Nikhef

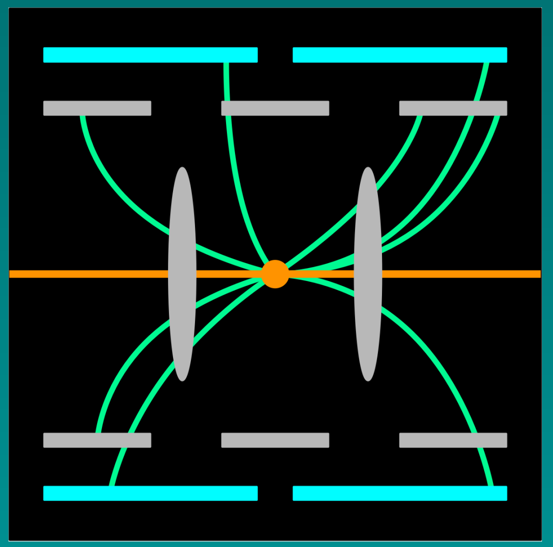
University of Twente - Nikhef

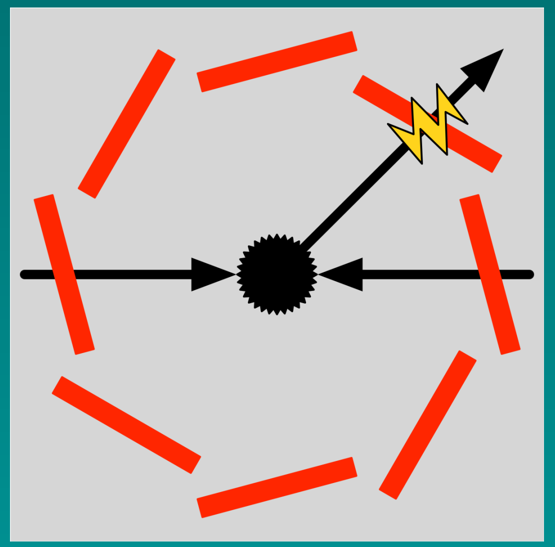
ATLAS team

Efficient Tracking Algorithm Evaluations through Multi-Level Reduced Simulations

Uraz Odyurt,
Ana-Lucia Varbanescu, Sascha Caron

2024-10-22
CHEP Conference





UNIVERSITY OF TWENTE.

ATLAS EXPERIMENT

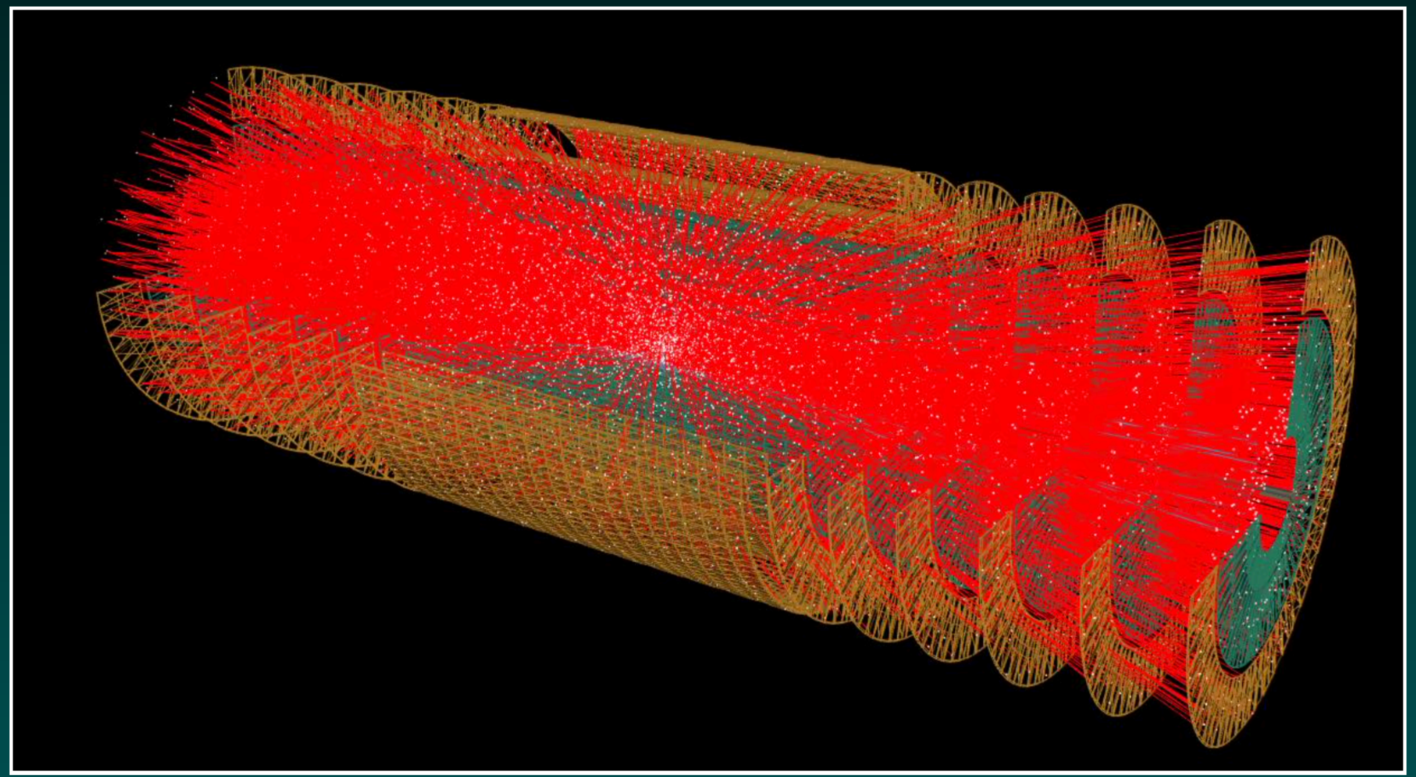
Motivation & challenge overview

Particle tracking

- Tracking: Reconstruct **particle trajectories** from recorded hits
  => Has to happen at every event

- Why we do this? Two main measurements:
  => Tracking and calorimetry -> **Momentum** and **energy**
  => Discover/study **particle behaviour**

- Present algorithms use **Kalman filters**
  => **Not linearly** scalable
  => Multiple steps, multiple passes
  => Cannot support HL-LHC era [2029-] (pile-up)
  => Numerous detector HW upgrades
  => SW and algorithm upgrades



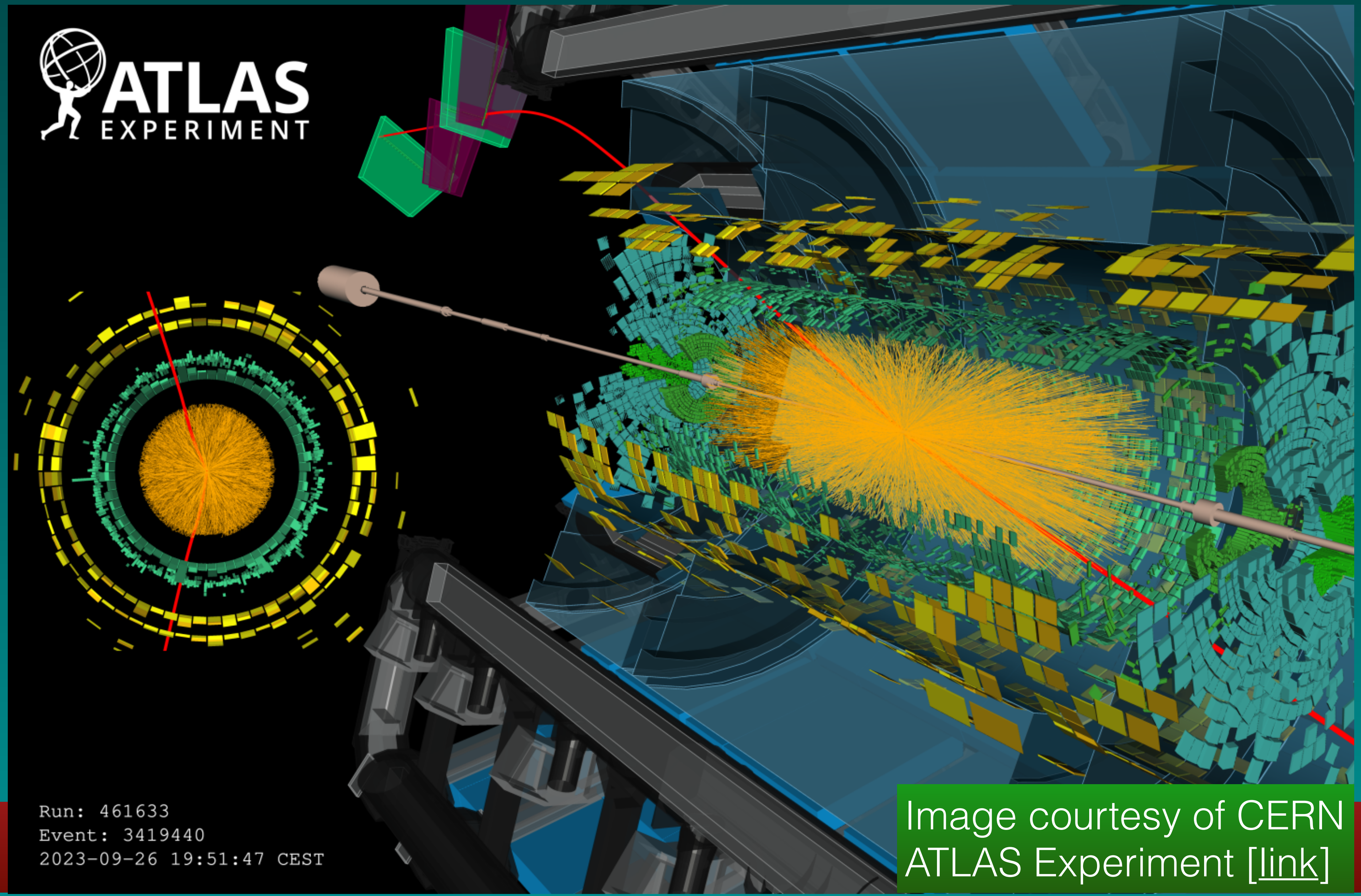


Image courtesy of CERN ATLAS Experiment [link]

Pile-up!

