THE EUROPEAN
PHYSICAL JOURNAL C

Special Article - Tools for Experiment and Theory

The GAMBIT Universal Model Machine: from Lagrangians to likelihoods

Sanjay Bloor[1,2,a]**, Tomás E. Gonzalo**[3,4,b]**, Pat Scott**[1,2,c]**, Christopher Chang**[2]**, Are Raklev**[5]**, José Eliel Camargo-Molina**[1,6]**, Anders Kvellestad**[1,5]**, Janina J. Renk**[1,2,7]**, Peter Athron**[4,8]**, Csaba Balázs**[4]

[1] Blackett Laboratory, Department of Physics, Imperial College London, Prince Consort Road, London SW7 2AZ, UK
[2] School of Mathematics and Physics, The University of Queensland, St. Lucia, Brisbane, QLD 4072, Australia
[3] Institute for Theoretical Particle Physics and Cosmology (TTK), RWTH Aachen University, 52056 Aachen, Germany
[4] School of Physics and Astronomy, Monash University, Melbourne, VIC 3800, Australia
[5] Department of Physics, University of Oslo, 0316 Oslo, Norway
[6] Department of Physics and Astronomy, Uppsala University, Box 516, 751 20 Uppsala, Sweden
[7] Oskar Klein Centre for Cosmoparticle Physics, AlbaNova University Centre, 10691 Stockholm, Sweden
[8] Department of Physics and Institute of Theoretical Physics, Nanjing Normal University, Nanjing 210023, Jiangsu, China

Abstract We introduce the GAMBIT Universal Model Machine (GUM), a tool for automatically generating code for the global fitting software framework GAMBIT, based on Lagrangian-level inputs. GUM accepts models written symbolically in FeynRules and SARAH formats, and can use either tool along with MadGraph and CalcHEP to generate GAMBIT model, collider, dark matter, decay and spectrum code, as well as GAMBIT interfaces to corresponding versions of SPheno, micrOMEGAs, Pythia and Vevacious (C++). In this paper we describe the features, methods, usage, pathways, assumptions and current limitations of GUM. We also give a fully worked example, consisting of the addition of a Majorana fermion simplified dark matter model with a scalar mediator to GAMBIT via GUM, and carry out a corresponding fit.

Contents

[a] e-mail: sanjay.bloor12@imperial.ac.uk
[b] e-mail: gonzalo@physik.rwth-aachen.de (corresponding author)
[c] e-mail: pat.scott@uq.edu.au

1 Introduction

The Standard Model (SM) has been exceptionally successful in explaining the fundamental particle physics that underpins the workings of the Universe. Despite these successes, there are still both experimental and theoretical considerations in tension with the SM: the nature of dark matter (DM), the generation of neutrino masses, the hierarchy problem, cosmological matter-antimatter asymmetry, various experimental anomalies (such as flavour or $g - 2$), and the stability of the electroweak vacuum. Clearly, the search for physics beyond the SM (BSM) is a multidisciplinary endeavour.

Performing global fits of BSM theories ensures complementarity between experimental and theoretical efforts. This entails consistently predicting multiple observables, comparing them rigorously with experimental results, and using sophisticated statistical sampling techniques in order to obtain meaningful, quantitative inference on theories and their parameters.

The GAMBIT software framework [1,2] aims to address this need, providing a flexible, efficient and scalable code for BSM phenomenology. GAMBIT provides dedicated modules for statistical sampling [3], DM [4], collider [5] flavour [6] and neutrino [7] physics, as well as spectrum generation, decay and precision physics [8] and cosmology [9].

The first version of GAMBIT shipped with various parameterisations of the minimal supersymmetric Standard Model (MSSM) at the weak [10] and grand unified theory (GUT) scales [11], a scalar singlet DM extension of the SM [12] and an effective field theory of flavour interactions [6,13]. Since then, studies of vacuum stability for scalar singlet DM [14], generic Higgs portal DM models [15], DM effective field theory models [16,17], axions and axion-like particles [18], additional MSSM models [19], both left- [20] and right-handed neutrinos [7], and inflationary and other cosmological models [9] have expanded the GAMBIT model database substantially.

One of the most notable features of GAMBIT is that it is open source. This allows users to study new models and add their own experimental and theoretical constraints to GAMBIT in a modular manner. Although adding a new model to GAMBIT is largely formulaic, it is still not a completely trivial task, as the user still requires some level of understanding of the underlying software design. To this end, we present the GAMBIT Universal Model Machine (GUM): a tool for interfacing symbolic Lagrangian-level tools to GAMBIT, to further automate the procedure of comparing theory to experiment [21].

Not only does automation increase efficiency and effectiveness in BSM physics, it also reduces the scope for human error, which is inevitably introduced when coding complicated expressions by hand. Development of Lagrangian-level tools has been a very important step in the development of automation in BSM physics. The original motivation for creating Lagrangian-level tools was to automatically write outputs that could be used for generating matrix element functions, which could in turn be used by Monte Carlo event generators to simulate new physics at particle colliders. The first tool to achieve this was LanHEP [22–25], originally created to compute vertices for CompHEP [26–28] from a simple Lagrangian input. With the release of FeynRules [29–32], this quickly expanded to generating output for other matrix element codes, such as MadGraph/MadEvent [33–37], CalcHEP [38,39], FeynArts [40–43], SHERPA [44] and WHIZARD/O'Mega [45,46]. SARAH [47–52] was also developed around the same time, initially with a particular focus on supersymmetry, but soon expanding to a much larger range of models.

The success of FeynRules and SARAH in generating Feynman rules for use by matrix element generators lead to the creation of a new filetype, the 'Universal FeynRules Output' (UFO) [53]. These UFO files encode information about the particles, the parameters and interaction vertices for a given model. They can be generated by both FeynRules and SARAH, and handled by a range of matrix element generators such as MadGraph, GoSam [54,55] and Herwig++ [56,57].

As the search for new physics spans more than just collider physics, it has been necessary for Lagrangian-level tools to generate output for tools in other areas of physics, outside of collider phenomenology. The UFO-compatible package MadDM [58–60] has been built on top of MadGraph, for computing DM relic densities and direct and indirect detection signals. From SARAH, inputs can now also be generated for DM phenomenology with micrOMEGAs [61–67], spectrum generation with SPheno [68,69] and FlexibleSUSY [70–72], flavour physics observables with SPheno and FlavorKit [73], and calculations of the stability of electroweak symmetry breaking (EWSB) vacuum with Vevacious [74].

Although FeynRules and SARAH were both created to solve essentially the same problem, they serve different purposes. FeynRules is concerned with computing Feynman rules for *any* given Lagrangian, including effective ones, and performing physics at tree level. SARAH on the other hand places far more emphasis on renormalisable theories. As a result, any UV-complete model can be implemented in both FeynRules and SARAH, and any output generated by FeynRules for such models can also be created by SARAH. However, SARAH is also able to compute renormalisation group equations (RGEs) at 2-loop order and particle self-energies at 1-loop order, allowing its 'downstream benefi-

Table 1 GAMBIT backends with GUM support and Lagrangian-level tools used to generate them. Apart from the external packages listed, GUM also produces GAMBIT Core and physics module code tailored to the model and observables of interest

Generated GAMBIT backends	FeynRules	SARAH	Usage in GAMBIT
CalcHEP	✓	✓	Decays, cross-sections
micrOMEGAs (via CalcHEP)	✓	✓	DM observables
Pythia (via MadGraph)	✓	✓	Collider physics
SPheno	✗	✓	Particle mass spectra, decay widths
Vevacious C++	✗	✓	Vacuum stability

ciaries' SPheno and FlexibleSUSY to generate corrected mass spectra at the 1-loop level.

Although the outputs of SARAH are more sophisticated than those of FeynRules, it also has limitations. Unlike in FeynRules, it is not generally possible to define non-renormalisable theories or higher-dimensional effective theories in SARAH. We therefore provide interfaces to both FeynRules and SARAH to allow the user to incorporate a vast range of theories into GAMBIT, from effective field theories (EFTs) via FeynRules to complex UV-complete theories in SARAH. We stress that if a model *can* be implemented in SARAH, then the user *should* use SARAH over FeynRules – both to use GAMBIT to its full potential, and to perform more detailed physics studies. The basic outputs available from GUM in each case are summarised in Table 1.[1]

This manual is organised as follows: in Sect. 2, we describe the code structure and outputs of GUM. In Sect. 3 we give usage details, including installation, the GUM file, and particulars of FeynRules and SARAH model files. In Sect. 4 we provide a worked example, where we use GUM to add a simplified DM model to GAMBIT, and perform a quick statistical fit to DM observables. Finally, in Sect. 5, we discuss future extensions of GUM and summarise. We include details of the new GAMBIT interfaces to CalcHEP, Vevacious and SARAH-SPheno (the auto-generated version of SPheno created using SARAH) in the Appendix.

GUM is open source and part of the GAMBIT 2.0 release, available from gambit.hepforge.org under the terms of the standard 3-clause BSD license.[2]

2 Code design

GAMBIT consists of a set of Core software components, a sampling module ScannerBit [3], and a series of physics modules [4–9]. Each physics module is in charge of a domain-specific subset of GAMBIT's physical calculations. GUM generates various snippets of code that it then adds to parts of the GAMBIT Core, as well as to some of the physics modules, enabling GAMBIT to employ the capabilities of those modules with the new model.

Within the Core, GUM adds code for any new particles to the GAMBIT particle database, and code for the new model to the GAMBIT models database, informing GAMBIT of the parameters of the new model so that they can be varied in a fit. GUM also generates interfaces (frontends) to the external codes (backends) that it is able to generate. The backends supported by GUM in this manner are those listed as outputs in Table 1.

Within the physics modules, GUM writes new code for the SpecBit [8] module, responsible for spectrum generation within GAMBIT, DecayBit [8], responsible for calculating the decays of particles, DarkBit [4], responsible for DM observables, and ColliderBit [5], the module that simulates hard-scattering, hadronisation and showering of particles at colliders, and implements subsequent LHC analyses.

GUM is primarily written in Python, with the exception of the Mathematica interface, which is written in C++ and accessed via Boost.Python.

Initially, GUM parses a `.gum` input file, using the contents to construct a singleton `gum` object. Details of the input format can be found in Sect. 3.3. GUM then performs some simple sanity and consistency checks, such as ensuring that if the user requests DM observables, they have also specified a DM candidate. GUM then opens an interface to either FeynRules or SARAH via the Wolfram Symbolic Transfer Protocol (WSTP), loads the FeynRules or SARAH model file that the user has requested into the Mathematica kernel, and performs some additional sanity checks using the inbuilt diagnostics of each package.

Once GUM is satisfied with the FeynRules or SARAH model file, it extracts all physical particles, masses and parameters (e.g. mixings and couplings). The minimal infor-

---

[1] Some readers will note the absence of FlexibleSUSY from this list; this is due to the complex C++ templates used in FlexibleSUSY and the fact that supporting it fully as a backend in GAMBIT requires significant development of the classloading abilities of the backend-on-a-stick script (BOSS) [1]. Once this challenge has been overcome, future versions of GUM will also generate code for FlexibleSUSY and its other flexi-brethren.

[2] http://opensource.org/licenses/BSD-3-Clause.

Table 2 Details of the new files that GUM writes or modifies in each part of GAMBIT, for the .gum file entry model:*new_model*. Some files are only written or edited when a model-specific version of a particular backend is requested in the .gum file. This is achieved by including the corresponding option *backend_name*:**true**, e.g. spheno:**true**, pythia:**true**, etc. Entries in the rightmost column indicate where this is the case, i.e. for rows with an entry present in the rightmost column, the file listed in that row will only be modified/generated if the backend(s) named in the rightmost column are requested in the .gum

file. Where no such entries exist in the right column, the addition or modification of the GAMBIT source is always performed, regardless of the contents of the .gum file. The entry "wimp_candidate" indicates that the output is only generated if the option of the same name is set in the .gum file, as wimp_candidate:*pdg*. Note that the entry "decaying_dm_candidate:*pdg*" can be used in place of wimp_candidate if the DM candidate decays instead of annihilates. All filenames containing *new_model* are newly created by GUM; all others are existing files that GUM amends

GAMBIT component	Entries and/or amendments	Required .gum entry or backend(s)
Models	src/SpectrumContents/*new_model*.cpp	
	include/gambit/Models/models/*new_model*.hpp	
	include/gambit/Models/SimpleSpectra/*new_model*_SimpleSpec.hpp	
SpecBit	src/SpecBit_*new_model*.cpp	
	include/gambit/SpecBit/SpecBit_*new_model*_rollcall.hpp	
	src/SpecBit_VS.cpp	Vevacious and SPheno
	include/gambit/SpecBit/SpecBit_VS_rollcall.hpp	Vevacious and SPheno
DarkBit	src/*new_model*.cpp: Dark Matter ID	wimp_candidate
	src/*new_model*.cpp: Process Catalogue	wimp_candidate and CalcHEP
	include/gambit/DarkBit/DarkBit_rollcall.hpp: Direct detection	wimp_candidate and micrOMEGAs
DecayBit	src/DecayBit.cpp	CalcHEP or SPheno
	include/gambit/DecayBit/DecayBit_rollcall.cpp	CalcHEP or SPheno
ColliderBit	src/models/*new_model*.cpp	Pythia
	include/gambit/ColliderBit/models/*new_model*.hpp	Pythia
	src/ColliderBit_Higgs.cpp	SPheno
	include/gambit/ColliderBit/ColliderBit_Higgs_rollcall.hpp	SPheno
Backends	src/frontends/CalcHEP_3_6_27.cpp	CalcHEP
	src/frontends/MicrOmegas_*new_model*_3_6_9_2.cpp	micrOMEGAs
	src/frontends/SARAHSPheno_*new_model*_4_0_3.cpp	SPheno

mation required to define a new particle is its mass, spin, color representation, PDG code, and electric charge (if non-self conjugate). For a parameter to be extracted, it must have an associated LHA block in the FeynRules or SARAH model file, and an index within that block. Additionally for FeynRules files, the interaction order used in UFO files must be set. For details on the syntax required for all required particle and parameter definitions, see Sect. 3.4.3 for FeynRules model files, and Sect. 3.5.3 for SARAH model files.

After extracting particle and parameter information, GUM cross-checks that all particles in the new model exist in the GAMBIT particle database, and adds entries if they do not. GUM uses this same particle and parameter information to also write new entries in both the GAMBIT model database and the SpecBit module. All other calculations rely on a combination of new code within GAMBIT and backends. In the following sections we provide details of the new code generated by GUM in the GAMBIT model database (Sect. 2.1), within GAMBIT physics modules (Sect. 2.2), and in the form of new backends and their corresponding frontend interfaces in GAMBIT (Sect. 2.3).

Many of the GAMBIT code outputs are only generated if the user elects to generate relevant new backend codes with GUM. Details of which backends must be generated with GUM for it to generate different GAMBIT source files can be found in Table 2.

At the end of its run, GUM outputs to screen a set of suggested commands for reconfiguring and rebuilding GAMBIT and the new backends. It also emits an example GAMBIT input file for running a scan of the new model (yaml_files/*new_model*_example.yaml, where *new_model* is the name of the new model).

2.1 The GAMBIT model database

For every new model requested, GUM adds a new entry to GAMBIT's hierarchical model database. GUM operates under the condition that no model of the same name exists already in the hierarchy, and there are no entries for it in either Models, SpecBit or DarkBit, and will throw an error if it does. If the requested model is a new model, GUM creates a new model file *new_model*.hpp in the Models directory (see

Table 2), with the parameters extracted from FeynRules or SARAH.

In addition to the model file, GUM creates a list of expected contents for the model's particle spectrum object in `SpectrumContents/`*new_model*`.cpp`. This includes not just pole masses of BSM particles, but also the parameters of the model itself, mixing matrices and various SM parameters. GUM also writes a corresponding simple container for the spectrum *new_model*`_SimpleSpec` that defines functions for accessing the spectrum contents and exposes them to the GAMBIT `Spectrum` class.

2.2 Modules

2.2.1 SpecBit

SpecBit includes structures for storing and passing around the so-called mass spectrum, i.e. the pole masses, mixings and Lagrangian parameters of the model. If SPheno is used to obtain the spectrum then the R_ξ-gauge will be set to $\xi = 1$ and the Lagrangian parameters given in the $\overline{MS}$ scheme for non-superymmetric models and the $\overline{DR}'$ scheme for supersymmetric models. The scale at which these parameters are given will depend on the spectrum generator and will also be stored in the spectrum stored in SpecBit.

Following the structure of the simple spectrum container, GUM writes module functions in SpecBit that allow the construction of an object of the `Spectrum` class. The capability *new_model*`_spectrum` and its module functions are declared in the header file `SpecBit_`*new_model*`_rollcall.hpp` and defined in `SpecBit_`*new_model*`.cpp`. The spectrum is either defined directly in terms of phenomenological model parameters, or generated from the Lagrangian parameters using SPheno.

