Pre-Learning a Geometry Using Machine Learning To Accelerate High Energy Physics Detector Simulations

Vangelis Kourlitis, Walter Hopkins, Doug Benjamin

November 23rd 2020

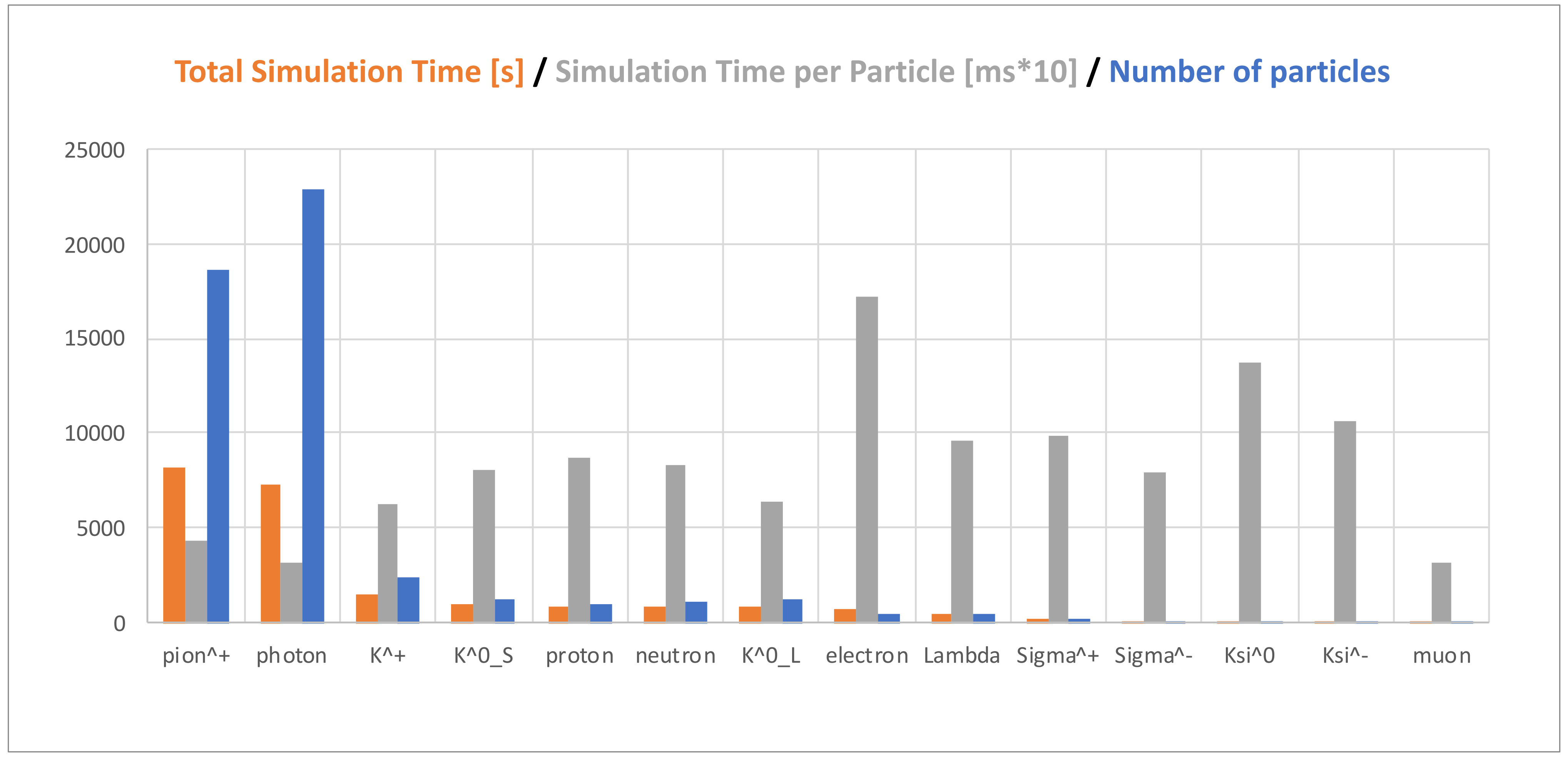
Introduction

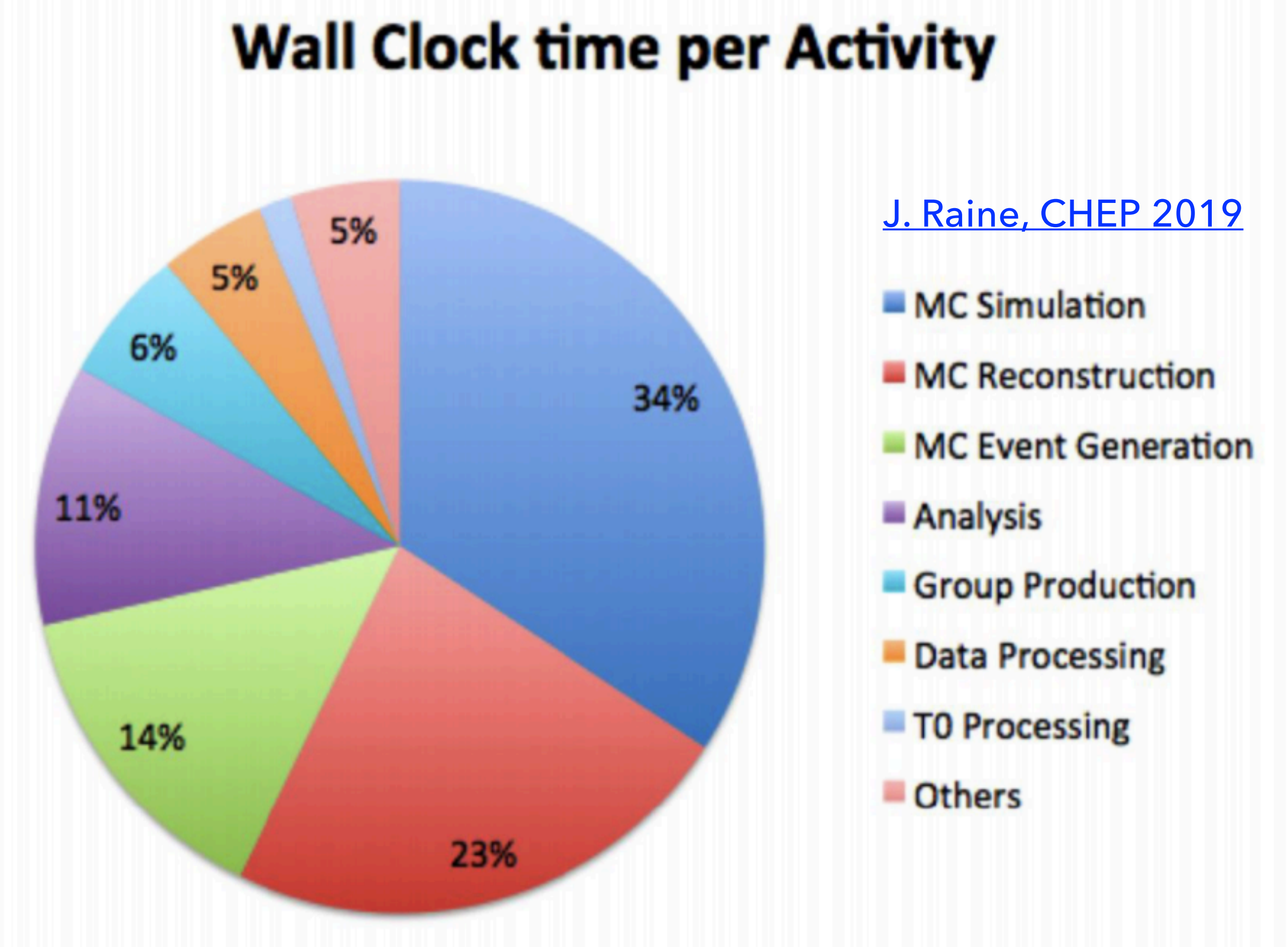
Accelerate Geant4:

