Status of Fast simulation

Anna Zaborowska
EP-SFT, CERN

25th Geant4 Collaboration Meeting
24/09/2020

Recent updates around fast simulation

1. Implementation of general tool for energy deposition
   - necessary for any model not to depend on GFlash
   - for classical parametrisation, neural network based models, ...
2. Demonstration of application of those tools for GFlash model
   - validation of new tools against existing GFlash shower model
3. GFlash study of significance of number of created deposits on simulation time
4. Demonstration of application of those tools for NN inference
5. On-going procedure for automated tuning of parameters for GFlash-inspired shower parametrisation
   - extraction of parameters (longitudinal and transverse)
   - on-going improvements and validation, requires significant changes to the algorithm

1. General tool for energy deposition

Status in GEANT4:

- Currently no general tool for depositing energy.
- GFlash implementation includes models, sensitive detector, ...
- They are all interlinked and connected to `GFlash*` classes.

1. General tool for energy deposition

Status in GEANT4:

- Currently no general tool for depositing energy.
- GFlash implementation includes models, sensitive detector, ...
- They are all interlinked and connected to `GFlash*` classes.

New developments:

- Assume simple implementation of fast simulation models.
  - energy E_i is deposited in $\overline{r_i}$, $i \in 1, \ldots N$ (no other information)

1. General tool for energy deposition

Status in GEANT4:

- Currently no general tool for depositing energy.
- GFlash implementation includes models, sensitive detector, ...
- They are all interlinked and connected to `GFlash*` classes.

New developments:

- Assume simple implementation of fast simulation models.
  - energy E_i is deposited in $\overline{r_i}$, $i \in 1, \ldots N$ (no other information)
- `G4FastHit` binding energy deposit and its position;
- `G4FastSimHitMaker` that keeps track of volumes, readout geometry, and calls sensitive detector to deposit energy;
- `G4VFastSimSensitiveDetector` is a base class for user's SD (additional to G4VSensitiveDetector) that includes method for processing hits based on `G4FastHit` (instead of the usual `G4Step`)

1. General tool for energy deposition: G4FastHit

```
/**
 * @brief Minimal hit created in the fast simulation
 *
 * Minimal hit containing energy and position, used in the fast simulation classes.
 *
 */

class G4FastHit
{
public:
  G4FastHit();
  G4FastHit(const G4ThreeVector &aPosition, G4double aEnergy);
  ~G4FastHit(){};

  /// Set hit's energy
  inline void SetEnergy(const G4double &aEnergy) { fEnergy = aEnergy; }
  /// Get hit's energy
  inline G4double GetEnergy() const { return fEnergy; }
  /// Set hit's position
  inline void SetPosition(const G4ThreeVector &aPosition) { fPosition = aPosition; }
  /// Get hit's position
  inline G4ThreeVector GetPosition() const { return fPosition; }

private:
  /// energy
  G4double fEnergy = 0;
  /// position
  G4ThreeVector fPosition = G4ThreeVector();
};
```

1. General tool for energy deposition: G4FastSimHitMaker

```
/**
 * @brief Helper class for hit creation
 *
 * Helper class that can be employed in the fast simulation models.
 * It allows to deposit energy at given position, provided it is within the sensitive detector that derives from
 * G4VFastSimSensitiveDetector base class.
 *
 */

class G4FastSimHitMaker
{
public:
  G4FastSimHitMaker();
  ~G4FastSimHitMaker();

  /// Deposit energy at given position, as described in aHit.
  void make(const G4FastHit &aHit, const G4FastTrack &aTrack);
  /// If sensitive detector class is in the parallel world, it must be specified.
  inline void SetNameOfWorldWithSD(const G4String &aName) { fWorldWithSdName = aName; };

private:
  /// Touchable
  G4TouchableHandle fTouchableHandle;
  /// Navigator
  G4Navigator *fpNavigator;
  /// Flag specifying if navigator was set up
  G4bool fNaviSetup;
  /// Name of the world containing the sensitive detector. If empty, default mass world is used.
  G4String fWorldWithSdName;
};
```

