ATLAS PUB Note
ATL-PHYS-PUB-2022-017
25th March 2022

SimpleAnalysis: Generator-level Analysis Framework

The ATLAS Collaboration

Almost all Beyond the Standard Model (BSM) searches in ATLAS provide auxiliary information uploaded to HEPData which can be used to, for example, reinterpret the search results on other BSM models than those evaluated in the search. This information often includes generator-level (*truth*) acceptance maps and C++ analysis code snippets defining all of the signal regions in the analysis. Inside of ATLAS, the SimpleAnalysis generator-level analysis framework is used to calculate the truth-level acceptance maps with the uploaded C++ analysis fragments as well as for some systematic uncertainty evaluations. This framework is now publicly available and presented in this note. For validation, a search for supersymmetry (SUSY) in a final state with one lepton and two-*b*-jets is evaluated through this framework.

1 Introduction

SimpleAnalysis (SA) is an analysis framework in C++ designed to run on the output of event generators (generator-level information, or *truth*) which is used in most ATLAS Supersymmetry (SUSY) results since 2016. Similar in scope to for example Rivet [1] and MadAnalysis [2], SA provides a software framework for encoding high energy physics analyses in a compact, easy to use format that captures the majority of published analysis selections. It is used internally in ATLAS to run on data formats belonging to the ATLAS Data Model (Derived Analysis Object Data, or DAOD files) [3], but can also run over HepMC [4] output or a simple form of ROOT [5] ntuples. SA provides various helper functions for complex kinematic variables typically used in SUSY analyses, encapsulates custom efficiency maps, and contains additional utilities for serializing more complex multi-variate analysis techniques such as neural networks and boosted decision trees. The outputs are used by ATLAS to compute truth acceptances and selection efficiencies for HEPData publishing [6], and evaluate sources of theory systematics.

Most searches will provide *acceptance* and *efficiency* for a variety of Beyond-Standard-Model (BSM) signal models described in Eq. (1). The acceptance ($\mathcal{A}$) is defined as the fraction of signal events *accepted* in the generator-level analysis provided by SA. The acceptance is highly dependent on the final state particles and kinematics of the signal model under study. The efficiency on the other hand mostly captures detector and reconstruction effects, such as reconstruction and identification inefficiencies, object resolutions and selections which cannot be included at generator level. The efficiency is much less model dependent unless the analysis uses objects with large variation in reconstruction efficiency or primarily selects misreconstructed events. The efficiency is calculated from the "acceptance times efficiency" ($\mathcal{A} \otimes \varepsilon$) as extracted from the actual analysis using fully reconstructed quantities.

$$\mathcal{A} = \frac{n_{\text{accept}}^{\text{generator}}}{n_{\text{total}}^{\text{generator}}}, \qquad \mathcal{A} \otimes \varepsilon = \frac{n_{\text{accept}}^{\text{reco}}}{n_{\text{total}}^{\text{reco}}} \qquad \Rightarrow \qquad \varepsilon = \frac{\mathcal{A} \otimes \varepsilon}{\mathcal{A}} \tag{1}$$

Most SUSY analyses published by ATLAS re-implemented their analysis selections at generator-level in a single C++ file for SA and published that file to HEPData. This enables external and internal collaborators to more easily reinterpret each analysis result on different SUSY and other BSM models which were not considered by the original analysis. New signal events can be generated and filtered through the provided selection file to calculate the truth acceptance for the new model and, together with the model cross section, used to predict the number of events expected in each signal region. A crude efficiency correction can be applied based on the typical signal efficiency numbers published for the analysis in HEPData. For signal models which differ significantly from the original ones, the efficiency is better estimated using a fast detector simulation which emulates the reconstruction inefficiencies and resolution effects. Such a simulation is not provided by SA, but through a conversion script, it is possible for SA to run on the output of a DELPHES simulation which has been tuned to match the efficiency of a specific analysis. The expected number of signal region events can either be directly compared to provided model-independent limits or input into a likelihood fit [7–10] to possibly provide stronger exclusion limits. The reinterpretation procedure can be validated by applying the procedure to the original models of the analysis and comparing the results to the truth acceptance and selection efficiency maps provided in HEPData. This framework can be utilized for summary or combination efforts, evaluating hundreds of thousands of models scanned over in phenomenological MSSM (pMSSM) studies such as those done by ATLAS in Run 1 [11]. For fast or large scale studies, such reinterpretations, while less accurate, complement the full analysis reinterpretations

such as those provided by the RECAST [12–15] and REANA [16] tools which are currently only available inside the collaboration.

The portion of the framework used to compute generator-level acceptances and efficiencies is now available to external collaborators. This allows non-ATLAS users to easily compile and execute the analysis at generator level as a first step towards analysis reinterpretation. The code release also provides the full details of the various helper functions used to keep the analysis selection code short and simple to read.

