Eur. Phys. J. C (2017) 77:795
https://doi.org/10.1140/epjc/s10052-017-5285-8

THE EUROPEAN PHYSICAL JOURNAL C

CrossMark

Special Article - Tools for Experiment and Theory

ColliderBit: a GAMBIT module for the calculation of high-energy collider observables and likelihoods

The GAMBIT Scanner Workgroup: Csaba Balázs[1,2], Andy Buckley[3,a], Lars A. Dal[4], Ben Farmer[5], Paul Jackson[2,6], Abram Krislock[4], Anders Kvellestad[7,b], Daniel Murnane[2,6], Antje Putze[8], Are Raklev[4,c], Christopher Rogan[9], Aldo Saavedra[2,10], Pat Scott[11,d], Christoph Weniger[12], Martin White[2,6,e]

[1] School of Physics and Astronomy, Monash University, Melbourne, VIC 3800, Australia
[2] Australian Research Council Centre of Excellence for Particle Physics at the Tera-scale, Australia, http://www.coepp.org.au/
[3] SUPA, School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK
[4] Department of Physics, University of Oslo, 0316 Oslo, Norway
[5] Oskar Klein Centre for Cosmoparticle Physics, AlbaNova University Centre, 10691 Stockholm, Sweden
[6] Department of Physics, University of Adelaide, Adelaide, SA 5005, Australia
[7] NORDITA, Roslagstullsbacken 23, 10691 Stockholm, Sweden
[8] LAPTh,Université de Savoie, CNRS, 9 chemin de Bellevue B.P.110, 74941 Annecy-le-Vieux, France
[9] Department of Physics, Harvard University, Cambridge, MA 02138, USA
[10] Faculty of Engineering and Information Technologies, Centre for Translational Data Science, School of Physics, The University of Sydney, Sydney, NSW 2006, Australia
[11] Department of Physics, Imperial College London, Blackett Laboratory, Prince Consort Road, London SW7 2AZ, UK
[12] GRAPPA, Institute of Physics, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands

Received: 15 March 2017 / Accepted: 6 October 2017

Abstract We describe ColliderBit, a new code for the calculation of high energy collider observables in theories of physics beyond the Standard Model (BSM). ColliderBit features a generic interface to BSM models, a unique parallelised Monte Carlo event generation scheme suitable for large-scale supercomputer applications, and a number of LHC analyses, covering a reasonable range of the BSM signatures currently sought by ATLAS and CMS. ColliderBit also calculates likelihoods for Higgs sector observables, and LEP searches for BSM particles. These features are provided by a combination of new code unique to ColliderBit, and interfaces to existing state-of-the-art public codes. ColliderBit is both an important part of the GAMBIT framework for BSM inference, and a standalone tool for efficiently applying collider constraints to theories of new physics.

Contents

[a] e-mail: andy.buckley@glasgow.ac.uk
[b] e-mail: anders.kvellestad@nordita.org
[c] e-mail: ahye@fys.uio.no
[d] e-mail: p.scott@imperial.ac.uk
[e] e-mail: martin.white@adelaide.edu.au

1 Introduction

Despite decades of searches for physics beyond the Standard Model (BSM), we still lack an unambiguous discovery of such physics. The many null results from the Large Hadron Collider (LHC) and other experiments allow us to constrain, to various degrees, the parameter spaces of many extensions of the Standard Model (SM). These include effective theories and simplified models of dark matter, supersymmetric theories, theories with extra space dimensions and composite Higgs models. Because even the most minimal realistic theories of BSM physics have observable consequences in multiple experiments, it is particularly important to combine collider exclusions with other experiments in a statistically rigorous way if one is to draw sound conclusions on the viability of a theory.

Rigorously taking into account the sum of data relevant to a given model from the many disparate experimental sources has become a challenging task. This problem is addressed in GAMBIT (the Global And Modular Beyond-the-Standard-Model Inference Tool) [1], which combines calculations of observables and likelihoods in collider, flavour, dark matter and precision physics with a model database, a flexible system for interfacing to external codes, and a wide selection of different statistical methods and parameter scanning algorithms that can be applied to the models [1]. In this paper, we introduce ColliderBit, a GAMBIT module for the application of high-energy collider constraints to BSM physics theories.

The ATLAS and CMS experiments [2,3] have made great progress in the search for evidence of BSM physics at high energies, but applying these constraints to a generic theory of such physics remains challenging. Searches for new particles at the LHC are typically presented either in specific planes of a restrictive high scale physics hypothesis, *e.g.* the constrained minimal supersymmetric model (CMSSM), or in simplified models that strictly apply only to a very small volume of the total allowed space of particle masses and branching ratios. The computational expense of simulating signal processes for hundreds of thousands of points in a candidate model prevents an extended treatment by the experiments. In addition, some LEP results remain useful [4–15], and are not always rigorously applied in the literature. Finally, the discovery of an SM-like Higgs boson [16,17] by the ATLAS and CMS experiments in 2012 – and the subsequent measurement of its properties – provides tight constraints on variations in the Higgs branching ratios, which must be included in any thorough exploration of a BSM physics model. Given the ever-growing list of constraints on BSM physics from experiments at the LHC, the need to rigorously test those limits against various models is ever more pressing.

Partial solutions to each of these issues exist, but there is as yet no comprehensive tool that tackles all of them. The package SModelS applies constraints to supersymmetric (SUSY) models based on a combination of simplified model results [18]. FastLim provides similar functionality for SUSY models, but is extendible (in principle) to non-SUSY models through the use of user-supplied efficiency tables [19]. Both of these tools will provide limits that are much more conservative than a more rigorous calculation, due to the limitations of simplified models. SUSY-AI [20] provides a random forest classifier for SUSY models based on LHC exclusions, but as seen in earlier applications of machine learning to this problem [21–25], accuracy concerns exist when applying the method to large-volume parameter spaces, due to the relative sparsity of the training data [26] in the model parameter space. Other approaches to SUSY model exclusion based on machine learning can be found in [27,28]. CheckMATE provides a customised version of the Delphes detector simulation, an event analysis framework and a list of ATLAS and CMS analyses that can be used to apply LHC limits, and includes an interface to MadGraph5_aMC@NLO for event generation [29–32]. However, the time required to run a single BSM parameter combination through CheckMATE makes large-scale parameter scans a difficult prospect, and integration with a global fitting framework is not within the scope of the package. To the best of our knowledge, no general purpose tool exists to apply LEP BSM search limits, although many theorists have implemented their own local codes over the years. Packages such as HiggsBounds [33–36], HiggsSignals [37] and Lilith [38] allow the user to apply constraints on Higgs physics.

As ColliderBit is designed within the GAMBIT framework [1], it offers seamless integration with modules that provide statistical fitting [1,39], the ability to impose constraints from electroweak precision data [40], flavour physics [41] and a large range of astrophysical observations [42]. For LHC physics, we use a combination of parallelised Monte Carlo (MC) simulation and fast detector simulation to recast LHC limits without the approximations of the simplified model approach. The first release of the code comes with a list of ATLAS and CMS analyses that collectively present strong constraints on supersymmetry and dark matter scenarios [43–53]. It contains interfaces to the Pythia 8 MC event generator [54,55], to the Delphes detector simulation [30,31], and a customised detector simulation based on four-vector smearing (BuckFast). In this paper we show that that BuckFast gives comparable results to Delphes, but at a dramatically lower CPU cost. We also supply custom routines for re-evaluating LEP limits on supersymmetric particle production, and include interfaces to HiggsBounds and HiggsSignals for calculating Higgs observables. ColliderBit follows the modular design of GAMBIT, thus enabling the user to easily swap components (e.g. choose a different detector simulation without affecting the LHC analysis framework), add new collider analyses, or provide interfaces to standard particle physics tools.

This paper serves as both a description of the physics and design strategy of ColliderBit, and a user manual for the first code release. In Appendix A, we provide a quick start guide for users keen to compile and use the software out of the box. Sect. 2 describes the physics and implementation of the ColliderBit software. The ColliderBit user interface is outlined in Sect. 3. In Sect. 4 we cover two use cases: first, we point to an annotated GAMBIT input file that details the application of collider constraints in a scan of the constrained minimal supersymmetric standard model (CMSSM). Second, we provide a detailed example of how the user can add their own model to the ColliderBit code. The second of these examples shows the flexibility of ColliderBit in tackling generic theories supplied by the user, using existing codes for automatic generation of matrix elements. After summarising in Sect. 5, we also provide Appendices B and C, where we detail the C++ classes defined by ColliderBit, and a glossary of common GAMBIT terms, respectively.

ColliderBit is released under the terms of the 3-clause BSD license,[1] and can be obtained from gambit.hepforge.org.

2 Physics and implementation

To perform any calculations, ColliderBit requires numerical values for the free parameters of a theory for new physics. If ColliderBit is run with other GAMBIT modules, these will come from a scanning algorithm implemented in the ScannerBit [39] module, and other GAMBIT modules will then perform the necessary spectrum generation and decay rate calculations. The user may also run ColliderBit as a standalone code, in which case the parameters can be supplied via a model description, such as an SLHA file for supersymmetric models [59,60]. In this case, the user must supply spectrum and/or decay calculations as appropriate. The ColliderBit output is a series of signal event rate predictions and likelihood terms derived from BSM searches at the LHC, as well as likelihood terms from SUSY searches at LEP and Higgs searches at LEP, the Tevatron and the LHC. The terms may then be combined according to the user's request, to form a composite likelihood. Here we describe the strategy for calculating each individual likelihood term, along with the code implementation.

2.1 LHC likelihood calculation

2.1.1 Overview of LHC constraints included in ColliderBit

As the flagship collider at the energy frontier, the LHC provides the most stringent constraints on BSM physics models in the majority of cases. The search groups of the ATLAS and CMS experiments provide long lists of results using data from LHC proton–proton collisions taken at $\sqrt{s} = 7, 8$ and 13 TeV, including searches for specific particles encountered in BSM physics models, and generic resonances in a multitude of final states [61–64].

Implementing the full list of LHC constraints is a daunting task. The initial approach taken in ColliderBit is to provide a representative set of searches that run out-of-the-box, supplemented by a framework that makes it easy to add new LHC analyses. ColliderBit includes a selection of Run I and Run II LHC analyses, chosen for their relevance to supersymmetry and dark matter simplified model applications.

The **Run I** analyses included are:

– *ATLAS 0-lepton supersymmetry search* This targets squark and gluino production, and is the most constraining single ATLAS SUSY analysis in cases where the gluino, some or all squarks are expected to be light. The analysis looks for an excess of events in various signal regions defined by the jet multiplicity, the missing energy and other kinematic variables [49].
– *ATLAS and CMS third generation squark searches* It is possible for supersymmetry to remain a natural theory with only the third generation squarks accessible at LHC energies. In the limit of large stop mixing, only one squark may be light enough to be observed. Increasing theoretical interest in naturalness has prompted a series of optimised searches for top squarks in recent years, focussing primarily on stop decays to a top quark and the lightest neutralino, or to b quarks and charginos with subsequent chargino decay via an on- or off-shell gauge boson. ColliderBit includes ATLAS searches for top squarks in 0-lepton, 1-lepton and 2-lepton final states [44,45,48], and the CMS 1- and 2-lepton searches [51,52]. We also include the ATLAS b-jets plus MET search [43], which

[1] http://opensource.org/licenses/BSD-3-Clause. Note that fjcore [56] and some outputs of FlexibleSUSY [57] (incorporating routines from SOFTSUSY [58]) are also shipped with GAMBIT 1.0. These code snippets are distributed under the GNU General Public License (GPL; http://opensource.org/licenses/GPL-3.0), with the special exception, granted to GAMBIT by the authors, that they do not require the rest of GAMBIT to inherit the GPL.

targets direct sbottom production. All of these searches are also expected to strongly constrain simplified dark matter models with a mediator that couples preferentially to third generation fermions. These models, which have gained popularity as explanations of the *Fermi*-LAT Galactic Centre excess [65], give rise to similar final states and, indeed, the CMS searches that we implement have already been used in a non-supersymmetric dark matter context [66].

– *ATLAS and CMS multilepton supersymmetry searches* In the case that all coloured superpartners are too heavy to observe at the LHC, electroweak gaugino searches are the only hope of finding evidence for supersymmetry. Even if coloured superpartners are accessible, direct searches for the electroweak gauginos would provide extra information on the parameters of the neutralino and chargino mixing matrices, in addition to telling us whether the weak gaugino sector is that of the MSSM, or an expanded sector from an exotic supersymmetric scenario. ColliderBit includes the 2- and 3-lepton ATLAS electroweak gaugino searches [46,47] and the CMS 3-lepton electroweak gaugino search [50], which should provide the dominant constraints on the electroweak sector of the MSSM.
– *Dark matter searches* The classic technique for searching for dark matter at colliders is to look for events with a monojet plus missing energy. This signature results from pair production of a dark matter candidate, with the jet arising from QCD radiation. ColliderBit includes the CMS monojet search [53], which provides a constraint on various dark matter scenarios, in addition to supersymmetric scenarios with compressed spectra. Some caution must be taken when applying this to e.g. dark matter effective field theories. In cases where NLO QCD effects are significant, the user will need to interface GAMBIT to a suitable Monte Carlo generator capable of modelling these effects.

We also provide the ATLAS and CMS **Run II** (13 TeV) 0-lepton supersymmetry searches, based on 13 fb^{-1} of analysed data [67,68]. More **Run II** analyses will be added to ColliderBit in the near future, including searches sensitive to R-parity violating supersymmetry such as Ref. [69].

We consider this a reasonable minimum of LHC searches for covering a wide range of LHC phenomenology, but the average user will no doubt be keen to expand the collection. New analyses will be continuously added to the code repository, and information on how the user can add a new LHC analysis to ColliderBit is given in Sect. 3.1.1. It is worth noting, however, that the general treatment of the LHC analyses in ColliderBit means that even the LHC Run I results can provide previously unavailable insights when used to constrain models with large parameter spaces. We also emphasise that the above list does not contain searches for SUSY scenarios with compressed sparticle spectra. Since we use the LO Pythia generator in the current ColliderBit release, we would obtain less precise results than the ATLAS and CMS publications that use MadGraph5_aMC@NLO to explicitly model initial state jet radiation through the addition of the relevant diagrams to the tree-level sparticle production process.

In the rest of this section, we describe the process by which LHC analysis constraints are derived without employing any model-dependent assumptions, following a full simulation of proton–proton collisions, including detector effects and an approximation of the ATLAS and CMS statistical procedures.

2.1.2 Strategy for applying LHC constraints without model-dependent assumptions

A parameter point of a specified BSM model can in principle be expected to show up in a variety of LHC BSM searches. For counting analyses, the relevant data to model are the number of events that pass kinematic selection criteria (for brevity referred to in what follows as 'cuts') imposed in each analysis. If a model predicts that s signal events will pass the cuts for a given signal region, and b background events are expected from known SM processes, the likelihood of observing n events is given by the standard Poisson formula,

$$\mathcal{L} = \frac{e^{-(s+b)}(s+b)^n}{n!}. \tag{1}$$

For now, we neglect effects of systematic uncertainties in the signal and background yields – but we return to this point in Sect. 2.1.8. LHC BSM search papers provide details of b and n for each signal region, along with the background uncertainty, and some estimate of the signal uncertainty for representative models.

Calculating the likelihood for a given model thus requires an accurate estimate of s. This is given by

$$s = \sigma\epsilon A L, \tag{2}$$

where σ, ϵ and A are the process-specific production cross-section, detector efficiency and acceptance, respectively. L is the integrated luminosity of data used in the search.

The rigorous way to calculate s is to perform a cross-section calculation at the highest practically achievable level of accuracy in perturbation theory, before evaluating the acceptance and efficiency via a Monte Carlo simulation of the LHC collisions. This is usually augmented by simulating the reconstructed signatures of the Monte Carlo events in the relevant detector – ATLAS or CMS in the case of direct BSM searches at the LHC. One can then apply the analysis cuts for a given LHC search to the results of the detector

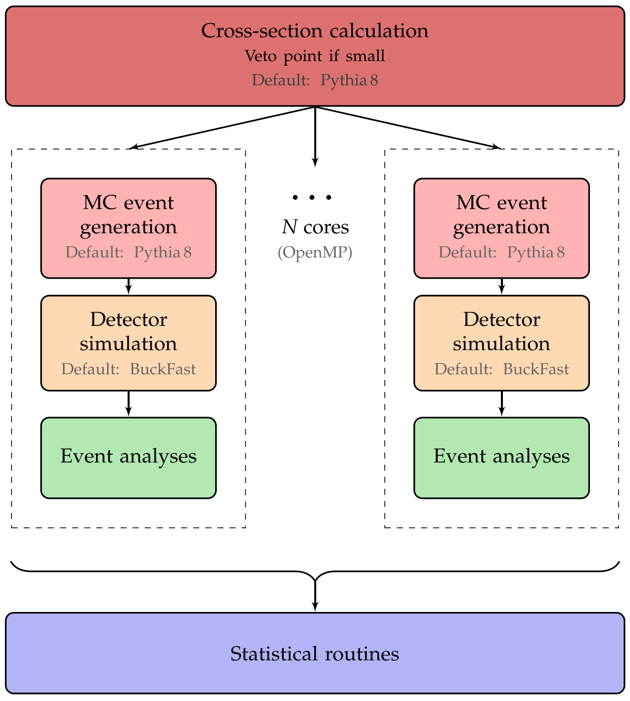


Fig. 1 Schematic diagram of the ColliderBit processing chain for LHC likelihoods

simulation. An approach using look-up tables for efficiencies and extrapolations from simplified models removes the need for time consuming simulation, but tends to give very conservative results as a consequence. This is because the approach misses models that do not resemble the simplified models under consideration, but still have some acceptance to the analysis cuts that are used to generate the simplified model results. Furthermore, generating look-up tables must be repeated for the parameter space of every physics model of interest, making it hard to produce a generic code for the application of LHC constraints.

The core strategy of the ColliderBit LHC module is instead to make each step of the simulation chain faster, using a combination of custom speed increases and parallel computing. The package thus performs a cross-section calculation, generates Monte Carlo events, performs an LHC detector simulation and then applies the analysis cuts for a range of LHC analyses, using a custom event analysis framework. The user can then utilise the GAMBIT statistical routines to return LHC likelihoods. The basic processing chain is illustrated in Fig. 1. The code is designed so that the user can choose, among available options, which software performs each step of this process, or, as an alternative, add an interface to an external code in place of an implemented option. Nevertheless, ColliderBit has a default chain implemented, and the current version contains the elements summarised in the following subsections.

2.1.3 *Cross-section calculations*

ColliderBit uses the LO + LL cross-sections calculated numerically by the Pythia 8 event generator [54,55]. For many models, these are the state-of-the-art. For models where an NLO (or better) calculation exists, e.g the MSSM, this is a conservative approximation, as the K-factors are predominantly greater than one. The LO + LL MSSM cross-sections are considerably quicker to evaluate than the full NLO results obtained using e.g. Prospino [70–72]. A single evaluation of just the strong production cross-sections for a CMSSM benchmark point, with all relevant processes kinematically available, takes around 15 minutes of CPU time on a modern processor using Prospino 2.1 (Intel Core i5 at 2.6 GHz). This is clearly unusable in a scan where the evaluation of a single parameter point must be done in times on the order of a few seconds. For strong production there exist pre-computed grids of NLO cross-sections with added (N)NLL corrections, which in combination with fast interpolation routines allow accurate cross-sections to be obtained within fractions of a second [73–79]. However, these interpolations are limited to models where all squarks except the stops are mass degenerate. While this approximation is suitable for many lower-dimensional parametrisations of the MSSM, it is not sufficiently general to serve as a default MSSM cross-section calculator for ColliderBit.

