GAMBIT
The Global and Modular BSM Inference Tool

Pat Scott

Imperial College London

on behalf of the GAMBIT Collaboration

Slides at: tinyurl.com/patscott
Code at: gambit.hepforge.org

Question

How do we find dark matter and new particles?

Global Fits

Question

How do we find dark matter and new particles?

Answer

Build colliders, dark matter detectors, satellites and other experiments to hunt for them, then compare results

Global Fits

Question

How do we find dark matter and new particles?

Answer

Build colliders, dark matter detectors, satellites and other experiments to hunt for them, then compare results

Question

How do we tell which theories are good and which are bad?

Global Fits

Question

How do we find dark matter and new particles?

Answer

Build colliders, dark matter detectors, satellites and other experiments to hunt for them, then compare results

Question

How do we tell which theories are good and which are bad?

Answer

Combine the results from different searches

Global Fits

Question

How do we **rigorously** tell which theories are better than others?

Global Fits

Question

How do we **rigorously** tell which theories are better than others?

Answer

1. Keep the **full likelihood functions**, include correlations, systematics, theoretical errors + consistent theory treatments across all experiments.

2. Test **complete** parameter spaces, not just single points!

3. Test **many different theories** the same way!

Global Fits

Question

How do we **rigorously** tell which theories are better than others?

Answer

1. Keep the **full likelihood functions**, include correlations, systematics, theoretical errors + consistent theory treatments across all experiments.
   (composite likelihood)
2. Test **complete** parameter spaces, not just single points!
3. Test **many different theories** the same way!

Global Fits

Question

How do we **rigorously** tell which theories are better than others?

Answer

1. Keep the **full likelihood functions**, include correlations, systematics, theoretical errors + consistent theory treatments across all experiments.
   (composite likelihood)
2. Test **complete** parameter spaces, not just single points!
   (parameter estimation)
3. Test **many different theories** the same way!

Global Fits

Question

How do we **rigorously** tell which theories are better than others?

Answer

1. Keep the **full likelihood functions**, include correlations, systematics, theoretical errors + consistent theory treatments across all experiments.
   (composite likelihood)
2. Test **complete** parameter spaces, not just single points!
   (parameter estimation)
3. Test **many different theories** the same way!
   (model comparison)

Global Fits

Question

How do we **rigorously** tell which theories are better than others?

Answer

1. Keep the **full likelihood functions**, include correlations, systematics, theoretical errors + consistent theory treatments across all experiments.
   (composite likelihood)
2. Test **complete** parameter spaces, not just single points!
   (parameter estimation)
3. Test **many different theories** the same way!
   (model comparison)

→ global fits

Global fits for dark matter and new physics

Current global fit codes are hardcoded to deal with only a few

- theories (MSSM and/or mSUGRA+friends)
- theory calculators (often interfaced in a very ad hoc way)
- datasets and observables (often missing detailed likelihoods)
- scanning algorithms and statistical methods (generally just one)

$\Longrightarrow$ *hitting the wall on theories, data & computational methods*

Global fits for dark matter and new physics

Current global fit codes are hardcoded to deal with only a few

- theories (MSSM and/or mSUGRA+friends)
- theory calculators (often interfaced in a very ad hoc way)
- datasets and observables (often missing detailed likelihoods)
- scanning algorithms and statistical methods (generally just one)

$\Longrightarrow$ *hitting the wall on theories, data & computational methods*

How to quickly recast data, likelihood functions, scanning code 'housekeeping' and even theory predictions to new theories?

$\Longrightarrow$ a new, very general global fitting framework

Global fits for dark matter and new physics

Current global fit codes are hardcoded to deal with only a few

- theories (MSSM and/or mSUGRA+friends)
- theory calculators (often interfaced in a very ad hoc way)
- datasets and observables (often missing detailed likelihoods)
- scanning algorithms and statistical methods (generally just one)

$\Longrightarrow$ *hitting the wall on theories, data & computational methods*

How to quickly recast data, likelihood functions, scanning code 'housekeeping' and even theory predictions to new theories?

$\Longrightarrow$ a new, very general global fitting framework

$\Longrightarrow$ GAMBIT

GAMBIT: The Global And Modular BSM Inference Tool

gambit.hepforge.org

- Fast definition of new datasets and theoretical models
- Plug and play scanning, physics and likelihood packages
- Extensive model database – not just SUSY
- Extensive observable/data libraries
- Many statistical and scanning options (Bayesian & frequentist)
- *Fast* LHC likelihood calculator
- Massively parallel
- Fully open-source

ATLAS	F. Bernlochner, A. Buckley, P. Jackson, M. White
LHCb	M. Chrząszcz, N. Serra
Belle-II	F. Bernlochner, P. Jackson
Fermi-LAT	J. Edsjö, G. Martinez, P. Scott
CTA	C. Balázs, T. Bringmann, M. White
CMS	C. Rogan
IceCube	J. Edsjö, P. Scott
XENON/DARWIN	B. Farmer, R. Trotta
Theory	P. Athron, C. Balázs, S. Bloor, T. Bringmann, J. Cornell, J. Edsjö, B. Farmer, A. Fowlie, T. Gonzalo, J. Harz, S. Hoof, F. Kahlhoefer, A. Kvellestad, F.N. Mahmoudi, J. McKay, A. Raklev, R. Ruiz, P. Scott, R. Trotta, A. Vincent, C. Weniger, M. White, S. Wild

29 Members in 9 Experiments, 12 major theory codes, 11 countries

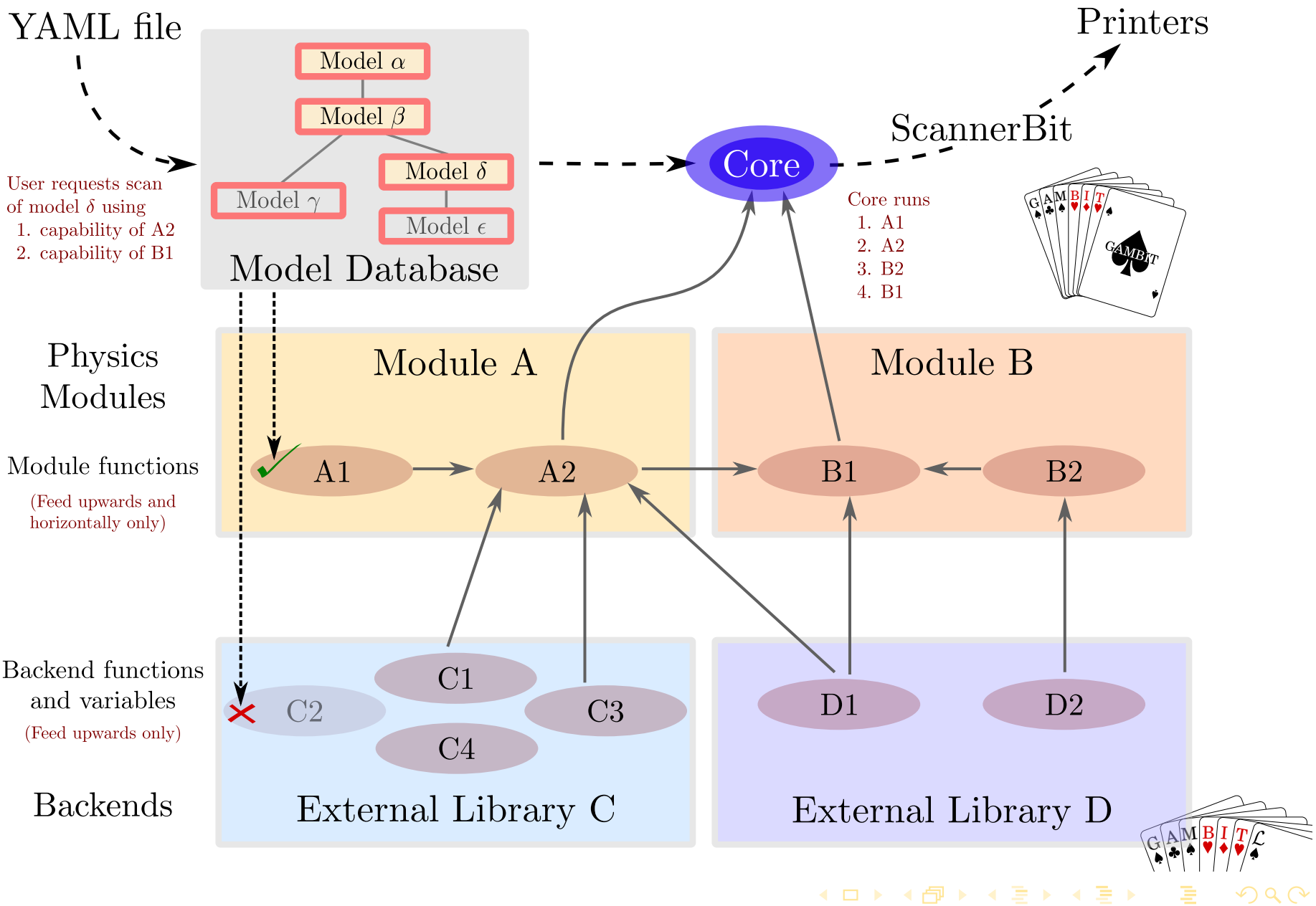
Physics modules

- DarkBit – dark matter observables (relic density, direct + indirect detection) (arXiv:1705.07920)
- ColliderBit – collider observables inc. Higgs + SUSY searches from ATLAS, CMS + LEP (arXiv:1705.07919)
- FlavBit – flavour physics inc. $b \to s\gamma$, B/D decays (new channels, angular obs., theory uncerts, LHCb likelihoods) (arXiv:1705.07933)
- SpecBit – generic BSM spectrum object, providing RGE running, masses, mixings, etc via interchangeable interfaces to different spectrum generators (arXiv:1705.07936)
- DecayBit – decay widths for all relevant SM & BSM particles (arXiv:1705.07936)
- PrecisionBit – SM likelihoods, precision BSM tests (W mass, $g-2$, $\Delta\rho$ etc) (arXiv:1705.07936)