This note documents the public part of the SA framework. The analysis codes that have been uploaded to HEPData across various analyses will in addition be co-located [17, 18] with the framework for convenience. Section 2 describes the overall structure of the SA framework as well as the associated documentation. This code has been validated for each analysis implementation, as shown in Section 3 for the SUSY electroweak one lepton, two b-jet analysis [19].

2 Code Structure

SimpleAnalysis provides an executable standalone program which can read user-supplied generator-level events, filter them through one or more selected analysis selection codes and calculate the acceptance (weighted fraction of accepted events) for each signal region. Optionally, histograms and ntuples filled by each analysis code for more detailed studies can be enabled. Currently, SA requires the input events to be either in an ATLAS-specific DAOD format, in HepMC format or in the form of a ROOT [5] ntuple. The latter contains a small set of standard variables like four vectors of all generator-level leptons and jets from the hard-scatter process as well as the missing transverse momentum. A python script is also provided to convert from the ROOT output format of Delphes to the SA ROOT ntuple format in order to simplify the inclusion of a fast detector simulation, though Delphes will need to be tuned to the specific analysis of interest.

There are two pieces of code in SA that are public: the framework which contains all of the machinery for driving the analysis, and the analysis code implementation for the analyses which have already been made public. In addition, the SA public documentation [20] using `mkdocs` [21] is available. The SA public documentation is a living document that describes the technical details of the code, the interface, the SA specific ntuple and how to use SA. The documentation contains a tutorial that should be followed to understand how to use the framework described in this note. Section 2.1 provides a high-level overview of how the ATLAS Collaboration organizes the associated code repositories and the development workflow. Section 2.2 discusses fundamental pieces of the SA application interface.

2.1 Code Organization

There are three important locations that contain public resources associated with the SA framework.

- Top-level: `https://gitlab.cern.ch/atlas-sa/simple-analysis/`
- Framework: `https://gitlab.cern.ch/atlas-sa/framework`
- Documentation: `https://simpleanalysis.docs.cern.ch/`

The framework contains the functionality for computing the object kinematics, including the more complex variables such as object-based $E_{\mathrm{T}}^{\mathrm{miss}}$ significance and neural network scores, or loading in published efficiency maps associated with a particular analysis. For most use cases, all interaction will be with the public top-level repository. This is organized as shown in Listing 1 and contains:

- `SimpleAnalysisCodes`: the individual analysis codes with associated data files (BDT weights, ONNX files, efficiencies, etc...) described in more detail in Section 2.2,
- `Ext_RestFrames`: a submodule providing `RestFrames` [22], a recursive jigsaw reconstruction package used by a few SUSY analyses,
- and `SimpleAnalysisFramework`: a submodule providing the SA framework, which contains definitions and functionality for performing generator-level analysis.

In addition to these core pieces, the top-level repository ships docker images in the associated GitLab registry for running the code without needing to compile [23]. As ATLAS continues to release more search results, new SA implementations will be added to the top-level repository. If those SA implementations require additional functionality from the SA framework, the framework will be updated and the corresponding submodule link in the top-level repository will also be updated.

```
.
├── CMakeLists.txt
├── CODE_OF_CONDUCT.md
├── Dockerfile
├── Ext_RestFrames (submodule)
├── LICENSE
├── README.md
├── SimpleAnalysisCodes
│   ├── CMakeLists.txt
│   ├── data
│   │   ├── StopOneLepton2016_BDT-tN_diag_high.weights1.xml
│   │   ├── StopOneLepton2016_BDT-tN_diag_high.weights2.xml
│   │   ├── StopOneLepton2016_BDT-tN_diag_low.weights1.xml
│   │   ├── StopOneLepton2016_BDT-tN_diag_low.weights2.xml
│   │   ├── StopOneLepton2016_BDT-tN_diag_med.weights1.xml
│   │   ├── StopOneLepton2016_BDT-tN_diag_med.weights2.xml
│   │   ├── ...
│   │   └── ZeroLepton2018-SRBDT-GGo4_weight2.xml
│   └── src
│       ├── ANA-SUSY-2016-16.cxx
│       ├── ...
│       └── ANA-SUSY-2019-08.cxx
├── SimpleAnalysisFramework (submodule)
├── ci
├── docs
├── scripts
│   └── Delphes2SA.py
```

Listing 1: A pared-down overview of the structure of the top-level repository of SimpleAnalysis.