Average of 200 simultaneous *pp* interactions are expected: $\langle \mu \rangle = 200$

Image courtesy of CERN ATLAS Experiment [link]

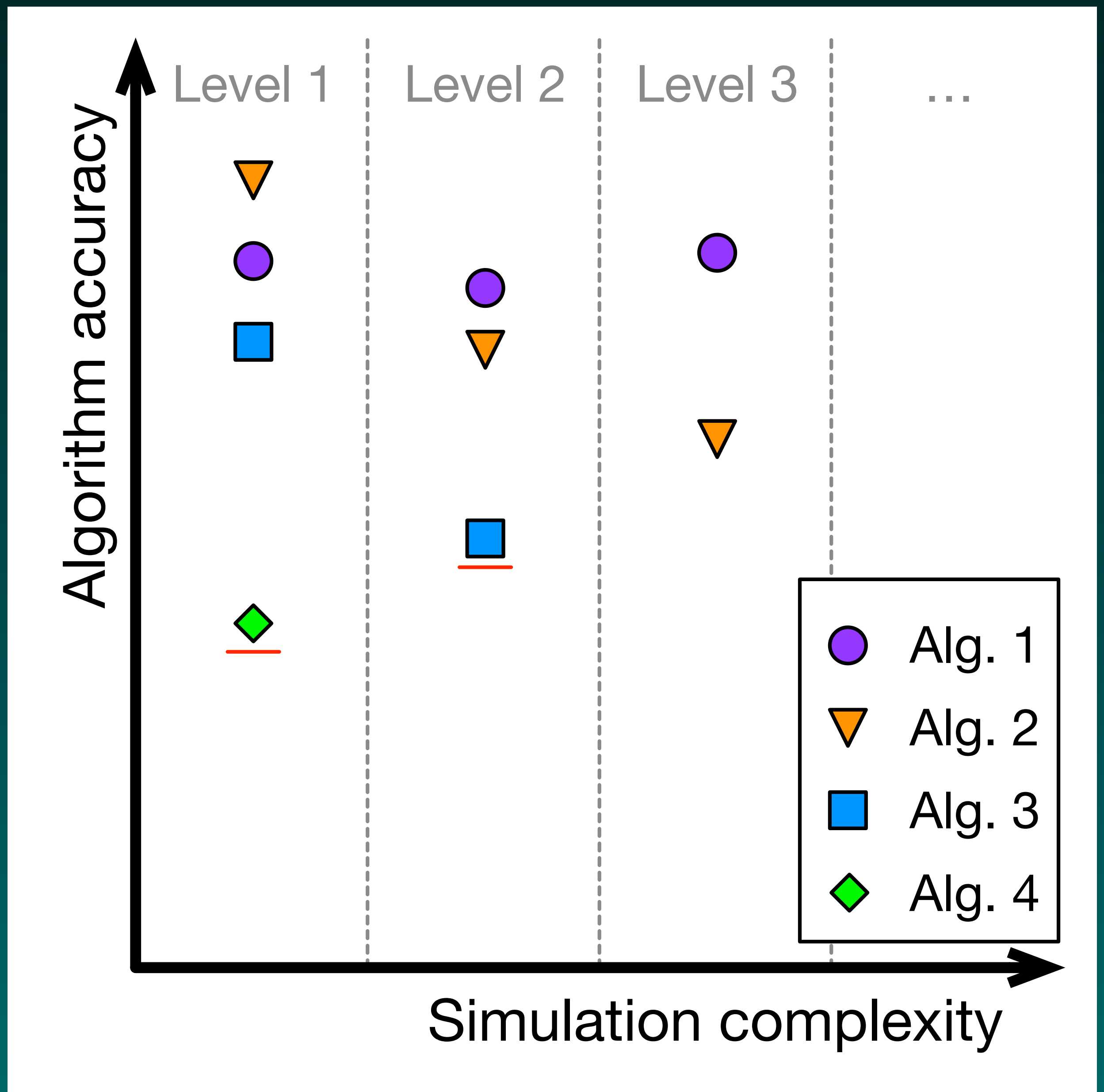
ML model design practices

Current

- Primarily **ad hoc** efforts (art, skill, experience)
  => Is it the best model? Most robust?
  => Is it the best data processing approach?
  => Is it the best data representation? Enough corner case data?
  => Is it addressing **secondary requirements**?
  Energy consumption? Explainability? …

Ideal

- Design and train best ML model(s) for tracking
  => Eliminate designer bias (experience??)
  => Better corner case response
  => Detector agnostic
  => Address **secondary requirements** (parallelisability)



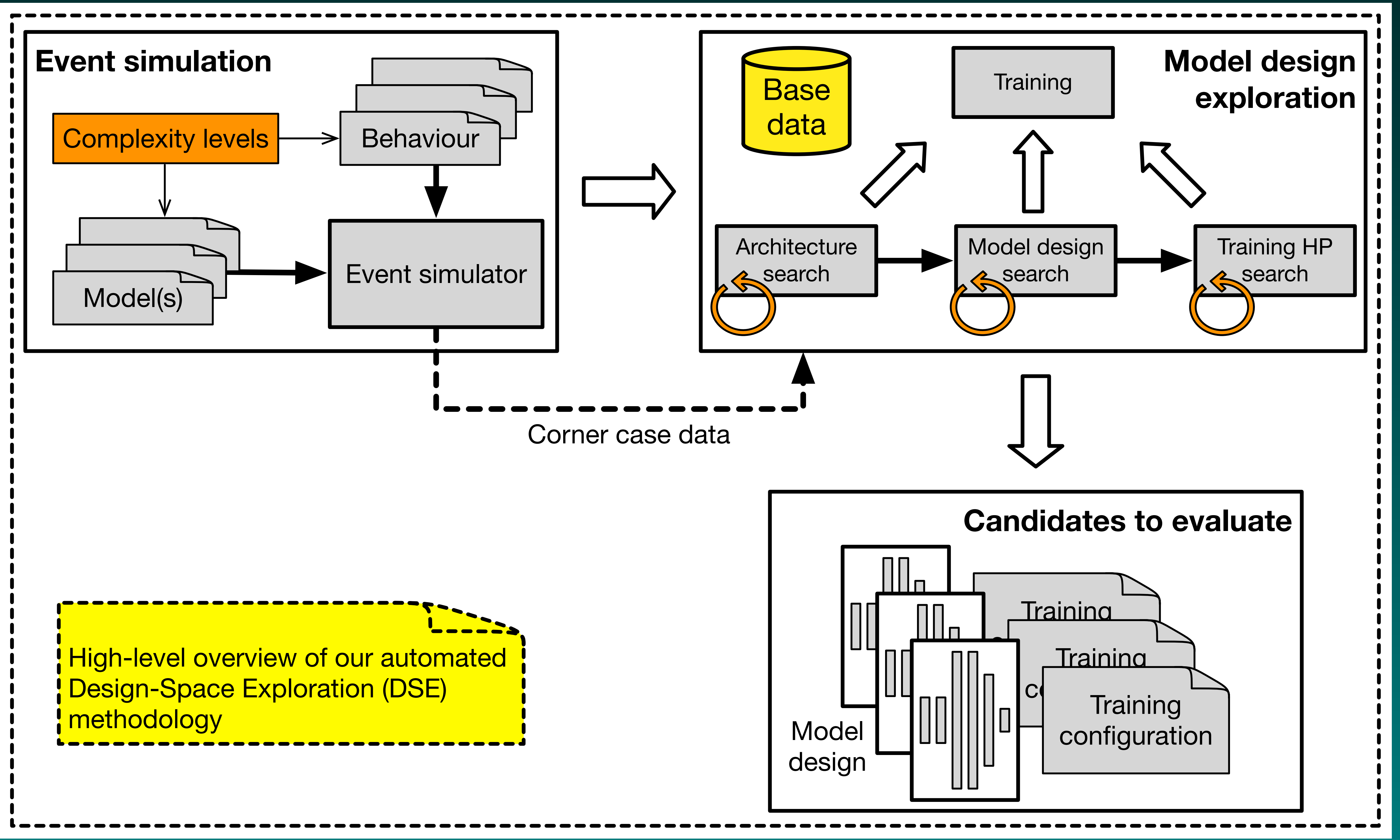
ML-assisted solutions: Higher **data capacity**, **specialised HW** (GPU, TPU, FPGA, Neuromorphic)

Can we do better?

Systematic ML model design

- Automated and multi-objective **Design-Space Exploration (DSE)**
  => Hyperparameter search
  => Neural-Architecture Search (NAS)
  => Important to minimise **search time**
  => Important to minimise **computational cost of search**

- What is the best way to do it?
  => **Complexity-aware simulations**
  => **Synthetic** data



DSE: Systematic analysis and pruning of unwanted design points based on parameters of interest.

Systematic ML model design

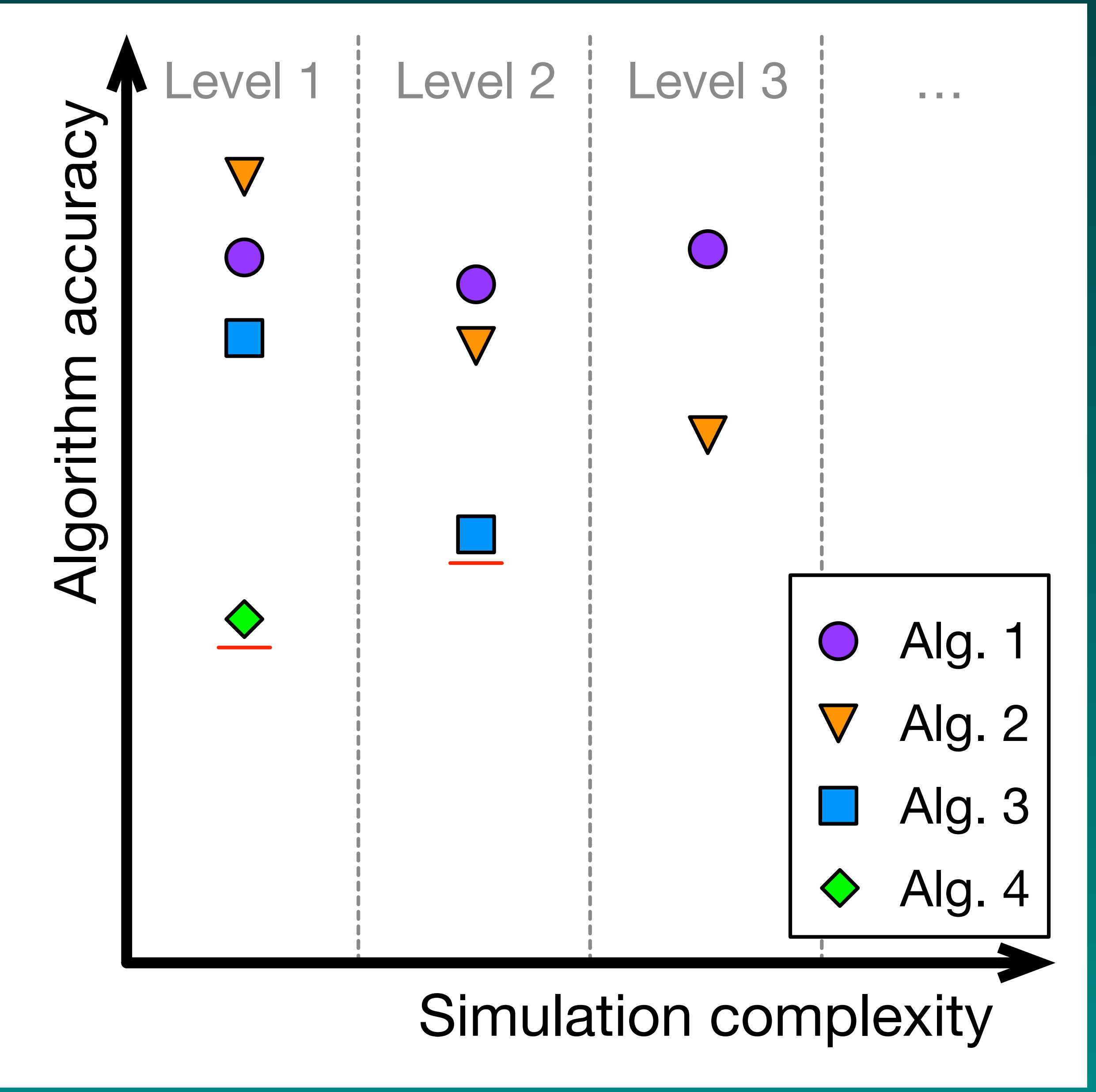
Ingredients

- Well-defined search space(s)
  => Well-defined problem space as a **complexity spectrum**
  => Each complexity level will have a model search space
  => **Continuous** space is highly advantageous