Each consists of a number of **module functions** that can have **dependencies** on each other

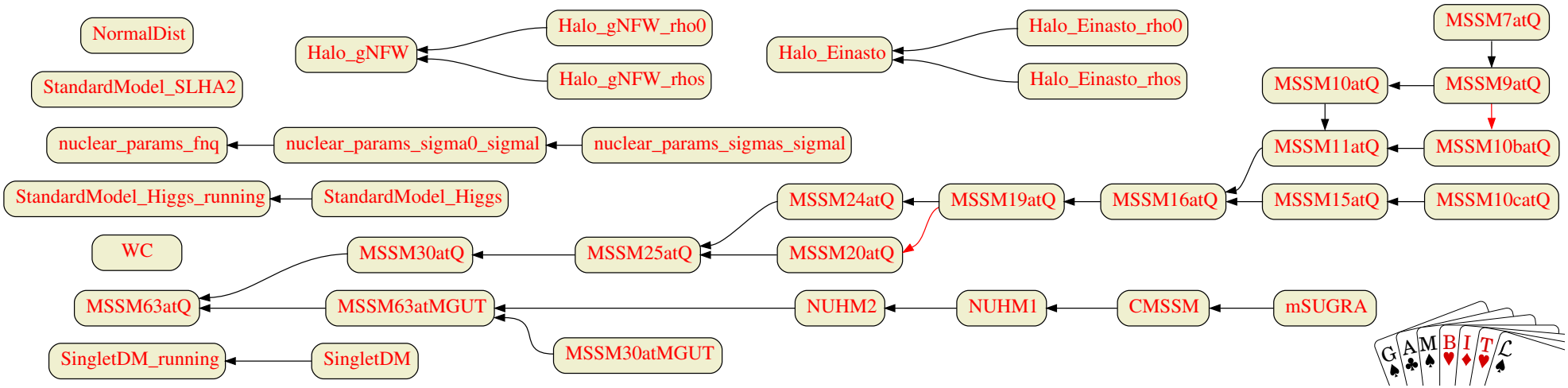
Functional overview of GAMBIT

(arXiv:1705.07908)



- Models are defined by their parameters and relations to each other
- Models can inherit from (be subspaces of) **parent models**
- Points in child models can be **automatically translated** to ancestor models
- **Friend models** also allowed (cross-family translation)
- Model dependence of every function/observable is tracked

  $\Longrightarrow$ maximum safety, maximum reuse



Backends: mix and match (arXiv:1705.07908)

- Module functions can require specific functions from **backends**
- Backends are external code libraries (DarkSUSY, FeynHiggs, Pythia, SUSYHD, etc) that include different functions
- GAMBIT automates and abstracts the interfaces to backends $\rightarrow$ backend functions are tagged according to what they calculate
- $\rightarrow$ with appropriate module design, different backends and their functions can be used interchangeably
- GAMBIT dynamically adapts to use whichever backends are actually present on a user's system (+ provides details of what it decided to do of course)
- backends can be C/C++, Fortran, Mathematica or Python

Backends: mix and match (arXiv:1705.07908)

pat@xpspedition: ~/gambit 163x45

All relative paths are given with reference to /home/pat/gambit.

BACKENDS	VERSION	PATH TO LIB	STATUS	#FUNC	#TYPES	#CTORS
DDCalc0	0.0	Backends/installed/DDCalc/0.0/libDDCalc0.so	OK	62	0	0
DarkSUSY	5.1.1	Backends/installed/DarkSUSY/5.1.1/lib/libdarksusy.so	OK	68	0	0
FastSim	1.0	Backends/installed/fastsim/1.0/libfastsim.so	absent/broken	1	0	0
FeynHiggs	2.11	Backends/installed/FeynHiggs/2.11.2/lib/libFH.so	OK	14	0	0
HiggsBounds	4.2.1	Backends/installed/HiggsBounds/4.2.1/lib/libhiggsbounds.so	OK	10	0	0
HiggsSignals	1.4	Backends/installed/HiggsSignals/1.4.0/lib/libhiggssignals.so	OK	11	0	0
LibFarrayTest	1.0	Backends/examples/libFarrayTest.so	OK	9	0	0
LibFirst	1.0	Backends/examples/libfirst.so	OK	8	0	0
	1.1	Backends/examples/libfirst.so	OK	15	0	0
LibFortran	1.0	Backends/examples/libfortran.so	OK	6	0	0
MicrOmegas	3.5.5	Backends/installed/micromegas/3.5.5/MSSM/MSSM/libmicromegas.so	OK	15	0	0
MicrOmegasSingletDM	3.5.5	Backends/installed/micromegas/3.5.5/SingletDM/SingletDM/libmicromegas.so	OK	13	0	0
Pythia	8.186	Backends/installed/Pythia/8.186/lib/libpythia8.so	absent/broken	0	27	105
	8.209	Backends/installed/Pythia/8.209/lib/libpythia8.so	OK	0	28	107
SUSYPOPE	0.2	no path in config/backend_locations.yaml	absent/broken	3	0	0
SUSY_HIT	1.5	Backends/installed/SUSY-HIT/1.5/libsusyhit.so	OK	55	0	0
SuperIso	3.4	Backends/installed/SuperIso/3.4/libsuperiso.so	OK	32	0	0
gamLike	1.0.0	Backends/installed/gamLike/1.0.0/lib/gamLike.so	OK	3	0	0
nulike	1.0.0	Backends/installed/nulike/1.0.0/lib/libnulike.so	OK	4	0	0

Gambit diagnostic backend line 1 (press h for help or q to quit)

Backends: mix and match

(arXiv:1705.07908)

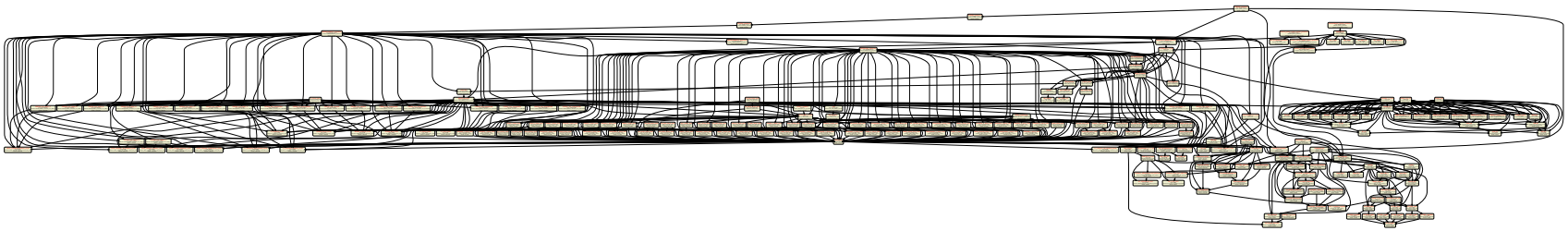
pat@xpspedition: ~/gambit 163x45

All relative paths are given with reference to /home/pat/gambit.

BACKENDS	VERSION	PATH TO LIB	STATUS	#FUNC	#TYPES	#CTORS
DDCalc0	0.0	Backends/installed/DDCalc/0.0/libDDCalc0.so	OK	62	0	0
DarkSUSY	5.1.1	Backends/installed/DarkSUSY/5.1.1/lib/libdarksusy.so	OK	68	0	0
FastSim	1.0	Backends/installed/fastsim/1.0/libfastsim.so	absent/broken	1	0	0
FeynHiggs	2.11	Backends/installed/FeynHiggs/2.11.2/lib/libFH.so	OK	14	0	0
HiggsBounds	4.2.1	Backends/installed/HiggsBounds/4.2.1/lib/libhiggsbounds.so	OK	10	0	0
HiggsSignals	1.4	Backends/installed/HiggsSignals/1.4.0/lib/libhiggssignals.so	OK	11	0	0
LibFarrayTest	1.0	Backends/examples/libFarrayTest.so	OK	9	0	0
LibFirst	1.0	Backends/examples/libfirst.so	OK	8	0	0
	1.1	Backends/examples/libfirst.so	OK	15	0	0
LibFortran	1.0	Backends/examples/libfortran.so	OK	6	0	0
MicrOmegas	3.5.5	Backends/installed/micromegas/3.5.5/MSSM/MSSM/libmicromegas.so	OK	15	0	0
MicrOmegasSingletDM	3.5.5	Backends/installed/micromegas/3.5.5/SingletDM/SingletDM/libmicromegas.so	OK	13	0	0
Pythia	8.186	Backends/installed/Pythia/8.186/lib/libpythia8.so	absent/broken	0	27	105
	8.209	Backends/installed/Pythia/8.209/lib/libpythia8.so	OK	0	28	107
SUSYPOPE	0.2	no path in config/backend_locations.yaml	absent/broken	3	0	0
SUSY_HIT	1.5	Backends/installed/SUSY-HIT/1.5/libsusyhit.so	OK	55	0	0
SuperIso	3.4	Backends/installed/SuperIso/3.4/libsuperiso.so	OK	32	0	0
gamLike	1.0.0	Backends/installed/gamLike/1.0.0/lib/gamLike.so	OK	3	0	0
nulike	1.0.0	Backends/installed/nulike/1.0.0/lib/libnulike.so	OK	4	0	0