By default, in the absence of a spectrum generator, GUM writes a simple module function in SpecBit, `get_`*new_model*`_spectrum`, that fills a `Spectrum` object with SM values and the input parameters of the model. If using SARAH to generate GAMBIT code, the pole masses of BSM particles are computed using the tree-level relations provided by SARAH. However, these tree-level masses from SARAH are only used for very simple models, such as those without additional Higgs states, as more complicated models include non-trivial mixings and modify the electroweak symmetry breaking (EWSB) conditions. In the latter case a spectrum generator (e.g. SPheno) should be used. When producing GAMBIT output from FeynRules, however, there are no such relations available, and thus the particle masses are model parameters and the `Spectrum` object is filled with those.

If the SPheno output is requested from SARAH for a model, GUM writes a module function, `get_`*new_model*`_spectrum_SPheno`, with the backend requirements necessary to generate the full spectrum, with all particle masses, mixing matrices, etc. Hence, for improved precision spectra, it is recommended that the user implement their model using SARAH, and request the spectrum to be provided by SPheno.

If Vevacious output is requested, for each new BSM model GUM writes new model-specific code in the SpecBit vacuum stability file, `SpecBit/src/SpecBit_VS.cpp`, and adds appropriate entries to the corresponding rollcall header. GUM provides two new module functions to interact with Vevacious. Firstly, `prepare_pass_`*new_model*`_spectrum_to_vevacious` with capability `pass_spectrum_to_vevacious`, which interfaces the `Spectrum` object to the Vevacious object. Secondly, `vevacious_file_location_`*new_model*, which directs GAMBIT to the location of the input Vevacious files generated by SARAH.

2.2.2 DecayBit

Whenever decay information is requested for a new model, GUM amends the header `DecayBit_rollcall.hpp` and source `DecayBit.cpp` files to add the decays of the particles in the model. The information for the decays can be provided separately by two backends in the GUM pipeline: CalcHEP and SPheno.

CalcHEP generates tree-level decays for each new BSM particle, plus new contributions to any existing particles in DecayBit such as the SM Higgs and the top quark. GUM adds these to DecayBit by adding the new decay channels wherever possible to any existing `DecayTable::Entry` provided by a module function with capability *particle_name*`_decay_rates`. If no such function exists, it instead creates a new module function `CH_`*new_model_particle_name*`_decays`, with this capability-type signature, where *particle_name* comes from the SARAH/FeynRules model file. GUM then modifies the module function `all_decays` to add the decays of any particles for which it has written new module functions. GUM presently only supports the generation of $1 \rightarrow 2$ decays, as support for 3-body decays within DecayBit is currently limited to fully integrated partial widths; consistent inclusion of integrated partial widths provided by SPheno and differential decay widths provided by CalcHEP into DecayBit will require further development of the `DecayTable` infrastructure, and a careful treatment of integration limits. Note also that GUM does not currently write any BSM contribution for *W* and *Z* boson decay via the CalcHEP interface, but this is planned for future releases.

SPheno by default computes leading order decays (tree-level or one-loop) for all BSM and SM particles, and adds universal higher-order corrections for specific decays, which are very important for Higgs decays. In addition it provides an alternative computation of full one-loop decays for all particles. The choice of method is left to the user via the SPheno

option `OneLoopDecays`. As SPheno provides decay widths for all particles in the spectrum, GUM creates a new module function `all_new_model_decays_from_SPheno`, which returns a `DecayTable` filled with all decay information computed by SPheno.

The default behaviour of GUM is to ensure that it always generates decay code of some sort when needed. This ensures that a complete `DecayTable` for the new model can be provided within GAMBIT for dependency resolution, by providing the capability `all_decays` in DecayBit. A viable GAMBIT `DecayTable` is required for the functioning of many external codes such as Pythia and micrOMEGAs: if any new particle is a mediator in a process of interest, then its width is needed. To this end, GUM activates CalcHEP output automatically if decays are needed by other outputs that have been activated in the `.gum` file, but neither CalcHEP nor SPheno output has been explicitly requested.

2.2.3 DarkBit

If the user specifies a Weakly-Interacting Massive Particle (WIMP) DM candidate for a model[3], GUM writes the relevant code in DarkBit. Each individual model tree in DarkBit has its own source file, so GUM generates a new source file `src/new_model.cpp`, and amends the DarkBit rollcall header accordingly. At a minimum, GUM includes a new module function `DarkMatter_ID_new_model` in this file. It then adds the remainder of the source code according to which backends the user selects to output code for in their `.gum` file; currently available options are CalcHEP and micrOMEGAs.

If the user requests CalcHEP output, then a new module `TH_ProcessCatalog_new_model` providing the Process Catalogue is written. The Process Catalogue houses all information about annihilation and decay of DM, and decays of all other particles in the spectrum. All computations of indirect detection and relic density likelihoods in DarkBit begin with the Process Catalogue. For details, see Sec. 6.3 of [4]. All processes that GUM adds to the Process Catalogue are $2 \rightarrow 2$ processes computed at tree level by CalcHEP.

The Process Catalogue is used to compute the relic abundance of annihilating DM via the DarkBit interface to the Boltzmann solver in DarkSUSY. The Process Catalogue interface does not currently fully support co-annihilations nor 3-body final states, so GUM does not generate these processes. Such functionality is planned for future releases of DarkBit, and will be supported by GUM at that time. Note however that GUM *does* support the full generation of micrOMEGAs versions including these effects via the CalcHEP-micrOMEGAs interface. If co-annihilations or three-body final states are expected to be important in a new physics model, the user should therefore use micrOMEGAs to compute the relic abundance from DarkBit, in preference to the DarkSUSY Boltzmann solver.

For decaying DM candidates, the user would need to implement their own relic density calculation, as appropriate for the specific model in question. Although DarkBit can calculate spectral yields from DM decay, at present the likelihoods for indirect detection in DarkBit do not support decaying dark matter (in large part because neither gamLike nor nulike presently support decaying DM). Existing direct detection likelihoods can still be used out of the box without any relic density calculation, if the user assumes that the decaying DM candidate constitutes all of the DM.

When writing micrOMEGAs output, GUM adds new entries to the `ALLOW_MODELS` macro for existing micrOMEGAs functions in DarkBit. To use micrOMEGAs' relic density calculator, GUM adds an entry to the module function `RD_oh2_Xf_MicrOmegas`. For more information, see Sec. 4.2 of Ref. [4]. GUM also provides an interface to the module function `DD_couplings_MicrOmegas`, which returns a `DM_nucleon_couplings` object containing the basic effective spin-independent and spin-dependent couplings to neutrons and protons G_{SI}^p, G_{SI}^n, G_{SD}^n, and G_{SD}^n. This object is readily fed to DDCalc for computing likelihoods from direct detection experiments. For more information, see Sec. 5 of Ref. [4].

For more complicated models where the standard spin-independent and spin-dependent cross-sections are not sufficient, micrOMEGAs is not able to compute relevant couplings. In this case, the user should perform a more rigorous calculation of WIMP-nucleon (or WIMP-nucleus) couplings by alternative means. This is required when, for example, scattering cross-sections rely on the momentum exchange between incoming DM and SM particles, or their relative velocity. The full set of 18 non-relativistic EFT (NREFT) DM-nucleon operators are defined in both DarkBit and DDCalc 2, and described in full in the Appendix of Ref. [15]. These operators fully take into account velocity and momentum transfer up to second order, and should typically be used in cases where the entirety of the physics is not captured by just σ_{SI} and σ_{SD}. Whilst the new 2.1 version of GAMBIT [16] allows for automated translation of high-scale relativistic effective DM-parton couplings to low-scale NREFT couplings via an interface to DirectDM [75,76], there is no established automated matching procedure for connecting other high-scale models (as defined in FeynRules or SARAH model files) to the Wilson coefficients of the relativistic EFT. GUM therefore does not automatically write any module functions connecting the GAMBIT `Spectrum` object to the NREFT interface of DDCalc; once such a procedure exists, GUM will be extended accordingly.

---

[3] We note that it is not currently possible to declare more than one DM candidate, and GUM will fail if attempted. Additionally, we do not recommend specifying a single DM candidate for models with multicomponent DM, as this could lead to inconsistent results.

2.2.4 ColliderBit

In ColliderBit, simulations of hard-scattering collisions of particles are performed using the Monte Carlo event generator Pythia [77]. These events are passed through detector simulation and then fed into the GAMBIT analysis pipeline, which predicts the signal yields for the new model. These can then be compared to the results of experimental searches for new particles.

For a new BSM model, the matrix elements for new processes unique to the model must be inserted into Pythia in order for it to be able to draw Monte Carlo events from the differential cross-sections of the model. To achieve this, GUM communicates with MadGraph to generate matrix element code for Pythia, and writes the appropriate patch to insert it into Pythia. Alongside the matrix elements, this Pythia patch also inserts any newly defined LHA blocks.

When Pythia output is requested, GUM writes a series of new ColliderBit module functions in the source file `ColliderBit/src/models/`*`new_model`*`.cpp`, and a corresponding rollcall header file. The new functions give ColliderBit the ability to

(i) collect the relevant `Spectrum` and `DecayTable` objects from other modules and provide them to the newly-generated copy of Pythia (capability `SpectrumAndDecaysForPythia`, function `getSpectrumAndDecaysForPythia_`*`new_model`*),

(ii) initialise the new Pythia for Monte Carlo events (capability `HardScatteringSim`, function `getPythia_`*`new_model`*), and

(iii) call the new Pythia in order to generate a Monte Carlo event (capability `HardScatteringEvent`, function `generateEventPythia_`*`new_model`*).

In addition to the likelihood from LHC new particle searches, ColliderBit also provides likelihoods associated with Higgs physics. This is done via interfaces to HiggsBounds [78–81] and HiggsSignals [82,83], which use information on Higgs signal rates and masses from the Tevatron, LEP and the LHC. When a new model is added to GAMBIT with GUM, if SPheno output is requested from SARAH, GUM constructs a new `HiggsCouplingsTable` used as input to the Higgs likelihoods, and amends the appropriate module function entries in `ColliderBit_Higgs_rollcall.hpp` and `ColliderBit_Higgs.cpp`.

2.3 Backends

In the GAMBIT framework, backends are external tools that GAMBIT links to dynamically at runtime, in order to compute various physical observables and likelihoods. Out of the full list of backends that can be interfaced with GAMBIT, a small selection of them can work for generic BSM models. In particular, GUM is able to produce output for SPheno [68,69], Vevacious [74], CalcHEP [39], micrOMEGAs [67], Pythia [77], HiggsBounds [78–81] and HiggsSignals [82,83]. Thus, we briefly describe here the specific outputs generated by GUM for each of these backends, along with and any corresponding GUM and GAMBIT YAML input file entries needed to use them. Unless otherwise stated, GUM has been developed to work with specific versions of the backends.

2.3.1 (SARAH-)SPheno 4.0.3

Required `.gum` **file entry:** `spheno:`**`true`**

SPheno is a spectrum generator capable of computing one-loop masses and tree-level decay branching fractions in a variety of BSM models. The model-specific code is generated by SARAH and combined with the out-of-the-box SPheno code into a single backend for GAMBIT. For each model GUM thus provides an interface between GAMBIT and the SPheno version via a new frontend, `SARAHSPheno_`*`new_model`*`_4_0_3.cpp`. Details about this interface, which differs significantly from the SPheno interface described in [8], can be found in Appendix B.2.

In order to generate SPheno output from SARAH, the user must provide a `SPheno.m` file in the same directory as the SARAH model files. For details of the contents of these files, we refer the reader to the SARAH manual [52].

Once the appropriate GAMBIT code is generated by GUM, the new capability *new_model*_spectrum is added to SpecBit to compute the spectrum using SPheno. The new SPheno generated `Spectrum` object can be obtained for a specific model in a run via the YAML entry in the GAMBIT input file:

```
ObsLikes:
- purpose: Observable
  capability: new_model_spectrum
```

As usual, if more than one module function can provide the same capability, as can happen, for example, if FlexibleSUSY is also present, the SPheno specific one can be selected by the rule in the GAMBIT input file.

```
Rules:
- capability: new_model_spectrum
  function: get_new_model_spectrum_SPheno
```

In addition to their masses and mixings, SPheno can compute the tree-level decay branching fractions for all particles in the spectrum, including some radiative corrections to the decays of Higgs bosons. The GUM-generated code in DecayBit includes the new module function `all_`*`new_model`*`_decays_from_SPheno`, which returns a GAMBIT `DecayTable` as computed by SPheno. This provides an alternative to the usual `all_decays` function in

DecayBit. When run with default settings, GAMBIT will preferentially select `all_`*`new_model`*`_decays_from_SPheno` whenever possible, rather than `all_decays`, as the former is more model-specific. If necessary, the user can also manually instruct GAMBIT to use this function by specifying a rule for the `decay_rates` capability:

```
Rules:
 - capability: decay_rates
   function: all_new_model_decays_from_SPheno
```

To build a newly-added SARAH-generated SPheno ("SARAH-SPheno") within GAMBIT, the appropriate commands to run in the GAMBIT build directory are

```
cmake ..
make sarah-spheno_new_model
```

where the backend name `sarah-spheno` is used to differentiate the corresponding code from the out-of-the-box version of SPheno.

2.3.2 *Vevacious 1.0 (C++)*

Required `.gum` **entries:** `vevacious:`**`true`**, `spheno:`**`true`**. Vevacious computes the stability of the scalar potential for generic extended scalar sectors [74]. A recent C++ version [84] has recently been interfaced to GAMBIT as a backend [85] and is the one used by GUM. The GAMBIT interface to Vevacious is explained in more detail in Appendix B.3.

To test the stability of the EWSB vacuum, Vevacious checks whether other deeper minima exist, in which case it computes the tunnelling probability to either the nearest (to the EWSB vacuum) or the deepest of such minima. The user can select whether to compute the tunnelling probability for either of them by using the `sub_capabilities` options in the GAMBIT input entry for the capability `VS_likelihood` as

```
ObsLikes:
 - purpose: LogLike
   capability: VS_likelihood
   sub_capabilities:
     - global
     - nearest
```

If both minima are chosen, Vevacious computes the probability of tunnelling to both if they are different. The capability `compare_panic_vacua` in GAMBIT checks if the nearest minimum is also the global minimum. If that is the case, the transition to the minimum is only computed once, which reduces the computation time significantly. In many instances, such as for the MSSM, the tunneling path optimization within Vevacious can be a very time-consuming step, therefore computing the tunnelling probability to both minima is not always recommended, as it requires running Vevacious twice for parameter points where the minima are

different, and thus can be prohibitively slow for very large scalar sectors.

For each minimum, Vevacious by default computes both the zero-temperature (*quantum*) tunnelling probability as well as the finite-temperature (*thermal fluctuation*) probability. To select `quantum` or `thermal`, it is possible to provide options to the `sub_capabilities` as

```
ObsLikes:
 - purpose: LogLike
   capability: VS_likelihood
   sub_capabilities:
     global: [quantum]
```

If both minima are selected, the same tunnelling strategy must be selected for both. GAMBIT computes the likelihood by combining the decay widths for all (independent) transitions as reported by Vevacious.

For each new model, SARAH generates the model files required by Vevacious and GUM moves them into the patch directory in GAMBIT. Note that if the user wishes to request Vevacious output, they must also request SPheno output (and provide a `SPheno.m` file). This is due to the GUM interface utilising Mathematica symbols provided by SARAH's SPheno routines.

Once a new model has been generated by GUM, Vevacious can be built from within the GAMBIT build directory with the command

```
make vevacious
```

which will either download and build Vevacious if it is not installed, or simply move the new model files from the GAMBIT patch directory to the Vevacious directory if it is already built. Note that building Vevacious for the first time will also download and install MINUIT [86], PHC [87], and HOM4PS2 [88].

2.3.3 *CalcHEP 3.6.27*

Required `.gum` **file entry:** `calchep:`**`true`**
Optional: `wimp_candidate:`*`pdg`* for annihilating DM, or
     `decaying_dm_candidate:`*`pdg`* for decaying DM
GUM uses the backend convenience function `CH_Decay_Width` provided by the new CalcHEP frontend (described in Appendix B.1), to compute tree-level decay widths.

For each new BSM decay, GUM generates a model-specific `DecayTable::Entry`. For each newly-added decaying particle, GUM writes a module function `CH_`*`new_model_particle_name`*`_decays`, which requires the ability to call the backend convenience function `CH_Decay_Width`. All new decays are then gathered up by the existing DecayBit function `all_decays`, which GUM modifies by adding an `if(ModelInUse(`*`new_model`*`))` switch for newly-added decaying particles in the new model.

The appropriate GAMBIT input rule for a CalcHEP-generated `DecayTable` is simply

```
Rules:
 # Use DecayBit (and CalcHEP), for decay
 # rates; not an SLHA file or SPheno
 - capability: decay_rates
   function: all_decays
```

If the user specifies the PDG code of a WIMP candidate via `wimp_candidate:pdg` or `decaying_dm_candidate:pdg`, then GUM creates a DarkBit entry for the new model.

In the case of `wimp_candidate:pdg`, GUM utilises the backend convenience function `CH_Sigma_V` provided by the CalcHEP frontend, to build the Process Catalogue. It does this by computing $2 \rightarrow 2$ scattering rates as a function of the relative velocity $v_{\rm rel}$, which are in turn fed to the appropriate module functions.