- **Event simulator**
  => Has to be reconfigurable for complexity levels

- Search algorithm

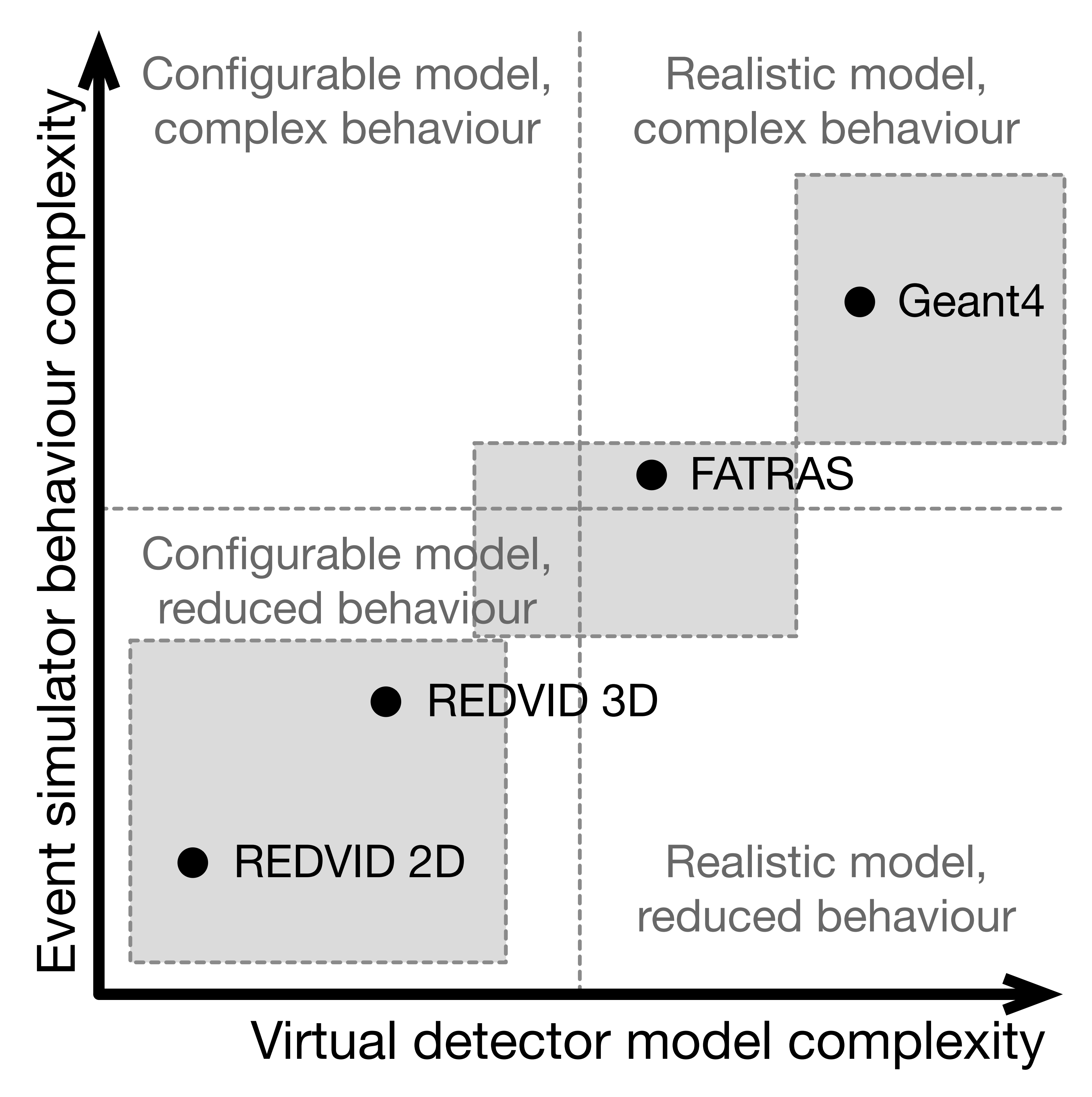
- ML model training and evaluation infrastructure



Different simulations

- **Complexity dimension** groups
  => Detector model
  => Simulator behaviour

- Two types could be considered
  => Parametric/**(re)configurable** simulations
  REDVID
  => **Physics-accurate** simulations
  Geant4, FATRAS, ATLFAST, …



Simulation: **Model of the system** (characteristics) + **Simulator** for events/actions/environment (behaviour)

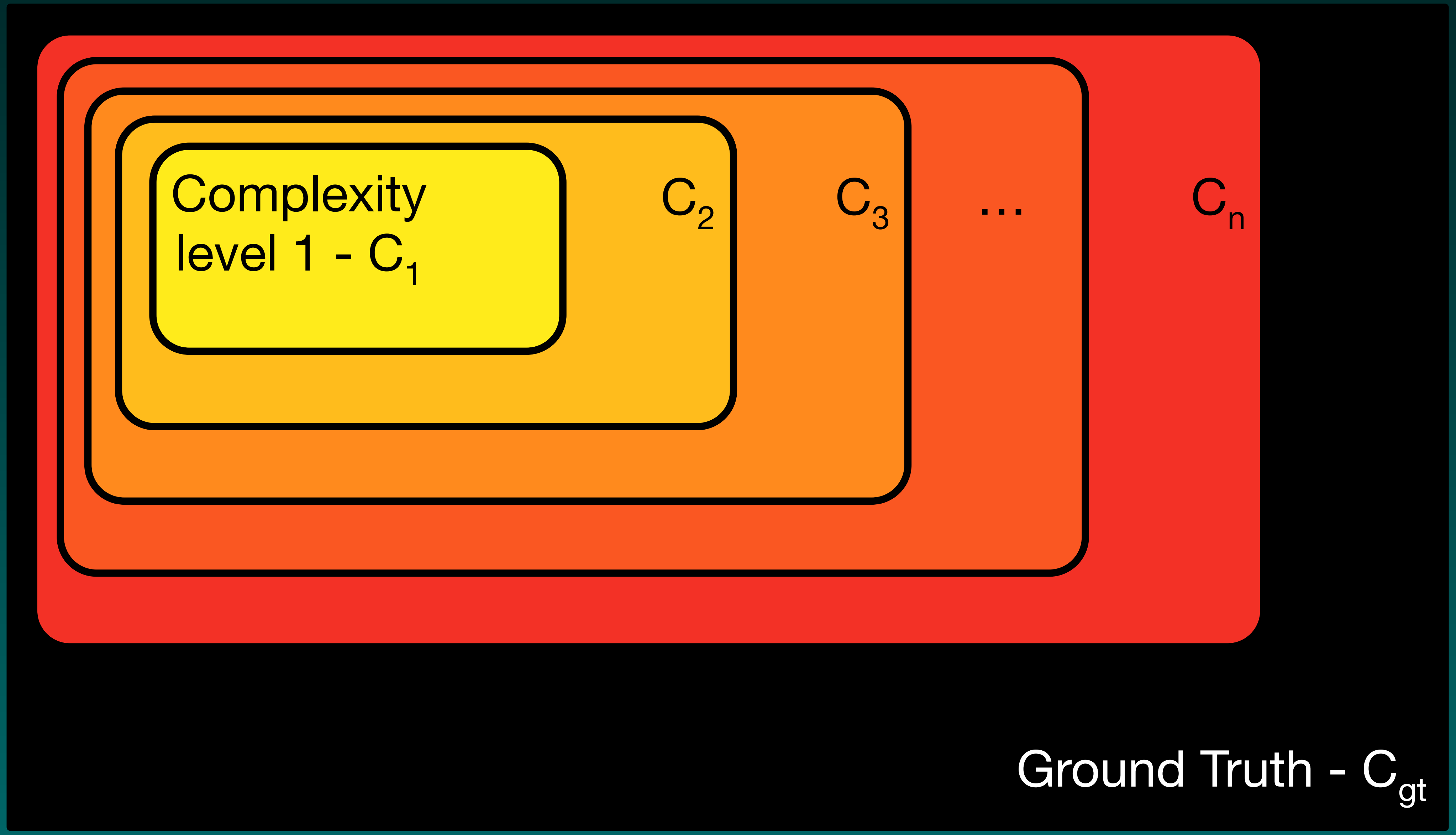
Problem transformation

Complexity spectrum - Layered approach

- For a problem that is too complex:
  => The likelihood of finding a solution is lower
  => The time it takes is longer (?)
  => The likelihood of an ad hoc solution is higher

- Evaluation of all requirements
  => Primary and **secondary requirements**

- Different **complexity levels** leading to the **ground truth**
  => Better understanding of the problem
  => Speeding up the automated search



$$C_1 \subset C_2 \subset \ldots \subset C_n \subset C_{gt}$$

Problem transformation

Complexity levels and complexity dimensions

Complexity level 1

		step
	i	1
dimensions	j	2
	k	1
	l	-
	m	-

Complexity level 2

		step
	i	2
dimensions	j	3
	k	1
	l	1
	m	-

Complexity level 3

		step
	i	2
dimensions	j	4
	k	1
	l	2
	m	1

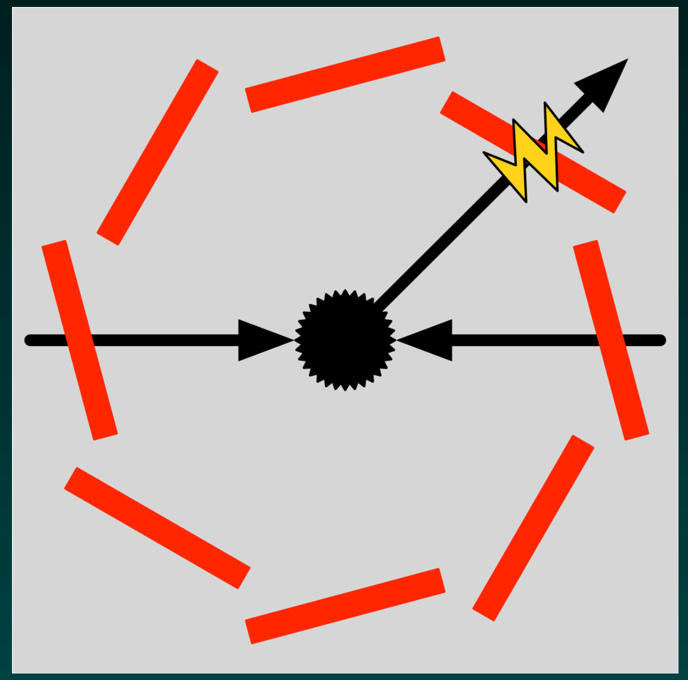
- Complexity levels:
  => Sets of adjusted **complexity dimensions**

- An obvious example:
  => **Scale** is an increasing complexity dimension
  (Event count, track count, concentration/sparsity, …)

$$C_1 = \{d_i, d_j, d_k\}$$
$$C_2 = \{d_i, d_j, d_k, d_l\}$$
$$C_3 = \{d_i, d_j, d_k, d_l, d_m\}$$

REDVID for reduced simulations

Reusable simulation tool



- **REDuced VIrtual Detector (REDVID)**
  => Fully (re)configurable, modular, complexity-aware
  => Reduced-Order Models (ROM) for detectors
  => Event simulator with complexity-reduced behaviour
  => Generates **synthetic data -> Tracks** and **associated hits**