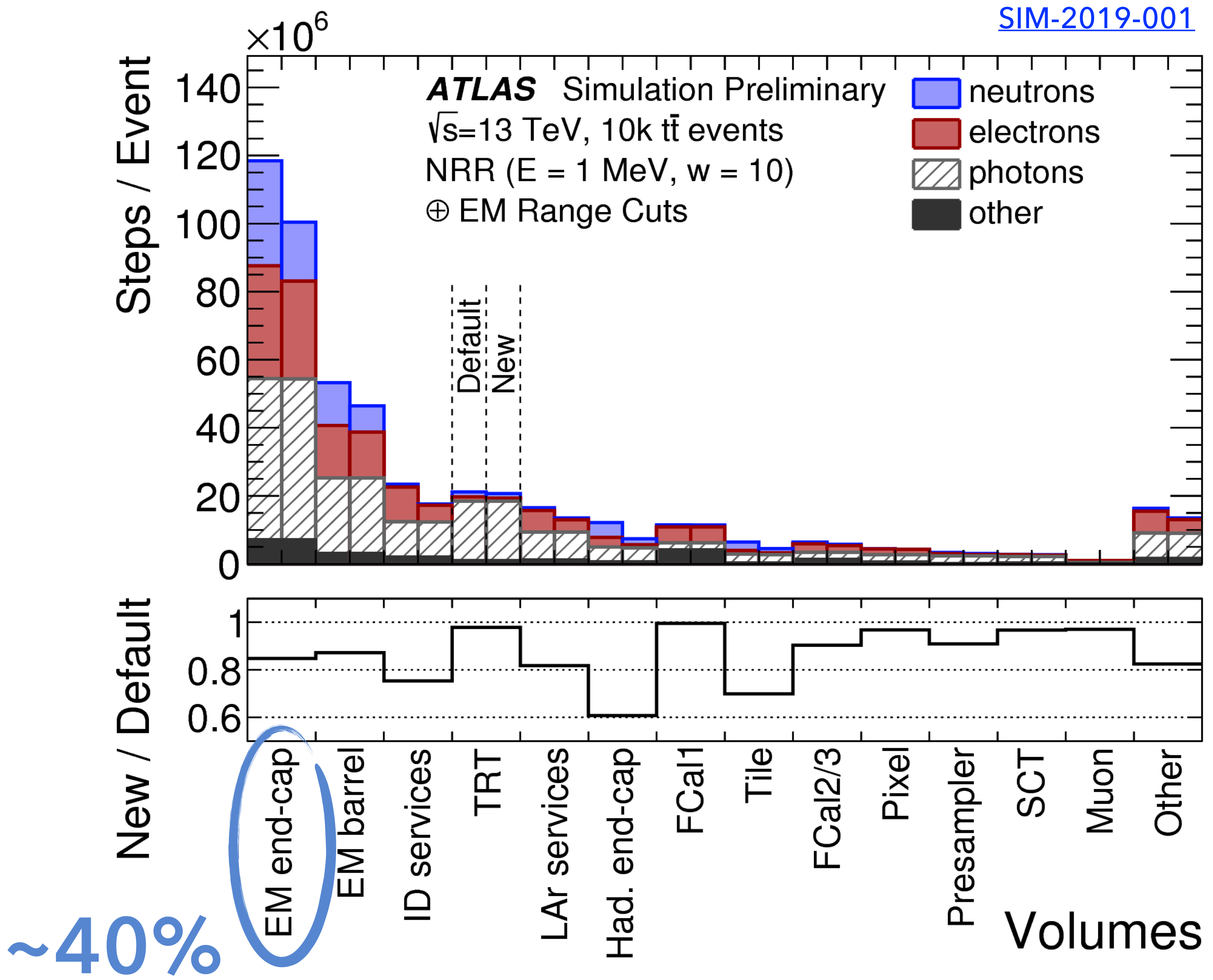
The ATLAS detector simulation paradigm

Facts

1. Studies have shown EM calorimeters dominate the simulation load (steps).
2. Electrons and neutrons require long simulation time but it is really **photons and pions** that drive the whole process.