1. General tool for energy deposition: G4FastSimHitMaker

```
22 void FastSimHitMaker::make(const G4FastHit &aHit, const G4FastTrack &aTrack)
23 {
```

```
51   G4VSensitiveDetector *sensitive;
52   if (currentVolume != 0)
53   {
54     sensitive = currentVolume->GetLogicalVolume()->GetSensitiveDetector();
55     G4VFastSimSensitiveDetector *fastSimSensitive = dynamic_cast<G4VFastSimSensitiveDetector
       ↪  *>(sensitive);
56     if (fastSimSensitive)
57     {
58       fastSimSensitive->Hit(&aHit, &aTrack, &fTouchableHandle);
59     }
60     else if (sensitive && currentVolume->GetLogicalVolume()->GetFastSimulationManager())
61     {
62       G4cerr « "ERROR - FastSimHitMaker::make()" « G4endl
63              « "        It is required to derive from the " « G4endl
64              « "        G4VFastSimSensitiveDetector in " « G4endl
65              « "        addition to the usual G4VSensitiveDetector class."
66              « G4endl;
67       G4Exception("FastSimHitMaker::make()", "InvalidSetup", FatalException,
68                   "G4VFastSimSensitiveDetector interface not implemented.");
69     }
70   }
71 }
```

1. General tool for energy deposition: G4VFastSimSensitiveDetector

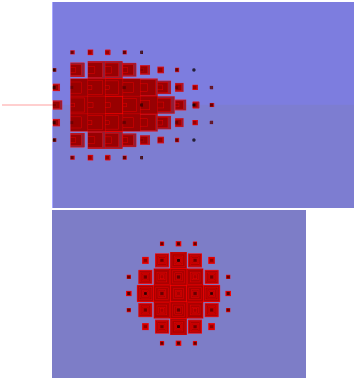
```
/**
 * @brief Base class for the sensitive detector used within the fast simulation
 *
 * Base class for a sensitive detector that allows to store hits created in the fast simulation.
 * It must me used in addition to inheritance from the usual `G4VSensitiveDetector` that describes the full simulation.
 * ProcessHits(...) method must be implemented and describe how hits should be saved in the collections.
 * It is invoked by Hit method which is public and can be called directly in the fast simulation model,
 * or via the helper class G4FastSimHitMaker that will allow to locate appropriate volume and retrieve its sensitive detector.
 *
 */

class G4VFastSimSensitiveDetector
{
```

```
private:
  /// Describes how energy and position of deposits are inserted into the hits collection.
  /// It is a private method and it will be invoked by Hit() method of the base class once the readout geometry
  /// that may be associated to the corresponding G4VSensitiveDetector is taken into account.
  virtual G4bool ProcessHits(const G4FastHit *aHit, const G4FastTrack *aTrack, G4TouchableHistory *aROhist) = 0;
};
```

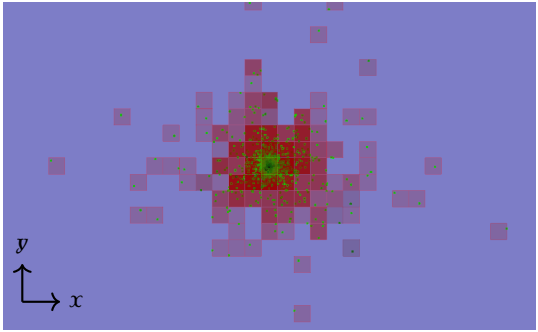
1. General tool for energy deposition: used on artificial example

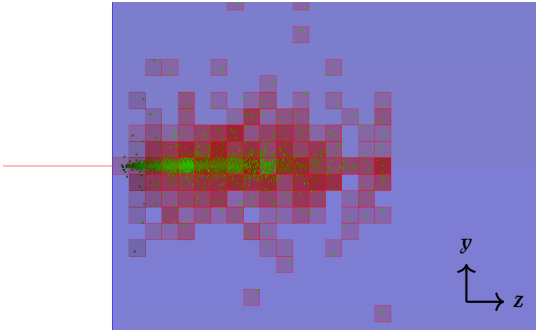
Create energy deposits



- Example for illustration purposes:
  - Gamma distribution for longitudinal profile
  - Gaussian distribution for transverse profile
  - arbitrary parameters
  - energy deposited on a mesh (Carthesian),
- Proper use would expect meaningful parameterisation:
  - with previously tuned parameters (e.g. GFlash-like parameterisation)
  - inference of previously trained NN models (on same mesh/readout that was used for training)