Publications

- *"Reduced Simulations for High-Energy Physics, a Middle Ground for Data-Driven Physics Research"*
  => Computational Science (ICCS) 2024
  [doi: 10.1007/978-3-031-63751-3_6] [arXiv:2309.03780]

- *"Novel Approaches for ML-Assisted Particle Track Reconstruction and Hit Clustering"*
  => Connecting The Dots (CTD) 2023
  [arXiv:2405.17325]

REDVID **code** and **reference data sets** available online: https://VirtualDetector.com/redvid

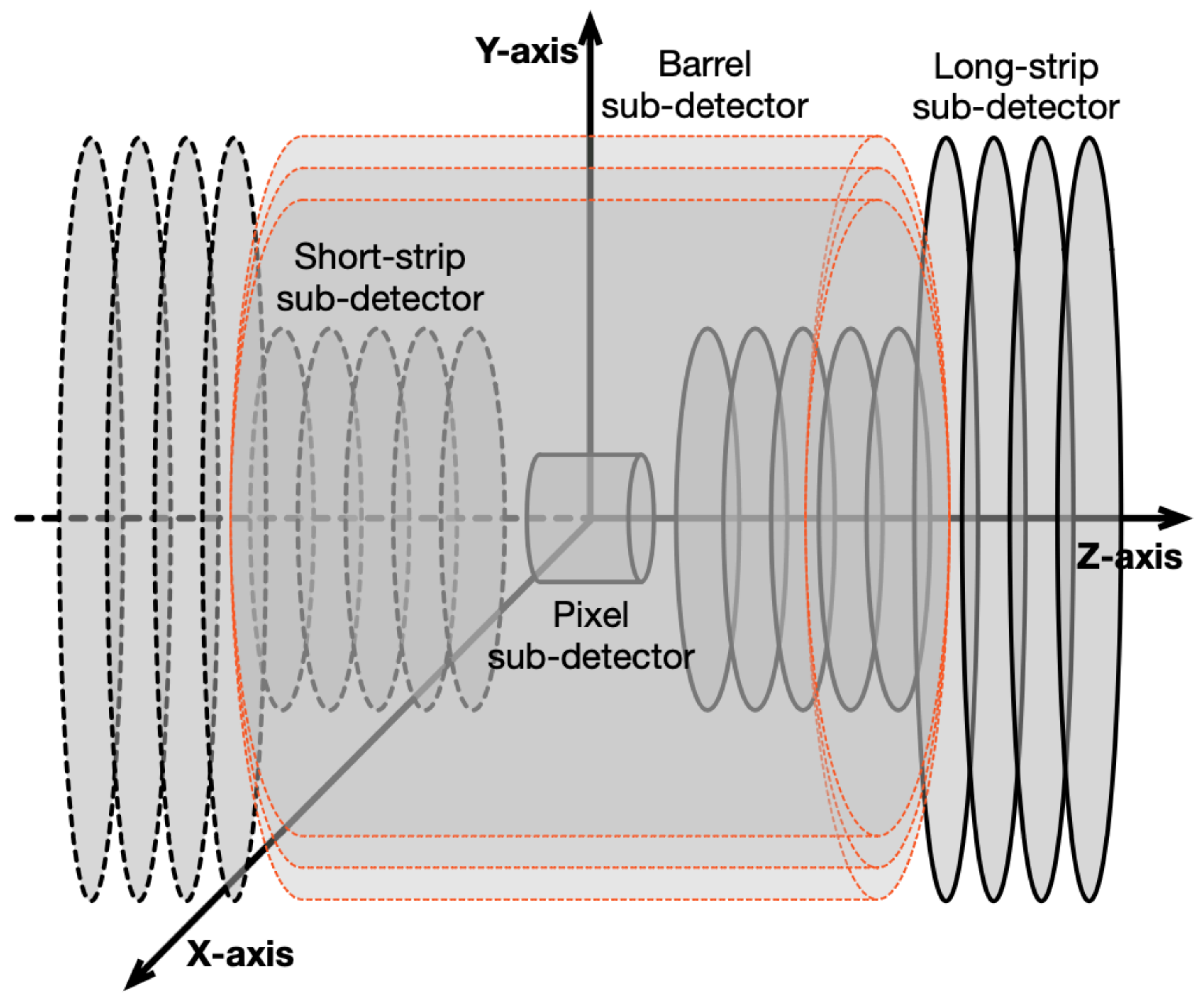
REDVID for reduced simulations

Available features

- Detector geometry
  => Fully customisable virtual detector
  => 2D (for edu.) and 3D experiments
  => A few simple shapes for detector elements

- Event simulation
  => Track counts (random and set)
  => Track randomisation protocols (propagation pattern)
  => Track types (linear, helical uniform, helical expanding)

- Hit point recording
  => Coordinate smearing
  => Recording probability (for Pixel and holes)



Lots of randomisation for **non-determinism**

Complete list on our website

REDVID for reduced simulations

Computational performance

- System resource utilisation
  => Execution time
  => CPU-time
  => Internal SW probes: Granular information per functionality

- Execution modes:
  Parallel execution, **batch** processing, **parallelised batch** processing

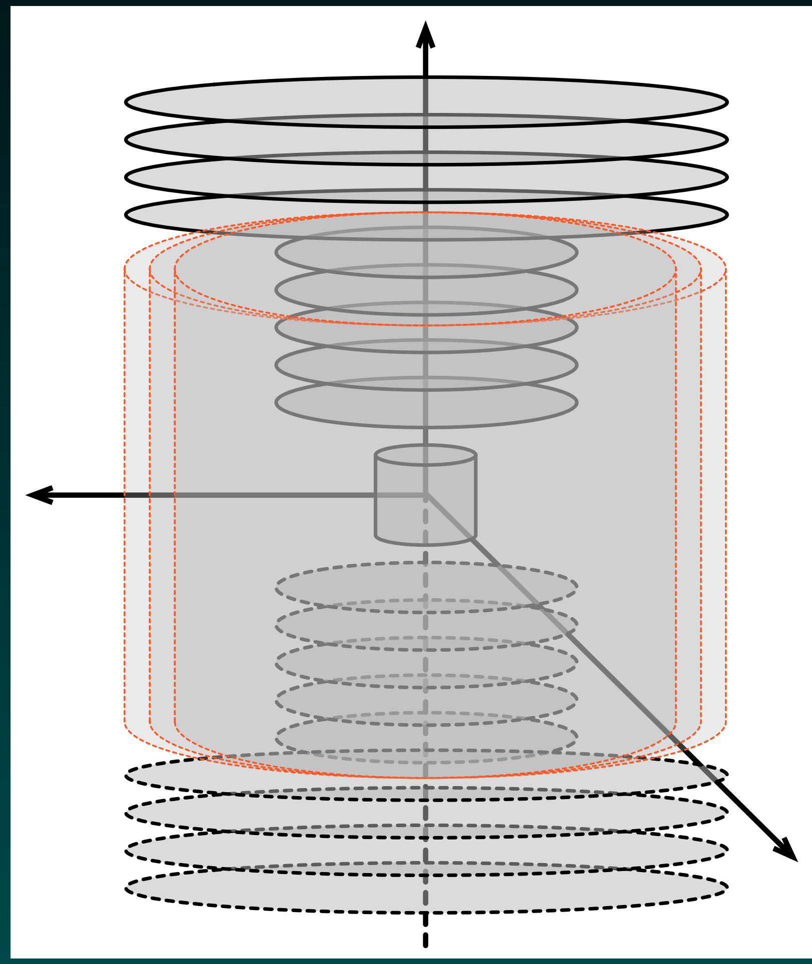
- **Scales linearly**
  => Very desirable!

Table 1: REDVID execution CPU-time cost for simulations of 1 000 events with various track concentrations. All values are in milliseconds. Full simulation times are provided in minutes as well. Even though REDVID is developed in Python, computational cost figures indicate efficiency for frequent executions.

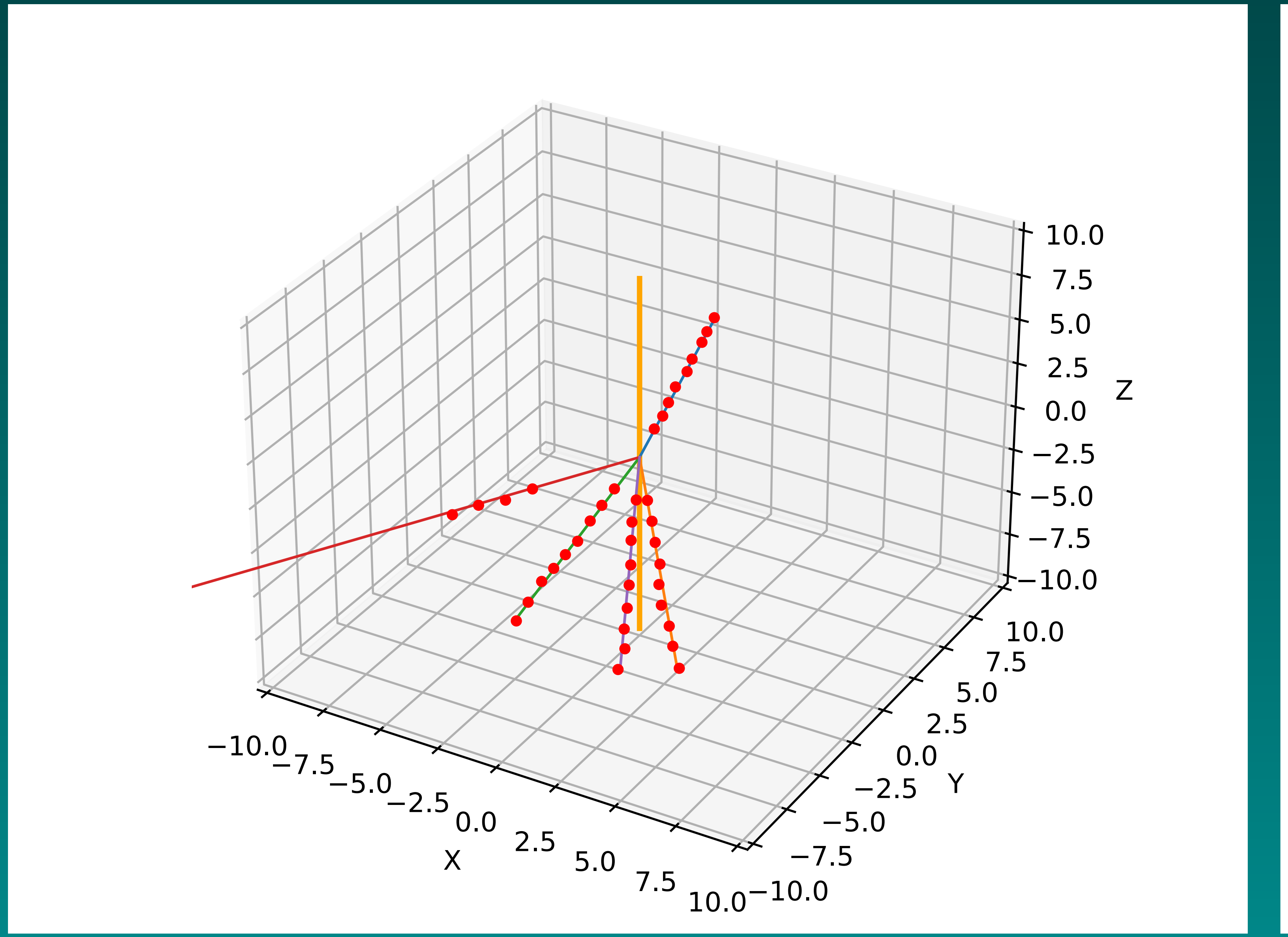
Recipe for 1 000 events	3D detector spawning	Track randomisation per event - Mean	Hit discovery per event - Mean	Full simulation of 1 000 events (minutes)
1 track per event	0.025	0.043	1.463	2 731.17 (0.05)
10 tracks per event	0.025	0.083	13.429	15 418.589 (0.26)
100 tracks per event	0.025	0.465	129.864	137 623.954 (2.29)
1 000 tracks per event	0.025	4.582	1 285.989	1 353 396.641 (22.56)
10 000 tracks per event	0.024	43.765	12 496.208	13 591 628.526 (226.53)