Gambit diagnostic backend line 1 (press h for help or q to quit)

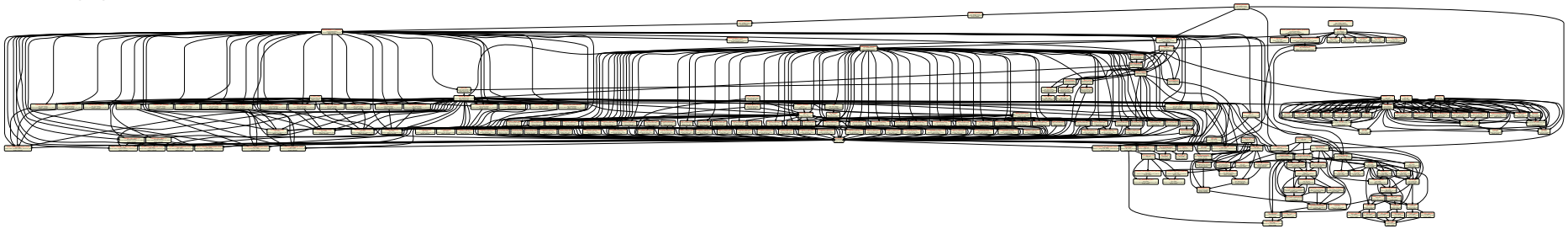
Dependency Resolution (arXiv:1705.07908)



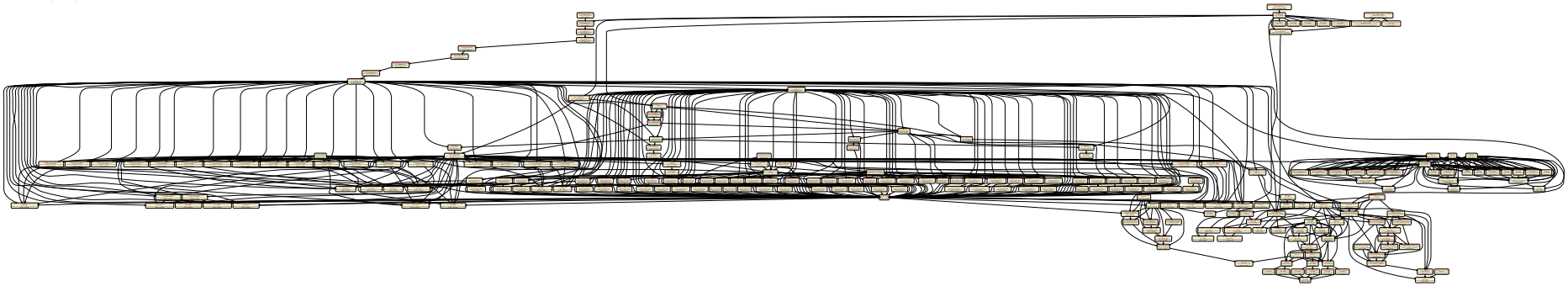
- Module functions and backend functions get arranged into a **dependency tree**
- Starting with requested observables and likelihoods, GAMBIT fills each dependency and backend requirement
- Obeys **rules** at each step: allowed models, allowed backends, constraints from input file, etc
- $\rightarrow$ tree constitutes a directed acyclic graph
- $\rightarrow$ GAMBIT uses graph-theoretic methods to 'solve' the graph to determine function evaluation order

Dependency Resolution (arXiv:1705.07908)

CMSSM:

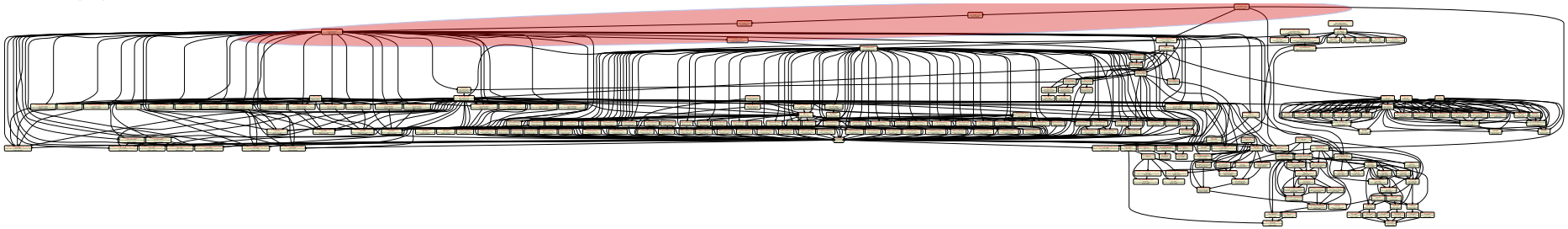


MSSM7:

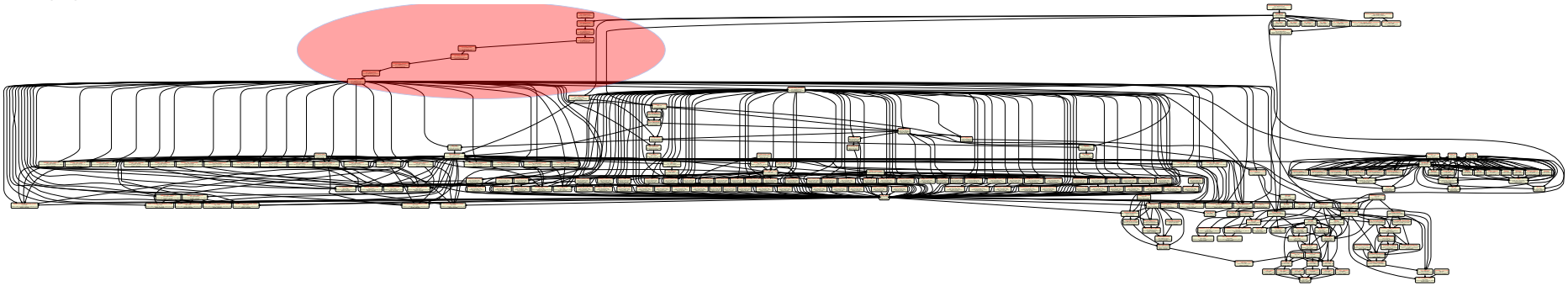


Dependency Resolution (arXiv:1705.07908)

CMSSM:



MSSM7:

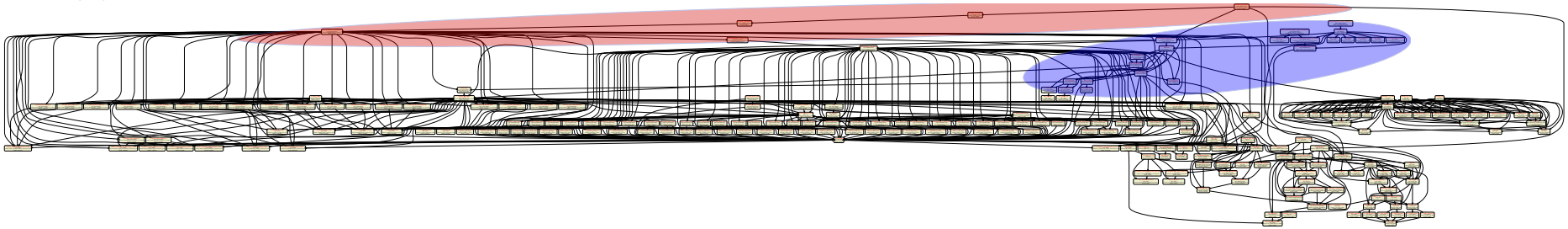


Red: Model parameter translations

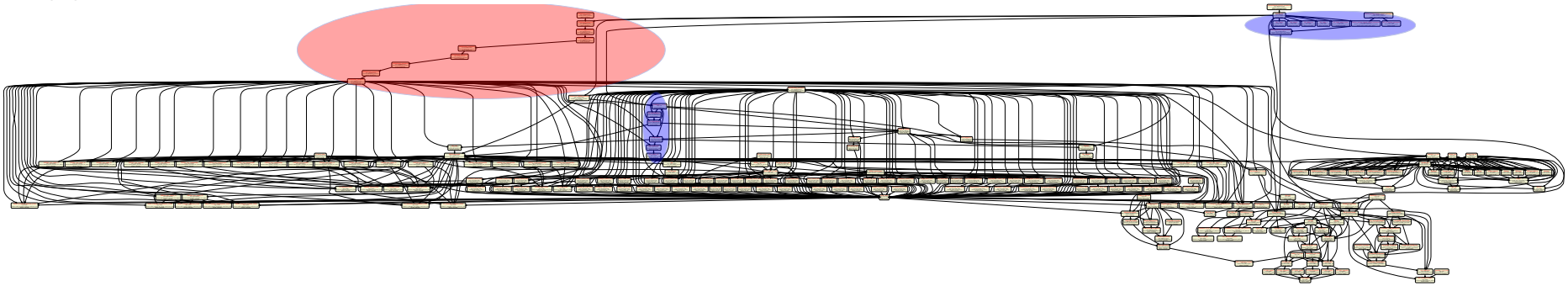
Dependency Resolution

(arXiv:1705.07908)