In order to facilitate this development workflow, ATLAS maintains an internal repository shown in Figure 1 which also uses a submodule of the public `SimpleAnalysisFramework` repository. This ensures that the

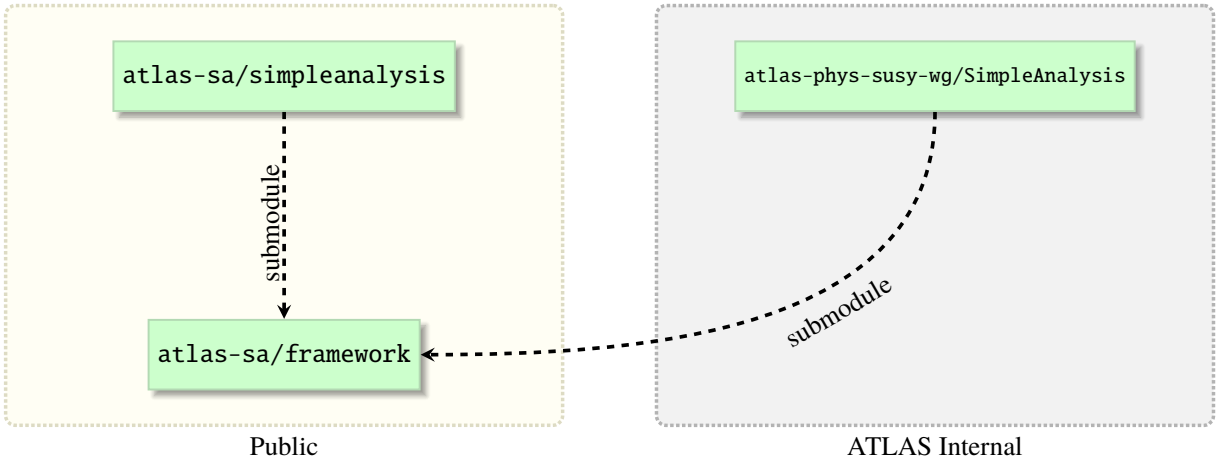


Figure 1: Overview of the GitLab repositories for the ATLAS SimpleAnalysis framework and their git relationship.

framework is actively developed to add any improvements or fixes required for an analysis – public or internal. This internal repository contains private code such as our fast detector simulation implementation as well as the SA implementations for analyses not yet made public. Whenever ATLAS wants to publish an analysis code, it requires a one-time manual copy operation to the public top-level repository, along with an update to the framework submodule if needed.

The code currently must be compiled and run on top of ATLAS software [24], `atlas/analysisbase`. This software is compiled and provided by ATLAS in docker images. The GitLab repository for SA provides a container registry[23] for these pre-built docker images containing SA binaries. The primary expectation is that external users will not implement new analyses within this framework, but use, and possibly tweak, the analyses that have already been implemented and published in the GitLab project.

2.2 Code Implementation

Listing 10 contains a pared-down example of an analysis code implementation which is contained in a single C++ file. It tends to be fairly readable, compact code that documents the analysis. An example of SA execution is shown in Listing 2.

```
simpleAnalysis -a EwkOneLeptonTwoBjets2018 my-evtgen.hepmc
```

Listing 2: An example of running a generator-level analysis using the top-level interface on the one lepton, two *b*-jet analysis [19].

2.2.1 Analysis Code Names

Analyses implemented in the project have names following the pattern `ANA-SUSY-XXXX-YY`. These are called "Glance Identifiers" [25] and are unique identifiers within ATLAS for referencing analyses. This will

be associated with a human-readable analysis name within the framework, such as `ANA-SUSY-2019-08` mapping to `EwkOneLeptonTwoBjets2018`[1] as demonstrated in Listing 3.

```
#include "SimpleAnalysisFramework/AnalysisClass.h"
DefineAnalysis(EwkOneLeptonTwoBjets2018)
```

Listing 3: A snippet of `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26] showing how the analysis name is defined.

This mapping makes it fairly easy for anyone to identify the associated public page for the given analysis. One can refer to the documentation[2] for SA to get links to the corresponding analyses as well.

2.2.2 Analysis Objects

Kinematic objects are loaded into the `AnalysisEvent` and retrieved via getter functions such as those shown in Listing 4.

```
// baseline objects
auto baseEle  = event->getElectrons(7,2.47, ELooseBLLH);
auto baseMuon = event->getMuons(6,2.70, MuMedium | MuNotCosmic | MuZ05mm);
auto baseJets = event->getJets(20.,4.5);
auto met_Vect = event->getMET();
auto weights  = event->getMCWeights();
```

Listing 4: A snippet of `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26] showing how to retrieve kinematic objects from an event.

For example, `event->getElectrons(7,2.47, ELooseBLLH)` will retrieve electrons that pass the `ELooseBLLH` isolation requirement. The full SA API is described in detail in the documentation [20] and the source code [26].

2.2.3 Region Definitions and Event Selection

Regions for an analysis can be defined as shown in Listing 5 and events can be flagged as being accepted as shown in Listing 6 by specific regions.

```
// Preselection for debugging
addRegions({"presel_1L", "presel_2J", "presel_bb", "presel_met", "presel_mbb"});
```

Listing 5: A snippet of `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26] showing how to define regions in an analysis.

[1] See https://gitlab.cern.ch/atlas-sa/simple-analysis/-/blob/master/SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx.

[2] See a list of analysis codes available at https://simpleanalysis.docs.cern.ch/analyses/.