In the case of `decaying_dm_candidate:pdg`, GUM instead utilises the branching ratios from the `DecayTable::Entry` for the decaying DM candidate, which can be constructed from CalcHEP, as described above.

The information contained within the Process Catalogue can be used by the GAMBIT native relic density solver for annihilating DM (using the function `RD_oh2_general` via DarkSUSY), and for all indirect detection rates. For annihilating DM, these utilise the velocity weighted annihilation cross-section $\sigma v_{\rm rel}$. This is usually evaluated at $v_{\rm rel} = 0$ (such as in the case of γ rays), but for solar neutrinos, $v_{\rm rel}$ is set to the solar temperature $T_{\rm Sun}$. For decaying DM, indirect detection rates are explicitly disabled until backend support for decaying DM becomes available.

In the first release of GUM, there is no support for 4-fermion interactions in CalcHEP, as neither FeynRules nor SARAH is able to produce output for these.

From the GAMBIT build directory the command

```
make calchep
```

will build CalcHEP if it is not installed; otherwise it will move the new CalcHEP model files from the GAMBIT patch directory to the CalcHEP model directory.

2.3.4 *microOMEGAs 3.6.9.2*

Required `.gum` **file entry:** `micromegas:`**true**
Optional: `wimp_candidate:pdg` for annihilating DM, or `decaying_dm_candidate:pdg` for decaying DM

MicrOMEGAs is a code capable of computing various DM observables for BSM models with WIMP candidates, such as the relic abundance, direct detection cross-sections, and indirect detection observables. Each microOMEGAs installation in GAMBIT is a separate backend, as the source is compiled directly with the model files. Therefore for each newly-added microOMEGAs model, GUM creates a new

single backend for microOMEGAs, via the new frontend `MicrOmegas_new_model_3_6_9_2.cpp`.

MicrOMEGAs uses CalcHEP files as input so is subject to the same caveats as CalcHEP, covered above. MicrOMEGAs assumes that there is an additional symmetry under which the SM is even and any dark matter candidate is odd. MicrOMEGAs distinguishes an odd particle by having its name begin with a tilde, such as `~chi`. If no particle name in a theory begins with a tilde, the microOMEGAs routines will fail.[4] The particle name is set by the `ParameterName` option in FeynRules, and the `OutputName` option in SARAH. If the indicated DM candidate does not have a particle name beginning with a tilde, GUM throws an error.

GUM provides a simple interface to the relic density calculation in microOMEGAs, in the case of annihilating DM. The GAMBIT input entry for computing the relic density with microOMEGAs is:

```
Rules:
 - capability: RD_oh2
   function: RD_oh2_MicrOmegas
```

If the user provides a decaying DM candidate, then GUM does not provide an interface to the microOMEGAs relic density routines. If the user wishes to compute the relic density for a decaying DM candidate, they must implement their own relic density calculation by hand.

GUM also provides a simple interface to the direct detection routines in microOMEGAs, which simply provide calculations of the spin-independent and spin-dependent cross-sections, which are added for both annihilating and decaying DM. This is fed to DDCalc [4,15] which computes expected rates for a wide range of direct detection experiments.

As each installation of microOMEGAs is a separate backend, each requires a specific build command to be run in the GAMBIT build directory:

```
make micromegas_new_model
```

Future versions of GAMBIT and GUM will interface to microOMEGAs 5 [89], which contains routines for computing the relic abundance of DM via freeze-in, and allows for two-component DM.

2.3.5 *Pythia 8.212*

Required `.gum` **entries:** `pythia:`**true**, `collider_processes:[...]`.

If the user requests Pythia output for a given model, either FeynRules or SARAH generates a collection of UFO files. GUM calls MadGraph directly using the UFO model files, and generates new output for Pythia in the form of matrix

---

[4] If the DM particle is not self-conjugate, its antiparticle should also begin with a tilde.

elements. GUM then writes the appropriate entries in the backend patch system and connects the new matrix elements with those existing in Pythia, and adds the corresponding entry to the file `default_bossed_versions.hpp`. Because Pythia is a C++ code, and the `Pythia` class it defines is used directly in ColliderBit, new versions of the backend must be processed for automated classloading from the corresponding shared library by BOSS (the backend-on-a-stick script [1]). This process generates new header files that must be `#include`(d) in GAMBIT itself, and therefore picked up by CMake. Therefore, a new version of Pythia is correctly built by running the commands

```
cmake ..
make pythia_new_model
cmake ..
make -jn gambit
```

in the GAMBIT build directory, where n specifies the number of processes to use when building. In the current version of ColliderBit, functions from nulike [90,91] are also required in order to perform inline marginalisation over systematic errors in the likelihood calculation; this can be built with

```
make -jn nulike
```

also in the GAMBIT build directory.

The user must provide a list of all processes to include in the new version of Pythia in the `.gum` file under the heading `collider_processes`; see Sect. 3.3 for details.

Once a new Pythia has been created, it has access to all implemented LHC searches within ColliderBit. The relevant GAMBIT input file entry to include Pythia simulations is

```
ObsLikes:
 # LHC likelihood from Pythia
 - purpose: LogLike
   capability: LHC_Combined_LogLike
```

along with rules specifying how to resolve the corresponding dependencies and backend requirements, e.g.

```
Rules:
 # Choose LHC likelihood form (assume normal
↪or
 # log-normal distribution for systematics)
 - capability: LHC_LogLikes
   backends:
   - {capability:
↪lnlike_marg_poisson_gaussian_error}

 # Choose to get cross-sections by Monte Carlo
 - capability: TotalCrossSection
   function: getEvGenCrossSection_as_base

 # Just use unweighted cross-sections
 - capability: EventWeighterFunction
   function: setEventWeight_unity

 # Select where to import model-specific
↪decays
```

```
 # from - CalcHEP in this instance
 # (Alternatively:
↪all_new_model_decays_from_SPheno)
 - capability: decay_rates
   function: all_decays
```

The matrix elements generated by MadGraph can include extra hard partons in the final state so that jet-parton matching is needed to avoid double counting between the matrix elements and the parton shower in Pythia. Currently, this is not automatic in GUM and must be implemented by the user as needed.

Traditional MLM matching, e.g. as found in MadGraph [36], applies a k_T jet measure cut on partons (`xqcut`) at the matrix element level, and separates events with different multiplicities. The optimal value of this cut should be related to the hard scale of the process, e.g. the mass of the produced particles, and tuned to ensure smoothness of differential jet distributions and invariance of the cross-section.

The hard scattering events are then showered and a jet finding algorithm (the k_T-algorithm [92,93] implemented in Pythia's `SlowJet` class in our case) is used on the final state partons to match the resulting jets to the original partons from the hard scatter. A jet is considered to be matched to the closest parton if the jet measure k_T (parton, jet) is smaller than a cutoff `qCut`. This parton shower cut, `qCut`, should be set slightly above `xqcut`. The event is rejected unless each jet is matched to a parton, except for the highest multiplicity sample, where extra jets are allowed below the k_T scale of the softest matrix element parton in the event.

While a full implementation of this matching procedure for use in GUM is still in development, to perform simple jet matching for models generated with GUM, the user can make use of the MLM matching machinery already present in Pythia. This can be accessed by the ColliderBit initialisation of Pythia to allow the relevant jet matching inputs to be passed through Pythia settings in the GAMBIT YAML file, for example:

```
Rules:
 - capability: HardScatteringSim
   type: Py8Collider_new_model_defaultversion
   function: getPythia_new_model
   options:
     LHC_13TeV:
       xsec_veto: 0.028
       pythia_settings:
         # Specify MLM matching method
         - JetMatching:merge = on
         - JetMatching:scheme = 1
         - JetMatching:setMad = off
         - JetMatching:jetAlgorithm = 2
         - JetMatching:slowJetPower = 1

         # Jet finding properties
         - JetMatching:coneRadius = 1.0
         - JetMatching:etaJetMax = 5.0
```

```
# Only light flavours in matching
- JetMatching:nQmatch = 4

# Maximum number of jets
# as defined in the Matrix Elements
- JetMatching:nJetMax = 1

# Minimal kT for a PYTHIA jet
- JetMatching:qCut = 30.0

# Phase space cut to approximate
# Matrix Element cuts
- PhaseSpace:pTHatMin = 20.0
```

Using the above approach one can approximate the matching for a single extra hard parton in dark matter pair production, by applying the matching cuts only on the Pythia side, on events from the matrix elements generated by MadGraph. Here, we also apply a p_T cut on the partons in the hard scatter in Pythia (`pTHatMin`). While this p_T cut and the `xqcut` are not equivalent, the difference is small for single-jet events because the geometrical part of the k_T jet measure becomes unimportant.

A side-effect of using `pTHatMin` is that it applies to all final state particles, and so it must be set well below any $\not{E}_T$ cuts in the analysis. Potential poor initial cross-section estimates in Pythia may lower the phase space selection efficiency, inflating computation time. To combat this, `pTHatMin` may be raised, requiring a sensible balance between efficiency and analysis cut constraints. Due to the limitations of this approach, the accuracy of jet matching cannot be guaranteed, and should be confirmed on a per-model, per-analysis basis.

We refer the reader to the ColliderBit manual [5] for additional details on the Pythia options used within the GAMBIT YAML file.

The particle numbering scheme used by both GAMBIT and Pythia is that of the PDG. For dark matter particles to be correctly recognised as invisible by both libraries, their PDG codes must be within the range 51–60. Other particles that Pythia and GAMBIT tag as invisible are the SM neutrinos, neutralinos, sneutrinos, and the gravitino. Where possible, all particles in SARAH and FeynRules files passed to GUM by the user should adhere to the PDG numbering scheme. For more details, see Sec. 43 of the PDG review [94].

GUM checks that any newly-added particle in the GAMBIT particle database is consistent with the definition in Pythia. If there is an inconsistency between the two, GUM will throw an error. For example, the PDG code 51 is not filled in the GAMBIT particle database by default, but is reserved for scalar DM in Pythia. GUM will throw an error if the user attempts to add a new particle with PDG code 51 but with spin 1/2.

2.3.6 HiggsBounds 4.3.1 *and* HiggsSignals 1.4.0

Required `.gum` **file entry:** `spheno:`**`true`**

Another advantage of using SPheno to compute decays is that all relevant couplings for HiggsBounds and HiggsSignals are automatically computed. Whenever SPheno output is generated, GUM also generates an interface to the GAMBIT implementations of HiggsBounds and HiggsSignals via the GAMBIT type `HiggsCouplingsTable`.

GUM achieves this by generating a function that produces an instance of the GAMBIT native type `HiggsCouplingsTable` from the decay output of SPheno. The `HiggsCouplingsTable` object provides all decays of neutral and charged Higgses, SM-normalised effective couplings to SM final states, branching ratios to invisible final states, and top decays into light Higgses. For more details, we refer the reader to the SpecBit manual [8].

GAMBIT categorises models into two types: 'SM-like' refers to models with only the SM Higgs plus other particles, and 'MSSM-like' refers to models with extended Higgs sectors. The appropriate type is automatically selected for each model by the GAMBIT dependency resolver, by activating the relevant one of the module functions in ColliderBit that can provide capability `HB_ModelParameters`.

For 'SM-like' models, GUM edits the ColliderBit module function `SMLikeHiggs_ModelParameters` to simply pass details of the single Higgs boson from the `Spectrum` object of the new model. For 'MSSM-like' models, GUM edits the ColliderBit function `MSSMLikeHiggs_ModelParameters`, which communicates the properties of all neutral and charged Higgses to HiggsBounds/HiggsSignals in order to deal with extended Higgs sectors.

To ensure the interface to the `HiggsCouplingsTable` works as expected, the user should make sure that the PDG codes of their Higgs sector mimics those of both GAMBIT and the SLHA:

```
# CP-even neutral Higgses
h0: [25, 35, 45]
# CP-odd neutral Higgses
A0: [36, 46]
# Charged Higgs
H+: 37
```

The `MSSMLikeHiggs_ModelParameters` function automatically supports all MSSM and NMSSM models within GAMBIT, as well as any model with a similar Higgs sector (e.g. a Two-Higgs Doublet Model or any subset of the NMSSM Higgs sector). If the user has extended Higgs sectors beyond this, i.e. with more Higgses than the NMSSM, then they will need to extend both GUM and GAMBIT manually.

On the GAMBIT side, if the Higgs sector has multiple charged Higgses, more than three CP-even or more than two CP-odd neutral Higgses, the user must write a new function in `ColliderBit/src/ColliderBit_Higgs.cpp` to construct the `HiggsCouplingsTable` correctly. If there are new CP-even Higgses, this will also require a new entry in `Elements/src/smlike_higgs.cpp` to determine the 'most SM-like' Higgs.

In GUM, the user must add the PDG codes of additional mass eigenstates to the function `get_higgses` in `gum/src/particledb.py` under the appropriate entries `neutral_higgses_by_pdg` and `charged_higgses_by_pdg`, and also make appropriate changes to the functions `write_spectrum_header` in `gum/src/spectrum.py` to reflect any changes to the construction of the `HiggsCouplingsTable`.

The appropriate GAMBIT input entries for using HiggsBounds and HiggsSignals likelihoods are simply

```
ObsLikes:
 # HiggsBounds LEP likelihood
 - purpose: LogLike
   capability: LEP_Higgs_LogLike

 # HiggsSignals LHC likelihood
 - purpose: LogLike
   capability: LHC_Higgs_LogLike
```

where the choice of function fulfilling the capability `HB_ModelParameters` is automatically taken care of by the dependency resolver.

HiggsBounds and HiggsSignals can both be built with

```
make higgsbounds higgssignals
```

but neither actually needs to be rebuilt once a new model is added by GUM.

3 Usage

3.1 Installation

GUM is distributed with GAMBIT 2.0.0 and later. The program can be found within the `gum/` folder in the GAMBIT root directory, and makes use of CMake. In addition to the minimum requirements of GAMBIT itself, GUM also requires at least

- Mathematica 7.0
- Python 2.7 or Python 3
- The Python future module
- Version 1.41 of the compiled Boost libraries Boost.Python, Boost.Filesystem and Boost.System
- libuuid
- libX11 development libraries.

Note that all CMake flags used in GUM are entirely independent from those used within GAMBIT. From the GAMBIT root directory, the following commands will build GUM:

```
cd gum
mkdir build
cd build
cmake ..
make -jn
```

where n specifies the number of processes to use when building.

3.2 Running GUM

The input for GUM is a `.gum` file, written in the YAML format. This file contains all of the information required for GUM to write the relevant module functions for GAMBIT, in a similar vein to the input YAML file used in GAMBIT. GUM is executed with an initialisation file *new_model* `.gum` with the `-f` flag, as in

```
./gum -f new_model . g u m
```

The full set of command-line flags are:

`-d`/`--dryrun`
to perform a dry run
`-h`/`--help`
to display help
`-f`/`--file` *file.gum*
to use the instructions from *file.gum* to run GUM
`-r`/`--reset` *file.mug*
to use the instructions from *file.mug* to run GUM in reset mode

There are three operational modes of gum: *dry run*, *regular* and *reset*. During a *dry run*, no code is actually written to GAMBIT. GUM checks that the Mathematica model file (either FeynRules or SARAH) is suitable for use, and writes a number of proposed source files for GAMBIT, but does not actually copy them to the GAMBIT source directories. This mode can be used for safe testing of new `.gum` and model files, without modifying any of GAMBIT.

A *regular* run of GUM will perform all necessary checks, add the new model to GAMBIT and generate all relevant GAMBIT code requested in the `.gum` file. After a regular GUM execution, GUM prints a set of commands to standard output for the user to run. It is recommended that the user copies these commands and runs them as instructed, as the order of the suggested build and CMake steps can be important, due to new templated C++ types being provided by backends (currently just Pythia).

In addition to the above, GUM outputs a reset (`.mug`) file after a successful run. This file is used in the *reset* mode, and

enables the user to remove a GUM-generated model from GAMBIT. Hence, after adding a new model, the user can run the command

```
./gum -r new_model.mug
```

which will remove all source code generated by GUM associated with the model *new_model*. Note that if the user manually alters any of the auto-generated code, the resetting functionality may not work as expected.

3.3 Input file and node details

GUM files are YAML files in all but name: they are written in YAML format and respect YAML syntax. The only mandatory nodes for a GUM input file are the `math` node, specifying details of the Mathematica package used, and the `output` node, which selects the GAMBIT backends that GUM should generate code for.

The full set of recognised nodes in a `.gum` file is

`math`: describes the Mathematica package used, and subsequently, the model name, plus any other information relating specifically to the package

`wimp_candidate`: give the PDG code for the annihilating DM candidate in the model

`decaying_dm_candidate`: give the PDG code for the decaying DM candidate in the model

`invisibles`: give the PDG codes for particles to be treated as invisibles in collider analyses

`output`: selects which backends GUM should write output for

`output_options`: specific options to use for each backend installation.