Geant4 Profile

photons on ATLAS end-cap calorimeters

Callees	CPU Time: Total
G4SteppingManager::Stepping	100.0%
G4SteppingManager::DefinePhysicalStepLength	66.7%
G4VProcess::AlongStepGPIL	58.2%
G4Transportation::AlongStepGetPhysicalInteractionLength	51.7%
G4Navigator::ComputeStep	34.1%
G4NormalNavigation::ComputeStep	20.9%
G4VoxelNavigation::ComputeStep	10.8%

The point: methods exploring the geometry* are taking significant amount of the simulation time.

* Locate position inside geometry tree and calculate distance to next boundary in order to limit step.

The Idea

Surrogate modeling within Geant4: *Could we speed-up the geometry exploration by using a* pre-defined/learned map instead of algorithmic calculations *in each step?*

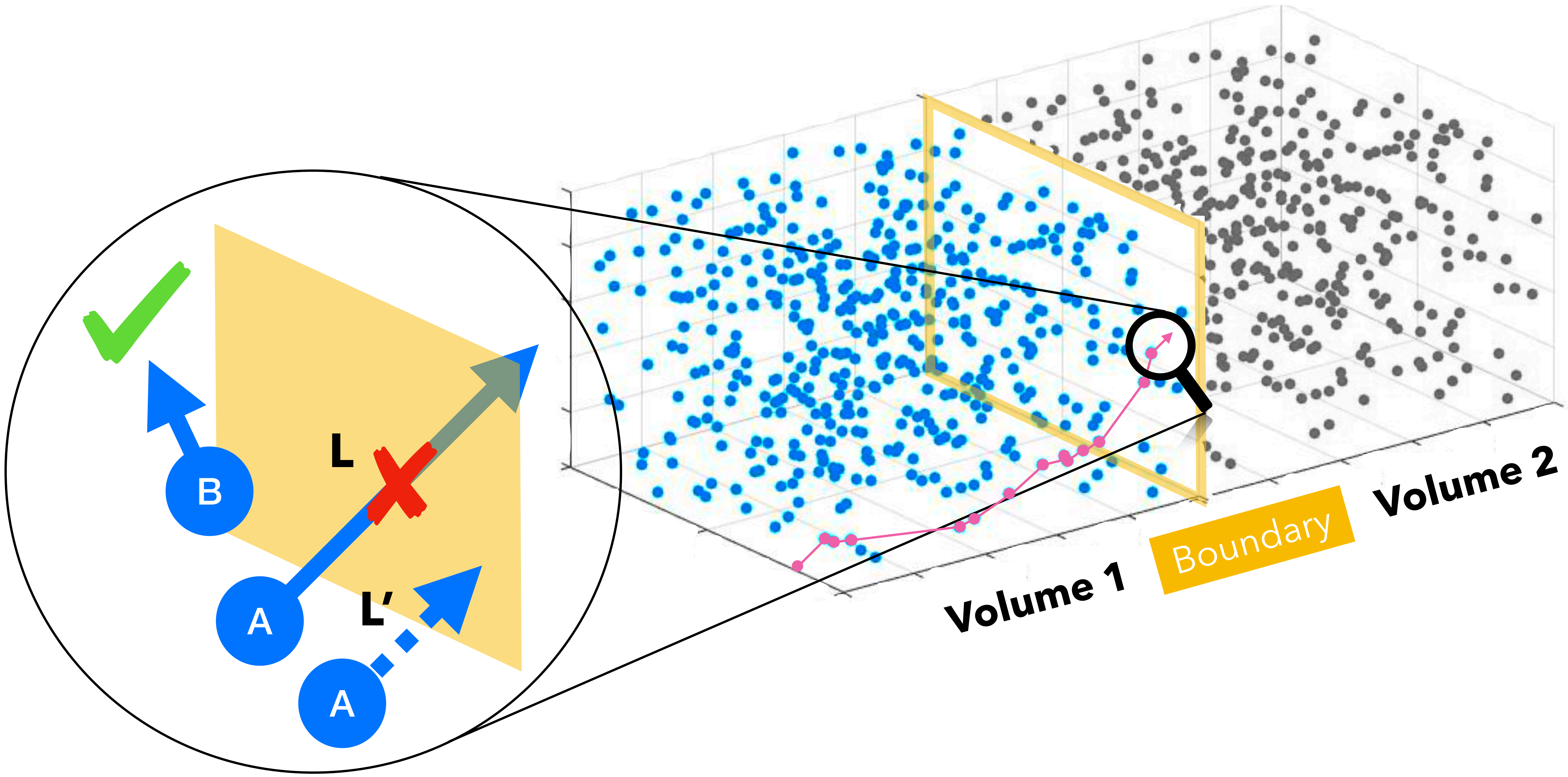
Machine learning regression technique trained for a particular geometry (e.g. ATLAS EMEC)