In Listing 5, `addRegions` is a function that takes a `std::vector<std::string>` of region labels. These are used to define branches (of type `float`, to hold event weights) in the resulting `ROOT::TTree`. Every event from the input file will be processed and correspond to an entry in the tree. To identify an event as falling within a region, the corresponding `accept("regionName")` call is used as shown in Listing 6. This allows an analyser to flag an event as being accepted by multiple non-disjoint regions at the same time.

```
if(N_signalLept == 1 && N_baseLept == 1) accept("presel_1L");
if(N_signalJets<=3 && N_signalJets >= 2) accept("presel_2J");
```

Listing 6: A snippet of `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26] showing how regions can accept events.

One can also just stop processing an event early with a **`return`** statement as in Listing 7. Combining early termination with `accept` provides a user-friendly way for building up a single-bin or multi-bin analysis.

```
// Preselection
if(N_baseLept != 1 || N_signalLept != 1) return;
if(N_signalJets>3 || N_signalJets < 2 || N_signalBJets != 2) return;
if(mt< 50. || met < 220.) return;
```

Listing 7: A snippet of `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26] showing the use of `return` for halting the processing of an event.

2.2.4 Multivariate Analysis Variables

Many searches are relying on multivariate analysis techniques such as boosted decision trees or neural networks for their event selection. In many cases these can also be evaluated meaningfully at generator level, and SA provides several helper functions for easily including some types of boosted decision trees and neural networks into the analysis code. In Listing 8 is an example of how to initialize a neural network stored in the open standard ONNX [27] format and calculate its value.

```
// In the initialization
addONNX("4jets","OneLeptonMultiJets2018_4jets.onnx");
...
// during event processing
MVA*  = getMVA("4jets");
// input to NN supplied in vector of floats
value = MVA->evaluate(nn_input_vector);
```

Listing 8: A snippet of code showing how to include and use a neural network in an analysis.

2.2.5 Additional Branches

Lastly, analysis teams might also provide additional output variables in the tree, alongside the regions, by using `ntupVar` as in Listing 9. The code currently supports simple numeric types: `int`, `float`,

`vector<int>`, and `vector<float>`. If there is no call to fill the branch for a given event, perhaps because of early termination via **`return`**, then this will be filled in with its corresponding C++ default value[3] for the branch type.

```
ntupVar("AnalysisType", (baseMuon.size() == 1) ? 2 : 1);
```

Listing 9: A snippet of `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26] showing how to create a new branch and fill it.

3 Implementation Validation

Analyses for SUSY in ATLAS implement the generator-level analysis code using the SimpleAnalysis framework. As part of the implementation, the analysis team performs validation using Monte Carlo samples by comparing events at the generator-level against the events at reconstruction-level. This comparison is done with the generator-level events as-is and with an ATLAS-internal fast detector simulation applied, which is not available in the public version of this framework. The purpose of the validation is to ensure that the generator-level analysis code accurately reflects the selection used in the full analysis. Since differences are expected due to reconstruction inefficiencies and resolution effects, applying the fast detector simulation makes it easier to detect possible selection code mistakes. In this section, the one lepton, two b-jet analysis [19] is used to demonstrate the validation.

The analysis targets signal events with a leptonically decaying W boson and a Higgs boson decaying into a $b\bar{b}$ pair. The signal regions are required to have exactly one lepton (electron or muon), and either two or three jets, of which two must be b-tagged. Thus all distributions shown in this section use a loose preselection summarised in Table 1. The standard validation study in Section 3.1 is done by comparing distributions of events at the generator-level and reconstruction-level, while Section 3.2 shows the impact of overlap removal.

Each validation study looks at six different kinematic variables:

- leading jet p_{T},
- leading lepton p_{T},
- transverse mass, m_{T},
- invariant mass of the two b-jets, $m_{b\bar{b}}$,
- number of b-jets, without the b-jet preselection, and
- missing transverse momentum, $E_{\mathrm{T}}^{\mathrm{miss}}$.

More details about the object definitions and how the kinematic variables are constructed can be found in Ref. [19].

[3] Typically zero-initialized.

	Preselection
N_{lepton}	$= 1$
$E_{\text{T}}^{\text{miss}}$	$> 50\,\text{GeV}$
m_{T}	$> 50\,\text{GeV}$
N_{jet}	$\in [2, 3]$
$N_{b\text{-jet}}$	$= 2$

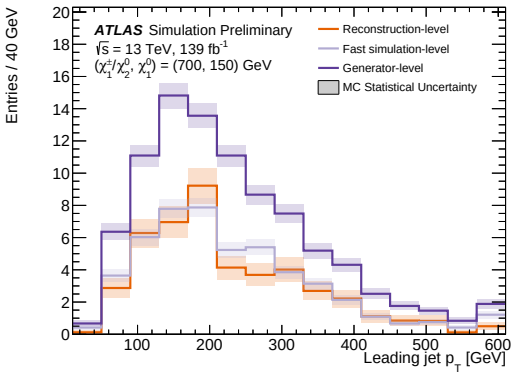
Table 1: Overview of the loose pre-selection criteria used for the SimpleAnalysis validation of the the one lepton, two b-jet analysis [19].