2. Demonstration of application of those tools for GFlash





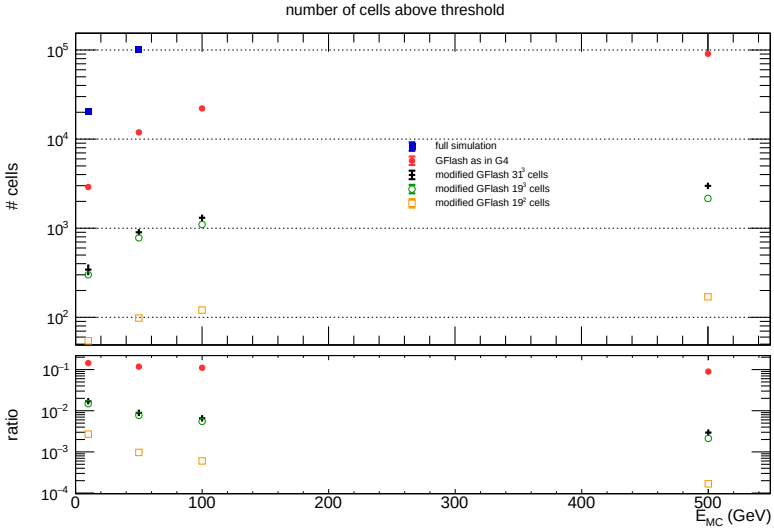
Green Energy deposited by current implementation of GFlash in GEANT4.

Red Additionally, assign deposits to (mesh) cells and deposit accumulated energy.

- Maintaining cell sizes relatively small (wrt readout cells) should not compromise accuracy.
- But should speed up the simulation.
- Size of cells subject of optimisation by users (specific detectors).

3. Study of GFlash — number of created deposits

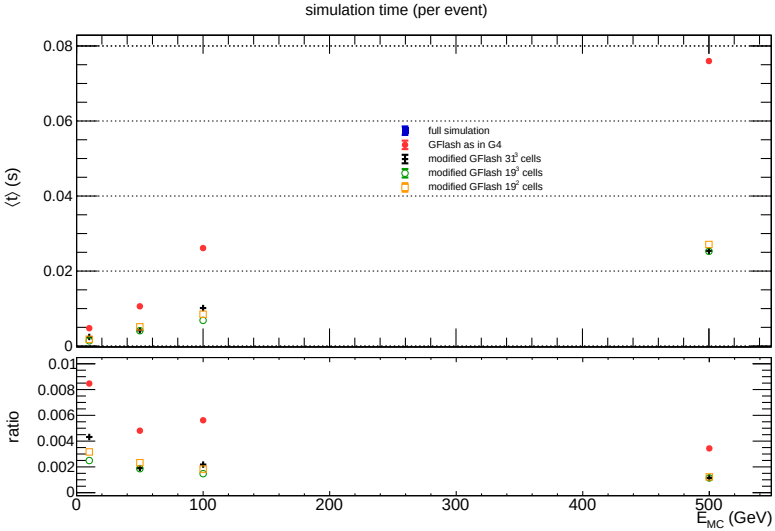
- number of deposits with >1 keV energy
- studied mesh size:
  - (black) 31x31x31,
  - (green) 19x19x19,
  - (orange)19x19x1 (no longitudinal segmentation)
- significant reduction of number of created deposits ($O(10)$ for 3D mesh)



3. Study of GFlash — simulation time

- $\sim 3 - 4$ times speed-up
- speed-up not as significant as reduction of number of deposits
- speed-up does not depend on segmentation
- parametrisation itself is a limit now
- result: more beneficial to tune number of GFlash-parametrised deposits

$\rightarrow$ another element to be tuned & studied



4. Demonstration of application of those tools for NN inference

- Inference of NN models requires G4 to load the model (architecture & parameters);
- Natural dependency on external packages;
- Skeleton for an example fast simulation model is prepared:
  - Initialization of the NN model;
  - Inference for each (triggered) parametrisation;
  - Returned energies (and positions) transformed to energy deposits using newly implemented tools;

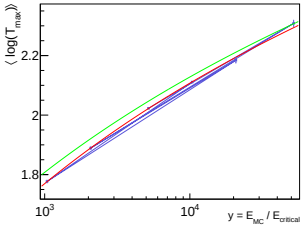
4. Demonstration of application of those tools for NN inference

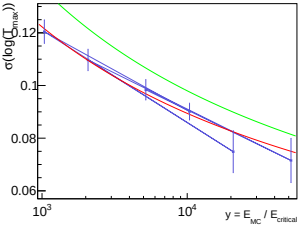
- Inference of NN models requires G4 to load the model (architecture & parameters);
- Natural dependency on external packages;
- Skeleton for an example fast simulation model is prepared:
  - Initialization of the NN model;
  - Inference for each (triggered) parametrisation;
  - Returned energies (and positions) transformed to energy deposits using newly implemented tools;
- How the model is initialized & used depends on the external library:
  - Implemented example depends on the light library (TensorFlow C API based) presented in RD Task Force parallel session by Ioana Ifrim;
  - Awaiting NN model to test (see the same presentation);
  - Should explore other, popular solutions for C++ inference, e.g. ONNX Runtime;