With the improvement to NLO + NLL, the error from the factorisation and renormalisation scales has been shown to be as low as 10% [76] for a wide range of processes and masses; however, PDF and α_s uncertainties must be included in the total error budget. These increase with the sparticle masses because the PDFs are most poorly constrained at large scales and at large parton x. As an example, at 8 TeV NLL-fast 2.1 [75–79] gives errors of $(+24.3, -22.2\%)$ and $(+8.3, -7.3\%)$, for the PDF and α_s, respectively, using the MSTW2008 NLO PDF set [80], with gluino and squark masses set to 1.5 TeV. Because 1.5 TeV is at the edge of the LHC reach at that energy, the total error budget here will not drop much below 25% even with NLO + NLL cross-sections.[2]

In light of the above, we take the conservative path of calculating likelihoods with the LO Pythia 8 cross-sections for the LHC. Assigning errors to these cross-sections is rather meaningless, considering the monotonic nature of LO scale-dependence, and the fact that the LO cross-sections in BSM models are known to almost always lie significantly below the NLO and higher order cross-section, sometimes by as

[2] With the CTEQ6.6M PDF set [81], the errors increase to $(+63.1, -38.5\%)$ and $(+15.6, -10.3\%)$; *these* uncertainties will reduce somewhat as PDF fits including higher-x LHC data become available.

Table 1 Time taken for the ColliderBit LHC likelihood calculation as a function of the number of cores, for 100,000 SUSY events at the SPS1a parameter point [83,84], including all sub-processes. The processes were run on a single computer node, with ISR, FSR, and full hadronisation enabled, but multiple parton interactions and tau decay spin correlations disabled. GAMBIT was compiled with full optimisation settings (cf. Sect. 11 of Ref. [1])

Num. cores	t (10^5 events) (s)	Speed-up
1	421	1
4	128	3.3
8	67	6.3
16	38	11.1
20	33	12.8

Table 2 Single-thread CPU effects of sequentially disabling event simulation components, for 100,000 SUSY events at the SPS1a parameter point [83,84], including all sub-processes. The disabled components have a major effect on CPU, and a minor (sometimes even positive) effect on physics performance. The third row corresponds to the first row in Table 1. Note that the few percent difference is typical of the variation with local CPU load on the cluster on which this was tested

Configuration	t (10^5 events) (s)	Speed-up
All	1529	1
$\hookrightarrow$ −MPI	516	3.0
$\hookrightarrow$ −τ correlations	434	3.5
$\hookrightarrow$ −FSR	195	7.8
$\hookrightarrow$ −hadrons	102	15.0

much as a factor of two.[3] The LO cross-sections are hence nearly always more conservative than the lower edge of the most pessimistic NLO uncertainty band due to renormalisation scale systematics. Accordingly, we do not apply any systematic theory error to our cross-sections, as any error due to finite statistics in the event generation is dwarfed by the systematic underestimation of the cross-sections due to the LO approximation. We have verified that these choices, combined with the approximations used in the event and detector simulation, result in limits equal to or more conservative than those in the included ATLAS and CMS analyses (see Sect. 2.1.7).

In future releases, we will allow the user to supply cross-sections and associated uncertainties as input to the LHC likelihood calculation, making it possible to calculate them using any preferred choice of external code (known in GAMBIT as a **backend**).

2.1.4 Monte Carlo event generation

For the ColliderBit event generation, we supply an interface to the Pythia 8 [54,55] event generator, alongside custom code that parallelises the main event loop of Pythia using OpenMP.[4] This substantially reduces the runtime, as seen in Table 1. In a parameter scan with GAMBIT the parameter sampling is parallelised using MPI. The additional OpenMP parallelisation of the LHC likelihood calculation in ColliderBit, along with similarly parallelised calculations in other GAMBIT modules, helps GAMBIT's overall scan performance to scale beyond the number of cores that the sampling algorithm alone can make efficient use of.

For the purposes of BSM searches, many time-consuming generator components also add little to the quality of relevant physics modelling, and can therefore be safely disabled. The single-threaded timing effects of sequentially disabling "soft physics" modelling such as multi-parton interactions (MPI), τ polarisation, QCD final-state radiation (FSR), and hadronisation are shown for a typical SUSY model point in Table 2. Of the model components shown, removal of MPI and tau correlations give the clearest gains. The detailed tau decay correlation mechanism is not generally relevant for BSM hard processes. LHC jet reconstruction includes a jet area correction [85] that removes the effects of pile-up and MPI on average, so disabling MPI is actually a *more* appropriate physical configuration than enabling it – and delivers a 60% CPU cost saving to boot.

The choices for FSR and hadronisation are less clear: these are responsible for production of realistic track and cluster multiplicities and energies on which detector simulation can be run. Completely disabling FSR – which mainly produces internal jet structure, not relevant to most BSM analyses – and all hadron-level processes including both hadronisation and decays, are both rather drastic options. In practice there are intermediate alternatives, such as raising the low-p_T cutoff of FSR evolution to balance CPU cost against physical accuracy, or to produce physical primary hadrons but elide simulation of their decays.

By default ColliderBit runs in the mode with MPI and "sophisticated" tau decays disabled; there is potential for further significant speed-up if the hadron-level or FSR simulation steps can be reduced, perhaps by use of specialised detector smearing to compensate for the biased final state particle distributions.

This combination of multi-threading and reduced generator functionality allows generation of 20,000 all-subprocess SUSY events in about 7 s on an Intel Core i7 processor using 8 cores, provided that the compilation makes use of the gcc option `--ffast-math`, or a suitable equivalent. Generating 100,000 events with the same settings and number of cores takes 19 s. When including FSR and hadronisation, as per the ColliderBit default, the time required to generate 20,000 and 100,000 events increase to 17 and 67 s, respectively.

In the above examples a factor 5 increase in the number of generated events only lead to a factor 2.5–4 increase in

[3] For a recent thorough exploration of K-factors in the MSSM up to approximate NNLO+NNLL order see [74] and Fig. 2 within.

[4] For an earlier similar approach, see Ref. [82].

the evaluation time. This illustrates that when the number of generated events is fairly low, other parts of the calculation besides the event loop itself account for a noticable fraction of the total evaluation time. The most important contribution comes from the initialisation of Pythia 8. While this step has not been parallelised, we have optimised the ColliderBit Pythia 8 initialisation so that per-thread copies of the Pythia objects are only constructed at the beginning of a GAMBIT sampling run, only requiring re-initialisation of process-specific physics components for each new model point. This produces a further speed increase in realistic applications.

In addition, in a GAMBIT-driven global fit, the event generation for a point can be skipped on the basis of the initial estimated maximum cross-section. If this is already too low to lead to observable consequences at the LHC, running the event generator is pointless, so skipping that step for some fraction of parameter points gives a further average speed increase. Event generation is also aborted if Pythia returns an error from the `pythia.next()` call. In both cases the contribution to the log likelihood, see Sect. 2.1.8, is set to zero.

Taken together, these routines make it computationally tractable to run a full Monte Carlo simulation in a global fit.

The choice of the Pythia generator is an acceptable compromise between generality and ease of use for the first ColliderBit release. It is sufficient for many BSM models, and is easily extendable with matrix elements for new models via the existing MadGraph5_aMC@NLO interface [32]. For an example, see Sect. 4.2. Pythia will prove insufficient, however, in cases where NLO corrections are significant – for example in the accurate treatment of some effective field theories of dark matter, where top quark loops become important [66]. These deficiencies can be addressed in the current release via a user-supplied interface to an appropriate Monte Carlo tool, and such interfaces will be supplied in future ColliderBit releases.

2.1.5 Event record

ColliderBit provides a custom set of event record classes that are independent of the particular choice of event generator or detector simulation. These are: a `P4` momentum 4-vector; `Particle` and `Jet`, which respectively add particle ID and flavour-tagging information to `P4`; and an `Event` container. The latter is used to store the particles in discrete categories of photons, electrons, muons, taus and invisibles, as well as a jet container and a missing momentum vector.

These event objects should be populated by conversion routines attached to the interface to each MC generator, allowing the different event structure conventions of each generator to be treated correctly. The conversion may be done either at parton or particle level. Parton-level conversion is primarily intended for speed, as it allows the most CPU-intensive parts of the event generation to be skipped, at some cost to physical accuracy. The description below concerns the complete particle-level variant, but the parton-level version only differs from it in a few minor details.

First, ColliderBit loops over the contents of each event, looking separately for decayed and stable particles. The former are only used to find b quarks[5] and hadronic taus for later jet tagging; following the established Rivet MC analysis system [87], only stable particles are used for constructing the kinematics of truth-level events, making the detector simulation and analysis more robust.

We require identified final-state leptons and photons to be "prompt", i.e. their ancestry is recursively checked to ensure that they have not been produced (even indirectly) in hadron decays. All final-state particles other than muons and invisibles are used as inputs to jet finding, which is performed using the FastJet [56] implementation of the anti-k_t jet algorithm [88]. We set the anti-k_t R parameter to 0.4 for Run I ATLAS BSM searches, 0.5 for corresponding CMS analyses, and 0.4 for both ATLAS and CMS Run II analyses. We use ΔR matching between jets and the unstable tagging objects to set appropriate jet attributes. ColliderBit computes missing momentum from the vector sum of the momenta of the invisible final-state particles within a geometric acceptance of $|\eta| < 5$.

The resulting `Event` is then passed on down the ColliderBit chain: first for modification by detector simulation, and then in read-only form to the analysis routines.

2.1.6 Detector simulation

ColliderBit is structured so that the detector simulation is run during the main parallelised event loop, implicitly speeding up the simulation step. The user has several options for this step.

No detector simulation The user can choose not to perform any detector simulation, in which case the truth-level MC events described above are passed directly to the event analysis framework without modification. Jets may be defined directly at the parton level, or at the hadron level. The former is only really sufficient for analyses in which leptons are the main species of interest, in which case turning off hadronisation can lead to a large speed increase, as seen in Table 2.

Delphes We provide an interface to the Delphes 3.1.2 detector simulation [30,31], which provides simulations of the ATLAS and CMS detectors. Delphes includes a simulation of track propagation in the magnetic field of an LHC

[5] We also tested final b-hadrons during validation, and found that their identification with ColliderBit differed significantly from the known performance of ATLAS [86]. As experimental flavour-tagging algorithms evolve, it will become necessary for the tagging algorithms in ColliderBit to be made more configurable.

detector, along with a simulation of the electron and hadron calorimeters, and the muon chambers. The user can configure the parameters of the simulation using the normal Delphes mechanism, but it should be noted that b- and τ-tagging, and the ATLAS lepton ID selection efficiencies ("medium", "tight", etc.), are controlled explicitly within the ColliderBit event analysis codes, to allow different analyses to use different calibration settings. Delphes has been interfaced with ColliderBit such that it can be passed single events via memory, rather than performing several passes over a large sample of MC events in pre-produced files, as in its usual mode of operation.

BuckFast For most purposes, a more approximate approach based on four-vector smearing is sufficient. We supply an internal ColliderBit detector simulation, BuckFast, which uses particle and jet resolution and efficiency functions based on those in Delphes, plus parametrised ATLAS electron identification efficiencies. New parametrisations are being added as Run 2 performance data becomes public.

The components of the BuckFast simulation are:

- **Electrons**: We apply the Delphes functions for electron tracking efficiency, electron energy resolution and electron reconstruction efficiency (in that order) to the truth-level electron four-vectors. In the analysis step, we apply parametrisations of the ATLAS electron identification efficiencies as appropriate, taken from Refs. [89,90].
- **Muons**: We apply the Delphes functions for the muon tracking efficiency, the muon momentum resolution and the muon reconstruction efficiency (in that order) to the truth-level muon four-vectors.
- **Taus**: Hadronic taus are identified at truth level. Leptonic taus are discarded. For both ATLAS and CMS the hadronic tau momentum is smeared by a 3% Gaussian resolution. Tau tags are applied to jets found within $\Delta R < 0.5$ of the true hadronic taus. Flat efficiencies are applied to tau tagging *in the analysis code* to allow use of different tagging configurations within the analyses of each experiment.
- **Jets**: Jets are reconstructed at hadron level using the anti-k_t algorithm, implemented in the FastJet package. All fiducial final-state particles other than invisibles and muons are used in jet finding, mimicking typical LHC jet calibration. For both ATLAS and CMS the jet momentum is smeared by a 3% Gaussian resolution chosen for compatibility with Delphes' constituent-level smearing.
- **b-jets**: Truth-level jet tags are obtained by matching jets to final b-partons for $\Delta R < 0.4$; a more robust approach using final b-hadrons is also available, but by construction agrees less well with the parton-based Delphes and LHC Run 1 tagging calibrations. As for taus, tagging efficiencies and mistag rates parametrised in η–p_T are applied in each analysis code to allow the use of different tagger configurations in different analyses.
- **Missing energy (MET)**: MET is constructed at generator level by summing the transverse momenta of invisible particles within the acceptance of the detector, and all particles outside the acceptance. No "soft-term" MET smearing is currently applied, since for events with real hard-process invisible particles the ATLAS reconstruction of $E_T^{\rm miss}$ is within a few percent of the true value at all scales, and within 1% above 70 GeV [91]. The same approach is taken to define the "truth MET" in the "no simulation" mode.

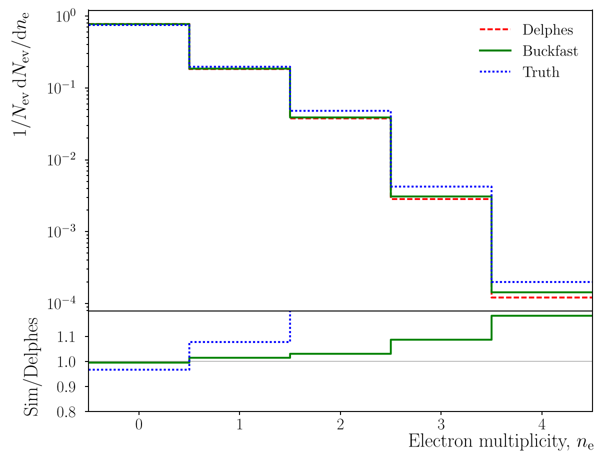
Figure 2 shows example performance of BuckFast, with comparisons to Delphes and no-simulation processing. For this example, we choose a CMSSM point close to the current ATLAS 95% exclusion contour, consistent with the measured Higgs boson mass: $m_0 = 2000$ GeV, $m_{1/2} = 600$ GeV, $A_0 = -4000$ GeV, $\tan\beta = 30$, $\mu > 0$.

The major effects of detector simulation are seen to be due to lepton efficiencies, with explicit resolution modelling producing relatively minor effects. BuckFast and Delphes typically agree to within a few percent for leptons, but some larger differences remain for b-jets (due to truth-tag definition) and missing E_T. The latter is currently unsmeared in BuckFast, but the origin of the deviation at high-$E_T^{\rm miss}$ is unclear since the reconstructed ATLAS $E_T^{\rm miss}$ closely matches the truth value in the BSM search region above 70 GeV [91]. The impact of these discrepancies on an ATLAS analysis dominated by b jets and $E_T^{\rm miss}$ is shown in Table 4.

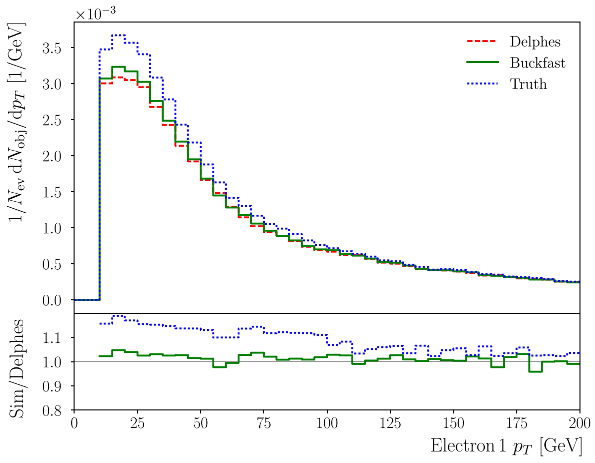
BuckFast is significantly faster than Delphes. One reason for this is that the operations it performs are computationally simpler, and should complete in fractions of a second. The other, more signficant, reason is that the ROOT framework on which Delphes is based is not thread-safe, so must be run serially within an OpenMP `critical` block. In contrast, BuckFast can be run in parallel along with our parallelised version of Pythia 8 (cf. Sect. 2.1.4), as it has no dependence on ROOT.

2.1.7 LHC event analysis framework

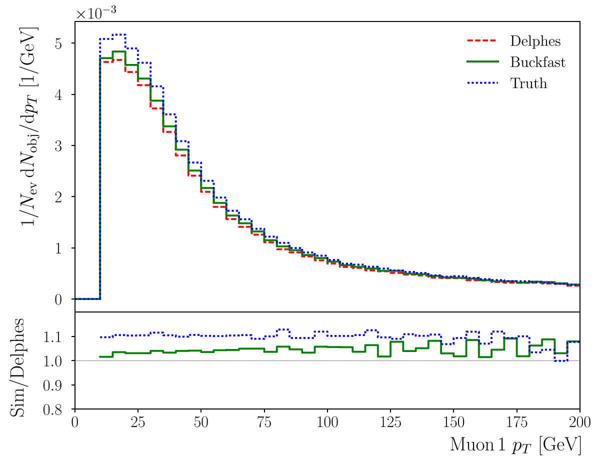
ColliderBit provides a simple analysis framework, built on the event record classes described in Sect. 2.1.5. Each analysis routine is a C++ class derived from the `BaseAnalysis` class, which provides the usual interface of a pre-run `init` method and an in-run `analyze` method to be called on each event. The user can choose which analyses to run in a given scan directly from the GAMBIT configuration file. Using the generic ColliderBit event record classes means that the analyses can be automatically run on either unsmeared truth records or ones to which detector effects (other than jet tagging rates) have been applied.



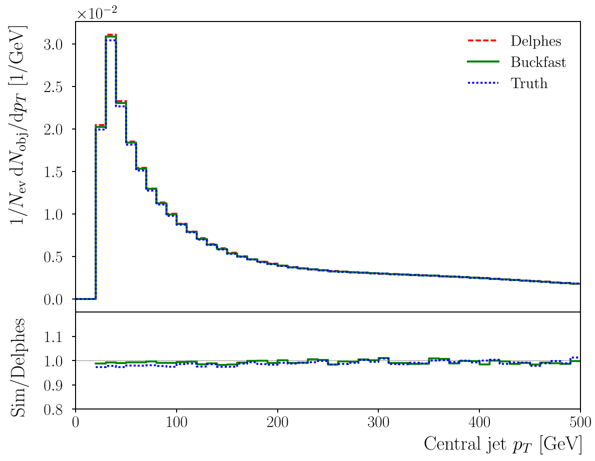
(a) Multiplicity of prompt electrons.



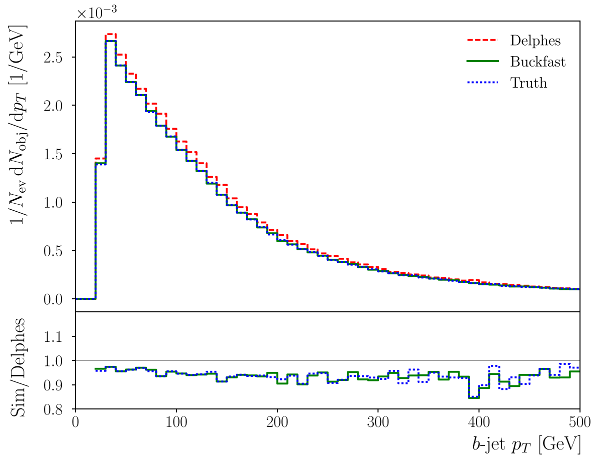
(b) $p_{\rm T}$ distribution of leading electrons.



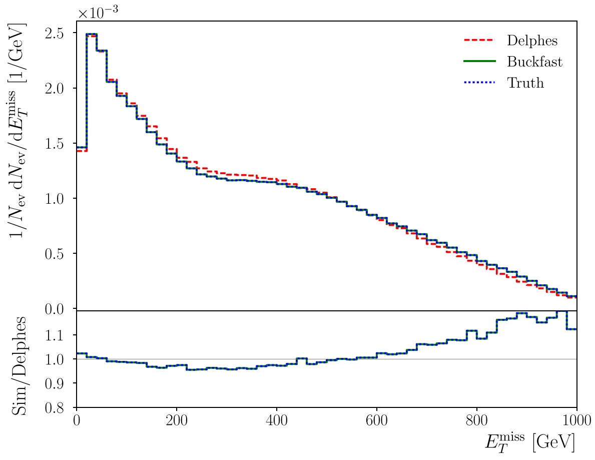
(c) $p_{\rm T}$ distribution of leading muons.



(d) $p_{\rm T}$ distribution of central ($|\eta| < 2.5$) jets.