The `math` node syntax is

```
math:

 # Select the Mathematica package to use,
 # either 'feynrules' or 'sarah'
 package: feynrules

 # Choose the name of the model
 model: new_model
```

For specific information on FeynRules files, see Sect. 3.4, and for SARAH files, see Sect. 3.5.

Information about the DM candidate of interest is given by either the `wimp_candidate` node or the `decaying_dm_candidate` node. Note that only one of these nodes can be passed; if both are present in the `.gum` file, GUM will throw an error. Although these nodes are optional, if neither is present, then no output will be written for DarkBit (including the Process Catalogue and direct detection interfaces). The syntax is

```
# Select the PDG code of the DM candidate
wimp_candidate: 9900001
```

in the case of annihilating DM, and similarly

```
# Select the PDG code of the DM candidate
decaying_dm_candidate: 9900009
```

in the case of decaying DM. Note that only one DM candidate can be specified at present. Future versions of GAMBIT will allow for multiple DM candidates, and for the lightest stable particle (LSP) to be determined by the `Spectrum` object.

Any additional particles that are to be treated as invisible when calculating missing momentum in collider analyses are given by the `invisibles` node. They are given in a list, with anti-particle PDG codes also required when necessary. If this node is not present, ColliderBit will use the default list provided in `contrib/heputils/include/HEPUtils/Event.h`.

```
# Specify the PDGs of any additional
↪invisibles
invisibles: [9900001,-9900001]
```

The `output` option specifies for which backends GUM should generate code.

```
# Specify outputs: calchep, pythia, spheno,
# vevacious, micromegas
output:
 calchep: true
 spheno: true
 vevacious: false
```

The default for each possible backend output is **false**. If the `output` node is empty, or if all backend output is set to **false**, GUM will terminate with an error message.

The `output_options` node allows the user to pass specific settings relevant for each backend to GUM. We briefly go through these in turn. The syntax for this is

```
output:
 backend_a: true
 backend_b: true

output_options:
 backend_a:
   # Option given by a single key
   option_a: value_a
 backend_b:
   # Option given by a list
   option_b:
   - entry_1
   - entry_2
   ...
```

To tell MadGraph which processes to generate Pythia matrix elements for, the user should provide a list of all BSM processes in MadGraph syntax under the `output_options::pythia::collider_processes`

sub-node. The user needs to know the names of each particle within MadGraph in order to fill in this information.

While Pythia is able to perform its own showering for initial jets, these will be very soft. If the user specifically requires hard ISR jets, such as for a monojet signal associated with DM pair production, these matrix elements should be explicitly requested. In doing so, the user must be careful and aware that collider events are not double counted, i.e. jet matching is performed. We explain our treatment of jet matching in Pythia in Sect. 2.3.5.

For example, to generate matrix elements for monojet and mono-photon production in association with pair production of a DM candidate X, one would include

```
output:
 pythia: true

output_options:
 pythia:
   # Processes in MadGraph syntax
   collider_processes:
     - p p > ~X ~X
     - p p > ~X ~X j
     - p p > ~X ~X a
```

The `collider_processes` sub-node is currently always required if `pythia:true` is set in the `.gum` file.

As with the existing Pythia functionality in ColliderBit, the new function getPythia_new_model introduced to ColliderBit by GUM recognises a YAML option pythia_settings, which can be provided by a user in their GAMBIT input file. In particular, the boolean sub-option *new_model*:all of the pythia_settings option allows one to activate all processes specified in the collider_processes entry of the `.gum` file.

Other sub-nodes of the `.gum` file's pythia entry offer the ability to use the native multiparticle description within MadGraph (multiparticles), and to select events with the relevant particles in the initial and final state (pythia_groups). An example including all available sub-nodes for the pythia entry is shown below:

```
math:
 package: feynrules
 model: newSUSY

output:
 # Generate output for Pythia
 pythia: true

output_options:
 pythia:
   # Define some multiparticles for convenience
   multiparticles:
     - chi0: [chi0_1, chi0_2, chi0_3, chi0_4]
     - chi0bar: [chi0_1, chi0_2, chi0_3,
↪chi0_4]
     - chi+: [chi+_0, chi+_1]
     - chi-: [chi-_0, chi-_1]
   # All processes we want to export to Pythia
```

```
   collider_processes:
     - p p > chi0 chi0bar j
     - p p > chi+ chi-
     - p p > chi- H+
   # Define some groups so we can import
↪processes
   # with these particles in the initial or
↪final
   # states when we do a scan
   pythia_groups:
     - Neutralino: [chi0_1, chi0_2, chi0_3,
↪chi0_4]
```

In this example, including the example pythia_groups node will generate an additional group of events known as newSUSYNeutralino:all, which can also be set in the pythia_settings option of the new ColliderBit module function getPythia_newSUSY. Setting this flag to on picks out all processes in which any of the particles in the pythia_group is an initial or a final state. This is useful for when one wishes to simulate events only for a specific subset of the processes for which matrix elements have been generated for the new model.

For SPheno, the user can request to turn loop decays off via the flag IncludeLoopDecays,

```
math:
 package: sarah
 model: newHDM

output:
 # Generate output for SPheno
 spheno: true

output_options:
 spheno:
   IncludeLoopDecays: false # default: true
```

3.4 FeynRules pathway

Here we describe the process by which GUM can parse a model defined in FeynRules. For details on how to correctly implement a model in FeynRules, we refer the reader to the FeynRules manual [32]. There are many examples of models available on the FeynRules website.[5]

3.4.1 Outputs

FeynRules is designed to study particle physics phenomenology at tree level, and does not directly interface to any spectrum generators. FeynRules is therefore well suited to EFTs and simplified models, as gauge invariance and renormalisability are not typically required in these cases. Because of this, when working from the outputs of FeynRules, GUM is only able to provide minimal interfaces to the SpecBit module and the GAMBIT model database.

---

[5] http://feynrules.irmp.ucl.ac.be/wiki/ModelDatabaseMainPage.

FeynRules is able to output two file formats usable by GUM: CalcHEP (`.mdl` files) and UFO files. GUM uses `.mdl` files with CalcHEP to compute tree-level decay rates and DM annihilation cross-sections, and with microOMEGAs to compute DM relic densities and direct detection rates. The UFO files are currently only used by the MadGraph-Pythia 8 chain, for collider physics. See Sect. 2 for details.

3.4.2 Porting a FeynRules model to GAMBIT

To add a model to GAMBIT based upon a FeynRules file, GUM tries to find the *new_model*`.fr` model file, and any restriction (`.rst`) files that the user may wish to include, first in

```
gum/contrib/FeynRules/Models/new_model/
```

where the folder `gum/` is located inside the GAMBIT root directory. If these model files do not come with the FeynRules installation, GUM instead looks in

```
gum/Models/new_model/
```

To emulate the FeynRules command `LoadModel["new_model.fr"]` the `.gum` file simply needs the entry

```
math:
 package: feynrules
 model: new_model
```

Many models hosted on the FeynRules website and elsewhere often utilise 'base' files and extensions, where one model builds upon another. For instance, a model called `SingletDM` that builds on the Standard Model could be loaded in a Mathematica session using the FeynRules command `LoadModel["SM.fr","SingletDM.fr"]`. This behaviour is also possible with GUM via the additional option `base_model`. In this case, GUM expects `SM.fr` to be located in `Models/SM/` and `SingletDM.fr` to be in `Models/SingletDM/` (where both paths can be independently relative to `gum/` or to `gum/contrib/FeynRules/`). A user would indicate this in their input file like so:

```
math:
 package: feynrules
 model: SingletDM
 base_model: SM
```

An additional FeynRules-only option for the `math` node includes the ability to load FeynRules restriction (`.rst`) files. A FeynRules restriction is useful when considering a restricted subspace of the model at hand. For example, to set the CKM matrix to unity, we can load the restriction file `DiagonalCKM.rst` that is shipped with FeynRules, as

```
math:
 ...
 # Specify any restriction files
 restriction: DiagonalCKM
```

Another FeynRules-only option is the ability to specify the name of the Lagrangian that FeynRules should compute the Feynman rules for. The definition of the Lagrangian can either be a single definition from the FeynRules file:

```
math:
 package: feynrules
 model: SingletDM
 base_model: SM
 # Total Lagrangian in SingletDM.fr
 lagrangian: LTotal
```

or can be given as a string of Lagrangians:

```
math:
 package: feynrules
 model: SingletDM
 base_model: SM
 # All symbols defined in SM.fr or
 ↪SingletDM.fr
 lagrangian: LSM + LDMinteraction + LDMKinetic
```

After loading the model, GUM performs some diagnostics on the model to ensure its validity, checking that the Lagrangian is Hermitian, and that all kinetic and mass terms are correctly diagonalised according to the FeynRules conventions. For more details, we refer the reader to the FeynRules manual [32].

3.4.3 Requirements for FeynRules files

GUM interacts with loaded FeynRules files via the `EParamList` and `PartList` commands. To successfully parse the parameter list, every parameter *must* have a `BlockName` and `OrderBlock` associated with it.

A model implemented in FeynRules will be parametrised in GAMBIT by the full set of parameters denoted as external, by `ParameterType -> External` in the input `.fr` file. Additionally, all masses for non-SM particles are added as input parameters, as they are not computed by spectra.

For example, the SM extended by a scalar singlet S via a Higgs portal with the interaction Lagrangian $\mathcal{L} \supset \lambda_{hs} H^{\dagger} H S^2$ would be parametrised in GAMBIT by the coupling λ_{hs}, as well as the mass of the new field m_S.

The user should not use non-alphanumeric characters (apart from underscores) when defining parameter names (including the `ExternalParameter` field), as this will typically result in errors when producing output. The exception to this is a tilde, which is often used to signify a conjugate field, or in the case of microOMEGAs a DM candidate.

For the MadGraph-Pythia pathway to work correctly, each new external parameter must have its

`InteractionOrder` set. See Sec. 6.1.7 of the FeynRules manual for details [32]. A fully compliant FeynRules entry for a parameter looks as follows:

```
M$Parameters = {
 ...
 gchi == {
   ParameterType    -> External,
   ComplexParameter -> False,
   InteractionOrder -> {NP, 1},
   BlockName        -> DMINT,
   OrderBlock       -> 1,
   Value            -> 1.,
   TeX              -> Subscript[g,\\[Chi]],
   Description      -> "DM-mediator
 coupling"
 },
 ...
```

where the `BlockName`, `OrderBlock` and `InteractionOrder` are all defined. We also set `ComplexParameter` to `False`, as FeynRules is not able to generate CalcHEP files for complex parameters. All parameters that are complex should be redefined as their real and imaginary parts, with all factors of i explicitly placed in the Lagrangian.

For a matrix, the `OrderBlock` does not need to be specified,

```
M$Parameters = {
 ...
 yL == {
   ParameterType    -> External,
   ComplexParameter -> False,
   InteractionOrder -> {BSM, 1},
   Indices          -> {Index[Generation],
                         Index[Generation]},
   BlockName        -> yL,
   Value            -> {yL[1,1]->1,
yL[1,2]->0,
                         yL[2,1]->0,
yL[2,2]->1},
   TeX              -> Subscript[y,L],
   InteractionOrder -> {NP, 1},
   Description      -> "Left-handed matrix"
 },
 ...
```

In this case, GUM will add 4 model parameters to the model for each matrix index, labelled by `matrixname_[i]x[j]`, i.e. `yL_1x1, yL_1x2, yL_2x1,` `↪yL_2x2` for the above entry. Note that the values for each entry can be set to anything; these will all be set by GAMBIT during a scan.

An example of a particle implementation, for the Majorana DM candidate used in Sect. 4, is

```
M$ClassesDescription = {
 ...
 F[5] == {
   ClassName      -> chi,
   SelfConjugate  -> True,
   Mass           -> {mchi, 1000.},
   Width          -> 0.,
```

```
   PDG            -> 52,
   ParticleName   -> "~chi",
 },
 ...
```

Here we see that the `ParticleName` begins with a tilde, so that microOMEGAs can correctly identify it as a WIMP DM candidate, the PDG code is assigned to 52 (generic spin-1/2 DM, as per the PDG), and the particle mass `mchi` will be added as an external parameter. Note that because this particle has `SelfConjugate -> True`, GUM does not require the electric charge to be set. If the particle were Dirac, i.e. `SelfConjugate -> False`, GUM would require the additional entry `QuantumNumbers -> {Q -> 0}`.

For a particle η that should decay, an appropriate entry for the particle width would look like `Width -> {weta, 1.}`, enabling the contents of the `DecayTable` to be passed to CalcHEP. Note that in this case, `weta` will not be set as a free parameter of the model in GAMBIT, but derived from the model parameters and accessible channels.

Although FeynRules is able to compute the Feynman rules for a theory containing 4-fermion interactions, it does not support generating CalcHEP files for these models.[6] The first release of GUM does not support theories implemented in FeynRules with 4-fermion interactions, however such support is planned for future releases.

3.5 SARAH pathway

3.5.1 Outputs

As shown in Table 1, SARAH is able to generate output for CalcHEP, microOMEGAs, Pythia, SPheno and Vevacious. As SARAH is able to generate CalcHEP, MadGraph/Pythia and microOMEGAs output, it can mirror the capabilities of FeynRules in the context of GUM.

SARAH has been labeled a 'spectrum generator generator', as it can also automatically write Fortran source code for SPheno for a given model. GUM is able to automatically patch the SPheno source code generated by SARAH, and write a frontend interface to that SARAH-SPheno version.

3.5.2 Porting a SARAH model to GAMBIT

To add a model to GAMBIT based upon a SARAH file, the model file *new_model*.m must be located in

```
gum/contrib/SARAH/Models/new_model/
```

or

```
gum/Models/new_model/
```

---

[6] At the time of writing, LanHEP is the only package that supports automatic generation of 4-fermion contact interactions for CalcHEP files.

The usual **SARAH** files `parameters.m` and `particles.m` should also be present in one of these locations. To generate spectra via **SPheno**, a `SPheno.m` file must also be provided in the same directory.

GUM loads a new model in **SARAH** by invoking the command `Start["`*new_model*`"]`, which is selected by the `.gum` entries

```
math:
 package: sarah
 model: new_model
```

In order to validate the model **GUM** uses the **SARAH** command `CheckModel[]`. **SARAH** provides the results of the `CheckModel[]` function only to `stdout` and via error messages. **GUM** therefore captures the output and message streams from **Mathematica** in order to gather this information, and decides whether the errors should be considered fatal or not. Non-fatal errors, including gauge anomalies, possible allowed terms in the Lagrangian or missing Dirac spinor definitions, are directed to **GUM**'s own standard output as warnings. Fatal errors, such as non-conservation of symmetries or those associated with particle and parameter definitions, cause **GUM** to abort, as subsequent steps are guaranteed to fail in these cases.

3.5.3 Requirements for *SARAH* files

As with **FeynRules**, **GUM** extracts information from **SARAH** about the parameters and particles in the model. These are collected by **SARAH** in the `ParameterDefinitions` and `ParticleDefinitions` lists, respectively.

Definitions for new model parameters are located in the `parameters.m` file within the **SARAH** model folder. A well-defined entry for a new **SARAH** parameter looks as follows:

```
ParameterDefinitions = {
 ...
 {gchi,
   {
     Description -> "DM-mediator coupling",
     LesHouches  -> {DMINT, 1},
     OutputName  -> "gchi",
     LaTeX       -> "g_\\chi"
   }
 },
 ...
}
```

where the `LesHouches` block and respective index are required fields.

For a matrix, the index does not need to be specified:

```
ParameterDefinitions = {
 ...
 {YN,
   {
     Description -> "Yukawa for N field",
```

```
     LesHouches  -> YN,
     LaTeX       -> "Y_{\\rm N}",
     OutputName  -> yn
   },
 },
 ...
}
```

This instructs **GUM** to add a `LesHouches` block `YN` to the `SimpleSpec` definition, which will be filled by a spectrum generator.

GUM is concerned with the properties of physical particles in the mass basis (designated `[EWSB]` in **SARAH**). An example particle implementation from the `particles.m` file is:

```
ParticleDefinitions[EWSB] = {
 ...
 {ss,
   {
     Description    -> "Scalar singlet",
     Mass           -> LesHouches,
     PDG            -> {51},
     ElectricCharge -> 0,
     OutputName     -> "~Ss",
     LaTeX          -> "S"
   }
 },
 ...
}
```

Here the important entries are

- the `Mass` entry, where `Mass -> LesHouches` signifies that the particle mass will be provided by the **GAMBIT** `Spectrum` object (whether that is filled using **SPheno** or a tree level calculation),
- the `PDG` entry, which specifies a list over all generations for the mass eigenstates (in this example there is just one), and
- the `ElectricCharge` field.

Note that **SARAH** has default definitions for many particles and parameters in *SARAH_dir*`/Models/particles.m` and *SARAH_dir*`/Models/parameters.m`. Their properties can be inherited, or overwritten, via the `Description` field.