CMSSM:



MSSM7:



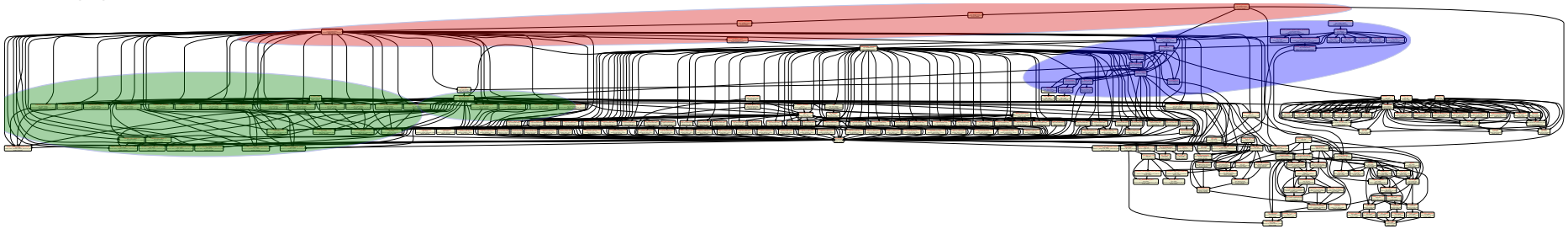
Red: Model parameter translations

Blue: Precision calculations

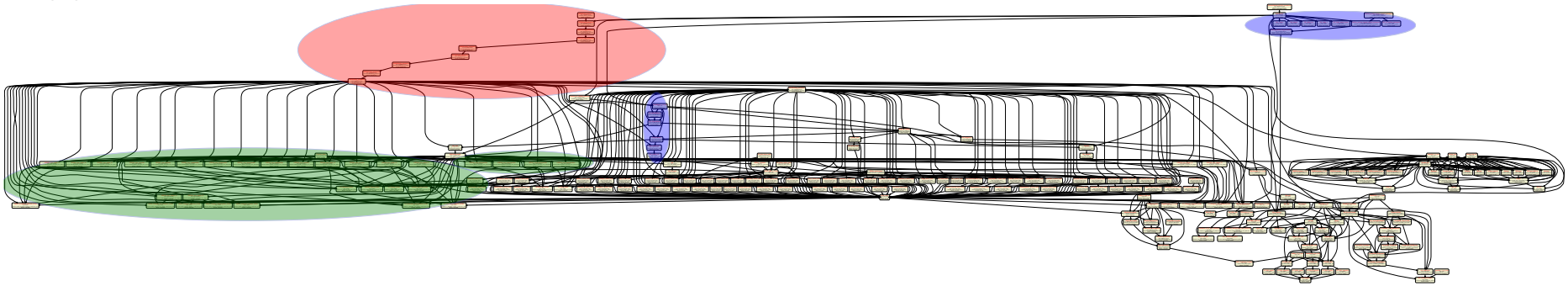
Dependency Resolution

(arXiv:1705.07908)

CMSSM:



MSSM7:



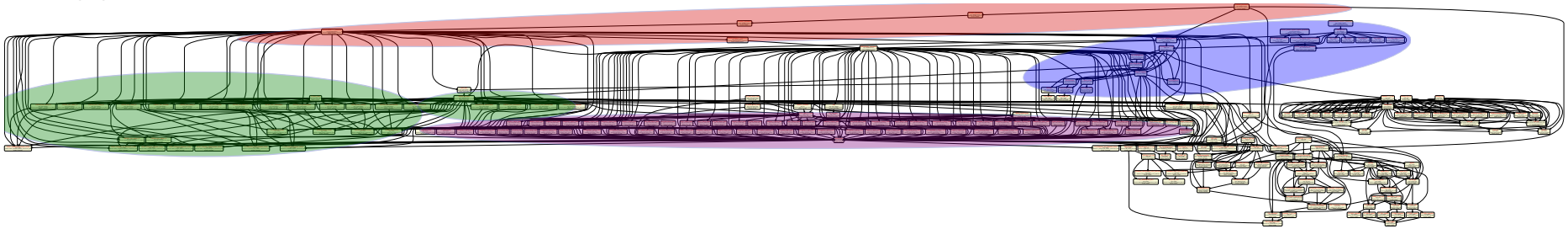
Red: Model parameter translations

Blue: Precision calculations

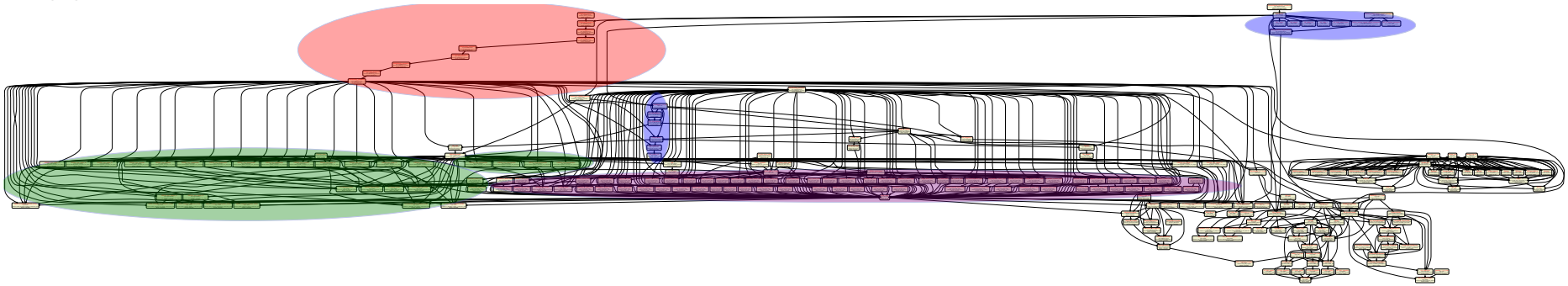
Green: LEP rates+likelihoods

Dependency Resolution (arXiv:1705.07908)

CMSSM:



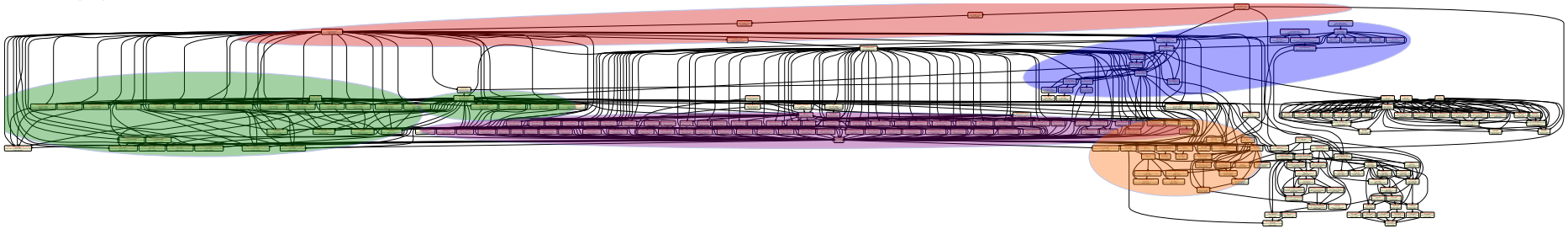
MSSM7:



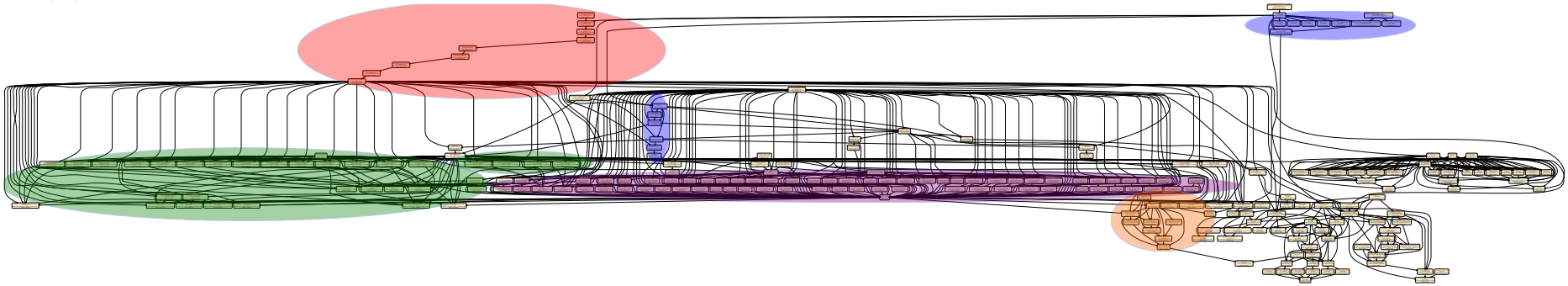
Red: Model parameter translations
Blue: Precision calculations
Green: LEP rates+likelihoods
Purple: Decays