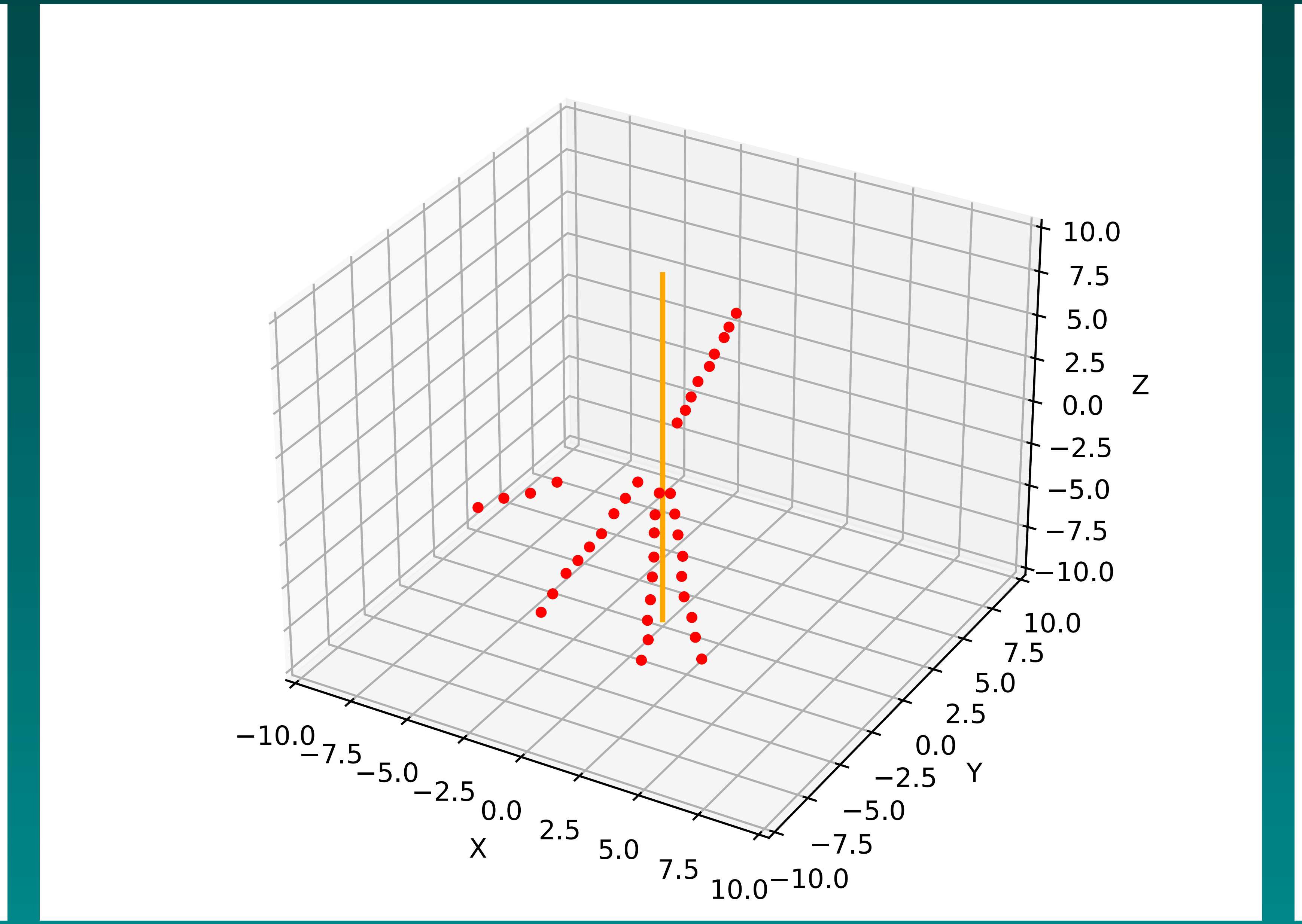
REDVID - Example events

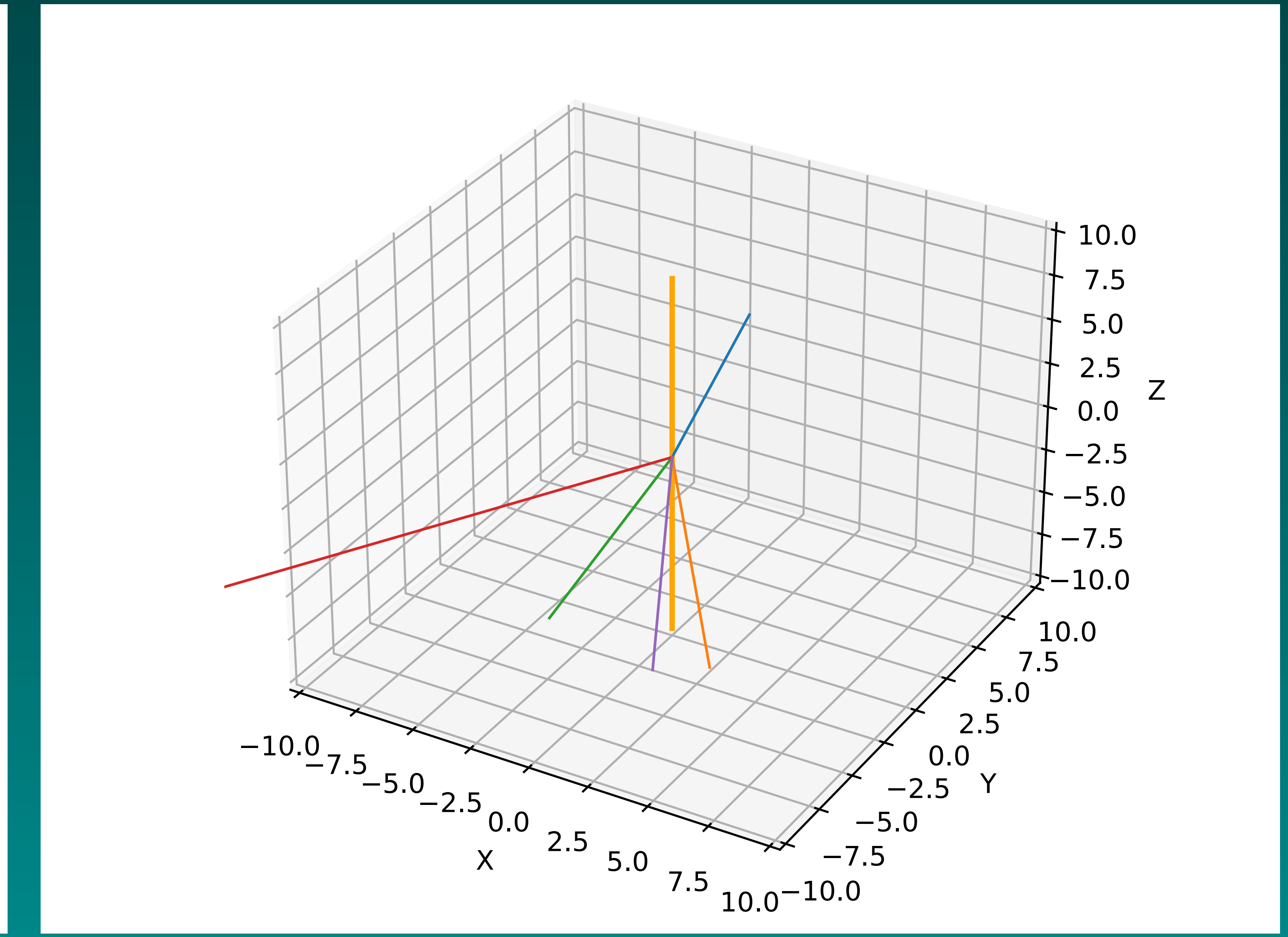
Linear track propagation & hits



5 tracks, linear, noisy

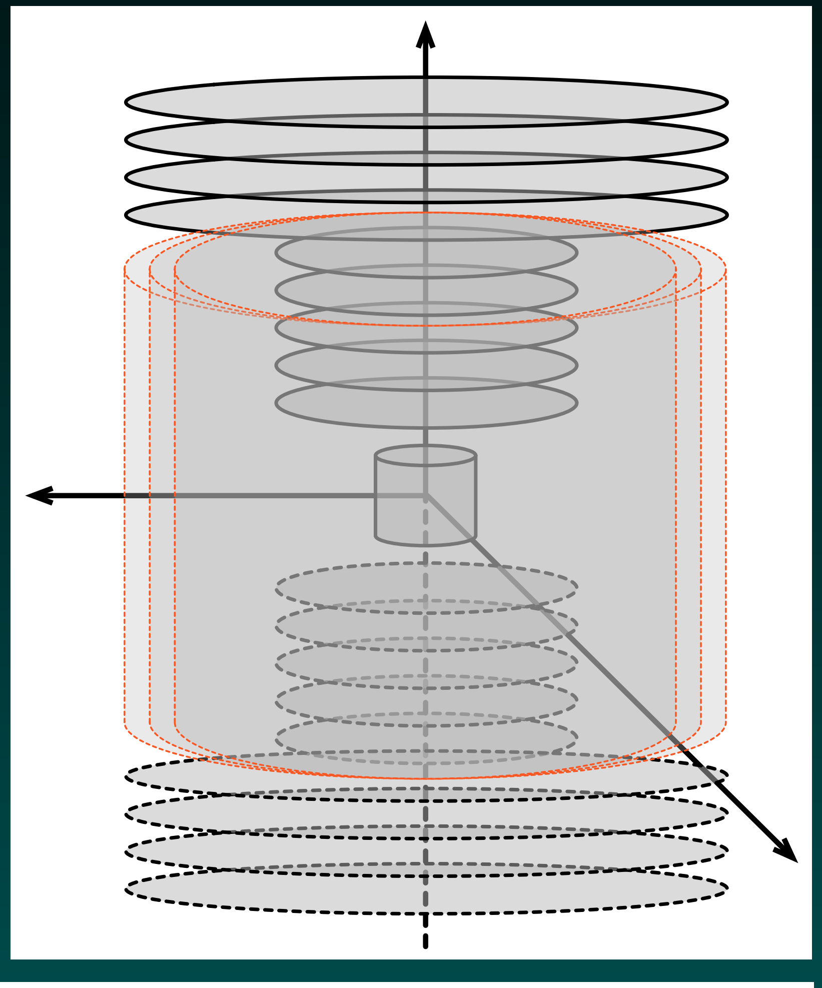