Information about mixing matrices is stored by **SARAH** in the variable `DEFINITION[EWSB][MatterSector]`. From this variable **GUM** learns the names of the mixing matrices associated with each particle. For Weyl fermions, **GUM** requests the name of the associated Dirac fermion, stored in the variable `DEFINITION[EWSB][MatterSector]`. As an example, the mixing matrices for the electroweakino sector of the MSSM are extracted as

```
DEFINITION[EWSB][MatterSector][[;;,2]] = {
...
 {L0, ZN},
 {{Lm, UM}, {Lp, UP}}
```

```
...
}
```

which associates the matrix `ZN` with the Weyl fermion `L0` (neutralinos) and the matrixes `UM` and `UP` with `Lm` (negative charginos) and `Lp` (positive charginos). As these are Weyl fermions, the Dirac eigenstates are

```
DEFINITION[EWSB][DiracSpinors] = {
...
  Chi -> {L0, conj[L0]},
  Cha -> {Lm, conj[Lp]}
...
}
```

GUM thus knows to assign the mixing matrix `ZN` to Dirac-eigenstate neutralinos `Chi`, as well as the matrices `UM` and `UP` to Dirac-eigenstate charginos `Cha`.

As opposed to FeynRules, where all parameters and particle masses become GAMBIT model parameters, the SARAH pathway attempts to optimise this list through various means. In the absence of a spectrum generator (e.g. SPheno, see below), almost all the parameters in `ParameterDefinitions` become model parameters. Only those with explicit dependencies on other parameters are removed, i.e. those with the `Dependence` or `DependenceSpheno` fields. In addition, SARAH provides tree-level relations for all masses, via `TreeMass[`*particle_name*`, EWSB]`, so even in the absence of a spectrum generator, none of the particle masses become explicit model parameters. For models with BSM states that mix together into mass eigenstates[7], the tree-level masses are not used and an error is thrown to inform the user of the need to use a spectrum generator.

If the user elects in their `.gum` file to generate any outputs from SARAH for specific backends, GUM requests that SARAH generate the respective code using the relevant SARAH commands. These are `MakeCHep[]` for CalcHEP and micrOMEGAs, `MakeUFO[]` for MadGraph/Pythia, `MakeSPheno[]` for SPheno and `MakeVevacious[]` for Vevacious.

When SPheno output is requested, GUM interacts further with SARAH in order to obtain all necessary information for spectrum generation:

1. Replace parameters and masses with those in SPheno. The parameter names are obtained using the `SPhenoForm` function operating on the lists `listAllParametersAndVEVs` and `NewMassParameters`. The particle masses are obtained just by using the `SPhenoMass[`*particle_name*`]` command.
2. Extract the names and default values of the parameters in the `MINPAR` and `EXTPAR` blocks, as defined in the model

---

[7] Technically this is done by checking if the PDG list for any of the BSM particles contains more than one entry.

file `SPheno.m`. For each of these, store the boundary conditions, also from `SPheno.m`, that match the `MINPAR` and `EXTPAR` parameters to those in the parameter list. Note that as of GUM 1.0, only the boundary conditions in `BoundaryLowScaleInput` are parsed.
3. Remove from the parameter list those parameters that will be fixed by the tadpole equations, as they are not free parameters. These are collected from the list `ParametersToSolveTadpoles` as defined in `SPheno.m`.
4. Get the names of the blocks, entries and parameter names for all SLHA input blocks ending in `IN`, e.g. `HMIXIN`, `DSQIN`, etc. SARAH provides this information in the list `CombindedBlocks`.
5. Register the values of various flags needed to properly set up the interface to SPheno. These are `"SupersymmetricModel"`, `"OnlyLowEnergySPheno"`, `"UseHiggs2LoopMSSM"` and `"SA`AddOneLoopDecay"`.

4 A worked example

To demonstrate the process of adding a new model to GAMBIT with GUM, in this section we provide a simple worked example. Here we use GUM to add a model to GAMBIT, perform a parameter scan, and plot the results with pippi [95]. This example is designed with ease of use in mind, and can be performed on a personal computer in a reasonable amount of time. For this reason we select a simplified DM model, implemented in FeynRules.

In this example, we consider constraints from the relic density of dark matter, gamma-ray indirect detection and traditional high-mass direct detection searches. It should be noted that this is an example, not a full global scan, so we do not use all of the information available to us – a real global fit of this model would consider nuisance parameters relevant to DM, as well as a full set of complementary likelihoods such as from other indirect DM searches, low-mass direct detection searches, and cosmology.

The FeynRules model file, `.gum` file, GAMBIT input file and `pip` file used in this example can be found within the `Tutorial` folder in GUM.

4.1 The model

The model is a simplified DM model, where the Standard Model is extended by a Majorana fermion χ acting as DM, and a scalar mediator Y with a Yukawa-type coupling to all SM fermions, in order to adhere to minimal flavour violation. The DM particle is kept stable by a $\mathcal{Z}_2$ symmetry under which it is odd, $\chi \to -\chi$, and all other particles are even. Both χ and Y are singlets under the SM gauge group.

Here, we assume that any mixing between Y and the SM Higgs is small and can be neglected. This model has been

previously considered in e.g. [96,97] and is also one of the benchmark simplified models used in LHC searches [98–100]. The model Lagrangian is

$$\mathcal{L} = \mathcal{L}_{\mathrm{SM}} + \frac{1}{2}\overline{\chi}\left(i\slashed{\partial} - m_\chi\right)\chi + \frac{1}{2}\partial_\mu Y \partial^\mu Y - \frac{1}{2}m_Y^2 Y^2$$
$$- \frac{g_\chi}{2}\overline{\chi}\chi Y - \frac{c_Y}{2}\sum_f y_f \overline{f} f Y \,. \tag{1}$$

Note that this theory is not $SU(2)_L$ invariant. One possibility for a 'realistic' model involves Y-Higgs mixing, which justifies choosing the $Y\overline{f}f$ couplings to be proportional to the SM Yukawas y_f.

The free parameters of the model are simply the dark sector masses and couplings, $\{m_\chi, m_Y, c_Y, g_\chi\}$. In this example we follow the FeynRules pathway, working at tree level.

4.2 The .gum file

Firstly, we need to add the FeynRules model file to the GUM directory. The model is named 'MDMSM' (Majorana DM, scalar mediator). Starting in the GUM root directory, we first create the directory that the model will live in, and move the example file from the Tutorial folder to the GUM directory:

```
mkdir Models/MDMSM
cp Tutorial/MDMSM.fr Models/MDMSM/
```

As we are working with FeynRules, the only backends that we are able to create output for are CalcHEP, micrOMEGAs and MadGraph/Pythia. For the sake of speed, in this tutorial we will not include any constraints from collider physics. This is also a reasonable approximation, as for the mass range that we consider here, the constraints from e.g. monojet, dijet and dilepton searches are subleading (see e.g. Ref. [96] and Appendix A). We therefore set pythia:**false**. The contents of the supplied .gum file are simple:

```
math:
 # Choose FeynRules
 package: feynrules
 # Name of the model
 model: MDMSM
 # Model builds on the Standard Model
↪FeynRules file
 base_model: SM
 # The Lagrangian is defined by the DM sector
↪(LDM),
 # defined in MDMSM.fr, plus the SM
↪Lagrangian (LSM)
 # imported from the 'base model', SM.fr
 Lagrangian: LDM + LSM
 # Make CKM matrix = identity to simplify
↪output
 restriction: DiagonalCKM
```

```
# PDG code of the annihilating DM candidate
# in the FeynRules file
wimp_candidate: 52

# Select outputs for DM physics.
# Collider physics is not as important in
↪this model.
output:
 pythia: false
 calchep: true
 micromegas: true
```

Note the selection of the PDG code of the DM particle as 52, so that if we were to use Pythia, χ would be correctly identified as invisible.

We can run this from the GUM directory,

```
./gum -f Tutorial/MDMSM.gum
```

and GUM will automatically create all code needed to perform a fit using GAMBIT. On a laptop with an Intel Core i5 processor, GUM takes about a minute to run. All that remains now is to (re)compile the relevant backends and GAMBIT, and the new model will be fully implemented, and ready to scan. GUM prints a set of suggested build commands to standard output to build the new backends and GAMBIT itself. Starting from the gum directory, these are

```
cd ../build
cmake ..
make micromegas_MDMSM
make calchep
make -jn gambit
```

where n specifies the number of processes to use when building.

Note that GUM does not adjust any CMake flags used in previous GAMBIT compilations, so the above commands assume that the user has already configured GAMBIT appropriately and built any relevant samplers before running GUM. A user wishing to instead configure and build GAMBIT from scratch after running GUM, in order to e.g. run the example scan of Sect. 4 using differential evolution sampling and MPI parallelisation, would need to instead do (again, starting from the gum directory)

```
cd ../build
cmake -D WITH_MPI=ON ..
make diver
cmake ..
make micromegas_MDMSM
make calchep
make gamlike
make ddcalc
make -jn gambit
```

For more thorough CMake instructions, see the README in the gum/Tutorial, and CMAKE_FLAGS.md in the GAMBIT root directory.

4.3 Phenomenology and constraints

The constraints that we will consider for this model are entirely in the DM sector, as those from colliders are less severe. (Collider constraints are investigated in Appendix A.) The dark matter constraints are:

– Relic abundance: computed by microOMEGAs, and employed as an upper bound, in the spirit of effective DM models.
– Direct detection: rates computed by microOMEGAs, likelihoods from XENON1T 2018 [101] and LUX 2016 [102], as computed with DDCalc [4,14,15].
– Indirect detection: *Fermi*-LAT constraints from gamma-ray observations of dwarf spheroidal galaxies (dSphs) [103]. Tree level cross-sections are computed by CalcHEP, γ ray yields are consequently computed via DarkSUSY [104,105], and the constraints are applied by gamLike [4].

As the relic density constraint is imposed only as an upper bound, we rescale all DM observables by the fraction of DM, $f = \Omega_\chi / \Omega_{DM}$.

Here we will use the GAMBIT input file `gum/ Tutorial/MDMSM_Tute.yaml`. Although it does contain a little more than the GAMBIT input file automatically generated by GUM (`yaml_files/MDMSM_example.yaml`), it is still fairly standard, so we will cover only the important sections here. For an overview of YAML files in GAMBIT, we refer the reader to Sec. 6 of the GAMBIT manual [1].

Firstly the parameters section indicates all models required for this scan: not just the MDMSM parameters, but also SM parameters, nuclear matrix elements and DM halo parameters. The parameter range of interest for the MDMSM model will be masses ranging from 45 GeV to 10 TeV, and dimensionless couplings ranging from 10^{-4} to 4π. We will scan each of these four parameters logarithmically.

```
Parameters:

 # Import some default GAMBIT SM values
 StandardModel_SLHA2: !import
  include/StandardModel_SLHA2_defaults.yaml

 # Higgs sector is defined separately in
↪GAMBIT
 StandardModel_Higgs:
  mH: 125.09

 # Our dark matter model, implemented by GUM
 MDMSM:
  mchi:
    range: [45, 10000]
    prior_type: log
  mY:
    range: [45, 10000]
    prior_type: log
```

```
  gchi:
    range: [1e-4, 12.566]
    prior_type: log
  cY:
    range: [1e-4, 12.566]
    prior_type: log

 # Default halo parameters for the example
 Halo_gNFW_rho0:
  rho0: 0.3
  v0: 240
  vesc: 533
  vrot: 240
  rs: 20.0
  r_sun: 8.5
  alpha: 1
  beta: 3
  gamma: 1

 # Nuclear matrix parameters, also default
 nuclear_params_sigmas_sigmal:
  sigmas: 43
  sigmal: 58
  deltau: 0.842
  deltad: -0.427
  deltas: -0.085
```

The `ObsLikes` section includes likelihoods concerning the relic density, indirect detection from dSphs, and direct detection experiments.

```
ObsLikes:
  # Relic density
  - capability: lnL_oh2
    purpose: LogLike

  # Indirect detection
  - capability: lnL_FermiLATdwarfs
    purpose: LogLike

  # Direct detection: LUX experiment
  - capability: LUX_2016_LogLikelihood
    purpose: LogLike

  # Direct detection: XENON1T experiment
  - capability: XENON1T_2018_LogLikelihood
    purpose: LogLike
```

The `Rules` section uniquely specifies the functions to use for the dependency resolver:

```
Rules:
 # Use MicrOmegas to compute the relic density
 - capability: RD_oh2
   function: RD_oh2_MicrOmegas

 # Choose to implement the relic density
 # likelihood as an upper bound, not a
↪detection
 - capability: lnL_oh2
   function: lnL_oh2_upperlimit

 # Choose to use detailed Fermi Pass 8 dwarf
 # likelihood from gamlike
 - capability: lnL_FermiLATdwarfs
```

```
    function: lnL_FermiLATdwarfs_gamLike

 # Choose to get decays from regular DecayBit
 # function, not from an SLHA file nor SPheno.
 - capability: decay_rates
    function: all_decays

 # Choose to rescale signals in direct and
↪indirect
 # detection by the relic density fraction
 - capability: RD_fraction
    function: RD_fraction_leq_one
```

The scanner section selects the differential evolution sampler **Diver** [3] with a fairly loose stopping tolerance of 10^{-3} and a working population of 10,000 points.

```
Scanner:

 # Select differential evolution (DE) scanner
 use_scanner: de

 scanners:

   # Select settings for DE with Diver
   de:
     plugin: diver
     like: LogLike
     NP: 10000
     convthresh: 1e-3
     verbosity: 1
```

To perform the scan we copy the **GAMBIT** input file to the `yaml_files` folder within the **GAMBIT** root directory. This is a necessary step, as we need to `!import` the appropriate Standard Model **YAML** file from the relative path `include` (i.e. the folder `yaml_files/include` in the **GAMBIT** root directory). From the **GAMBIT** root directory, we

```
cp gum/Tutorial/MDMSM_Tute.yaml yaml_files/
```

and run **GAMBIT** with n processes,

```
mpirun -n n gambit -f
↪yaml_files/MDMSM_Tute.yaml
```

The above scan should converge in a reasonable time on a modern personal computer; this took 11 hr to run using 4 cores on a laptop with an i5-6200U CPU @ 2.30GHz, sampling 292k points in total. The results of this scan are shown below.

Note that whilst the scan has converged statistically, the convergence criterion that we set in the input file above is not particularly stringent, so many of the contours presented in this section are not sampled well enough to be clearly defined. A serious production scan would typically be run for longer, and more effort made to map the likelihood contours more finely. Nonetheless, the samples generated are more than sufficient to extract meaningful physics.

Once the scan has finished, we can plot the result using **pippi** [95]. As **Diver** aims to finds the maximum likelihood

point, we will perform a profile likelihood analysis with **pippi**. Assuming that **pippi** is in `$PATH`, do

```
cd gum/Tutorial
pippi MDMSM.pip
```

which will produce plots of the four model parameters against one another, as well against as a raft of observables such as the relic abundance and spin-independent cross-section (rescaled by f).

4.4 Results

The upper panel of Fig. 1 shows the profile likelihood in the plane of the DM mass m_χ against the mediator mass m_Y. The relic density requirement maps out the structure in the same plane. There are two sets of solutions: firstly when the DM is heavier than the mediator, $m_\chi > m_Y$ (bordered by the red dashed line in Fig. 1), and secondly where DM annihilates on resonance, $2m_\chi \approx m_Y$ (centred on the purple dashed line in Fig. 1).

When $m_\chi < m_Y$ and the YY annihilation channel is not kinematically accessible, annihilation predominantly occurs via an s-channel Y to $b\bar{b}$ or $t\bar{t}$, depending on the DM mass. In this case, the only way to efficiently deplete DM in the early Universe is when annihilation is on resonance, $m_\chi \approx m_Y/2$. Away from the resonance when the YY channel is closed, even couplings of 4π are not large enough to produce a sufficiently high annihilation cross-section to deplete the thermal population of χ to below the observed value.

When kinematically allowed, $\chi\overline{\chi} \to Y \to t\bar{t}$ is the dominant process responsible for depleting the DM abundance in the early Universe. When $m_\chi < m_t$ and $m_\chi < m_Y$, the only way to produce the correct relic abundance is when exactly on resonance, $2m_\chi = m_Y$, annihilating mostly to $b\bar{b}$. The effect of the t threshold can clearly be seen in Fig. 1: as the $\chi\overline{\chi} \to t\bar{t}$ channel opens up, the contours do not trace the resonance $2m_\chi = m_Y$ quite as tightly. This is because the cross-section to $\bar{t}t$ is significantly larger, as the mediator coupling to the SM leptons are proportional to their Yukawas. This means that near the resonance region it is far easier to satisfy the DM abundance constraint, which leads to the spread about the purple line for $m_\chi > m_t$ in Fig. 1.

When the DM candidate is heavier than the mediator, the process $\chi\overline{\chi} \to YY$ is kinematically accessible, and proceeds via t-channel χ exchange. When this channel is open, the correct relic abundance, $\Omega_\chi h^2$, can be acquired independently of c_Y by adjusting m_χ and g_χ. This can be seen in Fig. 2.