(e) $p_{\rm T}$ distribution of b-jets.



(f) Distribution of missing $E_{\rm T}$.

Fig. 2 Comparisons of ATLAS event observables between the no-detector "truth" configuration, ColliderBit's BuckFast 4-vector smearing simulation, and the Delphes fast simulation code, for a CMSSM point near the current ATLAS search limit (see main text). The ratio plots are computed relative to Delphes, to best evaluate the performance of BuckFast

The result of an analysis is a set of `SignalRegionData` objects. Each of these encodes the predicted event counts in a particular signal region of the analysis, from both signal and background processes. The signal numbers are obtained by normalising the yields of simulated events to the integrated luminosity of the original experimental data analysis. The `BaseAnalysis` class provides additional methods for statistically combining analyses (either equivalent or orthogonal), and for specifying the effective luminosity simulated in the Monte Carlo step.

2.1.8 LHC statistics calculations

To determine the basic likelihood of observing n events in a certain signal region, given a signal prediction s, we use the marginalised form of Eq. (1) [92–94]. This allows us to incorporate systematic uncertainties on the signal prediction (σ_s)[6] and background estimate (σ_b) into the calculation, by marginalising over the probability distribution of a rescaling parameter ξ:

$$\mathcal{L}(n|s,b) = \int_0^\infty \frac{[\xi(s+b)]^n\, e^{-\xi(b+s)}}{n!} P(\xi)\mathrm{d}\xi\,. \tag{3}$$

Note that the use of a single rescaling parameter is an approximation to avoid the need for a time-consuming 2D integration. The probability distribution for ξ is peaked at $\xi = 1$, and has a width characterised by $\sigma_\xi^2 = \sigma_s^2 + \sigma_b^2$. The user can choose whether to assume a Gaussian form for this function,

$$P(\xi|\sigma_\xi) = \frac{1}{\sqrt{2\pi}\sigma_\xi} \exp\left[-\frac{1}{2}\left(\frac{1-\xi}{\sigma_\xi}\right)^2\right], \tag{4}$$

or a log-normal form,

$$P(\xi|\sigma_\xi) = \frac{1}{\sqrt{2\pi}\sigma_\xi}\frac{1}{\xi} \exp\left[-\frac{1}{2}\left(\frac{\ln\xi}{\sigma_\xi}\right)^2\right]. \tag{5}$$

The ColliderBit default is to use the log-normal version. This is slower but more correct, as it does not permit a finite probability for $\xi = 0$. In the limit of small σ_ξ, both likelihoods give extremely similar results. We use the highly optimised implementations of these functions contained in nulike [94,95].

The steps we have described so far allow ColliderBit to calculate the predicted number of events in any given signal region, defined by a specific set of observables and kinematic cuts, and to compute the likelihood for that region. However, certain ATLAS and CMS analyses make use of multiple signal regions, allowing analysis cuts to be optimised according to the specific characteristics of each model being tested. These signal regions may overlap, and so contain events in common. The likelihood functions from overlapping signal regions are therefore not independent. Ideally, information would be available from the experiments about the degree to which this overlap occurs, which would allow GAMBIT analyses to include all signal regions and their correlations in the final likelihood for a given analysis.

As this information is not presently available for most analyses, ColliderBit computes the likelihood for a given analysis on the basis of the signal region *expected* to give the strongest limit. It does this individually for each model, by calculating the expected number of events for every possible signal region considered in the the original ATLAS or CMS analysis. It then chooses the signal region with the maximally negative log-likelihood difference

$$\Delta \ln \mathcal{L}_{\text{pred}} = \ln \mathcal{L}(n=b|s,b) - \ln \mathcal{L}(n=b|s=0,b). \tag{6}$$

This difference is the log of the likelihood ratio between the signal plus background and background-only predictions, assuming that the observed counts match the background expectation. To calculate the likelihood for the analysis in question, ColliderBit then computes the likelihood of the actual data in the chosen signal region, and takes the difference with respect to the background-only expectation in that region, giving an effective log-likelihood

$$\ln \mathcal{L}_{\text{eff}} \equiv \Delta \ln \mathcal{L}_{\text{true}} = \ln \mathcal{L}(n|s,b) - \ln \mathcal{L}(n|s=0,b)\,. \tag{7}$$

It is necessary to define the effective log-likelihood in this way because of the selection step between different signal regions. Signal regions can in principle differ markedly in their number of analysis bins and expected numbers of events, leading to very different effective likelihood normalisations. Because of this, choosing the signal region on a per-model basis and then adopting the raw log-likelihood from the selected signal region would introduce erroneous model-to-model likelihood fluctuations. Taking the difference with respect to the background prediction not only removes the differing (but model-independent) offsets to the log-likelihood from the different signal regions' typical count rates, but also reduces the effective degrees of freedom of the resulting likelihood, from N (the number of analysis bins) to just one. This puts effective likelihoods from all signal regions on the same footing, and allows them to be compared correctly across different points in parameter space.

This is a conservative approach, but it is the best possible treatment when one lacks sufficient information to handle correlated data and systematic uncertainties. When such

[6] We choose to set this term to zero in all analyses in ColliderBit 1.0.0, owing to our already conservative use of LO cross-sections, the error from which dwarfs uncertainties arising from finite statistics in event generation; see Sect. 2.1.3 for more details. Future versions, employing alternative cross-section calculations, will make more extended use of σ_s.

Table 3 The published ATLAS cutflow for the $2jt$ signal region taken from Ref. [49], which searched for squarks and gluinos in events with jets and missing transverse momentum. The cutflow is generated for a squark pair-production simplified model (in which a pair of squarks is produced with direct decay to a quark and a lightest neutralino, all other sparticles being decoupled), with $m_{\tilde{q}} = 1000$ GeV and $m_{\tilde{\chi}_1^0} = 100$ GeV. This is compared with the GAMBIT cutflow obtained using Pythia 8 and BuckFast. Shown are the efficiencies for passing each cut (second and third columns), and the ratio of efficiencies (fourth column). For reference, we also show the cutflow results obtained using the CheckMATE package (taken from their public validation results) [29]

Cut	ATLAS (%)	GAMBIT (%)	Ratio	CheckMATE (%)
E_T^{miss} + jet p_T cuts	89.6	91.0	1.02	90.8
$\Delta\phi_{\text{min}} > 0.4$	81.0	82.5	1.02	82.1
$E_T^{\text{miss}}/\sqrt{H_T} > 15\,\text{GeV}^{-1/2}$	56.0	56.8	1.01	54.2
$m_{\text{eff}}^{\text{incl}} > 1600$ GeV	31.6	33.4	1.06	31.9

Table 4 The published ATLAS cutflow for SR A in Ref. [43], a search for new physics in events with two b jets and missing transverse momentum. The cutflow is generated for a bottom squark pair-production simplified model (in which a pair of bottom squarks is produced with direct decay to a b-quark and a lightest neutralino, all other sparticles being decoupled), with $m_{\tilde{b}} = 500$ GeV and $m_{\tilde{\chi}_1^0} = 1$ GeV. This is compared with the GAMBIT cutflow obtained using Pythia 8 and BuckFast. Shown are the percentages of the initial event sample after each cut, and the ratio of the GAMBIT and ATLAS numbers. We also provide the CheckMATE cutflow in the final column, taken from their public validation results

Cut	ATLAS (%)	GAMBIT (%)	Ratio	CheckMATE (%)
$E_T^{\text{miss}} > 80$ GeV	92.4	92.6	1.00	93.1
Lepton veto	86.6	92.6	1.07	90.7
$E_T^{\text{miss}} > 150$ GeV	76.1	79.1	1.04	76.8
Jet selection	6.85	8.79	1.28	6.46
$m_{bb} > 200$ GeV	5.52	7.40	1.34	4.60
$M_{CT} > 150$ GeV	4.72	5.93	1.26	4.01
$M_{CT} > 200$ GeV	3.86	4.76	1.24	3.32
$M_{CT} > 250$ GeV	2.93	3.46	1.18	2.50
$M_{CT} > 300$ GeV	2.01	2.34	1.16	1.69

information is made available (ideally in a standardised format), ColliderBit will use it for a more complete likelihood calculation. Refs. [96,97] are indeed very welcome recent steps in this direction.

To construct a compound likelihood from different *analyses*, we assume that all analyses have been chosen to be orthogonal, in the sense that they have disjoint selection criteria and no single event could contribute to the signal region counts of multiple analyses. This means that their effective log-likelihoods can be straightforwardly summed; ColliderBit does this for all analyses selected by the user, and returns the result to the GAMBIT Core as a final, combined LHC log-likelihood. It is the responsibility of the user to ensure that they only select mutually orthogonal analyses for combination in a ColliderBit run.

2.1.9 Validation of ColliderBit LHC constraints

We verified the ColliderBit LHC simulation and analysis chain by comparing cutflows for representative model parameter points against those published by the LHC experiments. Note that we use Pythia 8 and BuckFast for these comparisons, so we expect to see the agreement degrade in cases where effects not included in this chain become important, e.g. for compressed spectra, where a more appropriate treatment of initial state radiation is important.

Three sample cutflows are presented in Tables 3, 4 and 5, for a jets + MET search, 2b + MET search and a dilepton + MET search, respectively. These show close agreement for most signal regions, rising to no more than $\sim 50\%$ discrepancy in the worst case. These are a representative choice of sample cutflows for all signal regions considered. For reference, we also show the publically available CheckMATE cutflows where these are available. These confirm the expectation that BuckFast gives respectable performance, but does not match the ATLAS cutflows as closely as the CheckMATE package, which runs a heavily tuned version of the Delphes detector simulation. The compromise in performance in BuckFast is of course compensated for by the two-fold increase in speed, resulting from the quicker simulation step, plus the fact that it can be parallelised since it does not rely on the ROOT framework.

To illustrate the effect of changing the Pythia 8 settings on the physics performance of ColliderBit, we show the ATLAS

Table 5 The published ATLAS cutflow for Model 1 in Ref. [46], a search for new physics in events with two leptons and missing transverse momentum. This is compared with the GAMBIT cutflow obtained using Pythia 8 and BuckFast. Shown are the numbers of events expected in 20.1 fb^{-1} of 8 TeV ATLAS data, and the ratio of the GAMBIT and ATLAS numbers. Note that for the GAMBIT numbers, we used the same value of the SUSY production cross-section as that assumed in the ATLAS cutflow (and thus our cutflow does not include the effect of the LO cross-section that we use in our SUSY scans)

Cut	ATLAS	GAMBIT	Ratio
$e+e-$			
Two leptons	52.0	48.2	0.93
Jet veto	22.4	23.2	1.04
Z veto	21.2	21.6	1.02
SR MT2 90	12.7	12.6	0.99
SR MT2 120	9.4	9.5	1.01
SR MT2 150	6.2	6.3	1.02
$\mu^+\mu^-$			
Two leptons	47.8	51.2	1.07
Jet veto	20.7	25.5	1.23
Z veto	19.3	23.8	1.23
SR MT2 90	11.5	13.8	1.20
SR MT2 120	8.7	9.8	1.12
SR MT2 150	5.7	6.6	1.16
$e^\pm\mu^\mp$			
Two leptons	77.7	102.7	1.32
Jet veto	32.4	50.8	1.6
Z veto	32.4	42.1	1.49
SR MT2 90	19.1	27.2	1.42
SR MT2 120	14.7	20.1	1.37
SR MT2 150	10.1	13.6	1.34

0 lepton cutflow for four different Pythia 8 configurations in Table 6. This should be the most strongly affected cutflow, since the settings are all relevant for jet physics. In fact, we observe only a slight degradation of the cutflow performance as various approximations (e.g. removing hadronisation and FSR) are made, which validates the removal of certain Pythia 8 features in the interests of speed. We caution that for models with compressed particle decays where the effects of final state radiation may become more important, this conclusion is not expected to hold, but a thorough investigation is clearly physics-dependent and beyond the scope of this paper.

In Fig. 3, we compare the observed ATLAS Run 1 zero lepton CMSSM 95% CL exclusion limit in the m_0–$m_{1/2}$ plane from [49] with a GAMBIT ColliderBit scan performed with the same model. Here m_0 and $m_{1/2}$ are free parameters, $\tan\beta = 30$, $A_0 = -2m_0$ and $\mu > 0$. Since there are only two free parameters we perform a simple grid scan with 50×50 grid points. The ColliderBit likelihood includes only the LHC likelihood contribution, which in turn uses only the ATLAS zero lepton analysis, with 20,000 MC events generated per parameter point. The white solid line show the 95% CL exclusion contour, defined by the likelihood ratio $\mathcal{L}/\mathcal{L}_{max} = 0.05$. For comparison, the observed limit from the ATLAS analysis is plotted as a solid blue line, with dashed blue lines showing the reported $\pm 1\sigma$ theoretical uncertainty on this limit.

The ColliderBit exclusion limit is more conservative than the ATLAS result, as expected from the different cross-sections used (LO for ColliderBit, NLO + NLL for the ATLAS result). We have checked how our limit would change with NLO + NLL cross-sections from NLL-fast 2.1 for a number of points close to the observed ATLAS limit. In the region where $m_0 \gg m_{1/2}$ we find close agreement with the ATLAS limit; the rescaled ColliderBit limit ends within the uncertainty band of the ATLAS limit. In the low-m_0 part of the plane, where $m_0 \sim m_{1/2}$, we see a somewhat larger discrepancy with the observed ATLAS limit also after rescaling our results with NLO + NLL cross-sections. The reason for this discrepancy is that ColliderBit here differ from the ATLAS analysis in what signal region is predicted to have the best expected sensitivity. ATLAS uses the 4-jet region 4jt while ColliderBit chooses the 3-jet region 3j. Coinciden-

Table 6 Reproduction of the same ATLAS 0 lepton cutflow as Table 3, with each column representing different Pythia 8 settings. The baseline in the final "None" column has tau spin correlations turned off (since they have no effect for SUSY models in any case), ISR turned on, and hadronisation (HAD), FSR and multiple parton interactions (MPI) turned off. Pythia 8 is configured to produce light squark pairs only, and the parton-level events are reconstructed with the parton BuckFast settings. The first three columns add hadronization, FSR and multiple parton interactions. It is worth noting that none of these configurations match the cutflow configuration in Table 3, which includes a tuning of the minimum p_T threshold for the Pythia 8 `TimeShower`

Cut	MPI+FSR+HAD (%)	FSR+HAD (%)	HAD (%)	None (%)
E_T^{miss} + jet p_T cuts	91.0	90.7	91.4	91.0
$\Delta\phi_{min} > 0.4$	82.4	82.4	82.7	81.7
$E_T^{miss}/\sqrt{H_T} > 15\,\mathrm{GeV}^{-1/2}$	56.8	57.1	57.7	57.0
$m_{eff}^{incl} > 1600\,\mathrm{GeV}$	33.0	32.7	33.7	34.2

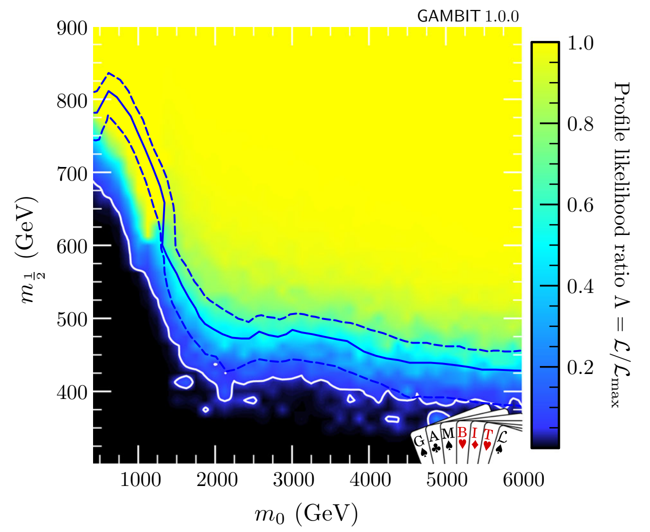


Fig. 3 Output from a ColliderBit CMSSM grid scan over m_0 and $m_{1/2}$ with $\tan\beta = 30$, $A_0 = -2m_0$ and $\mu > 0$, using 50 grid points in each direction. The likelihood only includes the ATLAS zero lepton SUSY search, with 20,000 MC events generated per point. The colour map shows the profile likelihood ratio $\mathcal{L}/\mathcal{L}_{max}$ and the solid white line indicates the GAMBIT 95% CL exclusion contour, defined by $\mathcal{L}/\mathcal{L}_{max} = 0.05$. The blue solid line shows the ATLAS 95% CL observed exclusion limit, taken from Ref. [49], with the blue dashed lines showing the reported $\pm 1\sigma$ theoretical (cross section) uncertainty

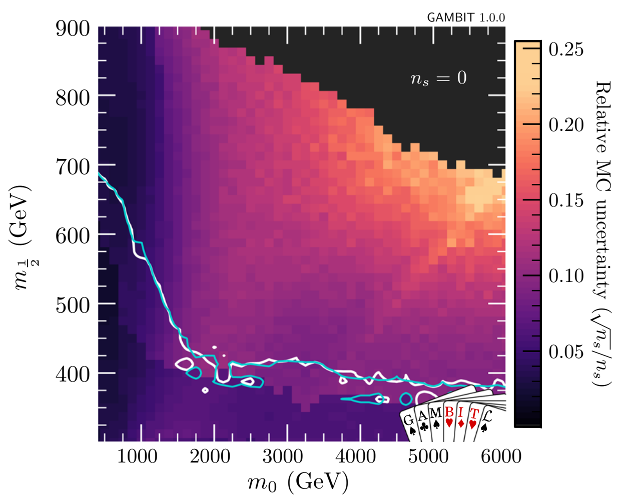


Fig. 4 The relative MC uncertainty $\sqrt{n_s}/n_s$ for ColliderBit simulations of the ATLAS zero lepton SUSY search across the plane of m_0 and $m_{1/2}$, using 20,000 MC events per parameter point. Here n_s is the number of accepted MC signal events. As in Fig. 3, the remaining CMSSM parameters are given by $\tan\beta = 30$, $A_0 = -2m_0$ and $\mu > 0$. The solid lines show the GAMBIT 95% CL exclusion contours obtained using 20,000 (white) and 100,000 (cyan) MC events

tally, the 4jt region observes a small downwards fluctuation relative to the background expectation, leading ATLAS to set stronger limits than expected, while at the same time there is a small upwards fluctuation in the event count for the region 3j, giving a weaker ColliderBit limit than expected. In this part of parameter space the squarks and gluinos are rather close in mass, implying that some of the jets will be soft. The choice between a 3-jet and a 4-jet signal region is therefore likely to be sensitive to the details of jet handling in the different event generators used (HERWIG++ 2.5.2 for the ATLAS result and Pythia 8.212 for ColliderBit). We note that if we by hand force ColliderBit to use the 4jt region, our limit again agrees nicely with the ATLAS limit after NLO + NLL scaling. For instance, our rescaled limit at $m_0 = 800$ GeV moves up to $m_{1/2} = 780$ GeV.

In the above 50×50 grid scan of the m_0–$m_{1/2}$ plane we only generated 20,000 MC events per parameter point. This scan completed in less than 80 minutes using 48 CPUs. Clearly, for a low-dimensional scan like this one can afford a much higher number of events per point to reduce the MC uncertainty. But for large global fits in many-dimensional parameter spaces 20,000 events may be a realistic trade-off between speed and accuracy. In Fig. 4 we show a colour map of the relative MC uncertainty, $\sqrt{n_s}/n_s$, across the m_0–$m_{1/2}$ plane for the ColliderBit simulation of the ATLAS zero lepton search using 20,000 events per point. Here n_s is the number of accepted MC events for the signal region chosen by ColliderBit for the given parameter point. As in Fig. 3, the white line depicts the 95% CL exclusion contour obtained with 20,000 generated events. For comparison the cyan line shows the limit obtained when generating 100,000 events per point. We see that for the signal regions used to calculate the likelihood for this analysis, the relative MC uncertainty stays below 13% in the parameter regions around the exclusion limit. At large m_0 and $m_{1/2}$ the uncertainty increases as we approach the production threshold. (The apparent cut-off at $\sqrt{n_s}/n_s \sim 0.25$ is due to the grid step size).