3.1 Yields at preselection

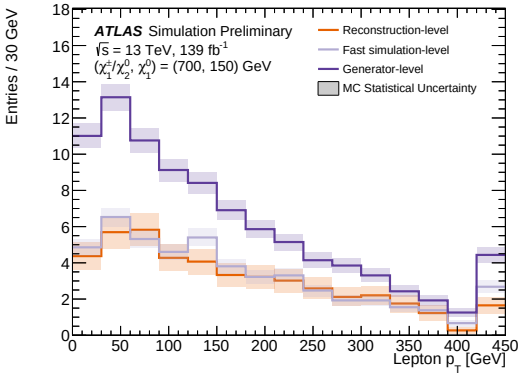
Figure 2 compares reconstruction-level kinematic distributions with generator-level distributions before and after a fast detector simulation. The largest impact on the overall normalisation of the generator-level distributions originates from the b-tagging efficiencies. Other object identification and reconstruction efficiencies considered mostly affect the kinematic distributions at low transverse momenta.

3.2 Impact of overlap removal

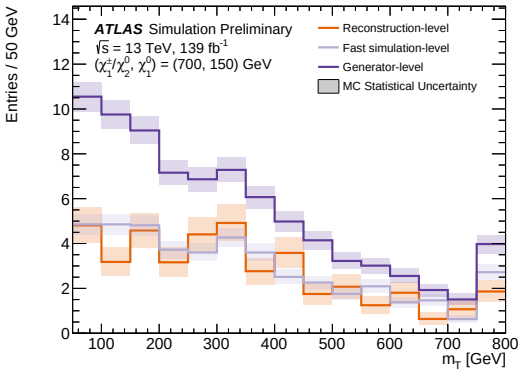
A single particle can be reconstructed as multiple physics objects, for instance an electron and a jet, and all ATLAS searches therefore apply a overlap removal procedure to remove objects that may be duplicates. Knowledge of the overlap removal procedure performed in an analysis is crucial to reproduce the final analysis results. Figure 3 shows the impact the overlap removal procedure has at generator-level using two kinematic observables and one signal mass point. Without the correct overlap removal, many events end up having additional objects in the final state, resulting in them not surviving the analysis selections.