5. On-going tuning of parameters for GFlash-inspired shower parametrisation

$$f(t) = \frac{((\alpha - 1)\, t)^{\alpha - 1}\, (\alpha - 1) \exp^{-\beta t}}{T \cdot \Gamma(\alpha)}$$

- $\log T$, $\log \alpha$ extracted from single particle longitudinal profiles for $PbWO_4$
- Fitted function (red)
- Original GFlash parameters are plotted in green

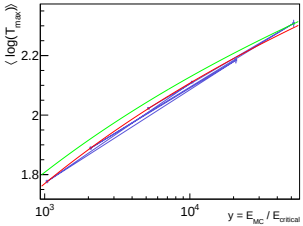




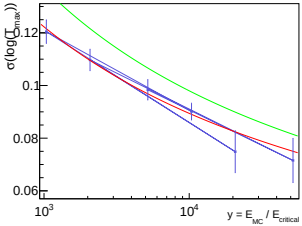
5. On-going tuning of parameters for GFlash-inspired shower parametrisation

$$f(t) = \frac{((\alpha-1)\,t)^{\alpha-1}\,(\alpha-1)\exp^{-\beta t}}{T\cdot\Gamma(\alpha)}$$

- $\log T$, $\log \alpha$ extracted from single particle longitudinal profiles for $PbWO_4$
- Fitted function (red)
- Original GFlash parameters are plotted in green



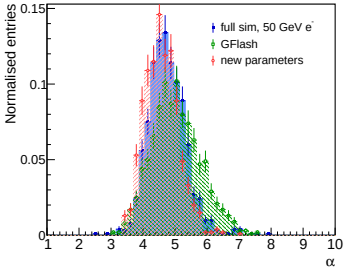
longitudinal profile fit - alpha parameter

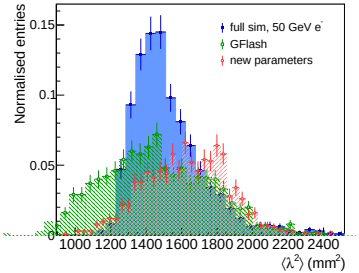


longitudinal second moment

Longitudinal profile not improved immensely - but extracted parameters (T, α), first/second moments are closer to full sim than GFlash.

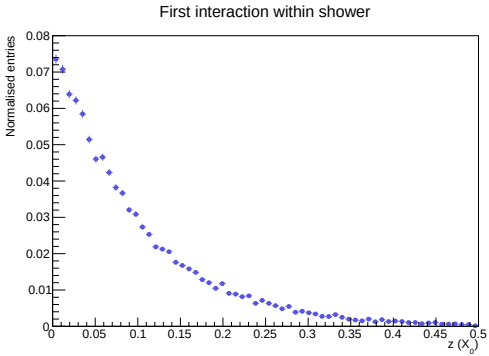
- full sim of 50 GeV e^- in $PbWO_4$
- GFlash parametrisation in G4
- New parameters





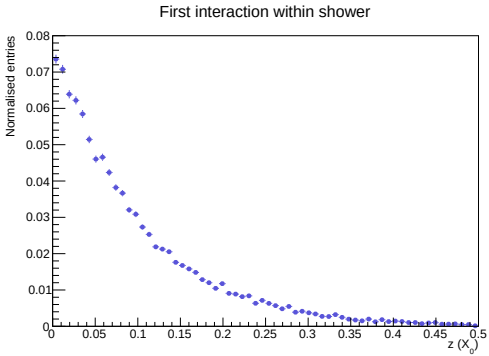
5. On-going tuning of parameters for GFlash-inspired shower parametrisation

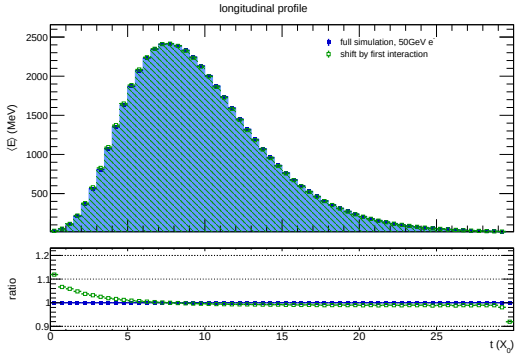
- GFlash starts parametrisation as soon as particle enters volume;
- But first interaction (shower start) may happen further;