In contrast to the simple grid scan presented in Figs. 3 and 4, a large global fit will typically involve sampling millions of parameter points. Limited MC event statistics then increases the chance of a few points ending up with a spurious good likelihood due to MC fluctuations. This may in particular affect the result of a frequentist analysis where the preferred parameter regions are determined relative to the best-fit point, as with the likelihood ratio $\mathcal{L}/\mathcal{L}_{max}$ used above. A spurious good likelihood $\mathcal{L}_{max}$ for the best-fit point will result in a falsly strong constraint on the preferred parameter regions. One simple way to ensure conservative and more stable limits in a large scan is to cap the ColliderBit effective log-likehood in Eq. (7) at the value given by the background-only expectation ($s = 0$), i.e. to force $\ln \mathcal{L}_{\text{eff}} \leq 0$. This then becomes an "exclusion only" likelihood, as all points where $s > 0$ gives an improved fit to the data are assigned the likelihood corresponding to $s = 0$. Of course, this method is not appropriate if the aim of the parameter scan is to fit the model to a potential new signal in the data.

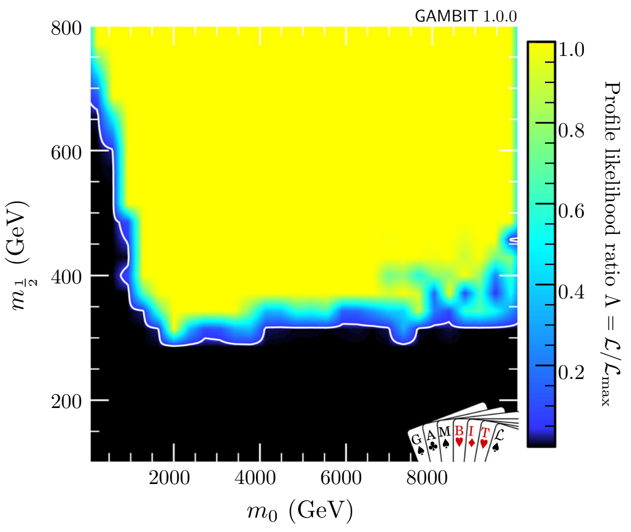


Fig. 5 Output from a ColliderBit CMSSM scan over m_0, $m_{1/2}$, $\tan\beta$, and A_0, with SM parameters set to default values, using Diver with GAMBIT production settings (`convthresh` $= 10^{-5}$ and `NP` $= 19200$). The likelihood includes all of the ATLAS and CMS Run I analyses summarised in the text, but no other contributions. The colour map shows the profile likelihood ratio $\mathcal{L}/\mathcal{L}_{max}$ and the solid white line indicate the GAMBIT 95% CL exclusion contours at $\mathcal{L}/\mathcal{L}_{max} = 0.05$

The approach of capping the effective likelihood was used for the result shown in Fig. 3. We also apply it in the analysis shown in Fig. 5. This gives an example of a typical use-case for ColliderBit, in which multiple LHC searches are included in the combined LHC likelihood and a larger parameter space is scanned. Here we show the CMSSM 95% CL exclusion limit in the m_0–$m_{1/2}$ plane, following a scan of m_0, $m_{1/2}$, $\tan\beta$, and A_0 using Diver [39] with GAMBIT production settings (`convthresh` $= 10^{-5}$ and `NP` $= 19200$), with the SM parameters held to their default values. All of the LHC Run I analyses listed above are included in the LHC combined likelihood, and no other likelihoods are used. One obtains an exclusion contour of similar shape to the ATLAS zero lepton limit for fixed A_0 and $\tan\beta$, but it is shifted to lower values of $m_{1/2}$.

2.2 LEP likelihood calculation

Despite the huge improvement in lower limits provided by high-energy LHC data, limits from direct searches at the LEP experiments are still important for some BSM models. This is true in particular for SUSY models that only have significant production of slepton or neutralino/chargino pairs with masses below half the maximum LEP centre-of-mass energy. In most available codes, LEP limits from direct searches take the form of hard lower limits on sparticle masses, at e.g. 95% CL. This is how such limits are implemented in DarkSUSY [98] and micrOMEGAs [99], for example. Such limits generally rely on model-dependent assumptions, which are not always clearly stated.

As an example, in DarkSUSY 5.1.3 the selectron mass is bounded by $m_{\tilde{e}_R} > 95$ GeV if $m_{\tilde{e}_R} - m_{\tilde{\chi}_1^0} > 15$ GeV, based on a search by the ALEPH experiment [8], and $m_{\tilde{e}_R} > 87.1$ GeV if $m_{\tilde{e}_R} - m_{\tilde{\chi}_1^0} > 5$ GeV, based on results from the OPAL experiment [100]. However, if one looks closely at the details of these limits there are indeed remaining model assumptions, e.g. the ALEPH experiment assumes $\mu = -200$ GeV and $\tan\beta = 2$ for the production cross-section, and a branching ratio $\mathrm{BR}(\tilde{e}_R \to e\tilde{\chi}_1^0) = 1$. In contrast, in an analysis of the same data using a more general MSSM parameter space (but still assuming gaugino mass unification, scalar mass unification, no slepton mixing, and negligible squark mixing), the selectron mass limit becomes 73 GeV [9]. The weakening of this limit is due to possibile cascade decays of the selectron.

In ColliderBit we take a different approach, which is free from model-dependent assumptions, using the direct cross-section limits for sparticle pair production of sleptons, neutralinos and charginos at LEP. Our approach includes not only model-dependent effects in the cross section, but also in the decay rates, where we make no assumptions on the branching ratios, relying instead on an explicit calculation.

Continuing the example of the selectron case, we now discuss how we model the cross-section limit for slepton pair production and decay into the lightest neutralino from the L3 experiment. This has been given as a function of the selectron and neutralino mass in Fig. 2a of Ref. [6], which we reproduce here in Fig. 6 for demonstration. Corresponding results for smuons and staus are used in the same manner. These results cover slepton masses from 45 GeV up to the kinematic limit of 104 GeV, with neutralino masses from zero up to the slepton mass. For a particular model point the theoretical slepton pair production cross-section is calculated in a separate routine. This uses leading order results on the cross-section taken from Refs. [101,102], which includes t-channel contributions from neutralinos. We treat contributions to a possible signal cross-section from $\tilde{e}_L^*\tilde{e}_L$ and $\tilde{e}_R^*\tilde{e}_R$ pair production separately, taking into account the relevant branching ratios for the decay to the lightest neutralino using DecayBit [40], which can be interfaced to, e.g., SUSY-HIT [103]. Hereafter, we refer to this cross-section times branching ratio as $\sigma \times \mathrm{BR}$.

We estimate the dominant theoretical uncertainty on $\sigma \times \mathrm{BR}$ using the mass uncertainties of the sleptons, as reported by SpecBit [40]. For slepton mass values of $m_1 \pm \delta_1$ and $m_2 \pm \delta_2$, we calculate the central value of $\sigma \times \mathrm{BR}$ for m_1 and m_2. Then, we recalculate $\sigma \times \mathrm{BR}$ with the upper and lower mass values and use the maximum and minimum of these as estimates for the overall $\sigma \times \mathrm{BR}$ uncertainty.

Once the $\sigma \times \mathrm{BR}$ has been calculated this way, we can look up the appropriate limit with which to compare from Fig. 6.

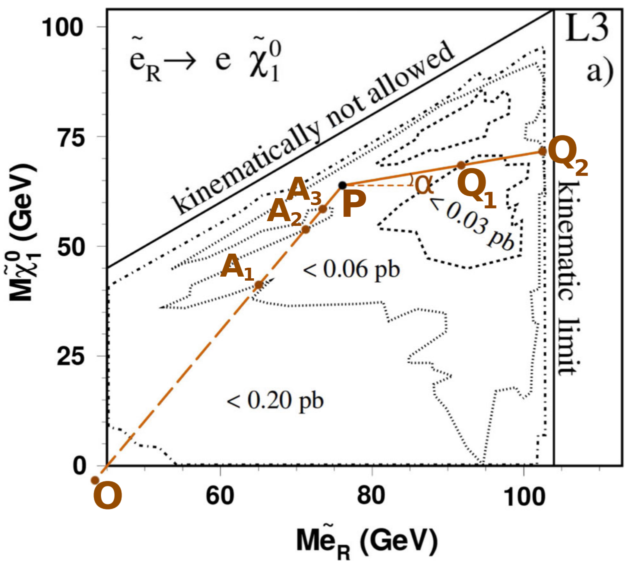


Fig. 6 Example of the limit interpolation process, based upon Fig. 2a of Ref. [6]. The line segment OP is used to find the intersection points A_1, A_2, and A_3, which determine that P is within the 0.06 pb limit. Then, for each angle $\alpha \in [0, 2\pi]$, the line segments PQ_1 and PQ_2 contribute to the weighted average limit at the point P. More details of this procedure are described in the main body of text

We do this by digitising each cross-section limit contour, and using inverse distance-weighted interpolation [104] to estimate the cross-section limits in regions between contours. The weighted averaging prevents the noisiness of the LEP limit curves from strongly influencing the interpolant (an advantage over e.g. spline or bilinear interpolation), whilst at the same time forcing the interpolant to exactly reproduce the published cross-section contours (an advantage over e.g. data smoothing algorithms).

Our algorithm works as follows. Given a point on the $m_{\tilde{e}}$–$m_{\tilde{\chi}_1^0}$ plane, such as point P in Fig. 6, we first determine which contours contain this point. This can be achieved by drawing a line segment from this point to any point O outside of the plot. Then, for each contour, if this line segment OP intersects the contour an odd number of times, say at A_1, A_2, and A_3, then the point lies within the contour. Using this method, we find the two limit contours, 0.06 pb and 0.03 pb, between which the point P lies. Next, for a large number of angles $\alpha \in [0, 2\pi]$, we draw a line segment PQ_2 from P to where it intersects the outer limit of 0.06 pb. We also note if this line segment intersects the 0.03 pb contour, such as at Q_1. If the point lies directly on top of one of the contours, we simply take that contour as the correct limit. If not, we calculate the limit as a weighted average of all bounding cross-section limits over all angles. We weight each cross-section limit sample by $\overline{\mathrm{PQ}}^{-p}$, where $\overline{\mathrm{PQ}}$ is the length of the line segment PQ, and p is the so-called 'power' parameter of the inverse distance-weighted interpolation algorithm. We choose $p = 0.5$, to avoid artificially endowing the interpolating functions with local minima and maxima around the sample points, a known shortcoming of the algorithm for choices of p greater than or equal to 1. The results of this interpolation proceedure can be seen in Fig. 7, which shows the interpolated 95% confidence limits for all of the results from the L3 experiment that we use here. In particular the top left plot can be compared directly to Fig. 6.

Comparing the values of $\sigma \times$ BR to the 95% confidence interpolated limit drawn from the digitised limit plot, we can now calculate the likelihood using the error function and the estimated theoretical uncertainty on $\sigma \times$ BR.

To increase the constraining power of the direct LEP searches, we also use the corresponding cross-section limits set by the ALEPH experiment, calculating a second likelihood in the same manner. For this, we consider the searches for scalar leptons in the same mass range, using the model-independent results of Fig. 3 in Ref. [8]. We treat the data of the two experiments as independent.[7]

In Fig. 8 we show the ColliderBit exclusion limits from the combination of ALEPH and L3 searches for slepton pair production, in the CMSSM $(m_0, m_{1/2})$-mass plane for two different values of $\tan\beta$. The results in Fig. 8 can be compared to the corresponding CMSSM exclusion limits from ALEPH alone given in [9] (dashed lines). We have checked that the observed differences are mainly due to the higher constraining power of the two experiments combined, with some remaining unavoidable differences caused by the RGE codes used for the spectrum generation.

We take similar limits for the neutralino and chargino pair production cross-sections, with decays into the lightest neutralino, from searches by the OPAL and L3 experiments. The corresponding theoretical leading-order cross-sections are from Refs. [105] and [106], again taking into account the relevant branching ratio for each model point. For neutralino pairs, the limits are set on $\tilde{\chi}_2^0\tilde{\chi}_1^0$ production with subsequent decay of the $\tilde{\chi}_2^0$. We take OPAL results from Fig. 9 in Ref. [12], which applies to hadronic decays, giving bounds for $m_{\tilde{\chi}_2^0}$ from 100 GeV to the kinematic limit of 204 GeV, while $m_{\tilde{\chi}_1^0}$ ranges from zero to $m_{\tilde{\chi}_2^0}$. The region $m_{\tilde{\chi}_1^0} + m_{\tilde{\chi}_2^0} <$ 100 GeV is not bounded. From L3 we have limits on leptonic decays $\tilde{\chi}_2^0 \to ll\tilde{\chi}_1^0$ from Fig. 3b of Ref. [4] for $m_{\tilde{\chi}_2^0}$ from 91 GeV to the kinematic limit of 189 GeV. Again, no limit applies in the low mass region $m_{\tilde{\chi}_1^0} + m_{\tilde{\chi}_2^0} < 91$ GeV. Our interpolation from the L3 results is shown in Fig. 7 (bottom right).[8]

[7] This should be a good approximation; common uncertainties across the experiments, such as luminosity, are subdominant to the systematic uncertainty from Monte Carlo statistics in the experimental result.

[8] In this region limits from the decay of the Z-boson would apply unless the neutralinos are purely bino/wino combinations. In that case there must also be a light chargino (wino), for which the limits below apply.

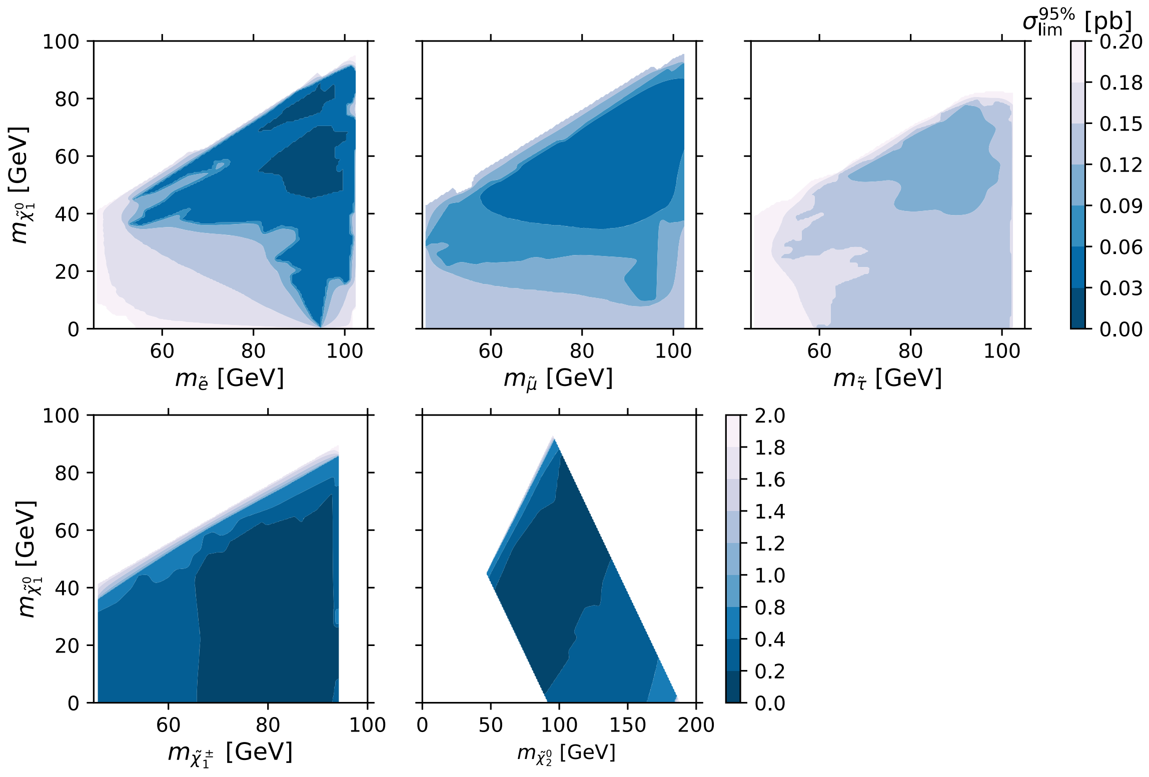


Fig. 7 Interpolated 95% CL on the cross section for pair production of selectrons (upper left), smuons (upper centre), staus (upper right), charginos (lower left) and the next-to-lightest neutralino (lower right), as a function of the produced sparticle mass and the mass of the lightest neutralino. This interpolation is based on results by the L3 experiment at LEP [4,6]

For chargino pair production, the OPAL experiment [12] sets limits on hadronic, semi-leptonic and leptonic decays separately in Figs. 5, 6 and 7 of that article. The limits are set from a chargino mass of 75 GeV up to the kinematical limit of 104 GeV, and for neutralino masses from zero up to the chargino mass. For each channel we take into account the branching ratios $\mathrm{BR}(\tilde{\chi}^\pm_1 \to q\bar{q}'\tilde{\chi}^0_1)$ and $\mathrm{BR}(\tilde{\chi}^\pm_1 \to l\nu\tilde{\chi}^0_1)$ of the model point. We use an older, compatible, limit from the L3 experiment on fully leptonic decays taken from Fig. 2b of Ref. [4]. This extends from 45 GeV chargino masses up to a kinematic limit of 94.5 GeV. Unfortunately, the L3 experiment does not give separate model-independent cross-section limits for the other two channels. Our interpolation from the L3 results is again shown in Fig. 7 (bottom left).

We also include results on chargino and neutralino pair production from both OPAL (Fig. 8 in Ref. [12]) and L3 (Figs. 2a and 3a in Ref. [4]), where the limits assume that the fermions in $\tilde{\chi}^\pm_1 \to ff'\tilde{\chi}^0_1$ and $\tilde{\chi}^0_2 \to f\bar{f}\tilde{\chi}^0_1$ are represented as per the normal W and Z branching ratios into two fermions. This must be used with some care, as light sfermions may affect the assumption.

We note that while the experimental limits have been set on $\tilde{\chi}^0_2\tilde{\chi}^0_1$ and $\tilde{\chi}^+_1\tilde{\chi}^-_1$ production, the ColliderBit likelihood can be calculated for production of any $\tilde{\chi}^0_i\tilde{\chi}^0_1$ and $\tilde{\chi}^+_i\tilde{\chi}^-_i$, as long as we consider the same experimental signature in the decay. Again, we show the resulting exclusion limits in the CMSSM in Fig. 8. Here we observe very good agreement with earlier ALEPH results on chargino pair production [9], and we have checked that the difference is dominated by differences in the RGE codes used. GAMBIT relies on FlexibleSUSY [57], while the ALEPH analysis was carried out with ISASUSY 7.51 [107].

2.3 Higgs likelihood calculation

ColliderBit includes likelihoods relating to constraints on extended Higgs sectors from collider experiments, and to measurements of the SM-like Higgs mass and production cross-sections at the LHC. These likelihoods are provided through an interface to HiggsBounds [33,34] and HiggsSignals [35].