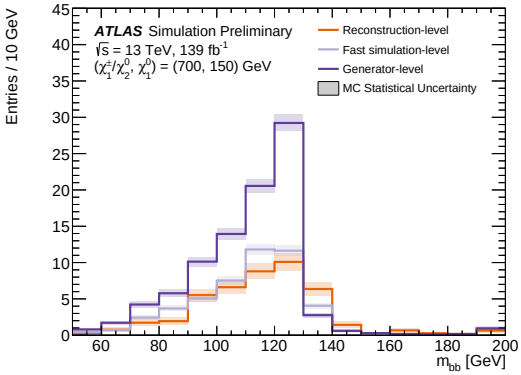
(a) Leading jet p_{T}



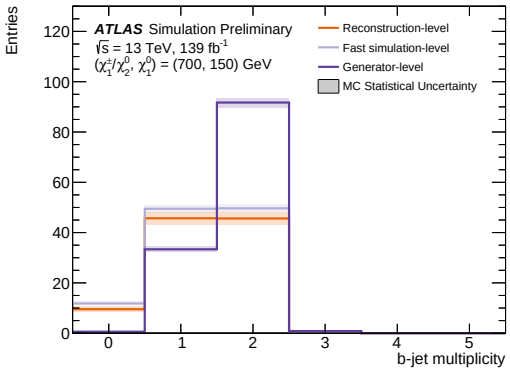
(b) Leading lepton p_{T}



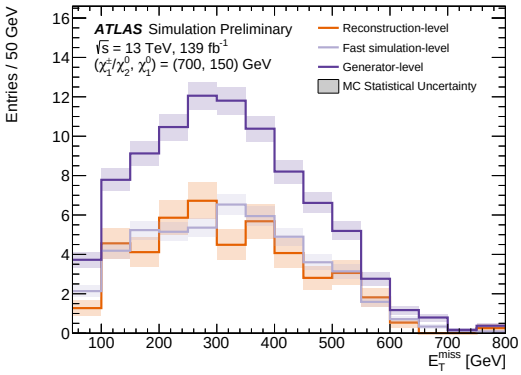
(c) Transverse mass



(d) Di-b-jet invariant mass

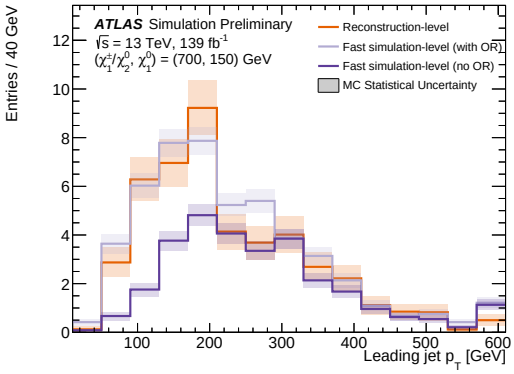


(e) Number of b-jets

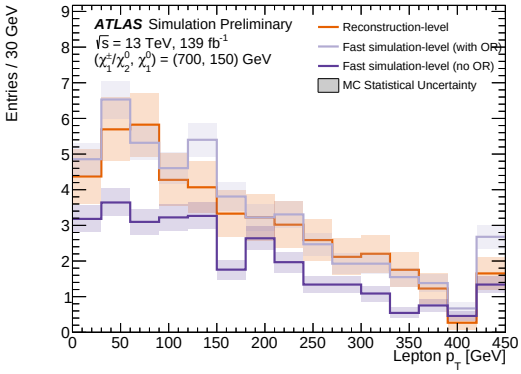


(f) Missing transverse momentum

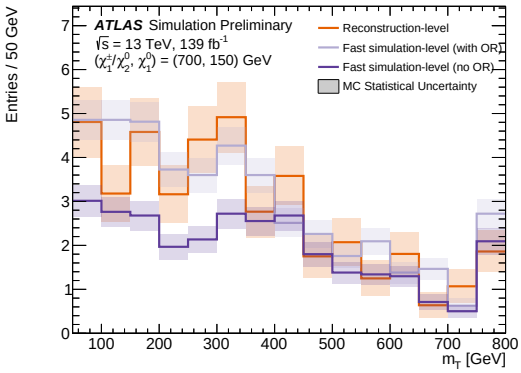
Figure 2: Comparisons of the kinematic distributions of key observables at generator- and reconstruction-level. The benchmark signal point with $m(\tilde{\chi}_1^{\pm}/\tilde{\chi}_2^0), m(\tilde{\chi}_1^0) = 700, 150\,\mathrm{GeV}$ is shown. Generator-level distributions without (dark purple) and with fast simulation (light purple) are compared to reconstruction-level distributions (orange). Only the MC statistical uncertainty is included in the error bars. For Subfigure (e) the requirement of two b-tagged jets has not been applied. The overflow appears in the last bin.