5. On-going tuning of parameters for GFlash-inspired shower parametrisation

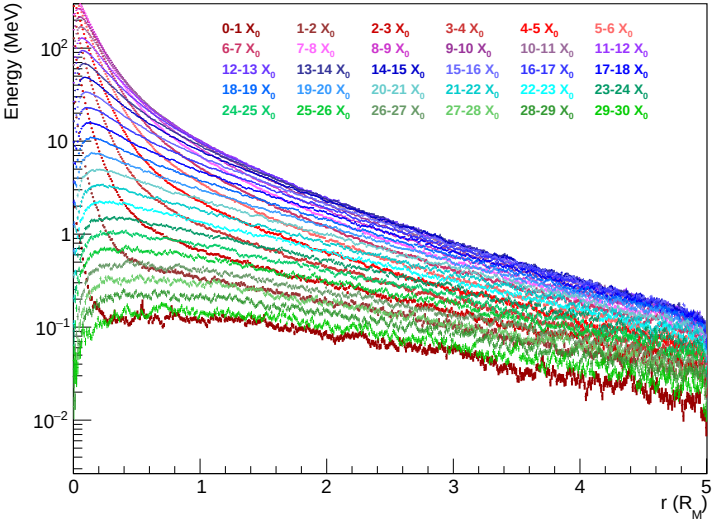
- GFlash starts parametrisation as soon as particle enters volume;
- But first interaction (shower start) may happen further;
- Including this shift in the longitudinal profile calculation does not impact significantly the parameters of the distribution, but certainly has impact on longitudinal profile for $t < 5X_0$
- Needs to be implemented for parametrisation;





5. On-going tuning of parameters for GFlash-inspired shower parametrisation

Transverse profile for 50 GeV electrons



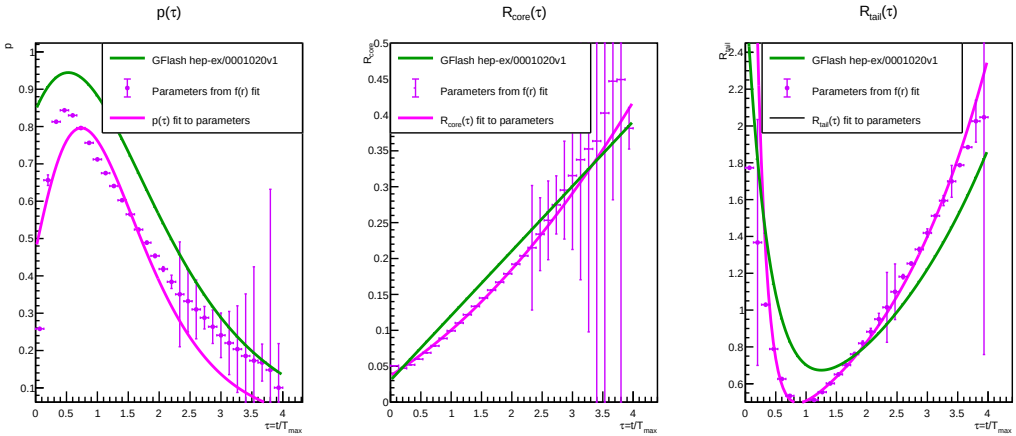
- Transverse profile changes drastically with shower depth;
- Parametrisation done on average transverse profile, for slices in shower depth;
- Same distribution is used to describe all slices:

$$f(r) = pf_{\text{core}}(r) + (1-p)f_{\text{tail}}(r)$$

$$f_{\text{core/tail}}(r) = \frac{2rR^2_{\text{core/tail}}}{\left(r^2 + R^2_{\text{core/tail}}\right)^2}$$