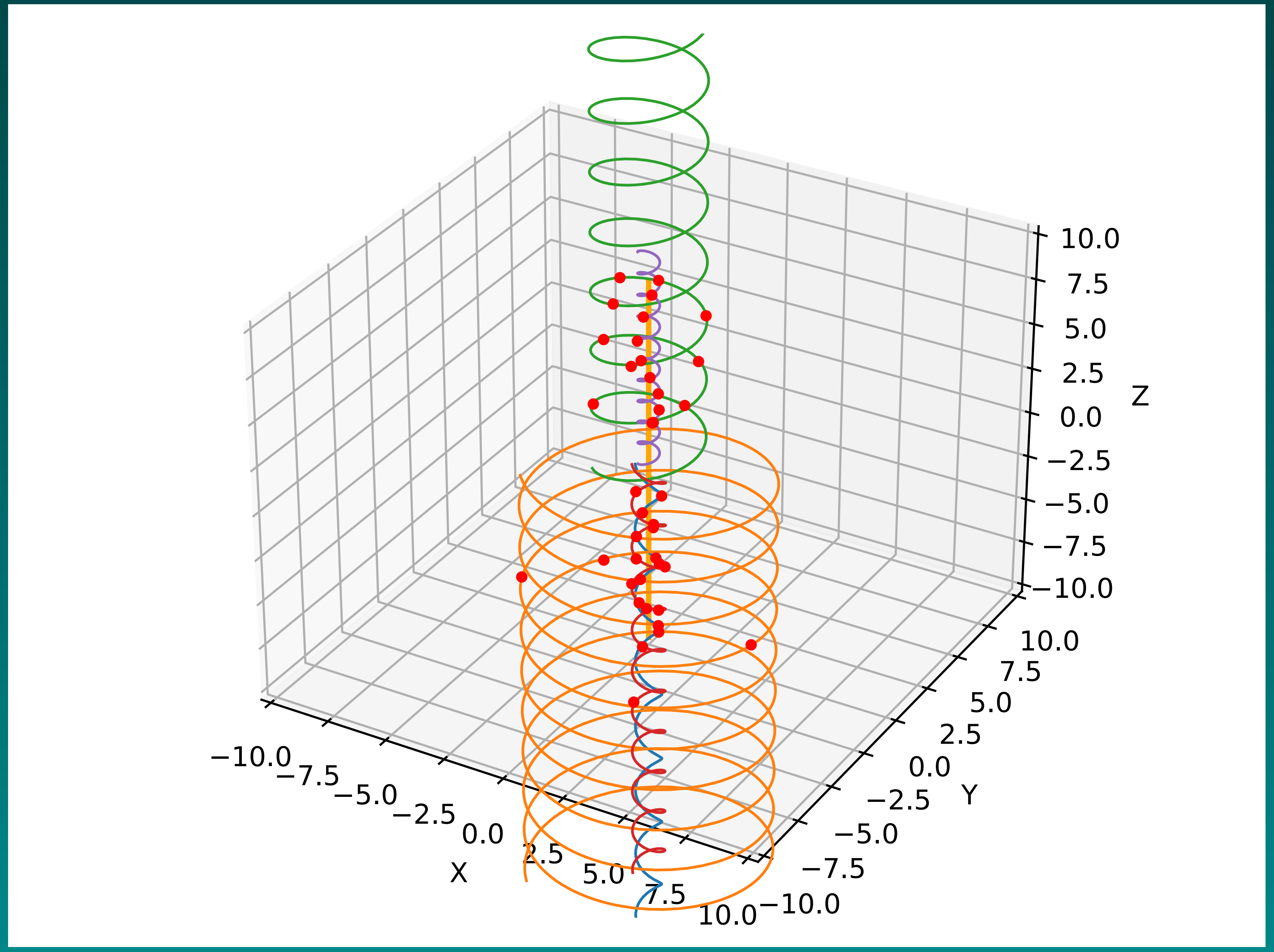


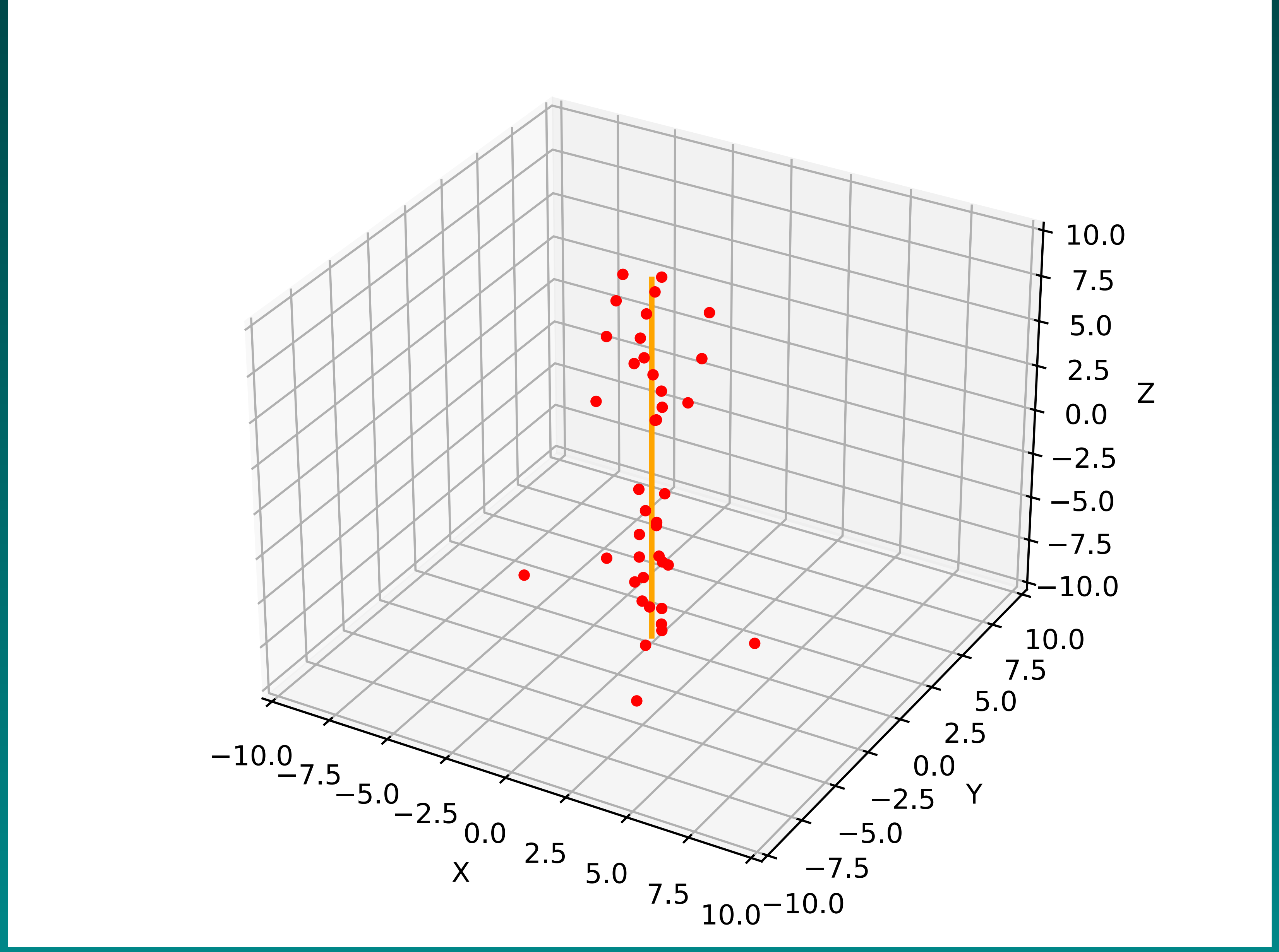
REDVID - Example events

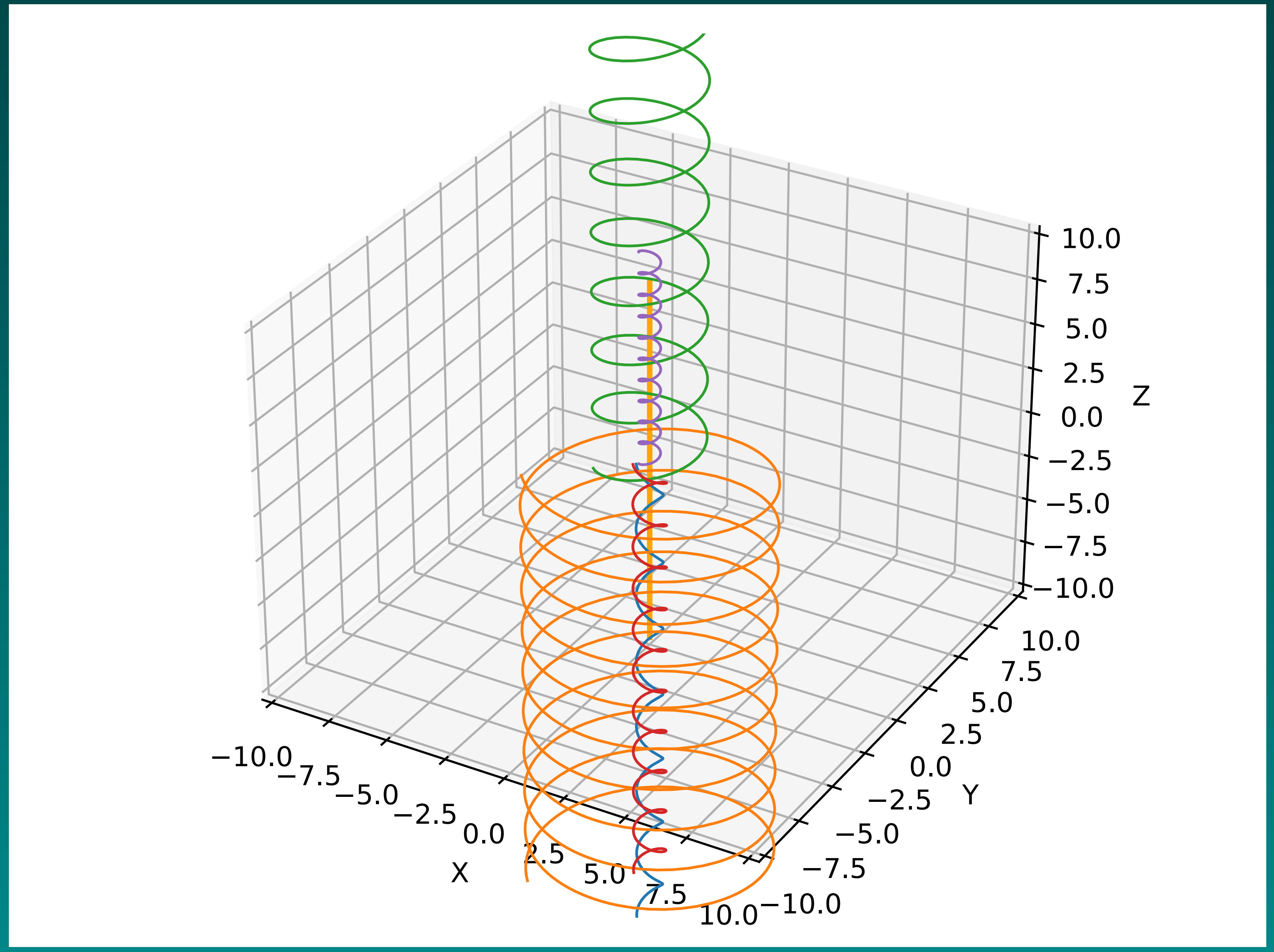
Helical uniform track propagation & hits