Dependency Resolution (arXiv:1705.07908)

CMSSM:



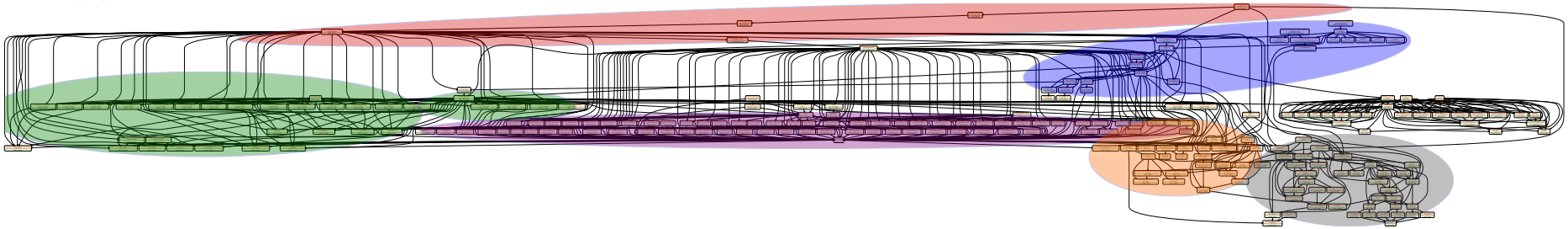
MSSM7:



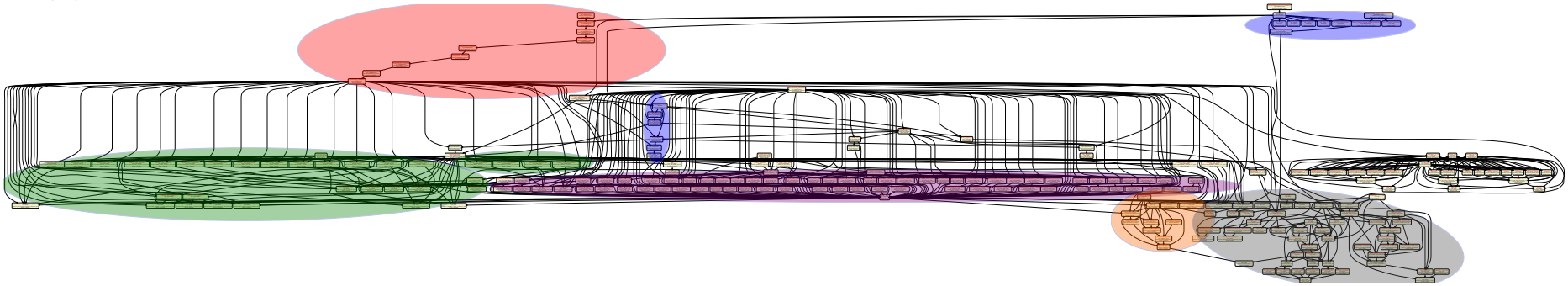
Red: Model parameter translations
Blue: Precision calculations
Green: LEP rates+likelihoods
Purple: Decays
Orange: LHC observables and likelihoods

Dependency Resolution (arXiv:1705.07908)

CMSSM:



MSSM7:

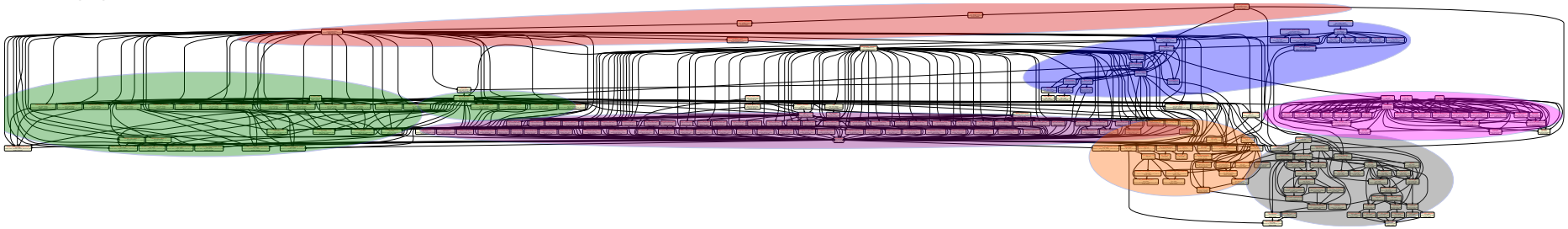


Red: Model parameter translations
Blue: Precision calculations
Green: LEP rates+likelihoods
Purple: Decays
Orange: LHC observables and likelihoods
Grey: DM direct, indirect and relic density

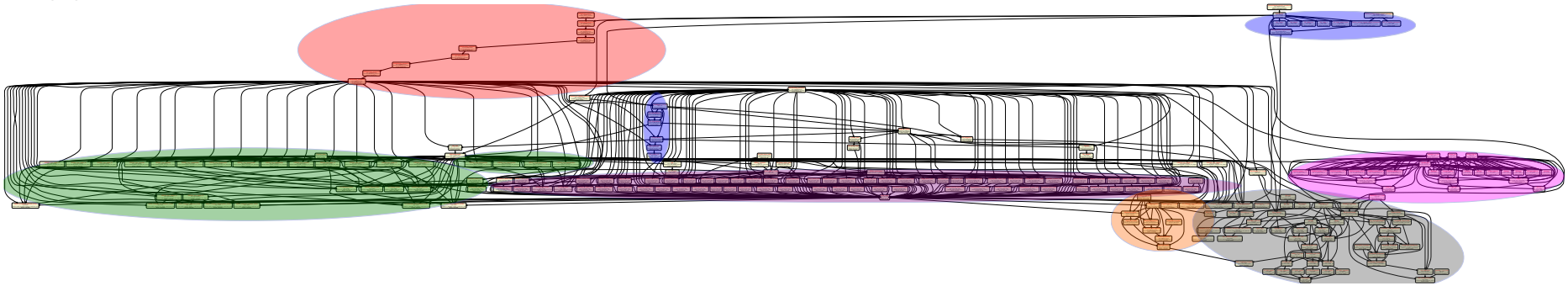
Dependency Resolution

(arXiv:1705.07908)

CMSSM:



MSSM7:



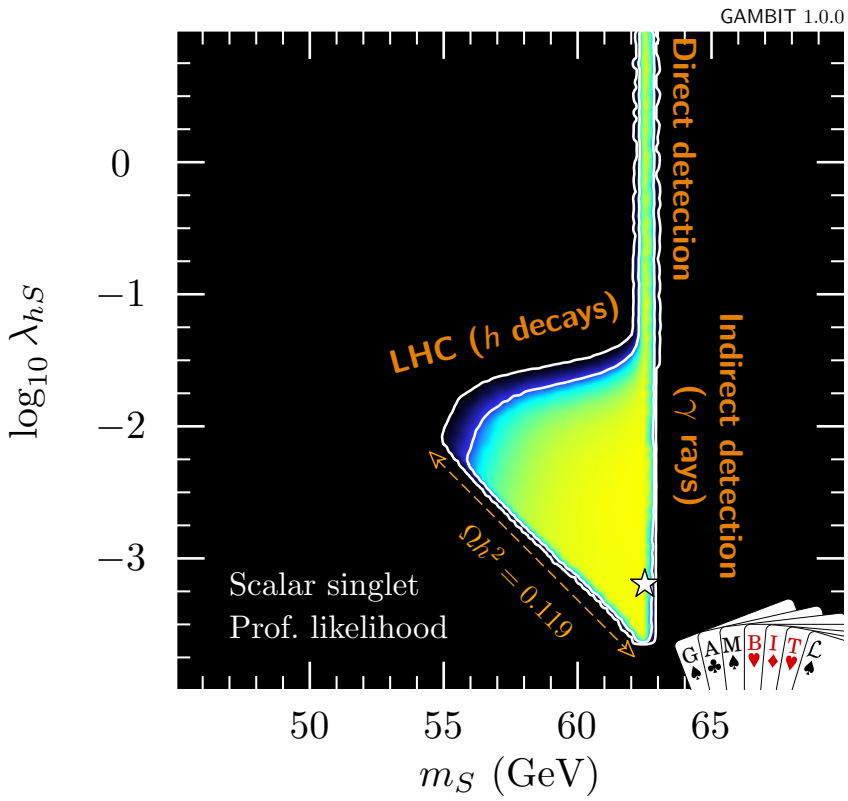
Red: Model parameter translations
Blue: Precision calculations
Green: LEP rates+likelihoods
Purple: Decays
Orange: LHC observables and likelihoods
Grey: DM direct, indirect and relic density
Pink: Flavour physics