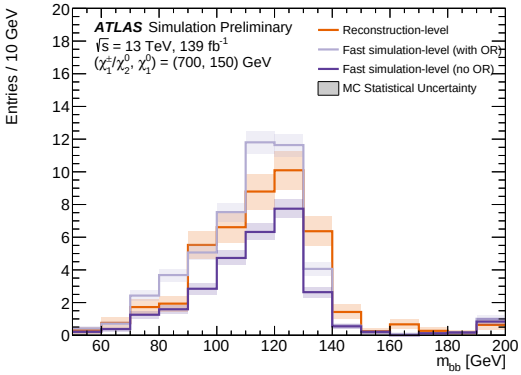
(a) Leading jet p_{T}



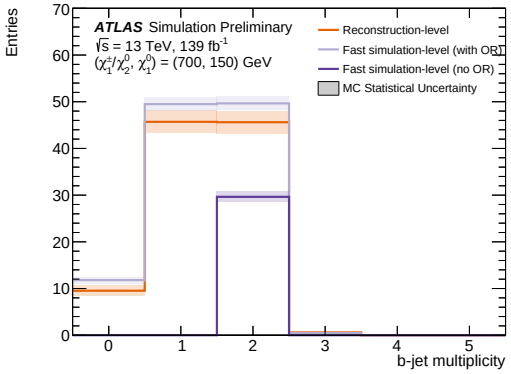
(b) Leading lepton p_{T}



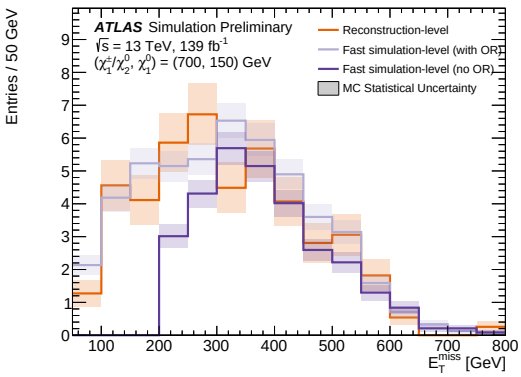
(c) Transverse mass



(d) Di-b-jet invariant mass



(e) Number of b-jets



(f) Missing transverse momentum

Figure 3: Comparisons of the kinematic distributions of key observables at generator- and reconstruction-level. The benchmark signal point with $m(\tilde{\chi}_1^{\pm}/\tilde{\chi}_2^0), m(\tilde{\chi}_1^0) = 700, 150\,\mathrm{GeV}$ is shown. Fast simulation-level distributions without (dark purple) and with (light purple) overlap removal are compared to reconstruction-level distributions (orange). Without overlap removal, many events do not satisfy the analysis selections due to additional objects in the final state. Only the MC statistical uncertainty is included in the error bars. For Subfigure (e) the requirement of two b-tagged jets has not been applied. The overflow appears in the last bin.

```cpp
#include "SimpleAnalysisFramework/AnalysisClass.h"
DefineAnalysis(EwkOneLeptonTwoBjets2018)

void EwkOneLeptonTwoBjets2018::Init() {
  // ...
  // Preselection for debugging
  addRegions({"presel_1L", "presel_2J", "presel_bb", "presel_met", "presel_mbb"});
}

void EwkOneLeptonTwoBjets2018::ProcessEvent(AnalysisEvent *event) {
  // baseline objects
  auto baseEle  = event->getElectrons(7,2.47, ELooseBLLH);
  auto baseMuon = event->getMuons(6,2.70, MuMedium | MuNotCosmic | MuZ05mm);
  auto baseJets = event->getJets(20.,4.5);
  auto met_Vect = event->getMET();
  auto weights  = event->getMCWeights();

  // overlap removal
  baseEle = overlapRemoval(baseEle, baseMuon, 0.01);
  baseJets = overlapRemoval(baseJets, baseEle, 0.2);
  // ...

  // signal objects
  auto signalEle = filterObjects(baseEle,7., 2.47, ETightLH | ED0Sigma5  | EZ05mm);
  auto signalMuon = filterObjects(baseMuon,6., 2.7, MuD0Sigma3 | MuZ05mm | MuIsoFCLoose);
  auto signalLept =  signalEle + signalMuon;
  auto signalJets = filterObjects(baseJets, 30., 2.80, JVT120Jet);
  auto signalBJets = filterObjects(signalJets, 30., 2.8, BTag77MV2c10);

  unsigned int N_baseLept = baseEle.size() + baseMuon.size();
  unsigned int N_signalLept = signalEle.size() + signalMuon.size();
  unsigned int N_signalJets = signalJets.size();
  unsigned int N_signalBJets = signalBJets.size();

  float mt=0, m_CT=0, mbb=0, mlb1=0;
  if (signalLept.size()==1 && signalBJets.size()==2) {
    mt   = calcMT(signalLept[0], met_Vect);
    m_CT = calcMCT(signalBJets[0],signalBJets[1],met_Vect);
    mbb  = (signalBJets[0]+signalBJets[1]).M();
    mlb1 = (signalBJets[0]+signalLept[0]).M();
  }

  if(N_signalLept == 1 && N_baseLept == 1) accept("presel_1L");
  if(N_signalJets<=3 && N_signalJets >= 2) accept("presel_2J");
  // ...

  // Preselection
  if(N_baseLept != 1 || N_signalLept != 1) return;
  if(N_signalJets>3 || N_signalJets < 2 || N_signalBJets != 2) return;
  if(mt< 50. || met < 220.) return;

  ntupVar("AnalysisType", (baseMuon.size() == 1) ? 2 : 1);
  // ...
  return;
}
```

Listing 10: A snippet of the SA implementation for the one lepton, two b-jet analysis [19]. This can be found in the SA framework under `SimpleAnalysisCodes/src/ANA-SUSY-2019-08.cxx` [26]. This version has been modified for this note.

4 Conclusions

Searches for new physics often use complicated variables or non-trivial techniques that make reinterpretation not-quite-so straightforward. Providing additional documentation in the form of an analysis implementation that has been validated is helpful for disseminating results and reinforces a publication. ATLAS has now released a project called SimpleAnalysis composed of the underlying base framework as well as the public analysis code implementations (and data files). This public version of the code incorporates all published analysis codes to date. This is the code being used for the generator-level SUSY studies, which was used for calculating theoretical uncertainties as well as being used in global scans. A central location [26] keeps all of the analysis codes that have been published to HEPData. The documentation [20] answers many questions about the code and includes a tutorial describing how to run and debug the framework.

The ATLAS collaboration is providing the code and logic to help readers understand the precise analysis techniques used, and SimpleAnalysis further enables the implementation of new analyses at generator-level. At the same time, more complicated analysis techniques are exposed such as boosted decision trees or neural networks. The analyses can be run over events in either ATLAS specific data formats, events in the HepMC standard MC event record format or the output of Delphes.

References

[1] C. Bierlich et al., *Robust Independent Validation of Experiment and Theory: Rivet version 3*, SciPost Phys. **8** (2020).

[2] E. Conte, B. Fuks and G. Serret, *MadAnalysis 5, A User-Friendly Framework for Collider Phenomenology*, Comput. Phys. Commun. **184** (2013) 222, arXiv: 1206.1599 [hep-ph].

[3] ATLAS Collaboration, *ATLAS HL-LHC Computing Conceptual Design Report*, CERN-LHCC-2020-015, 2020, URL: https://cds.cern.ch/record/2729668.