5 tracks, helical uniform, noisy

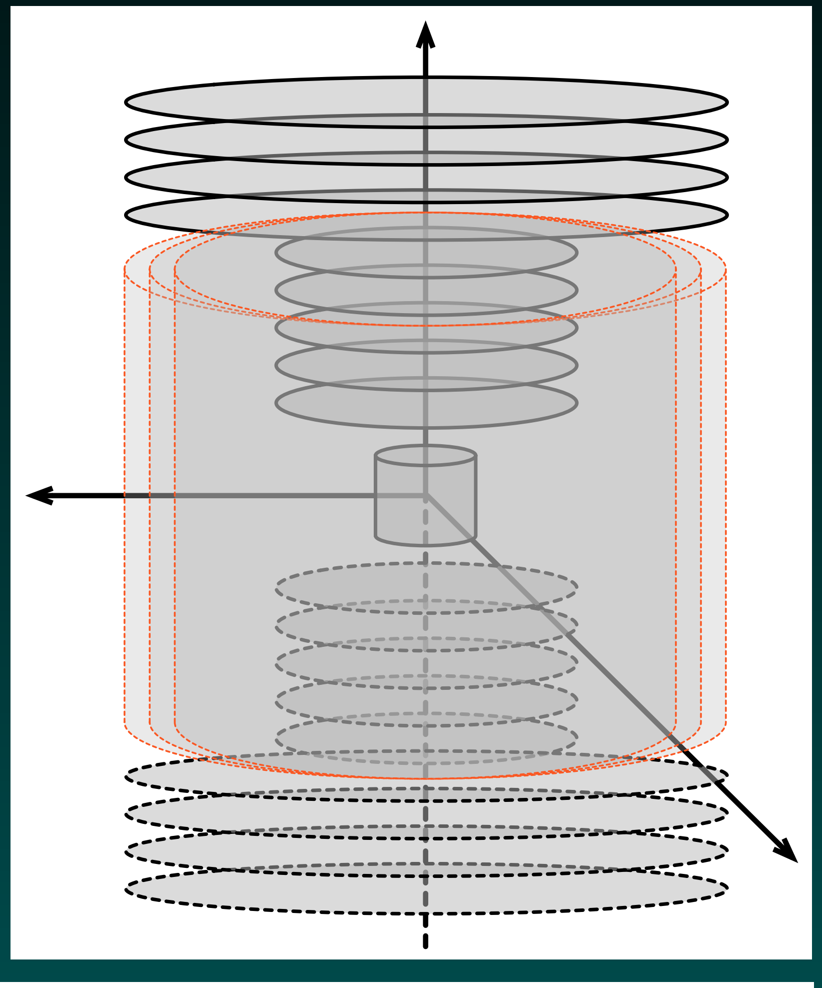




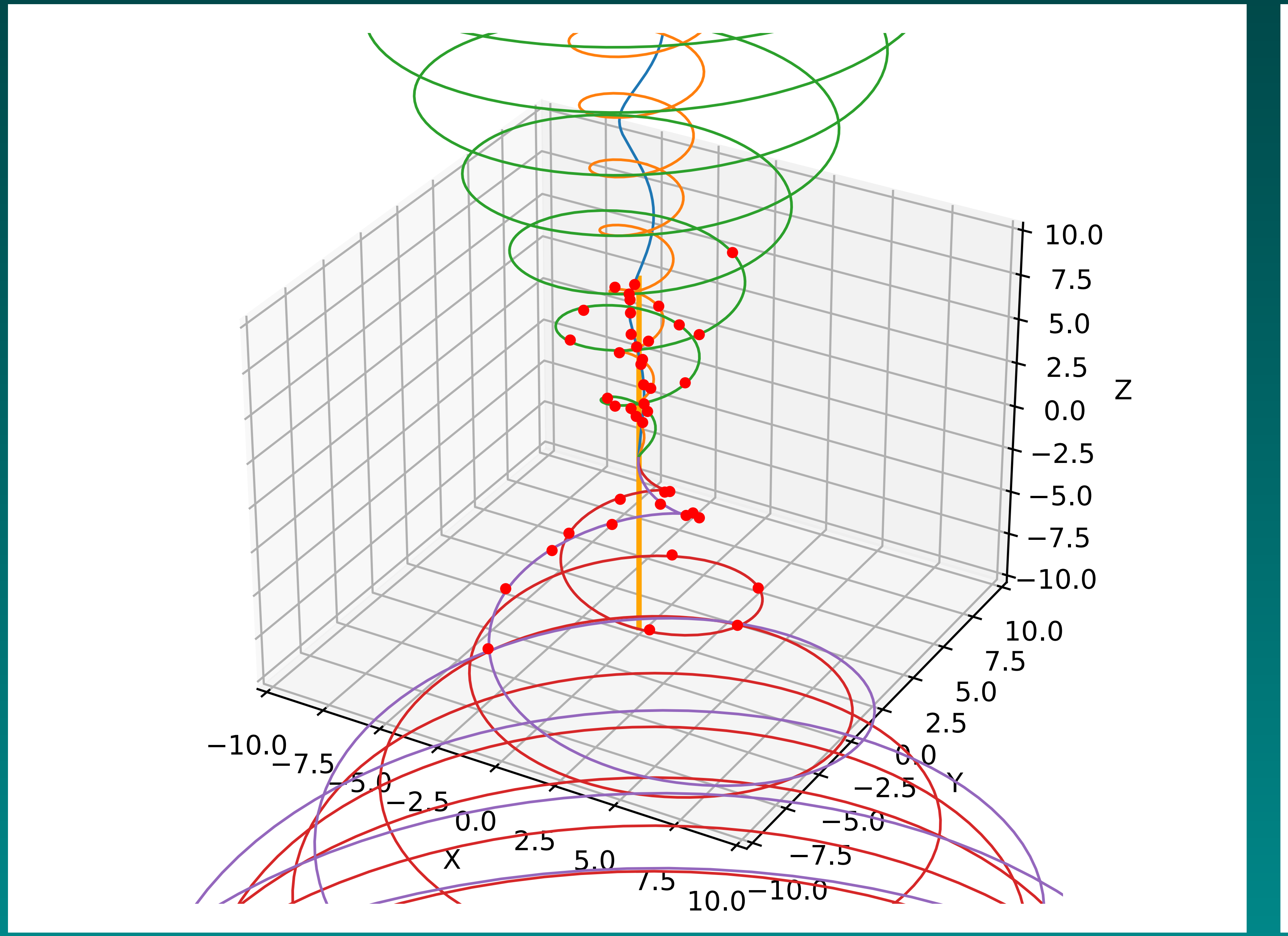


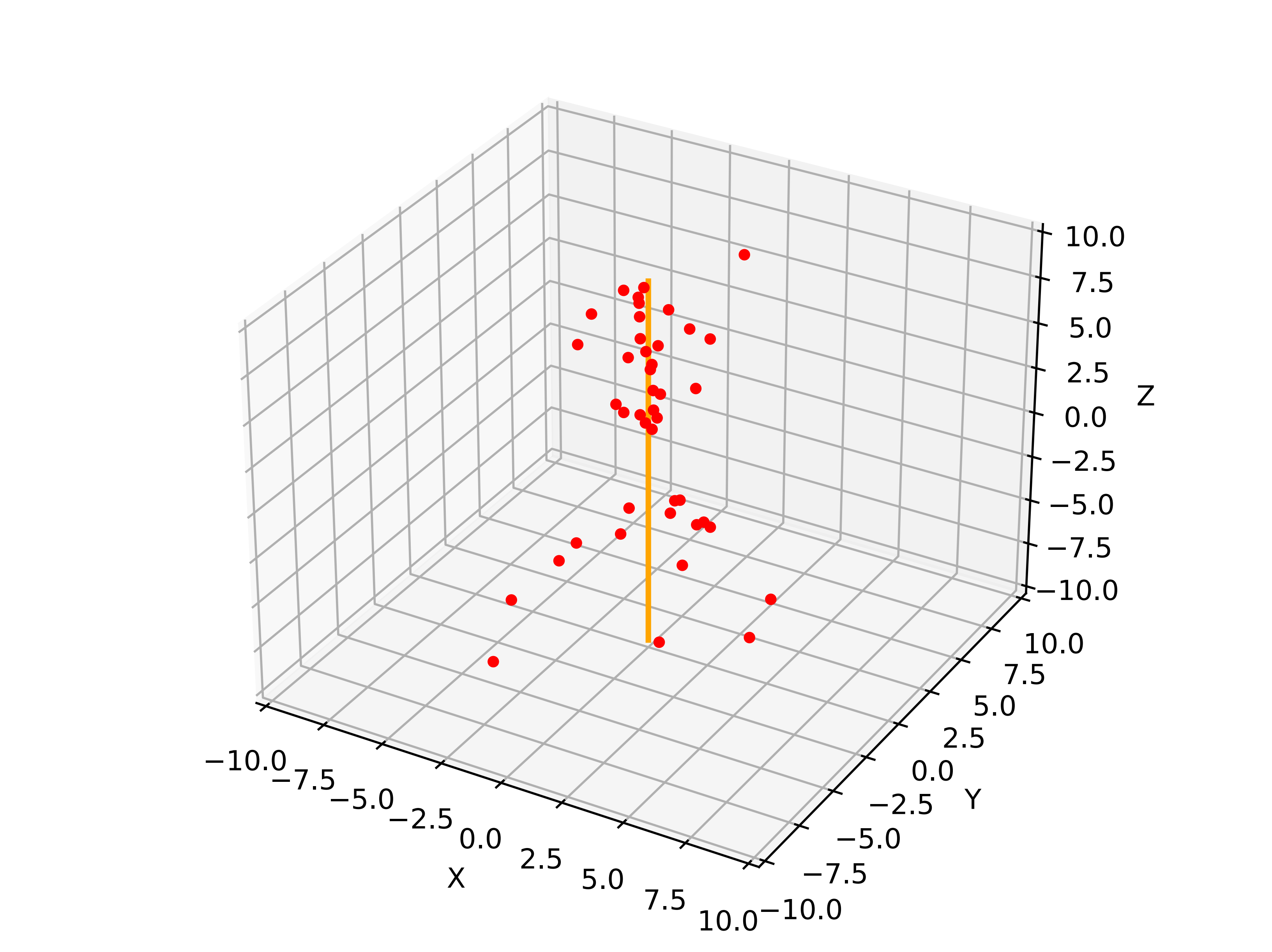
REDVID - Example events

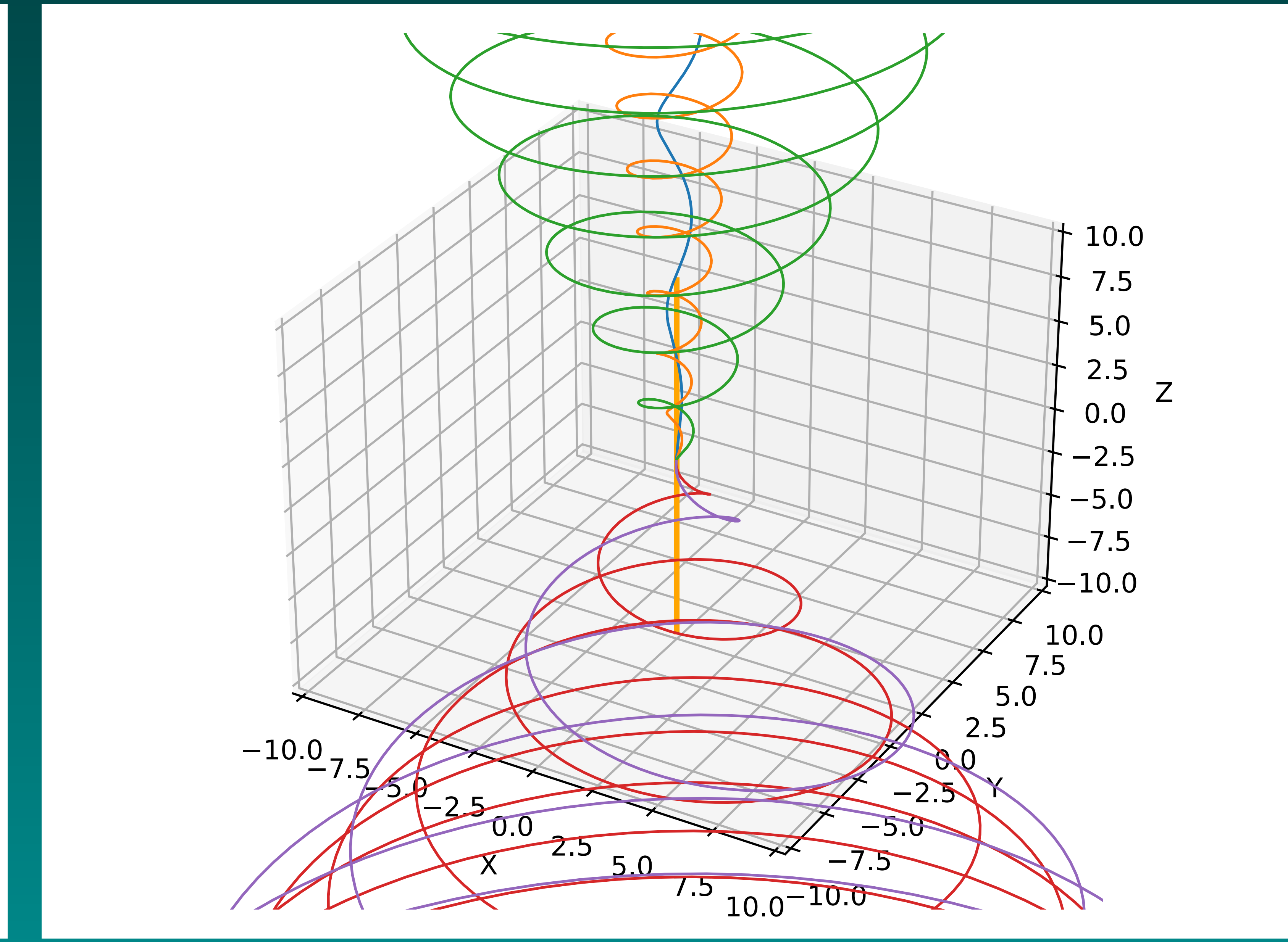
Helical expanding track propagation & hits



5 tracks, helical expanding, noisy





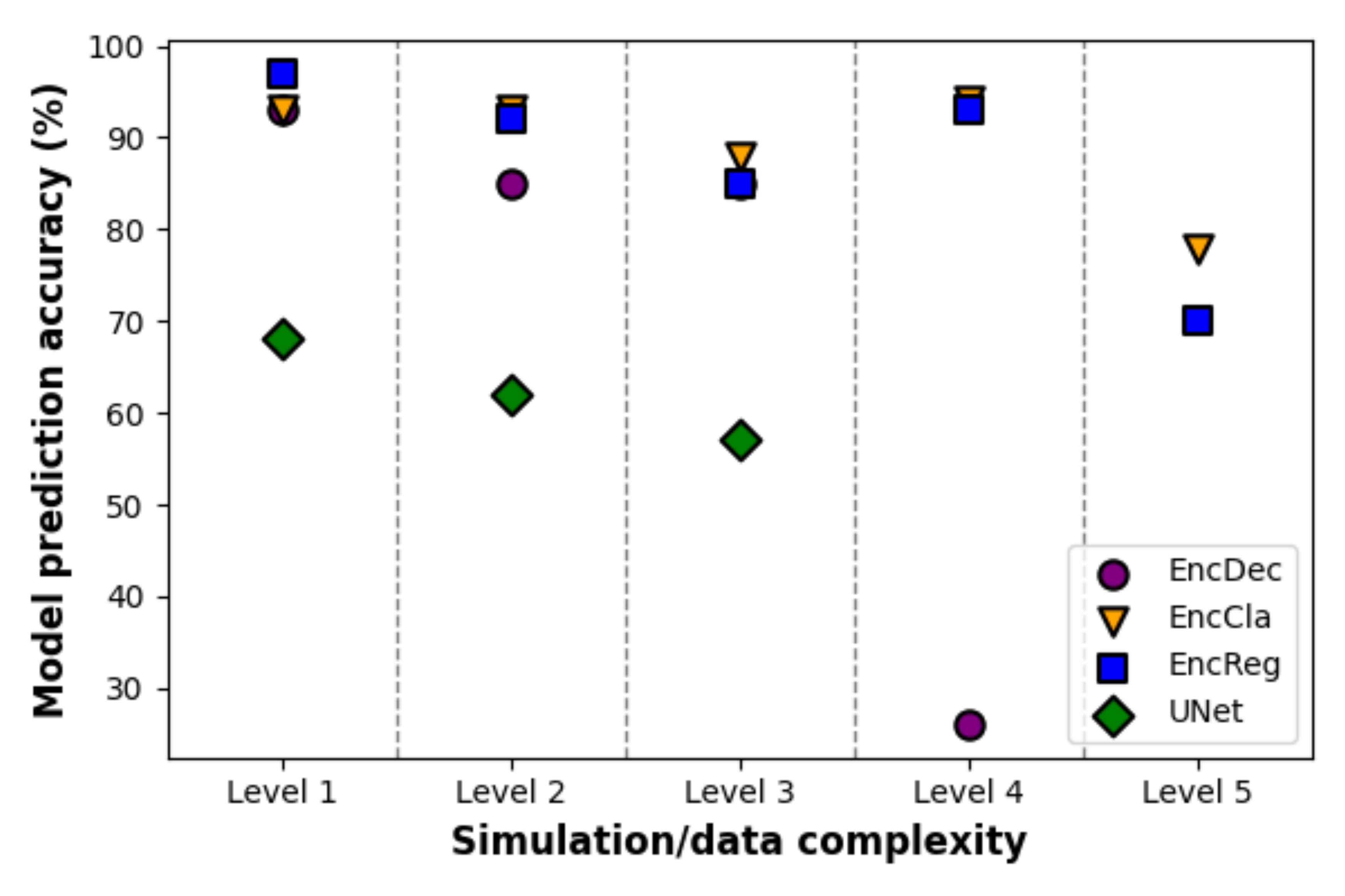


Usage in ML-model evaluation

- Two ML model architectures: **Transformers** and **U-Nets**
  A **manual** application of our method (not automated search)

- Transformer - Attention mech.:
  1. Similar to language translation, hits to tracks
  => Guess the next hit from a seed …
  2. Encoder-only transformer as a classifier,
  to assign hits to spatial bins