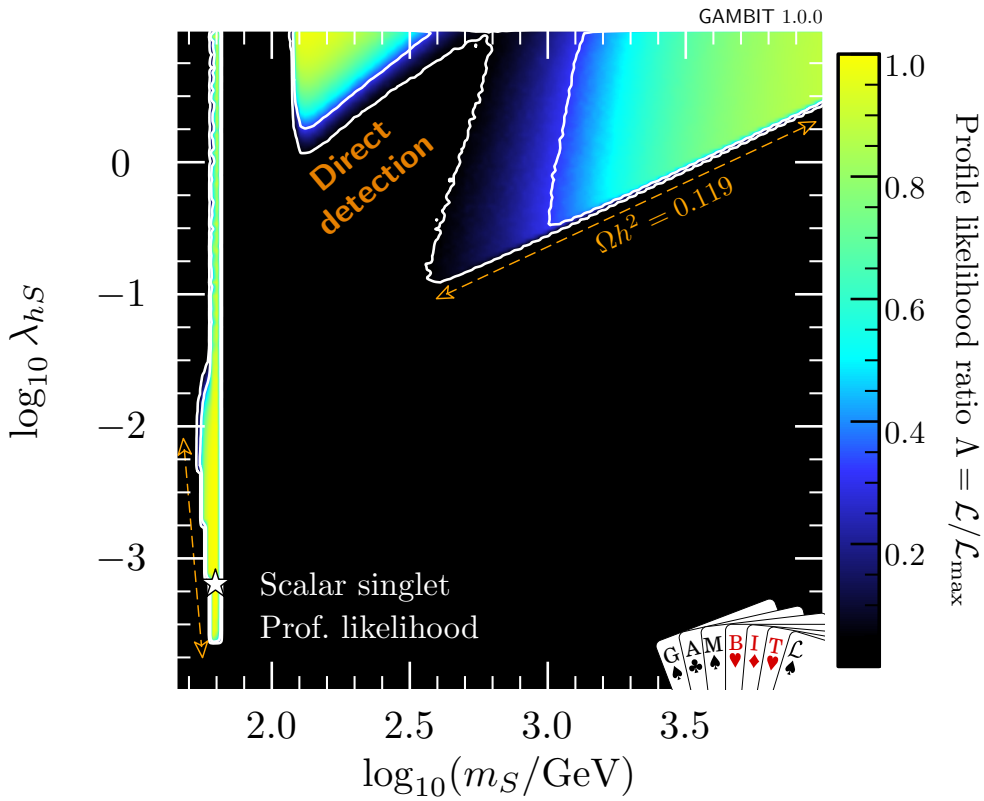
Other nice technical features (backup slides + **tutorial**)

- **User interface**: yaml file
- **Scanners**: Nested sampling, differential evolution, MCMC, T-Walk ensemble Monte Carlo...
- Mixed-mode **MPI + openMP** parallelisation, fully automated $\rightarrow$ scales to 10k+ cores
- diskless generalisation of various Les Houches Accords
- **BOSS**: dynamic loading of C++ classes from backends (!)
- **all-in or module standalone** modes – easily implemented from single cmake script
- **automatic getters** for downloading, configuring + compiling backends[1]
- **flexible output streams** (ASCII, databases, HDF5, ...)
- available as Docker plugin or Vagrant virtual machine
- more more more...

[1] if a backend won't compile/crashes/kills your hamster, blame the authors (not us... except where we **are** the authors...)

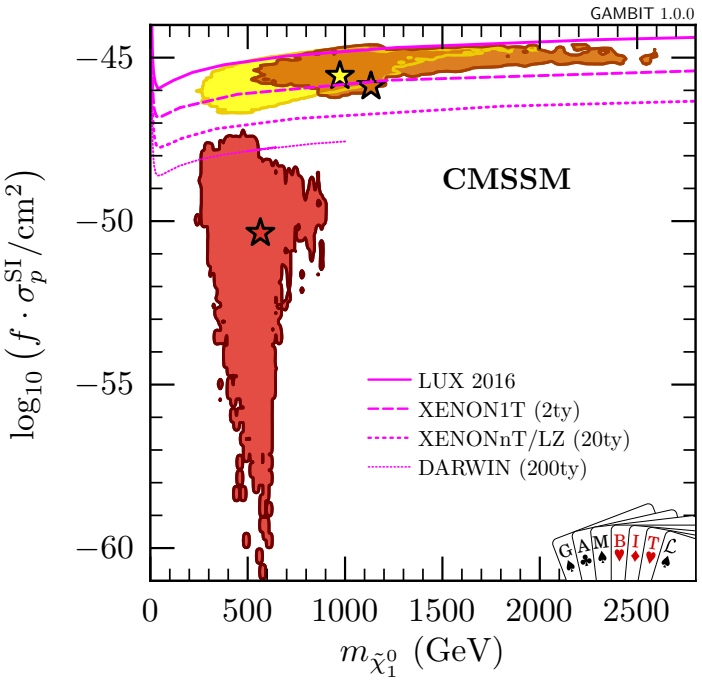
Scalar singlet DM (m_S, λ_{hS} + 13 nuisances) (arXiv:1705.07931)

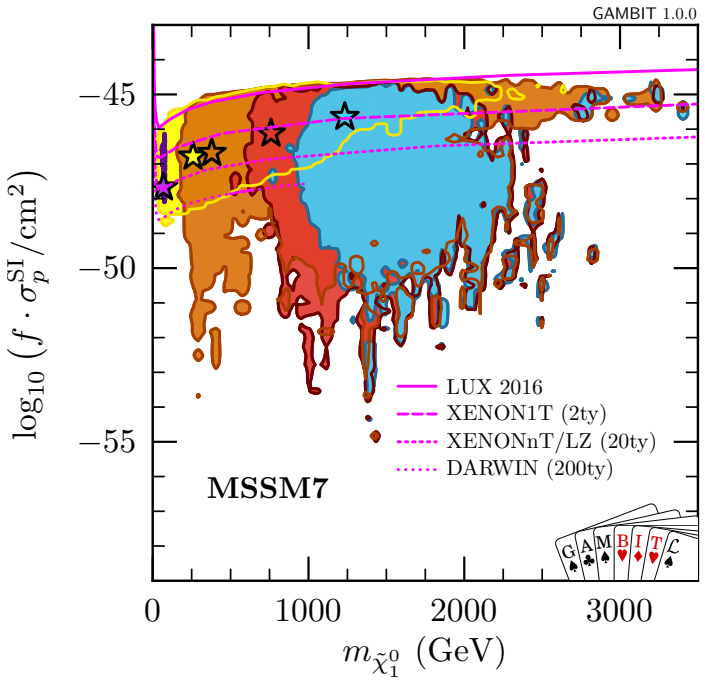




Simplest BSM example: $\mathcal{L}_S = -\frac{\mu_S^2}{2} S^2 - \frac{\lambda_{hs}}{2} S^2 H^\dagger H + \ldots$

All dark matter signals consistently scaled for predicted abundance





$\tilde{t}_1$ co-annihilation A/H funnel $\tilde{\chi}_1^\pm$ co-annihilation $\tilde{b}_1$ co-annihilation h/Z funnel

- CMSSM (4 parameters), MSSM7 (7 parameters) + 5 nuisances
- Includes LUX 2016, Panda-X + direct simulation of LHC Run 1 & Run 2 limits.
- $\tilde{\tau}$ **co-annihilation now ruled out in the CMSSM**
- $\tilde{t}$ co-annihilation tough to get at with DD, ID or LHC; some hope with LHC for seeing light stops at the 'tip'

Outlook

- Code is public: gambit.hepforge.org
- Version 1.1 due out in the next few weeks (adds support for SUSYHD and other Mathematica backends)
- SUSY + singlet results published/accepted in EPJC
- Axions, more Higgs portals & 2HDM coming soon
- **Next:** more **models**!

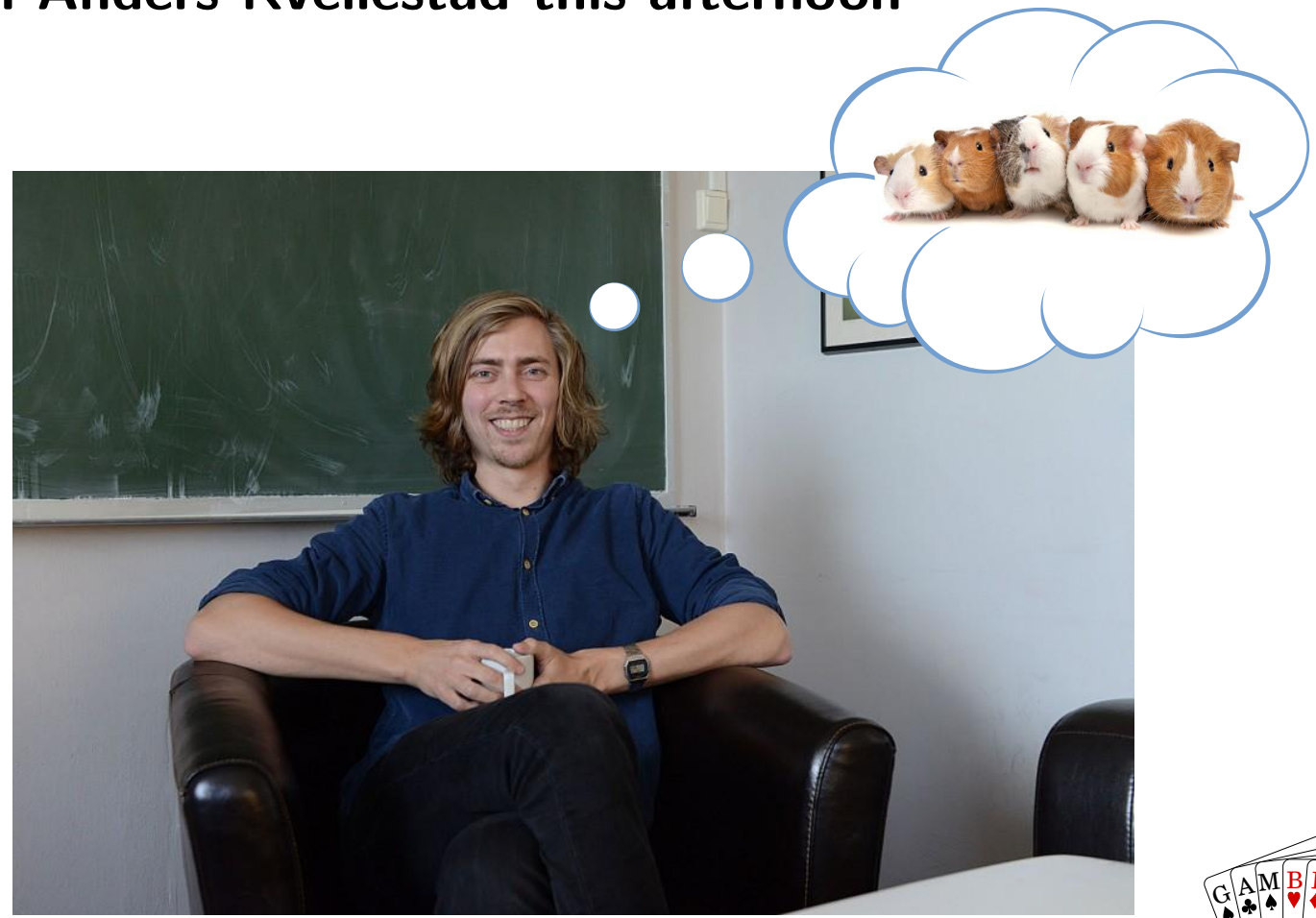
  $\rightarrow$ automated interfaces with SARAH, FeynRules, MadGraph, CalcHEP, etc.
- **Next:** more **observables**!