Although constructing a likelihood from null search results at colliders generally requires event simulation, the information provided by the combined LEP Higgs search results [108] allows for the construction of an approximate likelihood for neutral Higgs bosons. HiggsBounds interpolates the full CL_{s+b} distribution from the combined model-independent LEP searches, for all Higgs mass combinations,

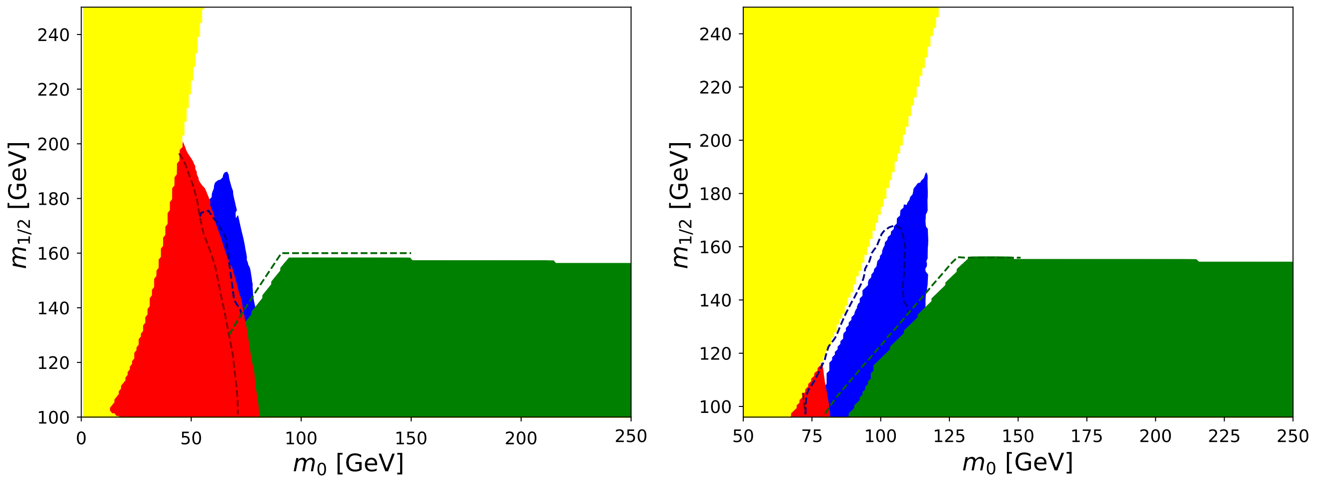


Fig. 8 Limits from direct sparticle pair production searches at LEP shown in the CMSSM $(m_0, m_{1/2})$-mass plane with fixed $\tan\beta = 15$ (left) and $\tan\beta = 30$ (right), $A_0 = 0$ GeV and $\mu > 0$. The 95% CL excluded areas from chargino searches (green), stau searches (blue), and selectron searches (red) are shown separately and overlaid in the sequence listed here. Theoretically forbidden regions are shown in yellow. Included for comparison are the corresponding results from the ALEPH experiment alone taken from Fig. 6 of [9], indicated by the dashed lines

over varying production cross sections. Using a Gaussian approximation valid in the asymptotic limit, it employs the CL_{s+b} distribution to calculate an approximate likelihood.

With the direct observation of an SM-like Higgs boson [16, 17], measurements of the new particle's mass, production cross section, and branching ratios can be used to constrain the neutral Higgs sector of BSM models. In channels where measurements of the neutral boson's mass are available, HiggsSignals calculates contributions to the mass likelihood as a χ^2, taking into account both experimental and theoretical uncertainties. For each channel, it minimises the χ^2 independently over the possibility of each neutral state in the Higgs sector being responsible for the signal, including the simultaneous appearance of multiple resonances if they are nearly degenerate in mass. For signal strengths, it uses measurements over all available channels to construct a single χ^2, using the associated N_{meas}-dimensional covariance matrix to account for reported experimental uncertainties, including correlations due to common channels between experiments and the uncertainty in the integrated luminosity. It then combines this signal-strength likelihood with the mass likelihood to form a combined LHC neutral Higgs sector likelihood.

For both the LEP and LHC likelihoods implemented in ColliderBit, the theoretical masses (with uncertainties, when available), couplings and branching ratios come from other GAMBIT modules, namely DecayBit and SpecBit [40]. In particular, we use the Higgs couplings provided via `HiggsCouplingsTable` objects from SpecBit to estimate the neutral Higgs boson production cross sections. We calculate the ratios of the production cross-sections for each Higgs in a given BSM theory to an SM Higgs of the same mass, assuming them to be given by the ratio of squared couplings for the relevant processes.

3 User interface

The GAMBIT code consists of a series of separate code modules that calculate likelihoods for new physics models using data from flavour physics [41], astrophysics [42], electroweak precision physics [40] and collider physics (the present paper). These modules can be used as standalone tools (using a custom C++ driving code), or they can be used via the GAMBIT core framework that resolves dependencies between calculations, and steers scans with the aid of a dedicated scanning and statistics module [39]. The advantage of using the latter is that it is by far the easiest way to define models, calculate spectra and perform decay width calculations.

There are thus two ways to take advantage of the high-energy collider likelihoods provided by ColliderBit: either via the GAMBIT framework or by interfacing to ColliderBit as a standalone tool. Here we describe each in turn.

3.1 GAMBIT interface

The GAMBIT framework [1] defines two sorts of function that can be used by each **module** within the framework:

– **Module functions**: C++ functions within the GAMBIT code itself.
– **Backend functions**: functions that live within an external code, such as Pythia 8.

In GAMBIT, each module function is given a tag, called a **capability**, that describes what it can calculate, be it an observable, e.g. the number of events expected at the LHC, or a likelihood, e.g. the combined likelihood of a set of LHC searches. Module functions may also have **dependencies** on other module functions – which may live either in the same module or in another GAMBIT module – or **backend requirements** that are satisfied by **backend functions** or **backend variables**. A concrete example from ColliderBit is the capability of the combined LHC likelihood calculation (Table 7), which has dependencies on the numbers of events expected in CMS and ATLAS searches, and has a backend requirement relating to the functional form of the likelihood required.

ColliderBit interfaces with the GAMBIT Core to communicate its capabilities, dependencies, and backend requirements. The Core then runs its dependency resolution routine to connect and execute the module functions in the order that fulfils all the dependencies. As most of this machinery is described in the main GAMBIT paper [1], in this section we shall simply describe each of the ColliderBit capabilities and as their uses.

3.1.1 LHC simulation capabilities

These capabilities are grouped within ColliderBit into three categories, which correspond to the three main steps of simulation: collider, detector, and analysis. There are also two additional capabilities needed to complete a collider simulation. One is a capability meant simply to control the parallelisation and execution of the event generation loop (`ColliderOperator`), and the other calculates the likelihood as a final result (`LHC_Combined_LogLike`). These two capabilities are shown in Table 7. Tables 8, 9, and 10 show the collider, detector, and analysis capabilities, respectively.

Since there can be a variety of configurations for collider simulation and a variety of experimental analyses, we designed these components so new configurations and analyses could be easily added.

For instance, a user who wishes to add a new Pythia 8 configuration must only complete the following steps:

1. *Create a* `SpecializablePythia` *initialisation function.* These functions are defined in the file `colliders/SpecializablePythia.cpp`.[9] Each such function must have its own namespace and a call signature of `void init (SpecializablePythia* specializeMe)`. Within the init function, settings can be sent to Pythia 8 as strings using the `SpecializablePythia::addToSettings` function. For example, the following would be a valid init function:

[9] Within this section, all header files (`*.hpp`) mentioned are found in the `ColliderBit/include/gambit/ColliderBit` directory, while source files (`*.cpp`) are found in the `ColliderBit/src` directory.

Table 7 The capabilities provided by ColliderBit that control the simulation event loop and calculate the likelihood. The `operateLHCLoop` function requires classes from Pythia 8, which is connected to GAMBIT via BOSS [1]. The options are read at runtime from the GAMBIT YAML file (or configured in the ColliderBit standalone code). For readability, here and in the following tables we suppress the namespace `std` for standard C++ types such as `std::vector` and `std::string`. The `pythiaNames` option tells the `operateLHCLoop` function the names of the Pythia 8 configurations for which it should run simulation loops (one loop per configuration). The `nEvents` option tells it how many events to generate per loop, while the `silenceLoop` option (default `true`) is used to suppress output to `stdout` during the simulation loops

Capability	Function (return type): brief description	Dependencies	Backend requirements	Options (type)
`ColliderOperator`	`operateLHCLoop` (`void`): controls the parallelisation and execution of the entire event loop of the collider simulation		Pythia 8	`pythiaNames` (`vector<string>`) `nEvents` (`vector<int>`) `silenceLoop` (`bool`)
`LHC_Combined_LogLike`	`calc_LHC_LogLike` (`double`): Combines the results from different analyses together into a single delta-log-likelihood value	`DetAnalysisNumbers` `ATLASAnalysisNumbers` `CMSAnalysisNumbers` `IdentityAnalysisNumbers` (Table 10)	nulike	

Table 8 The collider capabilities provided by ColliderBit. In addition to the dependencies shown above, all of these functions also depend on the ColliderOperator capability in Table 7, because they all execute within the event loop. These functions need classes from Pythia 8, which is connected to GAMBIT via BOSS [1]. The decay_rates and MSSM_spectrum dependencies can be fulfilled by DecayBit and SpecBit [40], respectively. The options are read at runtime from the GAMBIT YAML file (or configured in the ColliderBit standalone code). The Pythia_doc_path option points to the xmldoc directory of Pythia. The *Pythia_config* option is a list of Pythia settings. One such *Pythia_config* list is required per Pythia configuration name, as given in pythiaNames (Table 7). The SLHA_filenames option is a list of the SLHA files that the user wants to run using getPythiaFileReader. Finally, the xsec_vetos option specifies limits on the maximum total cross-section (in fb), as estimated by Pythia at the beginning of a run, below which the simulation should be skipped. One cross-section limit can be set per Pythia configuration (default 0)

Capability	Function (return type): brief description	Dependencies	Backend req.	Options (type)
HardScatte ringSim	getPythia (ColliderBit::Specializable Pythia): provides a Pythia 8 instance within a container that is ready to simulate collision events for a model chosen by ScannerBit	decay_rates A relevant Spectrum object	Pythia 8	Pythia_doc_path (string) *Pythia_config* (vector<string>) xsec_vetos (vector<double>)
	getPythiaFileReader (ColliderBit::SpecializablePythia): provides a Pythia 8 instance within a container that is ready to simulate collision events based upon some SLHA files		Pythia 8	Pythia_doc_path (string) *Pythia_config* (vector<string>) SLHA_filenames (vector<string>) xsec_vetos (vector<double>)
HardScatte ringEvent	generatePythia8Event (Pythia8::Event): uses the given HardScatteringSim to generate the next event of the collider simulation chain	HardScatteringSim	Pythia 8	

Table 9 The detector capabilities provided by ColliderBit. In addition to the dependencies shown above, all of these functions also depend on the `ColliderOperator` capability in Table 7, since they all execute within the event loop. Some of these functions need classes from Pythia 8, which is connected to GAMBIT via BOSS [1]. The options are read at runtime from the GAMBIT YAML file (or configured in the ColliderBit standalone code). The `delphesConfigFiles` option specifies the TCL files used by Delphes for its configuration. The `antiktR` options (default `0.4`) control the R value used by FastJet's anti-k_T jet algorithm. The `partonOnly` options (default `false`) tell the smearing sims to consider only the partonic states of the event. Finally, the `useDetector` option switches a given detector simulation on or off, along with all analyses relying on that detector (default `true` for `getBuckFastATLAS` and `getBuckFastCMS` and `false` for `getDelphes` and `getBuckFastIdentity`). All the options in this table are vectors that require one entry per Pythia configuration in `pythiaNames` (Table 7)

Capability	Function (return type): brief description	Dependencies	Backend req.	Options (type)
`DetectorSim`	`getDelphes` (`ColliderBit::Delphes Vanilla`): provides a Delphes instance within a container that is ready to perform detector simulation		Pythia 8	`delphesConfigFiles` (`vector<string>`) `useDetector` (`vector<bool>`)
`SimpleSmearingSim`	`getBuckFastATLAS` (`ColliderBit::BuckFast SmearATLAS`): provides a set of BuckFast functions within a container that is ready to apply ATLAS smearing and reconstruction efficiencies to an event			`antiktR` (`vector<double>`) `partonOnly` (`vector<bool>`) `useDetector` (`vector<bool>`)
	`getBuckFastCMS` (`ColliderBit::BuckFast SmearCMS`): provides a set of BuckFast functions within a container that is ready to apply CMS smearing and reconstruction efficiencies to an event			`antiktR` (`vector<double>`) `partonOnly` (`vector<bool>`) `useDetector` (`vector<bool>`)
	`getBuckFastIdentity` (`ColliderBit::BuckFast Identity`): provides a function that does absolutely nothing to a given event within a container similar to those returned by other `SimpleSmearingSim` capabilities			`antiktR` (`vector<double>`) `partonOnly` (`vector<bool>`) `useDetector` (`vector<bool>`)
`ReconstructedEvent`	`reconstructDelphes Event` (`HEPUtils::Event`): Uses the given `DetectorSim` to perform detector simulation upon the given `HardScattering Event`	`HardScatteringEvent` (Table 8) `DetectorSim`	Pythia 8	
`ATLASSmearedEvent`	`smearEventATLAS` (`HEPUtils::Event`): Uses the given `SimpleSmearingSim` to apply smearing and reconstruction efficiencies upon the given `HardScattering Event`	`HardScattering Event` (Table 8) `SimpleSmearingSim` of type `BuckFastSmear- ATLAS`		
`CMSSmearedEvent`	`smearEventCMS` (`HEPUtils::Event`): Uses the given `SimpleSmearingSim` to apply smearing and reconstruction efficiencies upon the given `HardScatteringEvent`	`HardScattering Event` (Table 8) `SimpleSmearingSim` of type `BuckFastSmearCMS`		
`CopiedEvent`	`copyEvent` (`HEPUtils::Event`): Uses the given `SimpleSmearingSim` to do absolutely nothing to the given `HardScatteringEvent`	`HardScatteringEvent` (Table 8) `SimpleSmearingSim` of type `BuckFastIdentity`		

Table 10 The analysis capabilities provided by ColliderBit. In addition to the dependencies shown above, all of these functions depend on the ColliderOperator capability in Table 7, since they all execute within the event loop. The options are read at runtime from the GAMBIT YAML file (or configured in the ColliderBit standalone code). The analyses options tell the getDetAnalysisContainer, getATLASAnalysisContainer, getCMSAnalysisContainer and getIdentityAnalysisContainer functions the names of all the analyses the user wishes to include for the collider simulations. One vector of analysis names (possibly empty) is required per Pythia configuration in pythiaNames (Table 7)

Capability	Function (return type): brief description	Dependencies	Options (type)
DetAnalysis Container	getDetAnalysisContainer (ColliderBit::HEPUtilsAnalysisContainer): provides a list of analyses within a container that is ready to apply them to an event	HardScatteringSim (Table 8)	analyses (vector<vector<string>>)
ATLASAnalysis Container	getATLASAnalysisContainer (ColliderBit::HEPUtilsAnalysisContainer): provides a list of ATLAS analyses within a container that is ready to apply them to an event	HardScatteringSim (Table 8)	analyses (vector<vector<string>>)
CMSAnalysis Container	getCMSAnalysisContainer (ColliderBit::HEPUtilsAnalysis Container): provides a list of CMS analyses within a container that is ready to apply them to an event	HardScatteringSim (Table 8)	analyses (vector<vector<string>>)
IdentityAnalysis Container	getIdentityAnalysisContainer (ColliderBit::HEPUtilsAnalysisContainer): provides a list of “identity” analyses (no detector smearing) within a container that is ready to apply them to an event	HardScatteringSim (Table 8)	analyses (vector<vector<string>>)
DetAnalysis Numbers	runDetAnalyses (ColliderBit::AnalysisNumbers): Uses the given DetAnalysisContainer to perform all its analyses upon the given ReconstructedEvent	ReconstructedEvent (Table 9) HardScatteringSim (Table 8) DetAnalysisContainer	
ATLASAnalysis Numbers	runATLASAnalyses (ColliderBit::AnalysisNumbers): Uses the given ATLASAnalysisContainer to perform all its analyses upon the given ATLASSmearedEvent	ATLASSmearedEvent (Table 9) HardScatteringSim (Table 8) ATLASAnalysis Container	
CMSAnalysis Numbers	runCMSAnalyses (ColliderBit::AnalysisNumbers): Uses the given CMSAnalysisContainer to perform all its analyses upon the given CMSSmearedEvent	CMSSmearedEvent (Table 9) HardScatteringSim (Table 8) CMSAnalysisContainer	
IdentityAnalysis Numbers	runIdentityAnalyses (ColliderBit::AnalysisNumbers): Uses the given IdentityAnalysisContainer to perform all its analyses upon the given CopiedEvent	CopiedEvent (Table 9) HardScatteringSim (Table 8) IdentityAnalysis-Container	

```
namespace Pythia_ttbar_LHC_13TeV
{
  void init(SpecializablePythia* specializeMe)
  {
    specializeMe->addToSettings(
        "Beams:eCM = 13000");
    specializeMe->addToSettings(
        "Top:qqbar2ttbar = on");
    specializeMe->addToSettings(
       "Top:gg2ttbar = on");
  }
}
```

This would initialize a Pythia 8 configuration named `Pythia_ttbar_LHC_13TeV` to simulate $t\bar{t}$ production through $q\bar{q} \to t\bar{t}$ and $gg \to t\bar{t}$ at 13 TeV. Init functions may also "inherit" from other existing init functions by explicitly calling them. For instance:

```
namespace Pythia_alltop_LHC_13TeV
{
  void init(SpecializablePythia* specializeMe)
  {
    Pythia_ttbar_LHC_13TeV::init
(specializeMe);
    specializeMe->addToSettings(
        "Top:qq2tq(t:W) = on");
    specializeMe->addToSettings(
        "Top:ffbar2ttbar(s:gmZ) = on");
    specializeMe->addToSettings(
        "Top:ffbar2tqbar(s:W) = on");
    specializeMe->addToSettings(
        "Top:gmgm2ttbar = on");
  }
}
```

This creates a second Pythia 8 configuration named `Pythia_alltop_LHC_13TeV` for simulating the full set of top quark production processes. The first line of the init function calls the init function of `Pythia_ttbar_LHC_13TeV`, which sets the energy to 13 TeV and turns on the $q\bar{q} \to t\bar{t}$ and $gg \to t\bar{t}$ processes. Then follows four calls to `addToSettings` that switch on the simulation of the additional production processes $qq' \to tq''$ (t-channel W exchange), $f\bar{f} \to t\bar{t}$ (s-channel γ/Z exchange), $f\bar{f}' \to tq''$ (s-channel W exchange) and $\gamma\gamma \to t\bar{t}$.

2. *Add the initialisation function namespace within* `SpecializablePythia::resetSpecialization`. This function is located at the end of `colliders/SpecializablePythia.cpp`, and it allows for runtime selection of the Pythia 8 specialisation via a `std::string`. The namespaces for the new init functions must be included here using the `IF_X_SPECIALIZEX` macro. Thus, for our example top production init functions above, we would add:

```
IF_X_SPECIALIZEX(Pythia_ttbar_LHC_13TeV)
IF_X_SPECIALIZEX(Pythia_alltop_LHC_13TeV)
```

3. *Recompile GAMBIT.* (See Appendix A).
4. *Activate the new init function.* The choice of init function is specified within the GAMBIT YAML file in the "Rules" section for the `operateLHCLoop` module function.

For example, to activate our "alltop" Pythia 8 configuration, this would be the YAML entry:

```
  - capability: ColliderOperator
    function: operateLHCLoop
    options:
      nEvents: [1000]
      pythiaNames: [
  "Pythia_alltop_LHC_13TeV"]
      silenceLoop: true
```

The last of these options removes much of the output from the Pythia event generator. We may also supply our chosen configuration with additional options right in the YAML file. We do this in the rules for the `getPythia` module function:

```
  - capability: HardScatteringSim
    function: getPythia
    options:
      Pythia_alltop_LHC_13TeV: [
  "Print:quiet = on",
            "PartonLevel:MPI = off",
            "PartonLevel:ISR = on",
            "PartonLevel:FSR = on",
            "HadronLevel:all = on"]
```

We recommend registering commonly-used Pythia 8 configurations in `colliders/SpecializablePythia.cpp` as described above. However, it is also possible to set up custom Pythia configurations directly in the YAML file by specifying all relevant Pythia options there. In this case the configuration name given in `pythiaNames` must not match any of the registered init functions in `colliders/SpecializablePythia.cpp`.

For each Pythia configuration, the choice of analyses and detector simulations can be varied. Many of the options detailed in Tables 8, 9 and 10 are therefore vectors expecting one element per Pythia configuration, in the same order as the configurations in `pythiaNames`.

Adding a new analysis is nearly as simple as adding a Pythia 8 configuration. An annotated minimal example is given in `analyses/Analysis_Minimum.cpp`. This contains the minimum required to print out the number of jets, b-jets and leptons, and the missing energy in each event, plus pass an arbitrary set of signal region cuts. To add a new analysis:

1. *Copy the template example,* `Analysis_Minimum.cpp`, to a new location in the `analyses` folder, for example `analyses/Analysis_ATLAS_TYPE_20invfb.cpp`. Within the new file, replace every instance of `Minimum` by `ATLAS_TYPE_20invfb`.
2. *Edit the new analysis file to include the required cuts.* This includes the option to add extra signal regions. The existing repository of analyses provides examples of how to apply complex cuts.