In this regime, the relic abundance constrains the DM coupling g_χ, as seen in the central panel of Fig. 2, with annihilation cross-section $\langle \sigma v \rangle \propto g_\chi^2 c_Y^2/m_\chi^2$. We plot m_χ against g_χ in Fig. 3; the lower bound is set by the resonance region, and is (unsurprisingly) poorly sampled for low values of m_χ.
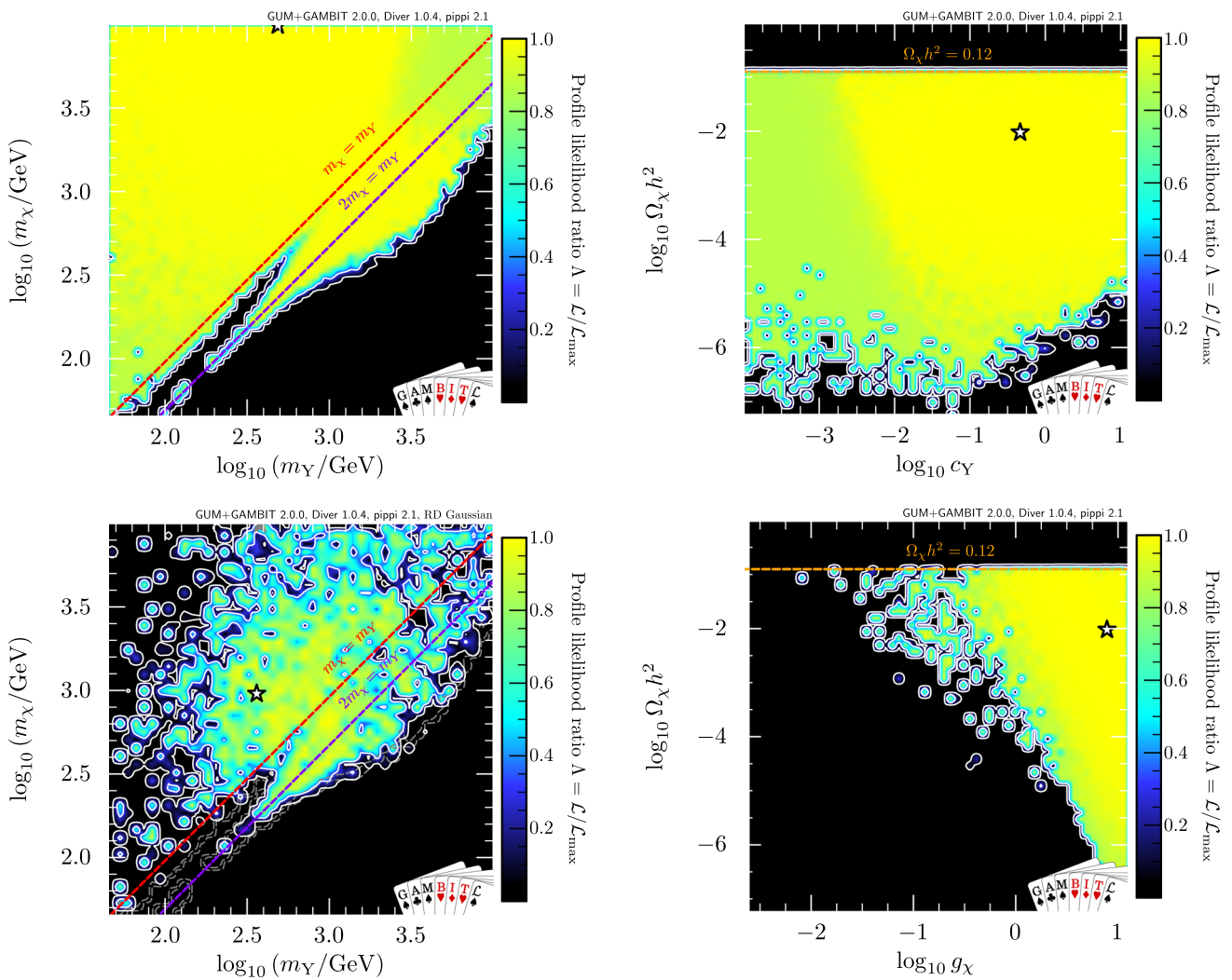
Fig. 1 Profile likelihood in the m_χ–m_Y plane with the relic density as an upper bound (upper panel) and as an observation (lower panel). Above the red dashed line at $m_\chi = m_Y$, DM can annihilate into Y bosons. The purple dashed line at $2m_\chi = m_Y$ indicates the region where DM can annihilate on resonance. Contour lines show the 1 and 2σ confidence regions. The white star shows the best-fit point. The grey contours in the lower panel the 1 and 2σ contours from the upper panel

To show the impact of allowing DM to be underabundant, we perform a separate scan where we instead employ a Gaussian likelihood for the relic abundance. This can be achieved by instead using the following entry in the `Rules` section of the GAMBIT input file:

```
# Choose to implement the relic density
↪likelihood
# as a detection, not an upper bound
- capability: lnL_oh2
  function: lnL_oh2_Simple
```

We show the m_χ–m_Y plane for this scan in the lower panel of Fig. 1. For a given point in the m_χ–m_Y plane, the couplings
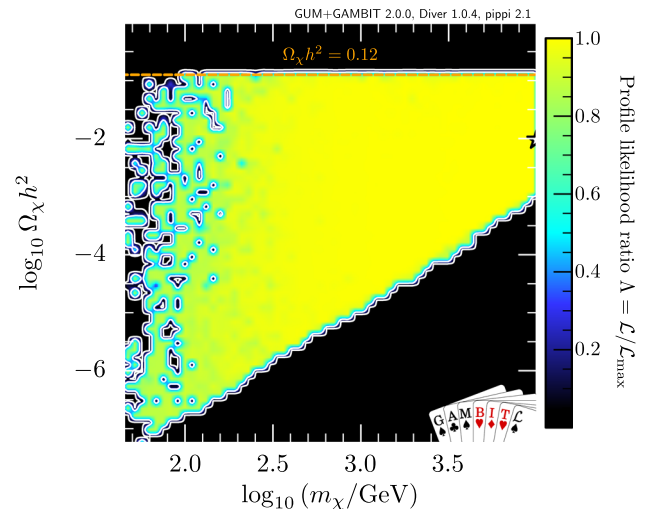
Fig. 2 Profile likelihood in the $\Omega_\chi h^2$–c_Y, (top) $\Omega_\chi h^2$–g_χ (centre), and $\Omega_\chi h^2$–m_χ (bottom) planes. The orange dashed line shows the standard ΛCDM limit from *Planck* [106]. Contour lines show 1 and 2σ confidence regions, and the white star the best-fit point
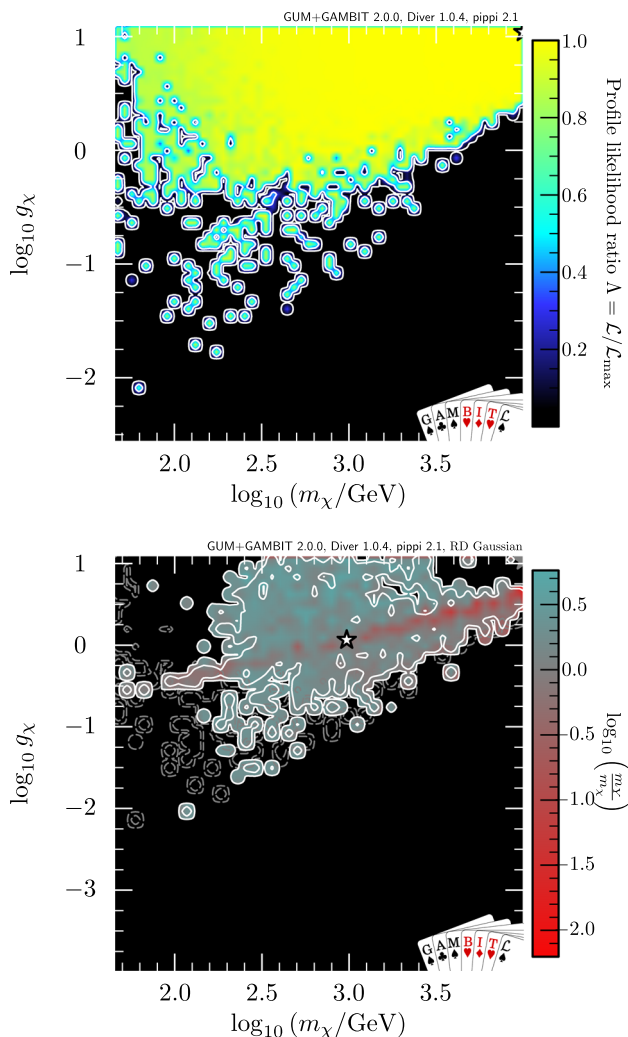
Fig. 3 Upper panel: profile likelihood in the m_χ–g_χ plane when the relic density is an upper bound. Lower panel: the m_χ–g_χ plane coloured by m_Y/m_χ, when treating the relic density as an observation. Shading and white contour lines represent 1 and 2σ confidence regions. Grey contours in the lower panel are the 1 and 2σ regions from the upper panel, where the relic density is used as an upper bound. The white star corresponds to the best-fit point

g_χ and c_Y must be correctly tuned to fit the relic density requirement: clearly, the scanner struggles to find such points compared to when DM can be underabundant. Notably, the sampler struggles to find the very fine-tuned points on resonance when $\bar{t}t$ is not kinematically accessible.

In the lower panel of Fig. 3 we show the m_χ–g_χ plane when requiring that χ fits the observed relic abundance, coloured by $\frac{m_Y}{m_\chi}$. There is a well-defined red area scattered along a straight line with $\frac{m_Y}{m_\chi} < 1$, corresponding to efficient annihilation to YY, i.e. for $\langle \sigma v \rangle \propto g_\chi^4/m_\chi^2$. This is reflected in the lower panel of Fig. 1: almost all of the valid samples for $m_\chi < m_t$ are in the regime where $m_\chi > m_Y$, i.e. above the red dashed line. Here we see that the slope of the line fol-

lowed by the red area in the lower panel of Fig. 3 is exactly half that of the lower bound on g_χ, due to the fact that the latter is instead set by resonant annihilation to fermions, which involves one less power of g_χ in the corresponding matrix element, i.e. $\langle \sigma v \rangle \propto g_\chi^2 c_Y^2/m_\chi^2$.

Direct detection processes proceed via t-channel Y exchange. The functional form of the spin-independent cross-section is [e.g. 98]:

$$\sigma_{\mathrm{SI}}^N = \frac{\mu_{\chi N}^2 m_N^2}{\pi} \left(\frac{g_\chi c_Y}{v m_Y^2} \right)^2 f_N^2 , \qquad (2)$$

where $\mu_{\chi N}$ is the DM-nucleon reduced mass, $N = n, p$, and the form factor

$$f_N = \sum_{q=u,d,s} f_N^q + \frac{2}{27} f_N^G . \qquad (3)$$

Here the light-quark form factors are

$$f_p^u = 0.0233, \; f_p^d = 0.0343, \; f_p^s = 0.0458 \qquad (4)$$

$$f_n^u = 0.0160, \; f_n^d = 0.0499, \; f_n^s = 0.0458, \qquad (5)$$

and the gluon factors $f_N^G = 1 - \sum_{q=u,d,s} f_N^q$ are

$$f_p^G = 0.8966, \; f_n^G = 0.8883. \qquad (6)$$

These follow directly from the values $\sigma_s = 43\,\mathrm{MeV}$, $\sigma_l = 58\,\mathrm{MeV}$ chosen in the GAMBIT input file presented in Sect. 4.3. Details of the conversion between the two parameterisations, and a discussion of possible values for σ_s and σ_l, can be found in Refs. [4,107].

Thus for a given DM mass m_χ, direct detection constrains the parameter combination $g_\chi c_Y/m_Y^2$, rescaled by the DM fraction $f \equiv \Omega_\chi/\Omega_{\mathrm{DM}}$.

Figure 4 shows the spin-independent cross-section on protons as a function of the DM mass. As it is possible for χ to be underabundant for all masses, it is easy to evade direct detection limits by simply tuning the couplings. We also plot the projection from LZ [108], which shows the significant effect that future direct detection experiments can have on the parameter space of this model, including the ability to probe the current best-fit point.

the best-fit region in Fig. 4 lies just below the XENON1T limit: this is due to a small excess (less than 2σ) in the data, which can be explained by this model. This excess is discussed in more detail in a GAMBIT study of scalar singlet DM [14].

Note that for all annihilation channels, the annihilation cross-section is proportional to the square of the relative velocity of DM particles in the direction perpendicular to the momentum transfer, i.e. $\langle \sigma v \rangle \propto v_\perp^2$. This means that annihilation is velocity suppressed, especially in the late Universe where $v_\perp \sim 0$. As annihilation processes are also sup-
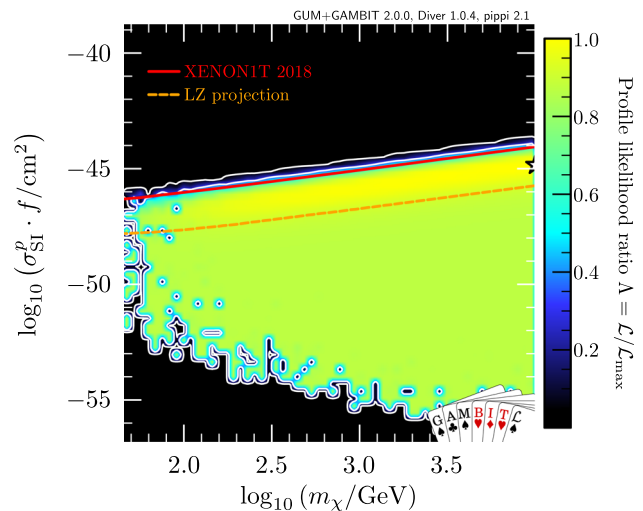
Fig. 4 Profile likelihood in the σ_{SI}^p–m_χ plane. The solid red line shows the exclusion from XENON1T [101] and the dotted orange line shows the projection from LZ [108]. The spin-independent scattering cross-section of dark matter with protons σ_{SI}^p is rescaled by the fraction of predicted relic abundance $f \equiv \Omega_\chi / \Omega_{DM}$. Contour lines show the 1 and 2σ confidence regions. The white star shows the best-fit point

Fig. 5 Profile likelihoods for the effective annihilation cross-section as a function of dark matter mass, at different epochs in the evolution of the Universe. The dashed orange line signifies the canonical thermal cross-section $\langle \sigma v \rangle = 3 \times 10^{-26} \text{cm}^3\text{s}^{-1}$. White contours with coloured shading correspond to the thermal average cross-section at dark matter freezeout. The contours show the 1 and 2σ confidence regions, and the white star the best-fit point. For comparison, in grey contours we show the 1 and 2σ confidence regions for the annihilation cross-section in the $v \to 0$ limit, rescaled by the square of the fraction of the predicted relic abundance, $f \equiv \Omega_\chi / \Omega_{DM}$. This is the effective annihilation cross-section that enters indirect detection rates in the late Universe

pressed by the square of the DM fraction f, indirect detection signals therefore do not contribute significantly to the likelihood function. The velocity dependence of the cross-section is fully taken into account by microMEGAs in computing the relic density, however, so the thermally averaged value at freezeout is much larger than the late-time value. We show the thermally-averaged value at freezeout in Fig. 5, which, as expected, overlaps the canonical thermal value $\langle \sigma v \rangle = 3 \times 10^{-26} \text{cm}^3\text{s}^{-1}$. For comparison, in grey contours we also plot $f^2 \langle \sigma v \rangle_{v \to 0}$, the effective cross-section for indirect detection. In this case, all parameter combinations give cross-sections several orders of magnitude below the canonical thermal value, heavily suppressing all possible indirect detection signals.

If we wish to remove the model from GAMBIT, we simply run the command

```
./gum -r mug_files/MDMSM.mug
```

and the GAMBIT source reverts to its state prior to the addition of the model.

5 Summary

The standard chain for a theorist to test a BSM theory against data has been greatly optimised, and largely automated in recently years, with the development of Lagrangian-level tools such as FeynRules and SARAH. On the phenomenological side, GAMBIT has been designed as a modular global fitting suite for extensive studies of BSM physics. GUM adds

the final major missing piece to the automation procedure. By providing an interface between GAMBIT, SARAH and FeynRules, it makes global fits directly from Lagrangians possible for the first time. This will make the process of performing statistically rigorous and comprehensive phenomenological physics studies far easier than in the past.

We have shown that GUM produces sensible results for a simplified model, in good agreement with previous results found in the literature. This is based on a scan that can be performed on a personal computer in a reasonable time frame.

The modular nature of GUM means that extension is straightforward. Since the first version of this paper appeared, GUM has already been extended in GAMBIT 2.1 to include a four-fermion EFT plugin connecting FeynRules and CalcHEP [16]. Other extensions planned include computation of modifications to SM precision observables and decays, multi-component and co-annihilating dark matter models, and interfacing to the GAMBIT flavour physics module FlavBit via FlavorKit. We also plan to add new interfaces to public codes not included in this release, including to spectrum generators and decay calculators associated with FlexibleSUSY, and to the dark matter package MadDM. We will also update supported backends to the latest versions, in particular microMEGAs 5 [89], HiggsBounds 5 [109], HiggsSignals 2 [110] and Pythia 8.3.