  $\rightarrow$ cosmological observables, neutrino physics + helioseismological searches for new physics

Outlook

Stay tuned for. . .

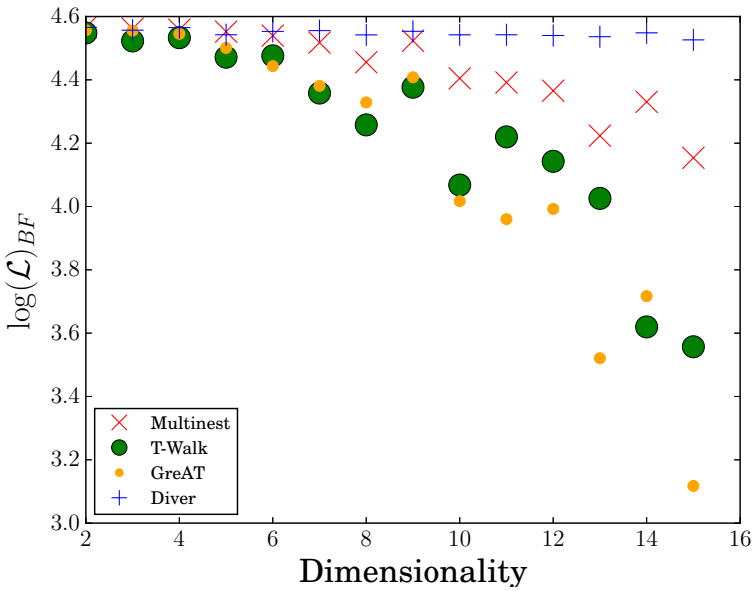
Tutorial with Anders Kvellestad this afternoon

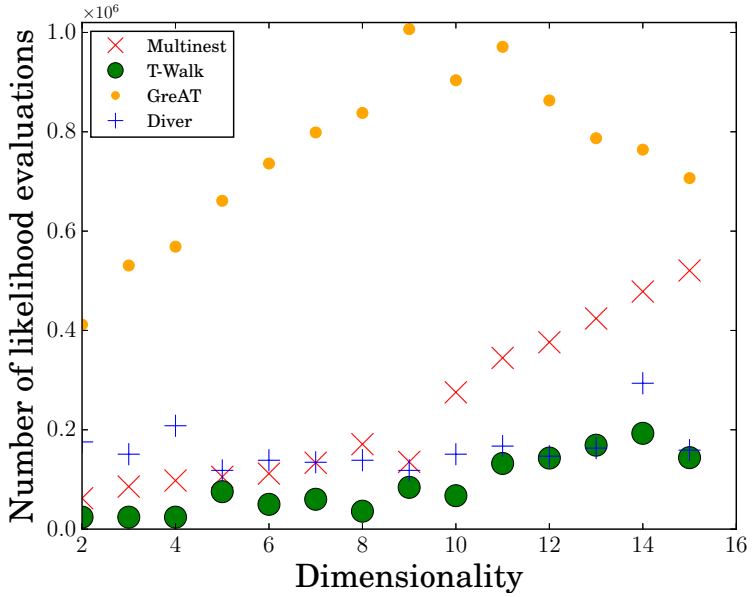


Backup slides

Scanner performance comparison (arXiv:1705.07959)

Extensive scanner tests on scalar singlet model with different numbers of nuisance paramters





Diver scales far better with dimensionality than MultiNest or other scanners

Expansion: adding new observables and likelihoods

Adding a new module function is easy:

1. Declare the function to GAMBIT in a module's **rollcall header**
   - Choose a capability
   - Declare any **backend requirements**
   - Declare any **dependencies**
   - Declare any specific **allowed models**
   - other more advanced declarations also available

```
#define MODULE FlavBit                                  // A tasty GAMBIT module.
START_MODULE

  #define CAPABILITY Rmu                                // Observable: BR(K->mu nu)/BR(pi->mu nu)
  START_CAPABILITY
    #define FUNCTION SI_Rmu                             // Name of a function that can compute Rmu
    START_FUNCTION(double)                              // Function computes a double precision result
    BACKEND_REQ(Kmunu_pimunu, (my_tag), double, (const parameters*)) // Needs function from a backend
    BACKEND_OPTION( (SuperIso, 3.6), (my_tag) )                      // Backend must be SuperIso 3.6
    DEPENDENCY(SuperIso_modelinfo, parameters)          // Needs another function to calculate SuperIso info
    ALLOW_MODELS(MSSM63atQ, MSSM63atMGUT)               // Works with weak/GUT-scale MSSM and descendents
    #undef FUNCTION
  #undef CAPABILITY
```

2. Write the function as a standard C++ function (one argument: the result)

Expansion: adding new models

1. Add the model to the **model hierarchy**:
   - Choose a model name, and declare any **parent model**
   - Declare the model's parameters
   - Declare any **translation function** to the parent model

```
#define MODEL NUHM1
#define PARENT NUHM2
  START_MODEL
  DEFINEPARS(M0,M12,mH,A0,TanBeta,SignMu)
  INTERPRET_AS_PARENT_FUNCTION(NUHM1_to_NUHM2)
#undef PARENT
#undef MODEL
```

2. Write the translation function as a standard C++ function:

```
void MODEL_NAMESPACE::NUHM1_to_NUHM2 (const ModelParameters &myP, ModelParameters &targetP)
{
   // Set M0, M12, A0, TanBeta and SignMu in the NUHM2 to the same values as in the NUHM1
   targetP.setValues(myP,false);
   // Set the values of mHu and mHd in the NUHM2 to the value of mH in the NUHM1
   targetP.setValue("mHu", myP["mH"]);
   targetP.setValue("mHd", myP["mH"]);
}
```

3. If needed, declare that existing module functions work with the new model, or add new functions that do.

Interface: yaml file

Basic interface for a scan is a YAML initialisation file

- specify parameters, ranges, priors
- select likelihood components
- select other observables to calculate
- define generic rules for how to fill dependencies
- define generic rules for options to be passed to module functions
- set global options (scanner, errors/warnings, logging behaviour, etc)

```yaml
Parameters:
  StandardModel_SLHA2: !import StandardModel_SLHA2_default
  MSSM25atQ: !import LesHouches.in.MSSM_1.yaml

Priors:
    # none: all parameters fixed in this example.

Scanner:

  use_scanner: toy_mcmc

  scanners:
    toy_mcmc:
      plugin: toy_mcmc
      point_number: 2000
      output_file:  output
      like:  Likelihood

ObsLikes:

  # Test DecayBit
  - purpose:      Test
    capability:   decay_rates
    type:         DecayTable

  # 79-string IceCube likelihood
  - capability: IceCube_likelihood
    purpose: Likelihood
    function: IC79_loglike

Rules:

  - capability: MSSM_spectrum
    function: get_MSSMatQ_spectrum
    options:
      invalid_point_fatal: true
```