3. *Add an analysis factory declaration to analyses/ HEPUtilsAnalysisContainer.cpp*, by adding the line:

```
DECLARE_ANALYSIS_FACTORY(ATLAS_TYPE_20invfb);
```

4. *Add a factory definition to analyses/ HEPUtilsAnalysis Container.cpp*, by adding a line:

```
IF_X_RTN_CREATEX(ATLAS_TYPE_20invfb);
```

5. *Recompile GAMBIT*. (See Appendix A).
6. *Activate the new analysis*. The user may now run the analysis by adding it to the list of analyses in the YAML file, within the rules for the relevant AnalysisContainer. For our ATLAS_TYPE_20invfb example, we would add it here:

```
  - capability: ATLASAnalysisContainer
    function: getATLASAnalysisContainer
    options:
      analyses: [["ATLAS_0LEP_20invfb",
            "ATLAS_TYPE_20invfb"]]
```

Although the current framework only supports cut-and-count analyses, the user could easily add more complicated likelihoods by adding new module functions.

3.1.2 LEP supersymmetry limit capabilities

ColliderBit contains functions that calculate the cross-section for various SUSY particle productions within the context of the LEP collider. The capabilities for these functions are described in Table 11. Using these functions along with SUSY particle decay information, we calculate the cross-section times branching ratio for each production mechanism associated with LEP model-independent limits. The capabilities and functions that compare this calculation with each LEP limit are described in Tables 12 and 13.

3.1.3 Higgs likelihood capabilities

ColliderBit provides likelihoods from experimental searches for Higgs bosons at LEP and the LHC, through interfaces to HiggsBounds and HiggsSignals. The capability LEP_Higgs_LogLike is provided by the function calc_HB_LEP _LogLike, which uses HiggsBounds to calculate an approximate likelihood constructed from the results from searches for neutral and charged Higgs bosons at LEP. Similarly, capability LHC_Higgs_LogLike is provided by function calc_HS_LHC _LogLike, which employs HiggsSignals to compute a likelihood including constraints from measurements of the Higgs boson production rates and mass at the LHC. These functions are detailed in Table 14, along with their dependencies.

Both functions depend on being provided with a HiggsBounds/Signals-specific data object containing all the input parameters needed to run either of these two external codes. ColliderBit constructs one of these objects from the Higgs_Couplings provided by SpecBit. There are separate functions to do this for a pure SM Higgs, an MSSM Higgs sector with three neutral and one charged Higgs, and a Higgs sector containing just one SM-like Higgs and possible invisible states for it to decay to, as in the scalar singlet and other such singlet Higgs portal models (e.g. [109,110]).

3.2 Standalone interface

As described in [1], GAMBIT routines can be called in a standalone code provided that the code specifies the module functions and backend functions that the user requires, along with any necessary options. In addition, the user must resolve the dependencies of each module function "by hand".

An annotated example program for running ColliderBit independently of the GAMBIT framework can be found in ColliderBit/examples/ColliderBit_standalone_ example.cpp. This example uses ColliderBit with a custom version of Pythia (8.212.EM) to calculate the LHC likelihood for a simple BSM model, with the required couplings, masses and branching ratios input via an SLHA file. The details of how to connect the custom Pythia version to ColliderBit and run the standalone are given in Sect. 4.2. Here we go through the structure of the code in ColliderBit_standalone_example.cpp.

The program consists of three main parts: dependency resolution, configuration of ColliderBit and Pythia, and execution of the simulation loop plus calculation of the LHC log-likelihood. It is the second part that the user typically will want to edit, as this is where the settings for event generation and detector simulation are specified, along with which LHC analyses to include.

To simplify the syntax a bit we use the following typedefs in ColliderBit_standalone_example.cpp:

```
using namespace std;
typedef vector<int> vint;
typedef vector<double> vdouble;
typedef vector<bool> vbool;
typedef vector<string> vstr;
typedef vector<vector<string> > vvstr;
```

The configuration section begins by setting up the function operateLHCLoop with settings for the LHC simulation loop. In this example we set up two Pythia configurations, "Pythia_EM_8Tev" and "Pythia_EM_13Tev", which will generate 20,000 events each. We will also allow detailed output to stdout during the event loops:

```
operateLHCLoop.setOption<vstr>("pythiaNames",
    vstr {"Pythia_EM_8Tev",
 ↪"Pythia_EM_13TeV"});
operateLHCLoop.setOption<vint>("nEvents",
    vint {20000, 20000});
operateLHCLoop.setOption<bool>("silenceLoop",
    false);
```

Table 11 The capabilities provided by ColliderBit that calculate SUSY particle production cross-sections within the context of the LEP collider. All of these functions return a triplet of `double`s. These correspond to the maximum, central, and minimum cross-sections calculated while varying the SUSY particle masses according to their estimated uncertainties. All of these functions depend on GAMBIT's `MSSM30atMGUT` model parameters [1], SpecBit's `MSSM_spectrum`, and DecayBit's `Z_decay_rates` [40]. Versions of these functions exist with many different E, X and Y values, corresponding to the energy (in GeV) and particle eigenstates used in the calculation

Capability	Function (return type): brief description	Energies for E	Eigenstates for X and Y
`LEP`E`_xsec_se`X`se`Y`bar`	`LEP`E`_SLHA1_convention_xsec_se`X`se`Y`bar` (`triplet<double>`): Calculates the LEP selectron pair production cross-section for centre of mass energy E, with selectron eigenstates X and Y	208 205 188	`l`, `r` (helicity) 1, 2 (mass)
`LEP`E`_xsec_smu`X`smu`Y`bar`	`LEP`E`_SLHA1_convention_xsec_smu`X`smu`Y`bar` (`triplet<double>`): Calculates the LEP smuon pair production cross-section for centre of mass energy E, with smuon eigenstates X and Y	208 205 188	`l`, `r` (helicity) 1, 2 (mass)
`LEP`E`_xsec_stau`X`stau`Y`bar`	`LEP`E`_SLHA1_convention_xsec_stau`X`stau`Y`bar` (`triplet<double>`): Calculates the LEP stau pair production cross-section for centre of mass energy E, with stau eigenstates X and Y	208 205 188	`l`, `r` (helicity) 1, 2 (mass)
`LEP`E`_xsec_chi00_`XY	`LEP`E`_SLHA1_convention_xsec_chi00_`XY (`triplet<double>`): Calculates the LEP neutralino pair production cross-section for centre of mass energy E, with neutralino mass eigenstates X and Y	208 205 188	1, 2, 3, 4
`LEP`E`_xsec_chipm_`XY	`LEP`E`_SLHA1_convention_xsec_chipm_`XY (`triplet<double>`): Calculates the LEP chargino pair production cross-section for centre of mass energy E, with chargino mass eigenstates X and Y	208 205 188	1, 2

Table 12 The slepton LEP limit capabilities provided by ColliderBit. In addition to the dependencies shown above, all of these functions also depend on GAMBIT's `MSSM30atMGUT` [1] model parameters and SpecBit's `MSSM_spectrum` [40] capability. Each of the `decay_rates` can be provided by DecayBit [40]. These functions have no options to be specified in the YAML file

Capability	Function (return type): brief description	Dependencies
`ALEPH_Selectron_LLike`	`ALEPH_Selectron_Conservative_LLike` (`double`): Compares the cross section times branching ratio for selectron pair production to the model-independent limit according to the ALEPH collaboration. Returns a log likelihood value	`LEP208_xsec_selselbar` `LEP208_xsec_serserbar` `selectron_l_decay_rates` `selectron_r_decay_rates`
`ALEPH_Smuon_LLike`	`ALEPH_Smuon_Conservative_LLike` (`double`): Compares the cross section times branching ratio for smuon pair production to the model-independent limit according to the ALEPH collaboration. Returns a log likelihood value	`LEP208_xsec_smulsmulbar` `LEP208_xsec_smursmurbar` `smuon_l_decay_rates` `smuon_r_decay_rates`
`ALEPH_Stau_LLike`	`ALEPH_Stau_Conservative_LLike` (`double`): Compares the cross section times branching ratio for stau pair production to the model-independent limit according to the ALEPH collaboration. Returns a log likelihood value	`LEP208_xsec_stau1stau1bar` `LEP208_xsec_stau2stau2bar` `stau_1_decay_rates` `stau_2_decay_rates`
`L3_Selectron_LLike`	`L3_Selectron_Conservative_LLike` (`double`): Compares the cross section times branching ratio for selectron pair production to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	`LEP205_xsec_selselbar` `LEP205_xsec_serserbar` `selectron_l_decay_rates` `selectron_r_decay_rates`
`L3_Smuon_LLike`	`L3_Smuon_Conservative_LLike` (`double`): Compares the cross section times branching ratio for smuon pair production to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	`LEP205_xsec_smulsmulbar` `LEP205_xsec_smursmurbar` `smuon_l_decay_rates` `smuon_r_decay_rates`
`L3_Stau_LLike`	`L3_Stau_Conservative_LLike` (`double`): Compares the cross section times branching ratio for stau pair production to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	`LEP205_xsec_stau1stau1bar` `LEP205_xsec_stau2stau2bar` `stau_1_decay_rates` `stau_2_decay_rates`

Then the `getPythiaFileReader` function can be configured with vectors of settings for the two Pythia configurations. For `"Pythia_EM_8Tev"` we have:

```
getPythiaFileReader.setOption<vstr>(
    "Pythia_EM_8Tev", vstr {
        "UserModel:all = on",
        "Beams:eCM = 8000",
        "PartonLevel:MPI = off",
        "PartonLevel:ISR = on",
        "PartonLevel:FSR = off",
        "HadronLevel:all = off",
        "TauDecays:mode = 0",
        "Random:setSeed = on"});
```

Here the `"UserModel:all = on"` setting turns on the processes in the new BSM model, as detailed in Sect. 4.2. The `"Pythia_EM_13Tev"` configuration is set up in a similar way, this time using `"Beams:eCM = 13000"`. The `getPythiaFileReader` function must also be given the path to the XML directory of Pythia 8.212.EM:

```
getPythiaFileReader.setOption<string>(
    "Pythia_doc_path", "Backends/installed/
        Pythia/8.212/share/Pythia8/xmldoc/");
```

In `ColliderBit_standalone_example.cpp` the path to a single input SLHA file is taken as a command line argument and stored in a variable `inputFileName`, which can then be passed to `getPythiaFileReader`:

```
getPythiaFileReader.setOption<vstr>(
    "SLHA_filenames", vstr {inputFileName});
```

Finally, we choose which detector simulators and LHC analyses to include. To use the ATLAS configuration of BuckFast with both `"Pythia_EM_8Tev"` and `"Pythia_EM_13`

Table 13 The gaugino LEP limit capabilities provided by ColliderBit. In addition to the dependencies shown above, all of these functions also depend on GAMBIT's MSSM30atMGUT [1] model parameters and SpecBit's MSSM_spectrum [40] capability. Each of the decay_rates can be provided by DecayBit [40]. These functions have no options to be specified in the YAML file. Note that the All_Channels likelihoods assume that the neutralino or chargino decay to fermions follows the same branching pattern as the corresponding on-shell gauge boson, and should be used with care

Capability	Function (return type): brief description	Dependencies
L3_Neutralino_All_Channels_LLike	L3_Neutralino_All_Channels_Conservative_LLike (double): Compares the cross section times branching ratio for neutralino pair production to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	LEP188_xsec_chi00_12 LEP188_xsec_chi00_13 LEP188_xsec_chi00_14 decay_rates
L3_Neutralino_Leptonic_LLike	L3_Neutralino_Leptonic_Conservative_LLike (double): Compares the cross section times branching ratio for neutralino pair production (with leptonically decaying Z bosons) to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	LEP188_xsec_chi00_12 LEP188_xsec_chi00_13 LEP188_xsec_chi00_14 decay_rates
L3_Chargino_All_Channels_LLike	L3_Chargino_All_Channels_Conservative_LLike (double): Compares the cross section times branching ratio for chargino pair production to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	LEP188_xsec_chipm_11 LEP188_xsec_chipm_22 decay_rates
L3_Chargino_Leptonic_LLike	L3_Chargino_Leptonic_Conservative_LLike (double): Compares the cross section times branching ratio for chargino pair production (with leptonically decaying W bosons) to the model-independent limit according to the L3 collaboration. Returns a log likelihood value	LEP188_xsec_chipm_11 LEP188_xsec_chipm_22 decay_rates
OPAL_Neutralino_Hadronic_LLike	OPAL_Neutralino_Hadronic_Conservative_LLike (double): Compares the cross section times branching ratio for neutralino pair production (with hadronically decaying Z bosons) to the model-independent limit according to the OPAL collaboration. Returns a log likelihood value	LEP208_xsec_chi00_12 LEP208_xsec_chi00_13 LEP208_xsec_chi00_14 decay_rates
OPAL_Chargino_All_Channels_LLike	OPAL_Chargino_All_Channels_Conservative_LLike (double): Compares the cross section times branching ratio for chargino pair production to the model-independent limit according to the OPAL collaboration. Returns a log likelihood value	LEP208_xsec_chipm_11 LEP208_xsec_chipm_22 decay_rates
OPAL_Chargino_Hadronic_LLike	OPAL_Chargino_Hadronic_Conservative_LLike (double): Compares the cross section times branching ratio for chargino pair production (with hadronically decaying W bosons) to the model-independent limit according to the OPAL collaboration. Returns a log likelihood value	LEP208_xsec_chipm_11 LEP208_xsec_chipm_22 decay_rates
OPAL_Chargino_Leptonic_LLike	OPAL_Chargino_Leptonic_Conservative_LLike (double): Compares the cross section times branching ratio for chargino pair production (with leptonically decaying W bosons) to the model-independent limit according to the OPAL collaboration. Returns a log likelihood value	LEP208_xsec_chipm_11 LEP208_xsec_chipm_22 decay_rates
OPAL_Chargino_SemiLeptonic_LLike	OPAL_Chargino_SemiLeptonic_Conservative_LLike (double): Compares the cross section times branching ratio for chargino pair production (with one leptonic and one hadronic decaying W boson) to the model-independent limit according to the OPAL collaboration. Returns a log likelihood value	LEP208_xsec_chipm_11 LEP208_xsec_chipm_22 decay_rates

Tev" we configure the ColliderBit function getBuckFastATLAS as follows:

```
getBuckFastATLAS.setOption<vbool>(
    "useDetector", vbool {true, true});
getBuckFastATLAS.setOption<vdouble>(
    "antiktR", vdouble {0.4, 0.4});
getBuckFastATLAS.setOption<vbool>(
    "partonOnly", vbool {false, false});
```

The names of the ATLAS analyses to include are then passed to the function getATLASAnalysisContainer. Here we include the 0-lepton searches at 8 and 13 TeV:

```
getATLASAnalysisContainer.setOption<vvstr>(
    "analyses", vvstr {{"ATLAS_0LEP_20invfb"},
        {"ATLAS_13TeV_0LEP_13invfb"}});
```

Table 14 The capabilities provided by ColliderBit for calculating LEP and LHC likelihoods from Higgs-sector-related experimental constraints. Final likelihood calculations are performed by the external code packages HiggsBounds and HiggsSignals, using interfaces for input and output of model parameters incorporated into the GAMBIT framework. `Higgs_Couplings` are typically provided by SpecBit [40]

Capability	Function (return type): brief description	Dependencies	Backend requirements
`LEP_Higgs_LogLike`	`calc_HB_LEP_LogLike`(`double`): Provides log-likelihood for combined model-independent LEP neutral Higgs searches	`HB_ModelParameters`	HiggsBounds
`LHC_Higgs_LogLike`	`calc_HS_LHC_LogLike`(`double`): Provides log-likelihood for LHC Higgs mass and signal strength measurements	`HB_ModelParameters`	HiggsSignals
`HB_ModelParameters`	`SMHiggs_ModelParameters`(`hb_ModelParameters`): Provides inputs for LEP and LHC Higgs likelihood calculations with HiggsBounds and HiggsSignals, for a Higgs sector consisting only of an SM Higgs	`Higgs_Couplings` `SM_spectrum`	
	`SMHiggs_ModelParameters`(`hb_ModelParameters`): Provides inputs for LEP and LHC Higgs likelihood calculations with HiggsBounds and HiggsSignals, for a Higgs sector consisting only of a single neutral Higgs, with possible decays to additional invisible particles	`Higgs_Couplings` A relevant `Spectrum` object	
	`MSSMHiggs_ModelParameters`(`hb_ModelParameters`): Provides inputs for LEP and LHC Higgs likelihood calculations with HiggsBounds and HiggsSignals, for an MSSM Higgs sector	`Higgs_Couplings` `MSSM_spectrum`	
`FH_HiggsProd`	`FH_HiggsProd`(`fh_HiggsProd`): Provides estimated MSSM Higgs production cross sections through an interface to FeynHiggs		FeynHiggs

Note that the two analyses are given in separate subvectors, one for each Pythia configuration (see Table 10). CMS analyses are similarly included by configuring `getBuckFastCMS` and `getCMSAnalysisContainer`. In our example, we only use a Run I CMS analysis, which is therefore only applied to the 8 TeV Pythia configuration.

The full LHC simulation loop and likelihood calculation is run in the third part of the main program, by executing the ColliderBit functions `operateLHCLoop` and `calc_LHC_LogLike`:

```
operateLHCLoop.reset_and_calculate();
calc_LHC_LogLike.reset_and_calculate();
```

4 Examples

4.1 CMSSM example

An annotated example of a YAML file for scanning the CMSSM with GAMBIT using only functions from ColliderBit is provided in `yaml_files/ColliderBit_CMSSM.yaml`. The file demonstrates how to specify the model parameters (and priors), choose and configure a sampler, choose a printer (either hdf5 or ascii), run the LHC and LEP collider likelihoods, run the HiggsBounds and HiggsSignals Higgs likelihoods, and configure details of the detector simulation and Monte Carlo event generator.

4.2 Generic Pythia model example

The recommended method of using ColliderBit with a new model is to define and run the model within the full GAMBIT framework, allowing access to the model declaration and scanning routines, in addition to non-collider likelihood functions should these be of interest. However, if the user only wants to check single parameter points with ColliderBit, the standalone interface described in the previous section presents a more minimal alternative. Regardless of which interface is used, ColliderBit must be set up to work with a version of Pythia that can generate events for the new model. Here we go through an example of how to achieve this.

Our physics model example consists of the SM augmented by a new scalar singlet field ϕ_1 and a new, coloured Dirac fermion U. The model is a stripped down version of that featured in [111], which contains a complete tutorial for how to implement the model in Monte Carlo generators. The new particles have the following mass terms:

$$\mathcal{L}_{\text{mass}} = -\frac{m_1^2}{2}\phi_1^2 + M_U \bar{U} U. \tag{8}$$

The new fermion interacts with the new scalar via the Lagrangian term

$$\mathcal{L}_{\text{yuk}} = \lambda_1 \phi_1 \bar{U} P_R u + h.c., \tag{9}$$

where u is the SM up-quark field. We will simulate the process $pp \to \bar{U}U$ where the U subsequently decays via $U \to u\phi_1$.

To use this model with ColliderBit, we make use of the MadGraph5_aMC@NLO–Pythia 8 interface to generate matrix element code that can be used to supplement the internal processes in Pythia. Sample Mathematica notebook and Feynrules model files for generating UFO output are provided in `ColliderBit/data/ExternalModel`. The MadGraph5_aMC@NLO commands for generating matrix element code for coloured fermion production in proton collisions are as follows, assuming that the UFO model has been placed in the MadGraph5_aMC@NLO `models` directory:

```
import model GambitDemo_UFO
generate p p > uv uv~
output pythia8
```

The resulting C++ code can be found in the `src` and `include` subdirectories of `Backends/patches/pythia/8.212.EM/ExternalModel`. This directory also contains two Pythia XML files that declare a new Pythia setting `UserModel:all`, and a version of the Pythia file `ProcessContainer.cc` that connects this setting to the generated matrix element code.