[4] M. Dobbs and J. B. Hansen, *The HepMC C++ Monte Carlo event record for High Energy Physics*, Comput. Phys. Commun. **134** (2001) 41.

[5] R. Brun and F. Rademakers, *ROOT – An object oriented data analysis framework*, Nucl. Instrum. Meth. A **389** (1997) 81, ISSN: 0168-9002.

[6] E. Maguire, L. Heinrich and G. Watt, *HEPData: a repository for high energy physics data*, J. Phys. Conf. Ser. **898** (2017) 102006, arXiv: 1704.05473 [hep-ex].

[7] ATLAS Collaboration, *Reproduction searches for new physics with the ATLAS experiment through publication of full statistical likelihoods*, ATL-PHYS-PUB-2019-029, 2019, URL: https://cds.cern.ch/record/2684863.

[8] ATLAS Collaboration, *Implementation of simplified likelihoods in HistFactory for searches for supersymmetry*, ATL-PHYS-PUB-2021-038, 2021, URL: http://cdsweb.cern.ch/record/2782654.

[9] L. Heinrich, M. Feickert and G. Stark, *pyhf: v0.6.2*, version 0.6.2, https://github.com/scikit-hep/pyhf/releases/tag/v0.6.2, URL: https://doi.org/10.5281/zenodo.1169739.

[10] L. Heinrich, M. Feickert, G. Stark and K. Cranmer,
pyhf: pure-Python implementation of HistFactory statistical models, JOSS **6** (2021) 2823.

[11] ATLAS Collaboration, *Summary of the ATLAS experiment's sensitivity to supersymmetry after LHC Run 1 — interpreted in the phenomenological MSSM*, JHEP **10** (2015) 134,
arXiv: 1508.06608 [hep-ex].

[12] K. Cranmer and L. Heinrich,
Analysis Preservation and Systematic Reinterpretation within the ATLAS experiment,
J. Phys. Conf. Ser. **1085** (2018) 042011.

[13] K. Cranmer and L. Heinrich,
Yadage and Packtivity - analysis preservation using parametrized workflows,
J. Phys. Conf. Ser. **898** (2017) 102019, arXiv: 1706.01878 [physics.data-an].

[14] K. Cranmer and I. Yavin, *RECAST: Extending the Impact of Existing Analyses*, JHEP **04** (2011) 038,
arXiv: 1010.2506 [hep-ex].

[15] ATLAS Collaboration,
Reinterpretation of the ATLAS Search for Displaced Hadronic Jets with the RECAST Framework,
ATL-PHYS-PUB-2020-007, 2020, URL: https://cds.cern.ch/record/2714064.

[16] T. Šimko, L. Heinrich, H. Hirvonsalo, D. Kousidis and D. Rodrìguez,
REANA: A System for Reusable Research Data Analyses, EPJ Web Conf. **214** (2019) 06034, ed. by A. Forti, L. Betev, M. Litmaath, O. Smirnova and P. Hristov.

[17] ATLAS Collaboration, *SimpleAnalysis*, version 1.1.0, 2022,
URL: https://doi.org/10.5281/zenodo.6365083.

[18] ATLAS Collaboration, *SimpleAnalysis*, 2022,
URL: https://doi.org/10.5281/zenodo.6328569.

[19] ATLAS Collaboration, *Search for direct production of electroweakinos in final states with one lepton, missing transverse momentum and a Higgs boson decaying into two b-jets in pp collisions at* $\sqrt{s} = 13$ *TeV with the ATLAS detector*, Eur. Phys. J. C **80** (2020) 691,
arXiv: 1909.09226 [hep-ex].

[20] ATLAS Collaboration, *SimpleAnalysis*, URL: https://simpleanalysis.docs.cern.ch/.

[21] MkDocs Team, *MkDocs*, 2021, URL: https://www.mkdocs.org/.

[22] P. Jackson and C. Rogan, *Recursive jigsaw reconstruction: HEP event analysis in the presence of kinematic and combinatoric ambiguities*, Phys. Rev. D **96** (2017).

[23] ATLAS Collaboration, *SimpleAnalysis Registry*,
https://gitlab.cern.ch/atlas-sa/simple-analysis/container_registry, 2021.

[24] ATLAS Collaboration, *Athena*, ATL-SOFT-PUB-2021-001, 21.2.158, 2021,
URL: https://cds.cern.ch/record/2767187.

[25] J. P. Araque Espinosa et al.,
A continuous integration and web framework in support of the ATLAS Publication Process, (2020),
arXiv: 2005.06989 [cs.DL].

[26] ATLAS Collaboration, *SimpleAnalysis Framework*,
https://gitlab.cern.ch/atlas-sa/simple-analysis, 2021.

[27] ONNX Community, *Open standard for machine learning interoperability*,
https://github.com/onnx/onnx, 2017.