Appendix A: Collider constraints on the Majorana DM simplified model with scalar mediator

We argued in Sect. 4 that the collider constraints are expected to be subleading for the MDMSM. To justify and clarify that argument, and to demonstrate GUM's ability to generate code for collider simulations, we here investigate the likelihood contribution from LHC searches.

It has been demonstrated that monojet searches are not necessarily the most constraining searches for the MDMSM [96,111]. In fact, given the large Yukawa couplings the tree level production of top quark pairs together with the mediator Y, despite the large final state masses, should be the most sensitive final state at the 13 TeV LHC. To investigate the constraints from this process, we select a $139\,\text{fb}^{-1}$ ATLAS search for final states with two leptons, jets and missing momentum, which is targeted to this specific final state [112].
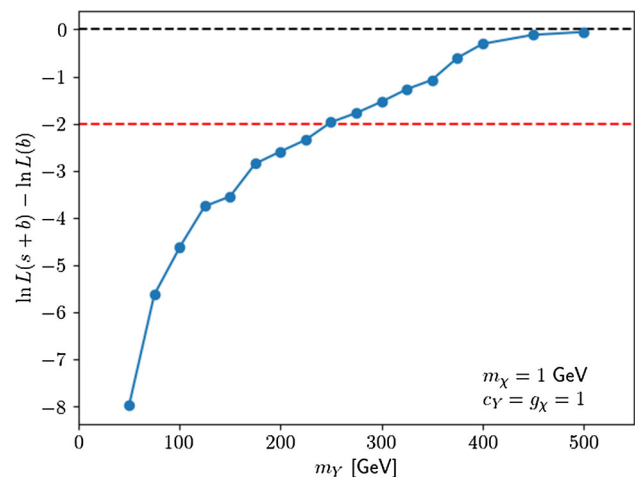

Fig. 6 The log-likelihood contribution $\Delta \ln L = \ln L(s+b) - \ln L(b)$ from a simulation of the ATLAS search in Ref. [112] in a scan of the mediator mass m_Y in the MDMSM. The mediator coupling to SM particles (c_Y) and to DM (g_χ) are set to $c_Y = g_\chi = 1$, and the DM mass is fixed at $m_\chi = 1$ GeV. The dashed black line denotes $\Delta \ln L = 0$, i.e. the limit where the signal plus background prediction fits the data equally as well as the background-only prediction. The dashed red line at $\Delta \ln L = -2$ shows the $\Delta \ln L$ limit corresponding to a 2σ confidence interval on m_Y. The most sensitive signal region differs for the $m_Y = 50$ GeV point, but is otherwise consistent across the scan

The computational requirement of a GAMBIT scan increases significantly when full collider simulations with ColliderBit are included. For this example scan we therefore only vary the mass m_Y of the mediator particle. The simulations are performed using the GUM-generated Pythia interface, as described in Sects. 2.2.4 and 2.3.5. For each parameter point in the scan we generate 12 million Pythia events. The events are then passed through fast detector simulation in ColliderBit and selection cuts emulating the ATLAS search. This search targets events with two opposite-charge leptons, jets and missing transverse momentum. No large excesses are observed in this search; across all signal regions the observed event counts agree with the Standard Model expectations to around the 2σ level.

The ATLAS analysis defines both exclusive and inclusive signal regions based on the 'stransverse mass' kinematic variable and the signal lepton flavours. For our scan we consider the exclusive signal regions. There is no publicly available full likelihood function for this analysis, nor any data on correlations, and we therefore take the conservative approach of only using the likelihood contribution from the single signal region with the best expected sensitivity at each point in our scan.

Figure 6 shows the resulting ATLAS likelihood function in our scan of the mediator mass m_Y, with the other model parameters set to $m_\chi = 1$ GeV and $c_Y = g_\chi = 1$. Following the standard approach in ColliderBit, we show the log-likelihood difference $\Delta \ln L = \ln L(s+b) - \ln L(b)$, where $L(s+b)$ denotes the likelihood when the predicted DM signal

(s) is added on top of the SM background expectation (b), and $L(b)$ is the likelihood for the background-only prediction. Further details on the likelihood evaluation in **ColliderBit** are given in Ref. [5]. The red dashed line shows $\Delta \ln L = -2$, corresponding to the $\Delta \ln L$ limit for the approximate 2σ confidence interval on m_Y. For $c_Y = g_\chi = 1$ and $m_\chi = 1\,\mathrm{GeV}$, mediator masses below $\sim 250\,\mathrm{GeV}$ are disfavoured at the 2σ level, in agreement with the constraint that the ATLAS analysis obtains on the mediator mass in a similar BSM scenario. When compared to the $[45, 10^4]\,\mathrm{GeV}$ scan range for m_Y in the MDMSM scan in Sect. 4, and further taking into account that the four-dimensional scan in Sect. 4 allowed couplings as small as 10^{-4}, we see that the collider likelihood would have had a minimal impact on the profile likelihood results of that scan.

Appendix B: New backend interfaces

B.1 CalcHEP

CalcHEP provides squared matrix elements for a given process at tree level. The **GAMBIT** interface to **CalcHEP** contains two simple convenience functions, `CH_Decay_Width` and `CH_Sigma_V`, which apply the correct kinematics to convert a matrix element into a $1 \rightarrow 2$ decay width, or a $2 \rightarrow 2$ DM annihilation cross-section.

The function `CH_Decay_Width` is used by **DecayBit** to add a new `Entry` to its `DecayTable`. To obtain the decay width, one simply passes the name of the model and the decaying particle as they are known internally in **CalcHEP**, along with a `std::vector<std::string>` containing the names of the decay products (also as known to **CalcHEP**). Note that at present only two-body final states are allowed by **CalcHEP**, but the interface generalises nearly automatically to higher-multiplicity final states.

The function `CH_Sigma_V` returns the product σv_{lab} for DM annihilation $\chi + \overline{\chi} \rightarrow A + B$. It does not support co-annihilations. This function is used by the **DarkBit** Process Catalogue. The arguments for `CH_Sigma_V` are identical to `CH_Decay_Width`, except that the in states must be a `std::vector<std::string>` containing the DM candidate and its conjugate. The function also requires the relative velocity in the centre-of-mass frame `double v_rel` (in units of c), and the `DecayTable`, to pass updated mediator widths to **CalcHEP**.

For matrix elements with numerical instabilities for zero relative velocity, we compute the cross-section at a reference velocity of $v_{\mathrm{lab}} = 1 \times 10^{-6}$.

B.2 SARAH-SPheno

SPheno is a spectrum generator capable of providing one-loop mass spectra as well as decay rates at tree and loop level.

GAMBIT has included a frontend interface to the release version of **SPheno** 3.3.8 since **GAMBIT** 1.0, and to 4.0.3 since **GAMBIT** 1.5. Details about the interface can be found in Appendix B of [8]. There are important differences between the frontend interfaces to the release version of **SPheno** and to the **SARAH**-generated version (which we refer to as **SARAH-SPheno**). We give details of these differences below.

SARAH generates the **Fortran SPheno** files to compute the spectrum and decays for a given model. These differ from the out-of-the-box **SPheno**, which only works with various versions of the MSSM. After generating these files with **SARAH**, **GUM** moves them to the main **GAMBIT** directory, to be combined with the downloaded version of **SPheno** at build time.

In order to improve the usability of **SARAH-SPheno** in **GAMBIT**, we have patched two variables into the **Fortran** code. The first, `ErrorHandler_cptr`, is a pointer to a `void` function that returns control to **GAMBIT** after a call to the **SPheno** subroutine `TerminateProgram`. This prevents **GAMBIT** being terminated when **SPheno** fails. Instead, it raises an `invalid_point` exception, and carries on. The second new variable is `SilenceOutput`, which provides a **GAMBIT** input option that allows the user to silence all output of **SARAH-SPheno** to `stdout`. This option defaults to **false**.

```
Rules:
 - capability: unimproved_new_model_spectrum
   function: get_new_model_spectrum_SPheno
   options:
     SilenceOutput: true #default: false
```

The interface to the spectrum computation from **SARAH-SPheno** remains fairly similar to that described for the release version of **SPheno** in Ref. [8]. Some variables and functions have changed names and library symbols. The computations have been re-ordered slightly, but otherwise remain unperturbed.

The major change to the spectrum is the computation of mass uncertainties, while the previous **SPheno** interface merely applied a universal uncertainty to all masses. These uncertainties are computed by **SPheno** for all spectrum masses and added to the **GAMBIT** spectrum object if requested, using the option `GetMassUncertainty`. This option defaults to **false**.

```
Rules:
 - capability: unimproved_new_model_spectrum
   function: get_new_model_spectrum_SPheno
   options:
     GetMassUncertainty: true #default: false
```

Setting `GetMassUncertainty:true` causes the mass uncertainties to be added to the spectrum in the SLHA block `DMASS`.

The most significant difference between the frontend interface to **SARAH-SPheno** compared to the release

version of **SPheno** is that the former includes computation of decays. The backend convenience function `run_SPheno_decays` provides a new capability `SARAHSPheno_`*`new_model`*`_decays`, which maps the decay widths computed by **SPheno** into a **GAMBIT** `DecayTable`. Internally, this backend function fills the **SARAH-SPheno** internal variables with the spectrum details and computes all the branching fractions using the **SARAH-SPheno** function `CalculateBR`. Note that the branching fractions for charged processes are rescaled within the frontend interface, as they are double-counted in **SPheno** so must be rescaled by a factor of 1/2. Various **GAMBIT** input options are added for the computation of the decays, the most notable of which are

- `OneLoopDecays`, which switches on the alternate computation of full one-loop decays,
- `MinWidth`, which specifies the minimum width value in GeV for a decay to be added to the table, and

These can be given as options of the `decay_rates` capability as

```
- capability: decay_rates
  function: all_new_model_decays_from_SPheno
  options:
    OneLoopDecays: false   #default: false
    MinWidth: 1e-10        #default: 1e-30
```

Lastly, the new **SARAH-SPheno** interface provides information about Higgs couplings via the backend convenience function `get_HiggsCouplingsTable`, which provides the capability `SARAHSPheno_`*`new_model`*`_HiggsCouplingsTable`. This function simply fills a GAM-BIT `HiggsCouplingsTable` object from various internal variables in SARAH-SPheno.

B.3 Vevacious (C++)

Vevacious computes the stability of the EWSB vacuum in BSM theories when deeper vacua exist. It does so by first finding all minima of the tree-level potential, calculating one-loop and thermal corrections, and computing the likelihood for our vacuum not to have decayed before the present epoch. **Vevacious** has been recently rewritten in C++ and without dependence on external tools for the tunnelling calculation. This is the version that **GUM** uses. The **GAMBIT** interface for **Vevacious** (C++) is described in detail in [85], so we will only summarise it briefly here.

Out of the box, **Vevacious** simply requires an SLHA2 file as input. To avoid file operations, from **GAMBIT** the spectrum object is passed to the central **Vevacious** object by the capability `pass_spectrum_to_vevacious`. It is this capability in **SpecBit** for which **GUM** can write a new module function for each model.

Besides the spectrum, **Vevacious** requires other information to initialise prior to running the main routines. The various operational options for **Vevacious** are set via the module function `initialize_vevacious` within **SpecBit**. These are described in Table 3. The specific minima to which **Vevacious** must compute the tunnelling probability are extracted from the **GAMBIT** input file by the `panic_vacua` capability, and the specific tunnelling strategy by `tunnelling_strategy`. The capability

Table 3 Table of the `runOptions` available to the module function `initialize_vevacious`, which is used to pass runtime options to **Vevacious**

Option	Utility and default value
`phc_random_seed`	Set the seed for the routines in **PHC**. Default: generated by **GAMBIT**
`minuit_strategy`	Select the strategy for **MINUIT** when minimising the one-loop potential and finding the optimal tunneling path. Higher values mean more function calls and more accuracy. Possible values: 0, 1, 2. Default: 0
`potential_type`	Selects the potential class within **Vevacious**. Current available options are (default) `FixedScaleOneLoopPotential`, `PotentialFromPolynomialWithMasses`
`homotopy_backend`	Selects choice of software to perform the homotopy continuation, choice of `hom4ps` and `phc`. Default: `hom4ps`
`path_finding_timeout`	Maximum time spent trying to find the optimal tunneling path, in seconds. Default: 3600
`survival_probability_threshold`	The threshold probability for which **Vevacious** stops trying to find a lower bounce action, as a fraction of age of the Universe. Default: 0.01
`radial_resolution_undershoot_overshoot`	Sets the choice of length scale resolution for the numerical integration of bounce solutions. Default: 0.1
`PathResolution`	Number of equally-spaced nodes along candidate paths in field space that are moved for finding the optimal tunneling path. Default: 1000

`compare_panic_vacua` checks whether the minima requested are identical.

The main **Vevacious** computations are performed using the method `RunPoint` from the `VevaciousPlusPlus` class, native to **Vevacious**. **GAMBIT** has access to this class dynamically via the class structure generated by **BOSS** and calls this method in the capability `check_vacuum_stability_vevacious`.

The likelihood of tunnelling to any minimum is provided by the capability `VS_likelihood`. The selection of which minimum to which to compute the transition and the tunnelling strategy is done by using its `sub_capabilities`, as described in Sect. 2.3.2. Lastly, the details of the tunnelling computations by **Vevacious** can be extracted as a map using the capability `VS_results`.

References

1. **GAMBIT** Collaboration: P. Athron, C. Balázs et al., GAMBIT: the global and modular beyond-the-standard-model inference tool. Eur. Phys. J. C **77**, 784 (2017). arXiv:1705.07908 [Addendum in [113]]

2. A. Kvellestad, P. Scott, M. White, GAMBIT and its application in the search for physics beyond the standard model. Prog. Part. Nucl. Phys. **113**, 103769 (2020). arXiv:1912.04079

3. **GAMBIT** Scanner Workgroup: G.D. Martinez, J. McKay et al., Comparison of statistical sampling methods with ScannerBit, the GAMBIT scanning module. Eur. Phys. J. C **77**, 761 (2017). arXiv:1705.07959

4. **GAMBIT** Dark Matter Workgroup: T. Bringmann, J. Conrad et al., DarkBit: a GAMBIT module for computing dark matter observables and likelihoods. Eur. Phys. J. C **77**, 831 (2017). arXiv:1705.07920

5. **GAMBIT** Collider Workgroup: C. Balázs, A. Buckley et al., ColliderBit: a GAMBIT module for the calculation of high-energy collider observables and likelihoods. Eur. Phys. J. C **77**, 795 (2017). arXiv:1705.07919

6. **GAMBIT** Flavour Workgroup: F.U. Bernlochner, M. Chrząszcz et al., FlavBit: a GAMBIT module for computing flavour observables and likelihoods. Eur. Phys. J. C **77**, 786 (2017). arXiv:1705.07933

7. M. Chrzaszcz, M. Drewes et al., A frequentist analysis of three right-handed neutrinos with GAMBIT. Eur. Phys. J. C **80**, 569 (2020). arXiv:1908.02302

8. **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. Eur. Phys. J. C **78**, 22 (2018). arXiv:1705.07936

9. **GAMBIT** Cosmology Workgroup: J.J. Renk, P. Stöcker et al., CosmoBit: a GAMBIT module for computing cosmological observables and likelihoods. JCAP **02**, 022 (2021). arXiv:2009.03286

10. **GAMBIT** Collaboration: P. Athron, C. Balázs et al., A global fit of the MSSM with GAMBIT. Eur. Phys. J. C **77**, 879 (2017). arXiv:1705.07917

11. **GAMBIT** Collaboration: P. Athron, C. Balázs et al., Global fits of GUT-scale SUSY models with GAMBIT. Eur. Phys. J. C **77**, 824 (2017). arXiv:1705.07935

12. **GAMBIT** Collaboration: P. Athron, C. Balázs et al., Status of the scalar singlet dark matter model. Eur. Phys. J. C **77**, 568 (2017). arXiv:1705.07931

13. J. Bhom, M. Chrzaszcz et al., A model-independent analysis of $b \rightarrow s\mu^+\mu^-$ transitions with GAMBIT's FlavBit. arXiv:2006.03489

14. P. Athron, J.M. Cornell et al., Impact of vacuum stability, perturbativity and XENON1T on global fits of $\mathbb{Z}_2$ and $\mathbb{Z}_3$ scalar singlet dark matter. Eur. Phys. J. C **78**, 830 (2018). arXiv:1806.11281

15. GAMBIT Collaboration: P. Athron et al., Global analyses of Higgs portal singlet dark matter models using GAMBIT. Eur. Phys. J. C **79**, 38 (2019). arXiv:1808.10465

16. GAMBIT: P. Athron et. al., Thermal WIMPs and the scale of new physics: global fits of Dirac dark matter effective field theories, Eur. Phys. J. C **81**, 992 (2021). arXiv:2106.02056