Interface: yaml file

Basic interface for a scan is a YAML initialisation file

- specify parameters, ranges, priors
- select likelihood components
- select other observables to calculate
- define generic rules for how to fill dependencies
- define generic rules for options to be passed to module functions
- set global options (scanner, errors/warnings, logging behaviour, etc)

```
Parameters:
  StandardModel_SLHA2: !import StandardModel_SLHA2_default
  MSSM25atQ: !import LesHouches.in.MSSM_1.yaml

Priors:
    # none: all parameters fixed in this example.

Scanner:

  use_scanner: toy_mcmc

  scanners:
    toy_mcmc:
      plugin: toy_mcmc
      point_number: 2000
      output_file:  output
      like:  Likelihood

ObsLikes:

  # Test DecayBit
  - purpose:      Test
    capability:   decay_rates
    type:         DecayTable

  # 79-string IceCube likelihood
  - capability: IceCube_likelihood
    purpose: Likelihood
    function: IC79_loglike

Rules:

  - capability: MSSM_spectrum
    function: get_MSSMatQ_spectrum
    options:
      invalid_point_fatal: true
```

Interface: yaml file

Basic interface for a scan is a YAML initialisation file

- specify parameters, ranges, priors
- select likelihood components
- select other observables to calculate
- define generic rules for how to fill dependencies
- define generic rules for options to be passed to module functions
- set global options (scanner, errors/warnings, logging behaviour, etc)

```yaml
Parameters:
  StandardModel_SLHA2: !import StandardModel_SLHA2_default
  MSSM25atQ: !import LesHouches.in.MSSM_1.yaml

Priors:
    # none: all parameters fixed in this example.

Scanner:

  use_scanner: toy_mcmc

  scanners:
    toy_mcmc:
      plugin: toy_mcmc
      point_number: 2000
      output_file:  output
      like:  Likelihood

ObsLikes:

  # Test DecayBit
  - purpose:      Test
    capability:   decay_rates
    type:         DecayTable

  # 79-string IceCube likelihood
  - capability: IceCube_likelihood
    purpose: Likelihood
    function: IC79_loglike

Rules:

  - capability: MSSM_spectrum
    function: get_MSSMatQ_spectrum
    options:
      invalid_point_fatal: true
```

Interface: yaml file

Basic interface for a scan is a YAML initialisation file

- specify parameters, ranges, priors
- select likelihood components
- select other observables to calculate
- define generic rules for how to fill dependencies
- define generic rules for options to be passed to module functions
- set global options (scanner, errors/warnings, logging behaviour, etc)

```yaml
Parameters:
  StandardModel_SLHA2: !import StandardModel_SLHA2_default
  MSSM25atQ: !import LesHouches.in.MSSM_1.yaml

Priors:
    # none: all parameters fixed in this example.

Scanner:

  use_scanner: toy_mcmc

  scanners:
    toy_mcmc:
      plugin: toy_mcmc
      point_number: 2000
      output_file:  output
      like:  Likelihood

ObsLikes:

  # Test DecayBit
  - purpose:      Test
    capability:   decay_rates
    type:         DecayTable

  # 79-string IceCube likelihood
  - capability: IceCube_likelihood
    purpose: Likelihood
    function: IC79_loglike

Rules:

  - capability: MSSM_spectrum
    function: get_MSSMatQ_spectrum
    options:
      invalid_point_fatal: true
```

Interface: yaml file

Basic interface for a scan is a YAML initialisation file

- specify parameters, ranges, priors
- select likelihood components
- select other observables to calculate
- define generic rules for how to fill dependencies
- define generic rules for options to be passed to module functions
- set global options (scanner, errors/warnings, logging behaviour, etc)

```yaml
Parameters:
  StandardModel_SLHA2: !import StandardModel_SLHA2_default
  MSSM25atQ: !import LesHouches.in.MSSM_1.yaml

Priors:
    # none: all parameters fixed in this example.

Scanner:

  use_scanner: toy_mcmc

  scanners:
    toy_mcmc:
      plugin: toy_mcmc
      point_number: 2000
      output_file:  output
      like:  Likelihood

ObsLikes:

  # Test DecayBit
  - purpose:      Test
    capability:   decay_rates
    type:         DecayTable

  # 79-string IceCube likelihood
  - capability: IceCube_likelihood
    purpose: Likelihood
    function: IC79_loglike

Rules:

  - capability: MSSM_spectrum
    function: get_MSSMatQ_spectrum
    options:
      invalid_point_fatal: true
```

ColliderBit details

LEP likelihoods

- complete model-independent recast of direct sparticle searches

Higgs likelihoods:

- for now: HiggSignals + HiggsBounds + constraints from invisible fits (Bernon, Dumont, Kraml et al)
- future: full simulation and ATLAS+CMS combination, more correlations, CP info, no SM-like coupling assumptions

Fast LHC likelihoods

- no simplified models, just faster direct simulation

ColliderBit details

LHC likelihoods:

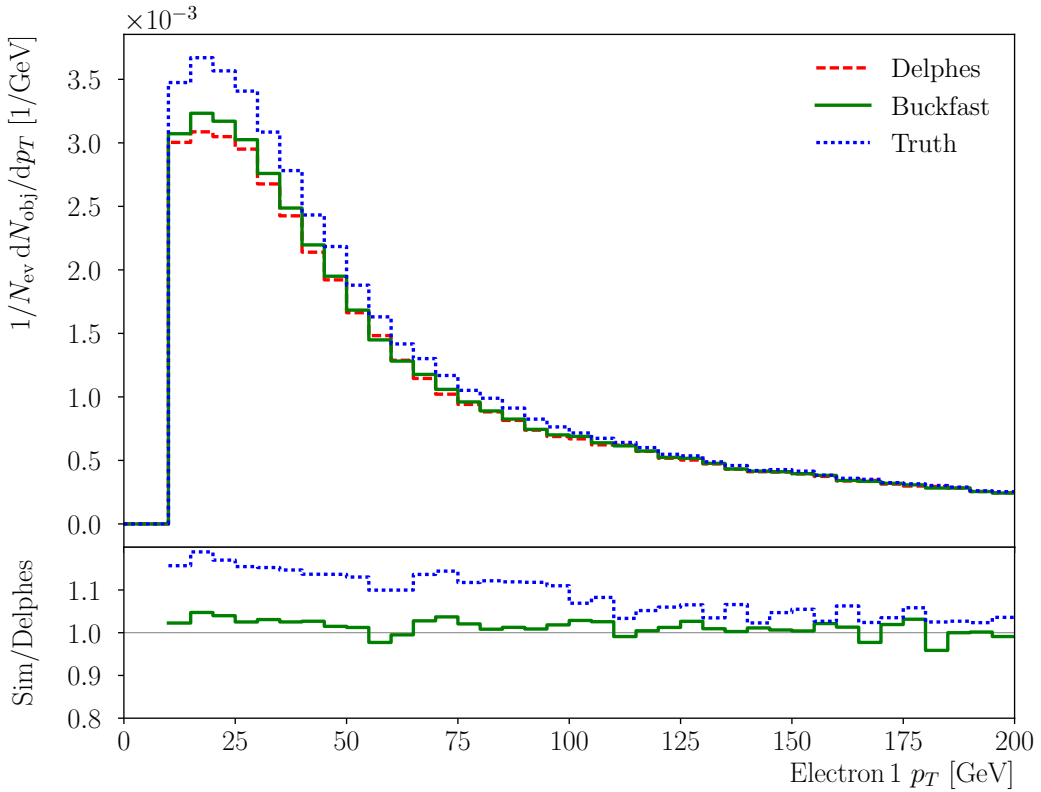
- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks

ColliderBit details

LHC likelihoods:

- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks
- Detector simulation: fast simulation based on 4-vector smearing
  $\rightarrow$ matches DELPHES results very closely (but much faster!)

Leading electron p_{T} distribution (CMSSM example):
red: detector-level simulation with DELPHES
green: 4-vector smearing with GAMBIT
blue: truth-level distribution



ColliderBit details

LHC likelihoods:

- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks
- Detector simulation: fast simulation based on 4-vector smearing
  $\rightarrow$ matches DELPHES results very closely (but much faster!)
- Cross-sections: LO + LL from MC generator by default (fast NLO in works for SUSY)

ColliderBit details

LHC likelihoods:

- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks
- Detector simulation: fast simulation based on 4-vector smearing
  $\rightarrow$ matches DELPHES results very closely (but much faster!)
- Cross-sections: LO + LL from MC generator by default (fast NLO in works for SUSY)
- Analysis framework: custom event-level, independent of experiment or simulation

ColliderBit details

LHC likelihoods:

- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks
- Detector simulation: fast simulation based on 4-vector smearing
  $\rightarrow$ matches DELPHES results very closely (but much faster!)
- Cross-sections: LO + LL from MC generator by default (fast NLO in works for SUSY)
- Analysis framework: custom event-level, independent of experiment or simulation
- Likelihood: inline systematic error marginalisation (via `nulike`)

ColliderBit details

LHC likelihoods:

- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks
- Detector simulation: fast simulation based on 4-vector smearing
  $\rightarrow$ matches DELPHES results very closely (but much faster!)
- Cross-sections: LO + LL from MC generator by default (fast NLO in works for SUSY)
- Analysis framework: custom event-level, independent of experiment or simulation
- Likelihood: inline systematic error marginalisation (via `nulike`)
- Initially shipping with:
  - ATLAS SUSY searches (0ℓ, $0/1/2\ell$ $\tilde{t}$, b jets + MET, $2/3\ell$ EW)
  - CMS multi-ℓ SUSY
  - CMS DM (t pair + MET, mono-b, monojet)
  - ATLAS + CMS Run II 0ℓ
  - refresh coming with more Run II analyses shortly

DarkBit overview