  3. Encoder-only transformer to regress the track parameters
  => Hit to road classification

- **4.** U-Net - Convolution mech.:
  => Track discovery with interpolation



Further details on October 24th, during this CHEP talk.

Complexity reduction => **Cost-effective and timely evaluation** of different solutions

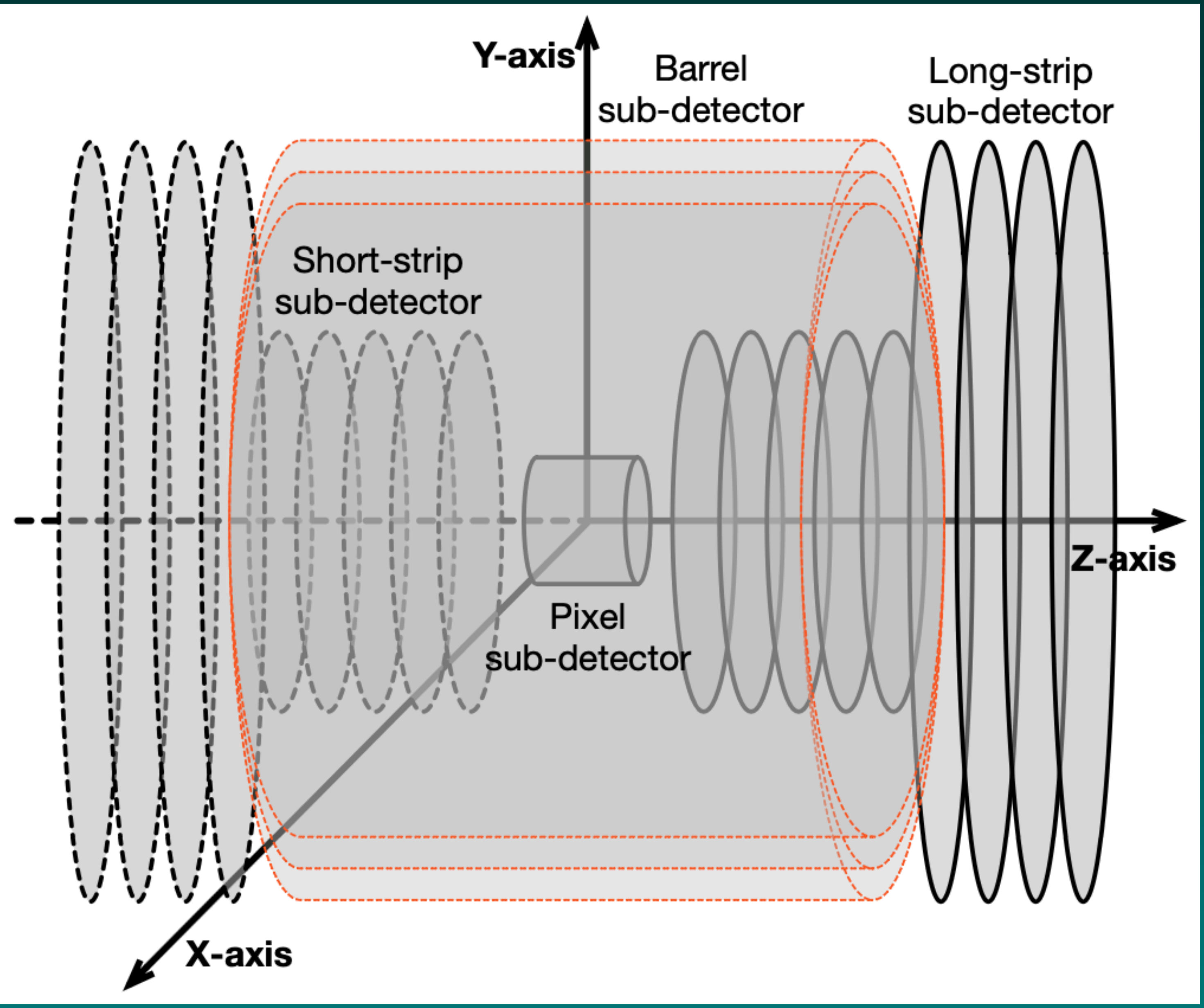
What is next for REDVID?

- Additional track **randomisation protocols** (in progress)

- New track types, multiple types (in progress)
  => **Jets**: Basically localised concentrations

- On demand simulations (bursts)
  => For the simulation-in-the-loop aspect

- Incomplete tracks
  => Early **termination** of tracks
  => **Secondary** tracks



and much more …

Thanks!
Questions?