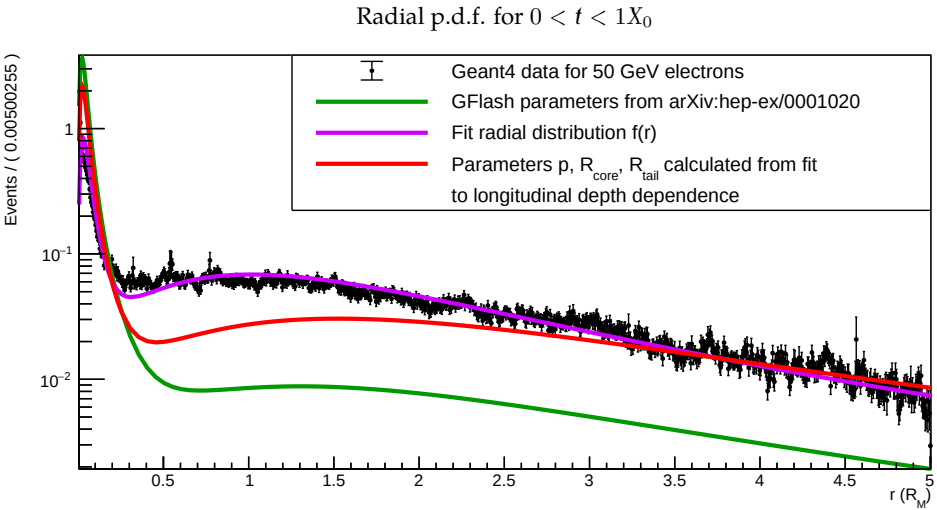
- several other distributions exist in literature (usually simpler);

5. On-going tuning of parameters for GFlash-inspired shower parametrisation

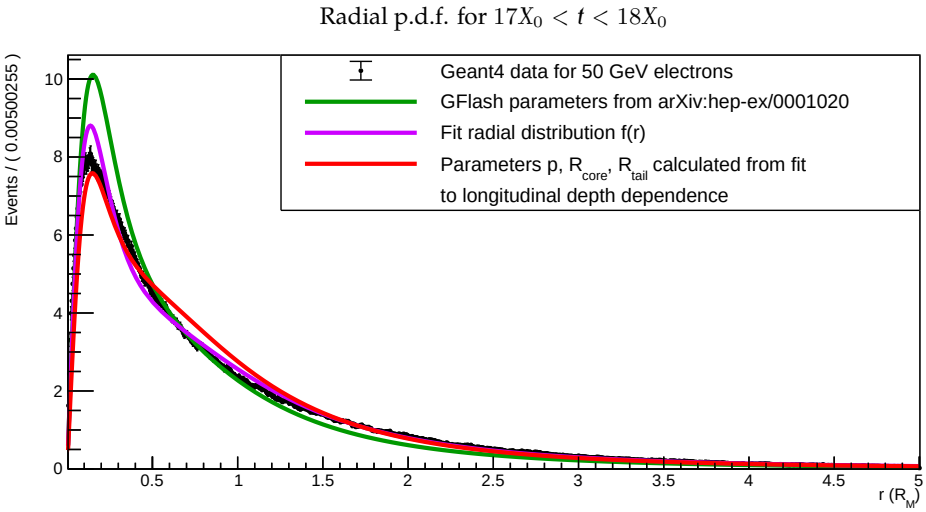


Fit-to-fit is done to describe shower depth ($\tau = t/T_{max}$) dependency.
Difficult to accurately describe low and high t values.
To be checked: use all values instead of a fit, fit only to energy of incoming particle.

5. On-going tuning of parameters for GFlash-inspired shower parametrisation



5. On-going tuning of parameters for GFlash-inspired shower parametrisation



Summary

- General tools for multiple energy depositions created (based on existing ones in GFlash);
- To be integrated into `processes/parameterisation` source code (12 classes now).

Summary

- General tools for multiple energy depositions created (based on existing ones in GFlash);
- To be integrated into `processes/parameterisation` source code (12 classes now).
- Deposits can be created by classical parametrisation models, or taken from inference of NN models;
- But inference of NN models depends on how the model is saved:
  - Current implementation depends on light library (TensorFlow based) presented in RD Task Force parallel session;
  - Investigate other popular solutions for C++ inference;

Summary

- General tools for multiple energy depositions created (based on existing ones in GFlash);
- To be integrated into `processes/parameterisation` source code (12 classes now).
- Deposits can be created by classical parametrisation models, or taken from inference of NN models;
- But inference of NN models depends on how the model is saved:
  - Current implementation depends on light library (TensorFlow based) presented in RD Task Force parallel session;
  - Investigate other popular solutions for C++ inference;
- On-going work on the automated tuning procedures:
  - Several ideas in place and being tested;

Additional slides

5. On-going tuning of parameters for GFlash-inspired shower parametrisation

- GFlash starts parametrisation as soon as particle enters volume;
- Including the start of the shower position should improve the longitudinal profile for $t < 5X_0$