17. GAMBIT: T.E. Gonzalo, BSM global fits with GAMBIT: a Dark Matter EFT fit, in *55th Rencontres de Moriond on QCD and High Energy Interactions* (2021). arXiv:2106.03490

18. S. Hoof, F. Kahlhoefer, P. Scott, C. Weniger, M. White, Axion global fits with Peccei–Quinn symmetry breaking before inflation using GAMBIT. JHEP **03**, 191 (2019). arXiv:1810.07192

19. GAMBIT Collaboration: P. Athron et al., Combined collider constraints on neutralinos and charginos. Eur. Phys. J. C **79**, 395 (2019). arXiv:1809.02097

20. GAMBIT Cosmology Workgroup: P. Stöcker et. al., Strengthening the bound on the mass of the lightest neutrino with terrestrial and cosmological experiments, Phys. Rev. D **103**, 123508 (2021). arXiv:2009.03287

21. T.E. Gonzalo, GAMBIT: the global and modular BSM inference tool, in *Tools for High Energy Physics and Cosmology* (2021). arXiv:2105.03165

22. A.V. Semenov, LanHEP: a package for automatic generation of Feynman rules in gauge models. arXiv:9608488 [hep-ph]

23. A. Semenov, LanHEP: a package for automatic generation of Feynman rules from the Lagrangian. Comput. Phys. Commun. **115**, 124–139 (1998)

24. A.V. Semenov, LanHEP: a package for automatic generation of Feynman rules in field theory. Version 2.0. arXiv:hep-ph/0208011

25. A. Semenov, LanHEP a package for the automatic generation of Feynman rules in field theory. Version 3.0. Comput. Phys. Commun. **180**, 431–454 (2009). arXiv:0805.0555

26. E.E. Boos, M.N. Dubinin, V.A. Ilyin, A.E. Pukhov, V.I. Savrin, CompHEP: specialized package for automatic calculations of elementary particle decays and collisions (1994). arXiv:hep-ph/9503280

27. A. Pukhov, E. Boos et al., CompHEP: a package for evaluation of Feynman diagrams and integration over multiparticle phase space. arXiv:hep-ph/9908288

28. CompHEP: E. Boos, V. Bunichev et al., CompHEP 4.4: automatic computations from Lagrangians to events. Nucl. Instrum. Methods A **534**, 250–259 (2004). arXiv:hep-ph/0403113

29. N.D. Christensen, C. Duhr, FeynRules—Feynman rules made easy. Comput. Phys. Commun. **180**, 1614–1641 (2009). arXiv:0806.4194

30. N.D. Christensen, P. de Aquino et al., A comprehensive approach to new physics simulations. Eur. Phys. J. C **71**, 1541 (2011). arXiv:0906.2474

31. N.D. Christensen, C. Duhr, B. Fuks, J. Reuter, C. Speckner, Introducing an interface between WHIZARD and FeynRules. Eur. Phys. J. C **72**, 1990 (2012). arXiv:1010.3251

32. A. Alloul, N.D. Christensen, C. Degrande, C. Duhr, B. Fuks, FeynRules 2.0—a complete toolbox for tree-level phenomenology. Comput. Phys. Commun. **185**, 2250–2300 (2014). arXiv:1310.1921

33. T. Stelzer, W.F. Long, Automatic generation of tree level helicity amplitudes. Comput. Phys. Commun. **81**, 357–371 (1994). arXiv:hep-ph/9401258

34. F. Maltoni, T. Stelzer, MadEvent: automatic event generation with MadGraph. JHEP **02**, 027 (2003). arXiv:hep-ph/0208156

35. J. Alwall, P. Demin et al., MadGraph/MadEvent v4: the new web generation. JHEP **09**, 028 (2007). arXiv:0706.2334

36. J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, T. Stelzer, MadGraph 5: going beyond. JHEP **06**, 128 (2011). arXiv:1106.0522

37. 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

38. A. Pukhov, CalcHEP 2.3: MSSM, structure functions, event generation, batchs, and generation of matrix elements for other packages. arXiv:hep-ph/0412191

39. A. Belyaev, N.D. Christensen, A. Pukhov, CalcHEP 3.4 for collider physics within and beyond the Standard Model. Comput. Phys. Commun. **184**, 1729–1769 (2013). arXiv:1207.6082

40. T. Hahn, M. Perez-Victoria, Automatized one loop calculations in four-dimensions and D-dimensions. Comput. Phys. Commun. **118**, 153–165 (1999). arXiv:hep-ph/9807565

41. T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3. Comput. Phys. Commun. **140**, 418–431 (2001). arXiv:hep-ph/0012260

42. T. Hahn, Automatic loop calculations with FeynArts, FormCalc, and LoopTools. Nucl. Phys. Proc. Suppl. **89**, 231–236 (2000). arXiv:hep-ph/0005029

43. T. Hahn, C. Schappacher, The implementation of the minimal supersymmetric standard model in FeynArts and FormCalc. Comput. Phys. Commun. **143**, 54–68 (2002). arXiv:hep-ph/0105349

44. T. Gleisberg, S. Hoeche et al., Event generation with SHERPA 1.1. JHEP **02**, 007 (2009). arXiv:0811.4622

45. W. Kilian, T. Ohl, J. Reuter, WHIZARD: simulating multi-particle processes at LHC and ILC. Eur. Phys. J. C **71**, 1742 (2011). arXiv:0708.4233

46. M. Moretti, T. Ohl, J. Reuter, O'Mega: an optimizing matrix element generator. arXiv:hep-ph/0102195

47. F. Staub, *SARAH*. arXiv:0806.0538

48. F. Staub, From superpotential to model files for FeynArts and CalcHep/CompHep. Comput. Phys. Commun. **181**, 1077–1086 (2010). arXiv:0909.2863

49. F. Staub, Automatic calculation of supersymmetric renormalization group equations and self energies. Comput. Phys. Commun. **182**, 808–833 (2011). arXiv:1002.0840

50. F. Staub, SARAH 3.2: Dirac Gauginos, UFO output, and more. Comput. Phys. Commun. **184**, 1792–1809 (2013). arXiv:1207.0906

51. F. Staub, SARAH 4: a tool for (not only SUSY) model builders. Comput. Phys. Commun. **185**, 1773–1790 (2014). arXiv:1309.7223

52. F. Staub, Exploring new models in all detail with SARAH. Adv. High Energy Phys. **2015**, 840780 (2015). arXiv:1503.04200

53. C. Degrande, C. Duhr et al., UFO—the universal FeynRules output. Comput. Phys. Commun. **183**, 1201–1214 (2012). arXiv:1108.2040

54. G. Cullen, N. Greiner et al., Automated one-loop calculations with GoSam. Eur. Phys. J. C **72**, 1889 (2012). arXiv:1111.2034

55. G. Cullen et al., GOSAM-2.0: a tool for automated one-loop calculations within the Standard Model and beyond. Eur. Phys. J. C **74**, 3001 (2014). arXiv:1404.7096

56. M. Bahr et al., Herwig++ physics and manual. Eur. Phys. J. C **58**, 639–707 (2008). arXiv:0803.0883

57. J. Bellm et al., Herwig 7.0/Herwig++ 3.0 release note. Eur. Phys. J. C **76**, 196 (2016). arXiv:1512.01178

58. M. Backovic, K. Kong, M. McCaskey, MadDM v.1.0: computation of dark matter relic abundance using MadGraph5. Phys. Dark Universe **5–6**, 18–28 (2014). arXiv:1308.4955

59. M. Backović, A. Martini, O. Mattelaer, K. Kong, G. Mohlabeng, Direct detection of dark matter with MadDM v.2.0. Phys. Dark Universe **910**, 37–50 (2015). arXiv:1505.04190

60. F. Ambrogi, C. Arina, et. al., MadDM v.3.0: a Comprehensive Tool for Dark Matter Studies, Phys. Dark Univ. **24**, 100249 (2019). arXiv:1804.00044

61. G. Bélanger, F. Boudjema, A. Pukhov, A. Semenov, MicrOMEGAs: a program for calculating the relic density in the MSSM. Comput. Phys. Commun. **149**, 103–120 (2002). arXiv:hep-ph/0112278

62. G. Bélanger, F. Boudjema, A. Pukhov, A. Semenov, micrOMEGAs: version 1.3. Comput. Phys. Commun. **174**, 577–604 (2006). arXiv:hep-ph/0405253

63. G. Bélanger, F. Boudjema, A. Pukhov, A. Semenov, MicrOMEGAs 2.0: a program to calculate the relic density of dark matter in a generic model. Comput. Phys. Commun. **176**, 367–382 (2007). arXiv:hep-ph/0607059

64. G. Bélanger, F. Boudjema, A. Pukhov, A. Semenov, Dark matter direct detection rate in a generic model with micrOMEGAs 2.2. Comput. Phys. Commun. **180**, 747–767 (2009). arXiv:0803.2360

65. G. Bélanger, F. Boudjema et al., Indirect search for dark matter with micrOMEGAs2.4. Comput. Phys. Commun. **182**, 842–856 (2011). arXiv:1004.1092

66. G. Bélanger, F. Boudjema, A. Pukhov, A. Semenov, micrOMEGAs 3: a program for calculating dark matter observables. Comput. Phys. Commun. **185**, 960–985 (2014). arXiv:1305.0237

67. G. Bélanger, F. Boudjema, A. Pukhov, A. Semenov, micrOMEGAs4.1: two dark matter candidates. Comput. Phys. Commun. **192**, 322–329 (2015). arXiv:1407.6129

68. W. Porod, SPheno, a program for calculating supersymmetric spectra, SUSY particle decays and SUSY particle production at e^+e^- colliders. Comput. Phys. Commun. **153**, 275–315 (2003). arXiv:hep-ph/0301101

69. W. Porod, F. Staub, SPheno 3.1: extensions including flavour, CP-phases and models beyond the MSSM. Comput. Phys. Commun. **183**, 2458–2469 (2012). arXiv:1104.1573

70. 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

71. P. Athron, M. Bach et al., FlexibleSUSY 2.0: extensions to investigate the phenomenology of SUSY and non-SUSY models. Comput. Phys. Commun. **230**, 145–217 (2018). arXiv:1710.03760

72. P. Athron, A. Büchner et al., FlexibleDecay: an automated calculator of scalar decay widths. arXiv:2106.05038

73. W. Porod, F. Staub, A. Vicente, A flavor kit for BSM models. Eur. Phys. J. C **74**, 2992 (2014). arXiv:1405.1434

74. J.E. Camargo-Molina, B. O'Leary, W. Porod, F. Staub, Vevacious: a tool for finding the global minima of one-loop effective potentials with many scalars. Eur. Phys. J. C **73**, 2588 (2013). arXiv:1307.1477

75. F. Bishara, J. Brod, B. Grinstein, J. Zupan, DirectDM: a tool for dark matter direct detection. arXiv:1708.02678

76. J. Brod, A. Gootjes-Dreesbach, M. Tammaro, J. Zupan, Effective field theory for dark matter direct detection up to dimension seven. JHEP **10**, 065 (2018). arXiv:1710.10218

77. T. Sjostrand, S. Ask et al., An introduction to PYTHIA 8.2. Comput. Phys. Commun. **191**, 159–177 (2015). arXiv:1410.3012

78. 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

79. 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

80. 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

81. 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

82. 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

83. P. Bechtle, S. Heinemeyer, O. Stål, T. Stefaniak, G. Weiglein, Probing the Standard Model with Higgs signal rates from the Tevatron, the LHC and a future ILC. JHEP **11**, 039 (2014). arXiv:1403.1582

84. https://github.com/JoseEliel/VevaciousPlusPlus. Accessed 25 May 2021

85. J.E. Camargo-Molina, B. Farmer etal., Vacuum stability in the otherwise allowed parameter regions of the CMSSM and NUHM1 SUSY-GUT Scenarios, in preparation (2021)

86. F. James, M. Roos, Minuit: a system for function minimization and analysis of the parameter errors and correlations. Comput. Phys. Commun. **10**, 343–367 (1975)

87. J. Verschelde, Polynomial homotopy continuation with PHCpack. ACM Commun. Comput. Algebra **44**, 217–220 (2011)

88. T.L. Lee, T.Y. Li, C.H. Tsai, HOM4ps-2.0: a software package for solving polynomial systems by the polyhedral homotopy continuation method. Computing **83**, 109–133 (2008)

89. G. Bélanger, F. Boudjema, A. Goudelis, A. Pukhov, B. Zaldivar, micrOMEGAs5.0: freeze-in. Comput. Phys. Commun. **231**, 173–186 (2018). arXiv:1801.03509

90. P. Scott, C. Savage, J. Edsjö, 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

91. 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

92. S. Catani, Y.L. Dokshitzer, M. Seymour, B. Webber, Longitudinally invariant K_t clustering algorithms for hadron hadron collisions. Nucl. Phys. B **406**, 187–224 (1993)

93. S.D. Ellis, D.E. Soper, Successive combination jet algorithm for hadron collisions. Phys. Rev. D **48**, 3160–3166 (1993). arXiv:hep-ph/9305266

94. Particle Data Group: M. Tanabashi et al., Review of particle physics. Phys. Rev. D **98**, 030001 (2018)

95. P. Scott, Pippi—painless parsing, post-processing and plotting of posterior and likelihood samples. Eur. Phys. J. Plus **127**, 138 (2012). arXiv:1206.2245

96. M.R. Buckley, D. Feld, D. Goncalves, Scalar simplified models for dark matter. Phys. Rev. D **91**, 015017 (2015). arXiv:1410.6497

97. G. Arcadi, M. Dutra et al., The waning of the WIMP? A review of models, searches, and constraints. Eur. Phys. J. C **78**, 203 (2018). arXiv:1703.07364

98. J. Abdallah et al., Simplified models for dark matter searches at the LHC. Phys. Dark Universe **9–10**, 8–23 (2015). arXiv:1506.03116

99. CMS: A.M. Sirunyan et al., Search for dark matter produced in association with a single top quark or a top quark pair in proton–proton collisions at $\sqrt{s} = 13$ TeV. JHEP **03**, 141 (2019). arXiv:1901.01553

100. ATLAS: M. Aaboud et al., Constraints on mediator-based dark matter and scalar dark energy models using $\sqrt{s} = 13$ TeV pp collision data collected by the ATLAS detector. JHEP **05**, 142 (2019). arXiv:1903.01400

101. XENON: E. Aprile et al., Dark matter search results from a one ton-year exposure of XENON1T. Phys. Rev. Lett. **121**, 111302 (2018). arXiv:1805.12562

102. D.S. Akerib, H.M. Araújo et al., Improved limits on scattering of weakly interacting massive particles from reanalysis of 2013 LUX data. Phys. Rev. Lett. **116**, 161301 (2016). arXiv:1512.03506

103. Fermi-LAT: M. Ackermann et al., Searching for dark matter annihilation from Milky Way dwarf spheroidal galaxies with six years of Fermi large area telescope data. Phys. Rev. Lett. **115**, 231301 (2015). arXiv:1503.02641

104. P. Gondolo, J. Edsjo et al., DarkSUSY: computing supersymmetric dark matter properties numerically. JCAP **0407**, 008 (2004). arXiv:astro-ph/0406204

105. T. Bringmann, J. Edsjö, P. Gondolo, P. Ullio, L. Bergström, DarkSUSY 6: an advanced tool to compute dark matter properties numerically. JCAP **1807**, 033 (2018). arXiv:1802.03399

106. Planck Collaboration, P.A.R. Ade et al., Planck 2015 results. XIII. Cosmological parameters. A&A **594**, A13 (2016). arXiv:1502.01589

107. J.M. Cline, K. Kainulainen, P. Scott, C. Weniger, Update on scalar singlet dark matter. Phys. Rev. D **88**, 055025 (2013). arXiv:1306.4710

108. LUX-ZEPLIN: D. S. Akerib et. al., Projected WIMP sensitivity of the LUX-ZEPLIN dark matter experiment, Phys. Rev. D **101**, 052002 (2020). arXiv:1802.06039

109. P. Bechtle, D. Dercks et al., HiggsBounds-5: testing Higgs sectors in the LHC 13 TeV era. Eur. Phys. J. C **80**, 1211 (2020). arXiv:2006.06007

110. P. Bechtle, S. Heinemeyer et al., HiggsSignals-2: probing new physics with precision Higgs measurements in the LHC 13 TeV era. Eur. Phys. J. C **81**, 145 (2021). arXiv:2012.09197

111. ATLAS: G. Aad et. al., Search for dark matter produced in association with a Standard Model Higgs boson decaying into b-quarks using the full Run 2 dataset from the ATLAS detector, arXiv:2108.13391

112. ATLAS: G. Aad et al., Search for new phenomena in events with two opposite-charge leptons, jets and missing transverse momentum in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector. JHEP **2104**, 165 (2021). arXiv:2102.01444

113. GAMBIT Collaboration: P. Athron, C. Balázs et al., GAMBIT: the global and modular beyond-the-standard-model inference tool. Addendum for GAMBIT 1.1: Mathematica backends, SUSYHD interface and updated likelihoods. Eur. Phys. J. C **78**, 98 (2018). arXiv:1705.07908 [Addendum to [1]]