format is extremely important for ongoing experiments and historical data and compatibility must be maintained. Other formats have important use-cases. Tools to translate between formats, and to enable various ecosystems to ingest and produce them, should be maintained.

Metadata

Finding: Experiments usually have global solutions for handling metadata involved in dataset cataloging and workflow control. However these solutions frequently are not available for individual analyses and users may need to develop bespoke solutions.

Recommendation: Effort should be put into developing user-friendly data provenance and metadata storage systems that can be easily integrated into typical analysis tasks.

Analysis Models

Finding: Users need to be able to scale their analyses from simple tests on a small debugging dataset to full deployment over all data. In many cases this transition requires working in a different environment (a common case is transitioning from a local workstation, referring to specific files, to a batch job running on a catalogued dataset). This can involve significant difficulties for users and cause support issues.

Recommendation: There is unlikely to be a one-size-fits-all solution for all experiments. Recent work with interactive analysis facilities which provide extreme scaling capabilities to users through a single interface may address many of these issues; if successful the resulting software stacks should be made available to small experiments in a turnkey way.

Long-Term Preservation

Finding: Transitioning from the way an analysis is actually done to a “packaged” version that can be rerun by an outsider from scratch is typically complex, as the entire workflow is rarely described in a single place. This causes people to see long-term preservation as an additional burden whose benefits they will not see.

Recommendation: provide pipelines to nudge users into choosing practices compatible with long-term preservation as the default. These need to be considered and built in to the structure of analysis systems at the start, not bolted on at the end.

Personnel Issues

Finding: A lot of transformative ideas are introduced to the HEP computing community and are implemented by early-career scientists (especially grad students and postdocs). Support for these physicists can be minimal, and the career trajectories are opaque.

Recommendation: software work (especially with cross-experiment application) should receive stronger consideration for funding. More cross-experiment/frontier computing physicist positions could be created. Funding agencies and frontiers need to work together to identify viable long-term funding patterns for this work.

Long-Term Support

Finding: Software typically has low barriers to initial entry but significant ongoing maintenance requirements. New software projects are frequently initiated without much concern for long-term support (and this is reasonable since many bottom-up projects do not succeed).

Recommendation: computing personnel should be funded specifically to maintain software projects identified as satisfying an important need in the community (for example, by the HEP Software Foundation).

Collaborative Software

Finding: The “full” analysis stack of an experiment also includes software that enables interaction between analyzers. This includes documentation of the experiment’s code, messaging between users, discussion forums, software version control, bug tracking, and document workflow management. Especially for small experiments, setting up this infrastructure from scratch can be daunting both in effort and cost.

Recommendation: host laboratories should provide a full stack of these services to experiments. XSEDE/ACES could also play an important role here as some laboratories have onerous access requirements.

Training

Finding: There are many cases where experiments are limited by the availability of personnel trained in various aspects of computing. All students need to be trained in the software and languages used for analysis; beyond that, specialized training may be needed for specific areas, such as GPU use or code optimization. Many traditional avenues (e.g. dedicated academic classes or industry training) may not be a good match to needs or affordable. There are significant efforts through the HEP Software Foundation and other initiatives to develop effective training for HEP software. Software training has a direct connection to concerns about diversity, equity, and inclusion.

Recommendation: the scalability of various approaches to training should be understood. Best practices from education research, industry, and assessment should be incorporated into efforts. Funding should be made available for the development and hosting of training materials and should include partnerships with education researchers.