The GAMBIT build system can be used to make a new version of the Pythia backend (8.212.EM) with

```
make pythia_8.212.EM
```

This command performs the following tasks:

- downloads Pythia in the usual way, but into a new location;
- copies the new matrix element code to the new location;
- updates the Pythia XML configuration files to define the new `UserModel` setting;
- updates the Pythia file `ProcessContainer.cc` to allow the user to run the new matrix elements using the setting `UserModel:all = on`;
- runs Pythia through BOSS [1] to construct the interface to GAMBIT;
- builds the new Pythia version.

To implement a different BSM physics model, the existing MadGraph5_aMC@NLO-generated files in `Backends/patches/pythia/8.212.EM/ExternalModel` must be replaced with the files generated for the new model, and `ProcessContainer.cc` must be updated accordingly. The GAMBIT build system will then take care of updating the Pythia backend to use the new code.

It remains to tell ColliderBit to use the new Pythia 8.212.EM backend rather than the old one. Since this contains all previous Pythia functionality in addition to the new matrix elements, it can be used in all places where Pythia 8.212 was previously used. To change the version of Pythia used, the user must change the default version in `Backends/include/gambit/Backends/default_bossed_versions.hpp`, using:

```
#define Default_Pythia 8_212_EM
```

Note that GAMBIT must be rebuilt after this change. Also, the ColliderBit option `Pythia_doc_path` (Table 8) must be set to `Backends/installed/Pythia/8.212.EM/share/Pythia8/xmldoc` in the input YAML file when ColliderBit is used as part of a GAMBIT run, or directly in the standalone code as shown in `ColliderBit_standalone_example.cpp`.

An example YAML file showing how to run GAMBIT with the new Pythia 8.212.EM backend can be found in `yaml_files/ColliderBit_ExternalModel.yaml`.

After compilation (see Appendix A), the standalone example that makes use of the new Pythia 8.212.EM backend can be run as

```
./ColliderBit_standalone
    ColliderBit/data/ExternalModel_point.slha
```

This instructs Pythia to produce $\bar{U}U$ pairs in proton collisions, but they will not decay unless instructed to do so via the input SLHA file. An example SLHA file generated with MadGraph5_aMC@NLO is provided in `ColliderBit/data/ExternalModel_point.slha`. This file contains a decay table for the U particle with a 100% branching ratio to an up quark and a ϕ_1.

We remind the reader that the standalone example is only intended as a minimal way of running single points of a new model through ColliderBit. For a comprehensive study, including scanning over model parameters, the user should add the model in the GAMBIT model database and implement spectrum and decay calculations through the GAMBIT modules SpecBit and DecayBit as required.

Finally, there is an important subtlety regarding invisible particles. At the time of writing, the default PDG ID codes of new particles in Feynrules do not always correspond to those of invisible, uncharged particles. In the ColliderBit simulation chain, this means that the particles will not appear as missing energy in the detector simulation. According to the PDG ID code standard, invisible particles may have a PID of 12, 14 or 16 (SM neutrinos), 1000022 (lightest neutralino in a superysmmetric model), or 50–60 (for generic new BSM particles). The user can thus obtain correct behaviour for an invisible species by including the PDG code definition in the Feynrules field definition as in the following example:

```
 S[10] == {
    ClassName        -> p1,
    SelfConjugate    -> True,
    Indices          -> {},
    Mass             -> {Mp1, 10},
    PDG              -> {51},
    Width            -> {Wp1, 0}
 }
```

A less satisfactory option is to change the following code in `contrib/heputils/include/HEPUtils/Event.h` that implements the PDG ID standard for invisibles:

```
if (p->abspid() == 12 || p->abspid() == 14 ||
    p->abspid() == 16 || p->pid() == 1000022 ||
    in_range(p->pid(), 50, 60))
  _invisibles.push_back(p);
```

5 Conclusions

ColliderBit is a new modular software code for the application of high-energy collider constraints to generic BSM physics models, written in the GAMBIT framework. This paper serves as an introduction to the code, and as a reference manual for users wishing to add new analyses or features.

The code provides a rigorous and fast implementation of LHC constraints through a parallelised Monte Carlo simulation interfaced with several detector simulation options, including a new simulation based on four-vector smearing. A custom event analysis class allows the user to apply the same LHC analysis code to any level of detector simulation, and we supply likelihood routines capable of reproducing LHC cut and count searches, or binned shape fits. An interface to the Pythia 8 event generator allows the user to add matrix elements for new models.

LEP constraints are handled via a new code based on a sophisticated interpolation of the cross-section limits on slepton, neutralino and chargino pair production. Higgs limits, for both LEP and the LHC, are currently handled via an interface to the HiggsSignals and HiggsBounds packages, but there exists scope to provide and interface new likelihood calculations in future ColliderBit releases.

The code can function either as a standalone tool for quick checks of specific model points, or it can be run within the GAMBIT framework to provide a complete tool for BSM inference from high energy collider data.

Acknowledgements We thank the other members of the GAMBIT Collaboration for helpful discussions, comments and support. We are very grateful to Torbjörn Sjöstrand and Peter Skands for helpful discussions on the use of the Pythia event generator, and for code modifications to improve the efficiency and flexibility of its process selection and settings database. We warmly thank the Casa Matemáticas Oaxaca, affiliated with the Banff International Research Station, for hospitality whilst part of this work was completed, and the staff at Cyfronet, for their always helpful supercomputing support. GAMBIT has been supported by STFC (UK; ST/K00414X/1, ST/P000762/1), the Royal Society (UK; UF110191), Glasgow University (UK; Leadership Fellowship), the Research Council of Norway (FRIPRO 230546/F20), NOTUR

(Norway; NN9284K), the Knut and Alice Wallenberg Foundation (Sweden; Wallenberg Academy Fellowship), the Swedish Research Council (621-2014-5772), the Australian Research Council (CE110001004, FT130100018, FT140100244, FT160100274), The University of Sydney (Australia; IRCA-G162448), PLGrid Infrastructure (Poland), Polish National Science Center (Sonata UMO-2015/17/D/ST2/03532), the Swiss National Science Foundation (PP00P2-144674), the European Commission Horizon 2020 Marie Skłodowska-Curie actions (H2020-MSCA-RISE-2015-691164), the ERA-CAN+ Twinning Program (EU and Canada), the Netherlands Organisation for Scientific Research (NWO-Vidi 680-47-532), the National Science Foundation (USA; DGE-1339067), the FRQNT (Québec) and NSERC/The Canadian Tri-Agencies Research Councils (BPDF-424460-2012).

Appendix A: Quick start guide

Instructions for how to get ColliderBit and GAMBIT can be found at gambit.hepforge.org. Here, we give a list of steps to follow in order to build and run ColliderBit, either in its standalone version or linked with GAMBIT. Additional details about configuring and building GAMBIT can be found in [1].

A.1: Building and running the standalone example

The basic commands to build the standalone example are:

```
cd gambit
mkdir build
cd build
cmake ..
make -jn ColliderBit_standalone
```

Here, n is the number of logical cores the user wishes to use during the compilation.

The backends used by the standalone example must also be built:

```
make nulike
make pythia_8.212.EM
```

The user can set the number of OpenMP threads to use during ColliderBit's parallelisation step with a system variable:

```
export OMP_NUM_THREADS=m
```

Here, m is the number of threads to use during runtime.

Finally, the standalone example can be run from the main GAMBIT directory:

```
cd ..
./ColliderBit_standalone
    ColliderBit/data/ExternalModel_point.slha
```

A.2: Running the ColliderBit example in GAMBIT

The basic commands to build GAMBIT and run a minimal ColliderBit example are very similar to those shown above, except that we now also need the backend SUSY-HIT, and we use the default version Pythia (8.212):

```
cd gambit
mkdir build
cd build
cmake ..
make -jn gambit
make nulike
make pythia
make susyhit
export OMP_NUM_THREADS=m
cd ..
./gambit -f yaml_files/ColliderBit_CMSSM.yaml
```

Appendix B: ColliderBit classes

Users who wish to add their own custom functions to ColliderBit may find it useful to use our inheritance scheme. For such users, we here describe the main base classes and inheritance scheme of ColliderBit. We expect such users to be familiar with adding capabilities, module functions, and (possibly) backend functions, as described in the main GAMBIT paper [1].

There are four categories of functions within ColliderBit associated with abstract base classes: Collider simulation is associated with the `BaseCollider` class, detector simulation with `BaseDetector`, analysis with `BaseAnalysis`, and the limit-setting application with `BaseLimitContainer`.

The `BaseCollider` class is templated on the type of collider event (`EventT`) that it can provide. Each subclass of `BaseCollider<EventT>` will inherit the virtual functions described in Table 15. Thus, creating a subclass of `BaseCollider` will force the user to define these functions, which are the usual things to be expected of collider simulation tools. A very simple example of this can be found in the header file[10] `colliders/SimplePythia.hpp`.

Within this file, we see the definition of the `SimplePythia` class, which inherits from `BaseDetector<Pythia8::Event>`. The class defines overrides for each of the virtual functions shown in Table 15. A more complicated example of this can be seen for the `SpecializablePythia` class, which also inherits from `BaseDetector<Pythia8::Event>`. It is declared and defined within the files `colliders/SpecializablePythia.hpp` and `colliders/SpecializablePythia.cpp`.

[10] Within this Appendix, the headers paths (`*.hpp`) are realtive to `ColliderBit/ include/gambit/ColliderBit` , while source files (`*.cpp`) are in `ColliderBit/src`.

Table 15 Inherited functions for subclasses of BaseCollider<EventT>. Functions marked with [a] do nothing unless overridden by the subclass author. Functions marked with [b] *must* be overridden by the subclass author

C++ function signature	Intended purpose
`virtual void clear()`[a]	Clear the internal memory of this instance so that it may be reused
`virtual void nextEvent(EventT & event) const`[b]	Generate the next collider event, storing the result into the given `event`
`virtual double xsec_pb() const`[b]	Return the total cross section (in pb) of generated events
`virtual double xsecErr_pb() const`[b]	Return the absolute error estimate of the cross section (in pb) of generated events
`virtual void init(const std::vector<std::string>&)`[a]	Initialise the collider simulator with a set of options given as a vector of strings
`virtual void init()`[a]	Initialise the collider simulator with no options.

Table 16 Inherited functions for subclasses of BaseDetector<EventIn, EventOut>. Functions marked with [a] do nothing unless overridden by the subclass author. Functions marked with [b] *must* be overridden by the subclass author

C++ function signature	Intended purpose
`virtual void clear()`[a]	Clear the internal memory of this instance so that it may be reused
`virtual void processEvent(const EventIn&, EventOut&) const`[b]	Apply detector simulation to the given `EventIn`, storing the result into the `EventOut`
`virtual void init(const std::vector<std::string>&)`[a]	Initialise the collider simulator with a set of options given as a vector of strings
`virtual void init()`[a]	Initialise the collider simulator with no options

The addition of custom detectors to ColliderBit involves subclasses of the `BaseDetector` class, which is templated on both the type of event that it can accept for simulation (`EventIn`), and the type of event that it will return after detector simulation (`EventOut`). In a similar way as described above for colliders, a user may add a fully custom detector by creating a subclass of `BaseDetector<EventIn, EventOut>` and writing overrides for the virtual functions, as described in Table 16.

The analysis base class, `BaseAnalysis`, is templated on the type of event that it can analyze (`EventT`). Subclasses of `BaseAnalysis<EventT>` inherit the functions as described in Table 17, some of which must be overridden by the subclass author. Existing analyses provide examples of how to do this.

The addition of custom limits and limit curve interpolation to ColliderBit requires that the user declare new module functions in `ColliderBit_rollcall.hpp` and define them in `ColliderBit.cpp`. However, if the user wishes to use ColliderBit's limit interpolation system (as described in Sect. 2.2), they can create a subclass of the `BaseLimitContainer` class and override the functions shown in Table 19.

Table 17 Inherited functions for subclasses of `BaseAnalysis<EventT>`. Functions that are `virtual` and marked with [a] perform only simple operations on variables in the base class, unless they are overridden by the subclass author. Functions that are `virtual` and marked with [b] *must* be overridden by the subclass author. Non-`virtual` functions are not intended to be overridden. Analysis results are contained within each subclass in a `SignalRegionData` instance, which is described in Table 18

C++ function signature	Intended purpose
`virtual void clear()`[a]	Clear the internal memory of this instance so that it may be reused
`void analyze(const EventT& e)`	This version of `analyze` simply calls the pointer version below
`virtual void analyze(const EventT*)`[a]	Analyze the given event, storing the result internally
`double num_events()`	Return the total number of events analyzed
`double xsec()`	Return the total cross section (in fb) of events analyzed
`double xsec_err()`	Return the cross section uncertainty (in fb)
`double xsec_relerr()`	Return the relative cross section uncertainty
`double xsec_per_event()`	Return the cross section per event (in fb) of events analyzed
`double luminosity()`	Return the integrated luminosity (in fb^{-1}) of events analyzed
`void set_xsec(double xs, double xserr)`	Set the cross section, and its uncertainty (in fb)
`void set_luminosity(double lumi)`	Set the luminosity (in fb^{-1})
`std::vector<SignalRegionData> get_results()`	Return the results as a `vector` of `SignalRegionData` objects
`void add_result(const SignalRegionData& res)`	Add a result to the internal results list
`virtual void collect_results()`[b]	Collect all results of this analysis together in preparation for a likelihood calculation
`virtual void init(const std::vector<std::string>&)`[a]	Initialise the analysis with a set of options given as a vector of strings
`virtual void init()`[a]	Initialise the analysis with no options
`virtual void scale(double factor)`[a]	Scale the results of this analysis by the given factor, which is optional. If no factor is given, the scale factor is set instead by the luminosity
`virtual void add(BaseAnalysis* other)`[a]	Adds the results of an identical analysis to this one
`void add_xsec(double xs, double xserr)`	Add the given cross section to the stored total and recompute the uncertainty
`void improve_xsec(double xs, double xserr)`	Improve the stored cross section by averaging it with the given one, and recompute the uncertainty

Table 18 Member variables of the `SignalRegionData` struct, which is used in the `BaseAnalysis` class as a container for the analysis results of each signal region. The `BaseAnalysis` class is described in Table 17

C++ Member Variable	Intended purpose
`std::string analysis_name`	The name of the analysis that contains this signal region
`std::string sr_label`	A label for this particular signal region
`double n_observed`	The number of events passing selection for this signal region, as reported by the experiment
`double n_signal`	The number of simulated model events passing selection for this signal region
`double n_signal_at_lumi`	The number of simulated model events passing selection for this signal region, scaled to the experimental luminosity
`double n_background`	The number of Standard Model events passing selection for this signal region, as reported by the experiment
`double signal_sys`	The absolute systematic error of `n_signal`
`double background_sys`	The absolute systematic error of `n_background`

Table 19 Inherited functions for subclasses of `BaseLimitContainer`. Functions that are `virtual` and marked with [b] *must* be overridden by the subclass author. Non-`virtual` functions are not intended to be overridden. Note that `P2` is a simple class that represents a point found on the limit plane and is defined in the header file `limits/PointsAndLines.hpp`

C++ function signature	Intended purpose
`virtual P2 convertPt() const`[b]	Convert a point from pixel units to axis units, creating a `P2`
`virtual bool isWithinExclusionRegion(double x, double y, double mZ)const`	Check to see if the point specified by `x` and `y` is within the exclusion region. This may depend on the Z boson mass, `mZ`
`virtual double specialLimit(double, double) const`	Return a special value for the limit when outside of the exclusion region. This function returns zero unless overridden
`double limitAverage(double x, double y, double mZ) const`	Uses the limit interpolation described in Sect. 2.2 to return a limit at the point specified by `x` and `y`. This may depend on the Z boson mass, `mZ`
`void dumpPlotData(double xlow, double xhigh, double ylow, double yhigh, double mZ, std::string filename, int ngrid) const`	Creates a file, `filename`, containing the results of `limitAverage` calls using a grid of `ngrid` × `ngrid` points within the rectangle defined by `xlow`, `xhigh`, `ylow`, and `yhigh`
`void dumpLightPlotData(std::string filename, int nperLine) const`	Creates a file, `filename`, containing the limit contour data as a series of points, using `nperLine` points for each line of the contour

Appendix C: Glossary

Here we explain some terms that have specific technical definitions in GAMBIT.

backend An external code containing useful functions (or variables) that one might wish to call (or read/write) from a **module function**.

backend function A function contained in a **backend**. It calculates a specific quantity indicated by its **capability**. Its capability and call signature are defined in the backend's **frontend header**.

backend requirement A declaration that a given **module function** needs to be able to call a **backend function** or use a **backend variable**, identified according to its **capability** and type(s). Backend requirements are declared in module functions' entries in **rollcall headers**.

backend variable A global variable contained in a **backend**. It corresponds to a specific quantity indicated by its **capability**. Its capability and type are defined in the backend's **frontend header**.

capability A name describing the actual quantity that is calculated by a module or backend function. This is one possible place for units to be noted; the other is in the documented description of the capability (see Sec. 10.7 of Ref. [1]).

dependency A declaration that a given **module function** needs to be able to access the result of another module function, identified according to its **capability** and type. Dependencies are declared in module functions' entries in **rollcall headers**.

frontend The interface between GAMBIT and a given **backend**, consisting of a **frontend header** plus optional source files and type headers.

frontend header The C++ header in which the **frontend** to a given **backend** is declared.

module A subset of GAMBIT functions following a common theme, able to be compiled into a standalone library. Although **module** often gets used as shorthand for **physics module**, this term technically also includes the GAMBIT scanning module ScannerBit.

module function A function contained in a **physics module**. It calculates a specific quantity indicated by its **capability** and **type**, as declared in the module's **rollcall header**. It takes only one argument, by reference (the quantity to be calculated), and has a void return type.

physics module Any **module** other than ScannerBit, containing a collection of **module functions** following a common physics theme.

rollcall header The C++ header in which a given **physics module** and its **module functions** are declared.

type A general fundamental or derived C++ type, often referring to the type of the **capability** of a **module function**.

References

1. GAMBIT Collaboration: P. Athron, C. Balazs et. al., GAMBIT: The Global and Modular Beyond-the-Standard-Model Inference Tool. arXiv:1705.07908
2. ATLAS Collaboration: G. Aad et. al., The ATLAS experiment at the CERN Large Hadron Collider. JINST **3**, S08003 (2008)

3. CMS Collaboration: S. Chatrchyan et al., The CMS experiment at the CERN LHC. JINST **3**, S08004 (2008)
4. L3 Collaboration: M. Acciarri et al., Search for charginos and neutralinos in e^+e^- collisions at $\sqrt{s} = 189$ GeV. Phys. Lett. B **472**, 420–433 (2000). arXiv:hep-ex/9910007
5. L3 Collaboration: M. Acciarri et al., Search for charginos with a small mass difference with the lightest supersymmetric particle at $\sqrt{s} = 189$ GeV. Phys. Lett. B **482**, 31–42 (2000). arXiv:hep-ex/0002043
6. L3 Collaboration: P. Achard et al., Search for scalar leptons and scalar quarks at LEP. Phys. Lett. B **580**, 37–49 (2004). arXiv:hep-ex/0310007
7. ALEPH Collaboration: A. Heister et. al., Search for charginos nearly mass degenerate with the lightest neutralino in e^+e^- collisions at center-of-mass energies up to 209 GeV. Phys. Lett. B **533**, 223–236 (2002). arXiv:hep-ex/0203020
8. ALEPH Collaboration: A. Heister et. al., Search for scalar leptons in e^+e^- collisions at center-of-mass energies up to 209 GeV. Phys. Lett. B **526**, 206–220 (2002). arXiv:hep-ex/0112011
9. ALEPH Collaboration: A. Heister et. al., Absolute lower limits on the masses of selectrons and sneutrinos in the MSSM. Phys. Lett. B **544**, 73–88 (2002). arXiv:hep-ex/0207056
10. ALEPH Collaboration: A. Heister et. al., Absolute mass lower limit for the lightest neutralino of the MSSM from e^+e^- data at $\sqrt{s}$ up to 209 GeV. Phys. Lett. B **583**, 247–263 (2004)
11. ALEPH Collaboration: A. Heister et al., Search for scalar quarks in e^+e^- collisions at $\sqrt{s}$ up to 209 GeV. Phys. Lett. B **537**, 5–20 (2002). arXiv:hep-ex/0204036
12. OPAL Collaboration: G. Abbiendi et al., Search for chargino and neutralino production at $\sqrt{s} = 192 GeV$ to 209 GeV at LEP. Eur. Phys. J. C **35**, 1–20 (2004). arXiv:hep-ex/0401026
13. OPAL Collaboration: G. Abbiendi et al., Search for anomalous production of dilepton events with missing transverse momentum in e^+e^- collisions at $\sqrt{s} = 183$ GeV to 209 GeV. Eur. Phys. J. C **32**, 453–473 (2004). arXiv:hep-ex/0309014
14. OPAL Collaboration: G. Abbiendi et al., Search for scalar top and scalar bottom quarks at LEP. Phys. Lett. B **545**, 272–284 (2002). arXiv:hep-ex/0209026
15. DELPHI Collaboration: J. Abdallah et al., Searches for supersymmetric particles in e^+e^- collisions up to 208 GeV and interpretation of the results within the MSSM. Eur. Phys. J. C **31**, 421–479 (2003). arXiv:hep-ex/0311019
16. ATLAS Collaboration: G. Aad et al., Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Phys. Lett. B **716**, 1–29 (2012). arXiv:1207.7214
17. S. Chatrchyan, V. Khachatryan et al., Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. Phys. Lett. B **716**, 30–61 (2012). arXiv:1207.7235
18. S. Kraml, S. Kulkarni et al., SModelS: a tool for interpreting simplified-model results from the LHC and its application to supersymmetry. Eur. Phys. J. C **74**, 2868 (2014). arXiv:1312.4175
19. M. Papucci, K. Sakurai, A. Weiler, L. Zeune, Fastlim: a fast LHC limit calculator. Eur. Phys. J. C **74**, 3163 (2014). arXiv:1402.0492
20. S. Caron, J.S. Kim, K. Rolbiecki, R. Ruiz de Austri, B. Stienen, The BSM-AI project: SUSY-AIâĂŞgeneralizing LHC limits on supersymmetry with machine learning. Eur. Phys. J. C **77**, 257 (2017). arXiv:1605.02797
21. H.K. Dreiner, M. Krämer, J.M. Lindert, B. O'Leary, SUSY parameter determination at the LHC using cross sections and kinematic edges. JHEP **4**, 109 (2010). arXiv:1003.2648
22. M. Bridges, K. Cranmer et al., A coverage study of CMSSM based on ATLAS sensitivity using fast neural networks techniques. JHEP **3**, 12 (2011). arXiv:1011.4306
23. A. Buckley, A. Shilton, M.J. White, Fast supersymmetry phenomenology at the Large Hadron Collider using machine learning techniques. Comput. Phys. Commun. **183**, 960–970 (2012). arXiv:1106.4613
24. B.O'Leary, LHC-FASER. http://github.com/benoleary/LHC-FASER
25. C. Balázs, A. Buckley, D. Carter, B. Farmer, M. White, Should we still believe in constrained supersymmetry? Eur. Phys. J. C **73**, 2563 (2013). arXiv:1205.1568
26. ATLAS Collaboration: ATLAS Collaboration, Summary of the ATLAS experiment's sensitivity to supersymmetry after LHC Run 1 – interpreted in the phenomenological MSSM. JHEP **10**, 134 (2015). arXiv:1508.06608
27. N. Bornhauser, M. Drees, Determination of the CMSSM parameters using neural networks. Phys. Rev. D **88**, 075016 (2013). arXiv:1307.3383
28. P. Bechtle, S. Belkner et al., SCYNet: testing supersymmetric models at the LHC with neural networks. arXiv:1703.01309
29. M. Drees, H. Dreiner, D. Schmeier, J. Tattersall, J.S. Kim, CheckMATE: confronting your favourite new physics model with LHC data. Comput. Phys. Commun. **187**, 227–265 (2014). arXiv:1312.2591
30. J. de Favereau et al., DELPHES 3. A modular framework for fast simulation of a generic collider experiment. JHEP **1402**, 057 (2014). arXiv:1307.6346
31. S. Ovyn, X. Rouby, V. Lemaitre, DELPHES, a framework for fast simulation of a generic collider experiment. arXiv:0903.2225
32. J. Alwall, R. Frederix et al., The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. JHEP **07**, 079 (2014). arXiv:1405.0301
33. P. Bechtle, O. Brein, S. Heinemeyer, G. Weiglein, K.E. Williams, HiggsBounds: confronting arbitrary Higgs sectors with exclusion bounds from LEP and the Tevatron. Comput. Phys. Commun. **181**, 138–167 (2010). arXiv:0811.4169
34. P. Bechtle, O. Brein, S. Heinemeyer, G. Weiglein, K.E. Williams, HiggsBounds 2.0.0: confronting neutral and charged Higgs sector predictions with exclusion bounds from LEP and the Tevatron. Comput. Phys. Commun. **182**, 2605–2631 (2011). arXiv:1102.1898
35. P. Bechtle, O. Brein et al., HiggsBounds-4: improved tests of extended Higgs sectors against exclusion bounds from LEP, the Tevatron and the LHC. Eur. Phys. J. C **74**, 2693 (2014). arXiv:1311.0055
36. P. Bechtle, S. Heinemeyer, O. Stål, T. Stefaniak, G. Weiglein, Applying exclusion likelihoods from LHC searches to extended Higgs sectors. Eur. Phys. J. C **75**, 421 (2015). arXiv:1507.06706
37. P. Bechtle, S. Heinemeyer, O. Stål, T. Stefaniak, G. Weiglein, HiggsSignals: confronting arbitrary Higgs sectors with measurements at the Tevatron and the LHC. Eur. Phys. J. C **74**, 2711 (2014). arXiv:1305.1933
38. J. Bernon, B. Dumont, Lilith: a tool for constraining new physics from Higgs measurements. Eur. Phys. J. C **75**, 440 (2015). arXiv:1502.04138
39. GAMBIT Scanner Workgroup: G.D. Martinez, J. McKay et al., Comparison of statistical sampling methods with ScannerBit, the GAMBIT scanning module. arXiv:1705.07959
40. GAMBIT Models Workgroup: P. Athron, C. Balázs et al., SpecBit, DecayBit and PrecisionBit: GAMBIT modules for computing mass spectra, particle decay rates and precision observables. arXiv:1705.07936
41. GAMBIT Flavour Workgroup: F.U. Bernlochner, M. Chrzaszcz et al., FlavBit: a GAMBIT module for computing flavour observables and likelihoods. arXiv:1705.07933
42. GAMBIT Dark Matter Workgroup: T. Bringmann, J. Conrad et al., DarkBit: a GAMBIT module for computing dark matter observables and likelihoods. arXiv:1705.07920

43. ATLAS Collaboration: G. Aad et al., Search for direct third-generation squark pair production in final states with missing transverse momentum and two b-jets in $\sqrt{s} = 8$ TeV pp collisions with the atlas detector. JHEP **1310**, 189 (2013). arXiv:1308.2631
44. ATLAS Collaboration: G. Aad et al., Search for direct top-squark pair production in final states with two leptons in pp collisions at $\sqrt{s} = 8$ TeV with the ATLAS detector. JHEP **1406**, 124 (2014). arXiv:1403.4853
45. ATLAS Collaboration: G. Aad et al., Search for direct pair production of the top squark in all-hadronic final states in proton–proton collisions at $\sqrt{s} = 8$ TeV with the ATLAS detector. JHEP **1409**, 015 (2014). arXiv:1406.1122
46. ATLAS Collaboration: Aad, Georges and others, Search for direct production of charginos, neutralinos and sleptons in final states with two leptons and missing transverse momentum in pp collisions at $\sqrt{s} = 8$ TeV with the ATLAS detector. JHEP **05**, 071 (2014). arXiv:1403.5294
47. ATLAS Collaboration: G. Aad et al., Search for direct production of charginos and neutralinos in events with three leptons and missing transverse momentum in $\sqrt{s} = 8$ TeV pp collisions with the atlas detector. JHEP **1404**, 169 (2014). arXiv:1402.7029
48. ATLAS Collaboration, Search for direct top squark pair production in final states with one isolated lepton, jets, and missing transverse momentum in $\sqrt{s} = 8$ TeV pp collisions using 21fb^{-1} of ATLAS data. ATLAS-CONF-2013-037, 2013
49. ATLAS Collaboration: G. Aad et al., Search for squarks and gluinos with the ATLAS detector in final states with jets and missing transverse momentum using $\sqrt{s} = 8$ TeV proton–proton collision data. JHEP **09**, 176 (2014). arXiv:1405.7875
50. CMS Collaboration: V. Khachatryan et al., Searches for electroweak production of charginos, neutralinos, and sleptons decaying to leptons and W, Z, and Higgs bosons in pp collisions at 8 TeV. Eur. Phys. J. C **74**, 3036 (2014). arXiv:1405.7570
51. CMS Collaboration: V. Khachatryan et al., Search for the production of dark matter in association with top-quark pairs in the single-lepton final state in proton–proton collisions at $\sqrt{s} = 8$ TeV. JHEP **06**, 121 (2015). arXiv:1504.03198
52. CMS Collaboration, Search for the production of dark matter in association with top quark pairs in the di-lepton final state in pp collisions at $\sqrt{s} = 8$ TeV. CMS-PAS-B2G-13-004 (2014)
53. CMS Collaboration, V. Khachatryan et al., Search for dark matter, extra dimensions, and unparticles in monojet events in proton–proton collisions at $\sqrt{s} = 8$ TeV. Eur. Phys. J. C **75**, 235 (2015). arXiv:1408.3583
54. T. Sjöstrand, S. Mrenna, P.Z. Skands, PYTHIA 6.4 physics and manual. JHEP **05**, 026 (2006). arXiv:hep-ph/0603175
55. T. Sjostrand, S. Ask et al., An introduction to PYTHIA 8.2. Comput. Phys. Commun. **191**, 159–177 (2015). arXiv:1410.3012
56. M. Cacciari, G.P. Salam, G. Soyez, FastJet user manual. Eur. Phys. J. C **72**, 1896 (2012). arXiv:1111.6097
57. P. Athron, J.-H. Park, D. Stöckinger, A. Voigt, FlexibleSUSY – a spectrum generator generator for supersymmetric models. Comput. Phys. Commun. **190**, 139–172 (2015). arXiv:1406.2319
58. B.C. Allanach, SOFTSUSY: a program for calculating supersymmetric spectra. Comput. Phys. Commun. **143**, 305–331 (2002). arXiv:hep-ph/0104145
59. P.Z. Skands et al., SUSY Les Houches accord: interfacing SUSY spectrum calculators, decay packages, and event generators. JHEP **07**, 036 (2004). arXiv:hep-ph/0311123
60. B.C. Allanach et al., SUSY Les Houches accord 2. Comput. Phys. Commun. **180**, 8–25 (2009). arXiv:0801.0045
61. https://twiki.cern.ch/twiki/bin/view/AtlasPublic/SupersymmetryPublicResults
62. https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ExoticsPublicResults
63. https://twiki.cern.ch/twiki/bin/view/CMSPublic/PhysicsResultsSUS
64. https://twiki.cern.ch/twiki/bin/view/CMSPublic/PhysicsResultsEXO
65. A. Berlin, D. Hooper, S.D. McDermott, Simplified dark matter models for the galactic center gamma-ray excess. Phys. Rev. D **89**, 115022 (2014). arXiv:1404.0022
66. M.R. Buckley, D. Feld, D. Goncalves, Scalar simplified models for dark matter. Phys. Rev. D **91**, 015017 (2015). arXiv:1410.6497
67. ATLAS Collaboration, Further searches for squarks and gluinos in final states with jets and missing transverse momentum at $\sqrt{s} = 13$ TeV with the ATLAS detector. ATLAS-CONF-2016-078 (2016)
68. CMS Collaboration, Search for supersymmetry in events with jets and missing transverse momentum in proton–proton collisions at 13 TeV. CMS-PAS-SUS-16-014 (2016)
69. ATLAS: M. Aaboud et al., Search for new phenomena in a lepton plus high jet multiplicity final state with the ATLAS experiment using $\sqrt{s} = 13$ Tev proton–proton collision data. arXiv:1704.08493
70. W. Beenakker, R. Hopker, M. Spira, P.M. Zerwas, Squark and gluino production at hadron colliders. Nucl. Phys. B **492**, 51–103 (1997). arXiv:hep-ph/9610490
71. W. Beenakker, R. Hopker, M. Spira, PROSPINO: a program for the production of supersymmetric particles in next-to-leading order QCD. arXiv:hep-ph/9611232
72. W. Beenakker, M. Kramer, T. Plehn, M. Spira, P.M. Zerwas, Stop production at hadron colliders. Nucl. Phys. B **515**, 3–14 (1998). arXiv:hep-ph/9710451
73. M. Beneke, J. Piclum, C. Schwinn, C. Wever, NNLL soft and Coulomb resummation for squark and gluino production at the LHC. JHEP **10**, 054 (2016). arXiv:1607.07574
74. W. Beenakker, C. Borschensky, M. Krämer, A. Kulesza, E. Laenen, NNLL-fast: predictions for coloured supersymmetric particle production at the LHC with threshold and Coulomb resummation. JHEP **12**, 133 (2016). arXiv:1607.07741
75. A. Kulesza, L. Motyka, Threshold resummation for squark–antisquark and gluino-pair production at the LHC. Phys. Rev. Lett. **102**, 111802 (2009). arXiv:0807.2405
76. A. Kulesza, L. Motyka, Soft gluon resummation for the production of gluino–gluino and squark–antisquark pairs at the LHC. Phys. Rev. D **80**, 095004 (2009). arXiv:0905.4749
77. W. Beenakker, S. Brensing et al., Soft-gluon resummation for squark and gluino hadroproduction. JHEP **0912**, 041 (2009). arXiv:0909.4418
78. W. Beenakker, S. Brensing et al., Supersymmetric top and bottom squark production at hadron colliders. JHEP **08**, 098 (2010). arXiv:1006.4771
79. W. Beenakker, S. Brensing et al., Squark and gluino hadroproduction. Int. J. Mod. Phys. A **26**, 2637–2664 (2011). arXiv:1105.1110
80. A.D. Martin, W.J. Stirling, R.S. Thorne, G. Watt, Parton distributions for the LHC. Eur. Phys. J. C **63**, 189–285 (2009). arXiv:0901.0002
81. P.M. Nadolsky, H.-L. Lai et al., Implications of CTEQ global analysis for collider observables. Phys. Rev. D **78**, 013004 (2008). arXiv:0802.0007
82. C.G. Lester, M.A. Parker, M.J. White, Determining SUSY model parameters and masses at the LHC using cross-sections, kinematic edges and other observables. JHEP **01**, 080 (2006). arXiv:hep-ph/0508143
83. B.C. Allanach et al., The Snowmass points and slopes: Benchmarks for SUSY searches. Eur. Phys. J. C **25**, 113–123 (2002). arXiv:hep-ph/0202233
84. J.A. Aguilar-Saavedra et al., Supersymmetry parameter analysis: SPA convention and project. Eur. Phys. J. C **46**, 43–60 (2006). arXiv:hep-ph/0511344

85. M. Cacciari, G.P. Salam, Pileup subtraction using jet areas. Phys. Lett. B **659**, 119–126 (2008). arXiv:0707.1378
86. Calibration of ATLAS b-tagging algorithms in dense jet environments. ATLAS-CONF-2016-001 (2016)
87. A. Buckley, J. Butterworth et al., Rivet user manual. Comput. Phys. Commun. **184**, 2803–2819 (2013). arXiv:1003.0694
88. M. Cacciari, G.P. Salam, G. Soyez, The anti-$k(t)$ jet clustering algorithm. JHEP **0804**, 063 (2008). arXiv:0802.1189
89. Electron efficiency measurements with the ATLAS detector using the 2012 LHC proton–proton collision data. ATLAS-CONF-2014-032 (2014)
90. Electron identification measurements in ATLAS using $\sqrt{s} = 13$ TeV data with 50 ns bunch spacing. ATL-PHYS-PUB-2015-041 (2015)
91. ATLAS Collaboration: G. Aad et al., Performance of missing transverse momentum reconstruction in proton–proton collisions at 7 TeV with ATLAS. Eur. Phys. J. C **72**, 1844 (2012). arXiv:1108.5602
92. J. Conrad, O. Botner, A. Hallgren, C. Pérez de Los Heros, Including systematic uncertainties in confidence interval construction for Poisson statistics. Phys. Rev. D **67**, 012002 (2003). arXiv:hep-ex/0202013
93. P. Scott, J. Conrad et al., Direct constraints on minimal supersymmetry from Fermi-LAT observations of the dwarf galaxy Segue 1. JCAP **1**, 31 (2010). arXiv:0909.3300
94. P. Scott, C. Savage, J. Edsjö, and the IceCube Collaboration: R. Abbasi et al., Use of event-level neutrino telescope data in global fits for theories of new physics. JCAP **11**, 57 (2012). arXiv:1207.0810
95. IceCube Collaboration: M.G. Aartsen et al., Improved limits on dark matter annihilation in the Sun with the 79-string IceCube detector and implications for supersymmetry. JCAP **04**, 022 (2016). arXiv:1601.00653
96. CMS Collaboration, Simplified likelihood for the re-interpretation of public CMS results. CMS-NOTE-2017-001 (2017)
97. CMS Collaboration, Search for new physics in the all-hadronic final state with the MT2 variable. CMS-PAS-SUS-16-036 (2017)
98. P. Gondolo, J. Edsjö et al., DarkSUSY: computing supersymmetric dark matter properties numerically. JCAP **0407**, 008 (2004). arXiv:astro-ph/0406204
99. G.Bélanger, F. Boudjema, A. Pukhov, A. Semenov, micrOMEGAs4.1: two dark matter candidates. Comput. Phys. Commun. **192**, 322–329 (2015). arXiv:1407.6129
100. OPAL Collaboration: G. Abbiendi et al., Search for anomalous production of acoplanar di-lepton events in e^+e^- collisions at $\sqrt{s} = 183$ GeV and 189 GeV. Eur. Phys. J. C **14**, 51–71 (2000). arXiv:hep-ex/9909052
101. S. Dawson, E. Eichten, C. Quigg, Search for supersymmetric particles in hadron–hadron collisions. Phys. Rev. D **31**, 1581 (1985)
102. A. Bartl, H. Fraas, W. Majerotto, Gaugino–Higgsino mixing in selectron and sneutrino pair production. Z. Phys. C **34**, 411 (1987)
103. A. Djouadi, M.M. Mühlleitner, M. Spira, Decays of supersymmetric particles: the program SUSY-HIT (SUspect-SdecaY-Hdecay-InTerface). Acta Phys. Polon. **38**, 635–644 (2007). arXiv:hep-ph/0609292
104. D. Shepard, A two-dimensional interpolation function for irregularly-spaced data. In Proceedings of the 1968 23rd ACM National Conference, ACM '68 (ACM, New York, 1968), pp 517–524
105. A. Bartl, H. Fraas, W. Majerotto, Production and decay of neutralinos in e^+e^- annihilation. Nucl. Phys. B **278**, 1 (1986)
106. A. Bartl, H. Fraas, W. Majerotto, Signatures of chargino production in e^+e^- collisions. Z. Phys. C **30**, 441 (1986)
107. H. Baer, F.E. Paige, S.D. Protopopescu, X. Tata, ISAJET 7.48: a Monte Carlo event generator for pp, anti-p, p, and $e^+ e^-$ reactions. arXiv:hep-ph/0001086
108. DELPHI Collaboration, OPAL Collaboration, ALEPH Collaboration, LEP Working Group for Higgs Boson Searches and L3 Collaboration: S. Schael et al., Search for neutral MSSM Higgs Bosons at LEP. Eur. Phys. J. C **47**, 547–587 (2006). arXiv:hep-ex/0602042
109. J.M. Cline, K. Kainulainen, P. Scott, C. Weniger, Update on scalar singlet dark matter. Phys. Rev. D **88**, 055025 (2013). arXiv:1306.4710
110. Collaboration: P. Athron, C. Balázs et al., Status of the scalar singlet dark matter model. Eur. Phys. J. C **77**, 568 (2017). https://doi.org/10.1140/epjc/s10052-017-5113-1
111. S. Ask et al., From Lagrangians to events: computer tutorial at the MC4BSM-2012 workshop. arXiv:1209.0297