Use industrial libraries, optimized for different architectures (CPU or GPU), as abstraction layer

Much easier & assured future portability

The Idea

Surrogate modeling within Geant4: *Could we speed-up the geometry exploration by using a* pre-defined/learned map instead of algorithmic calculations *in each step?*



Inputs:

- Position (x, y, z)
- Step direction (x′, y′, z′)

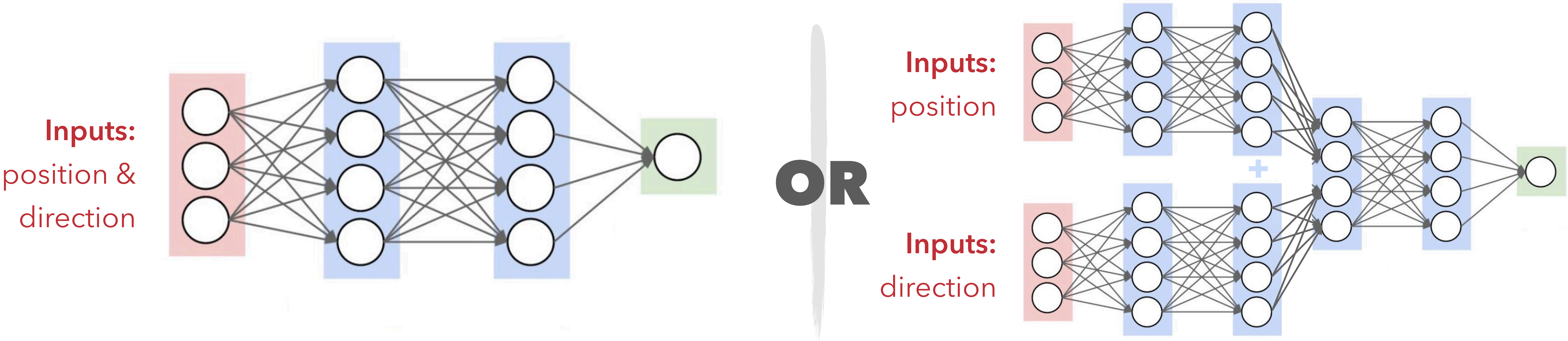
Output:

- Geometrically safe step length (L′)

ML Architecture

Baseline: fully connected layers, concatenated or split inputs

Main advantage: fast inference (compared to convolution operations)



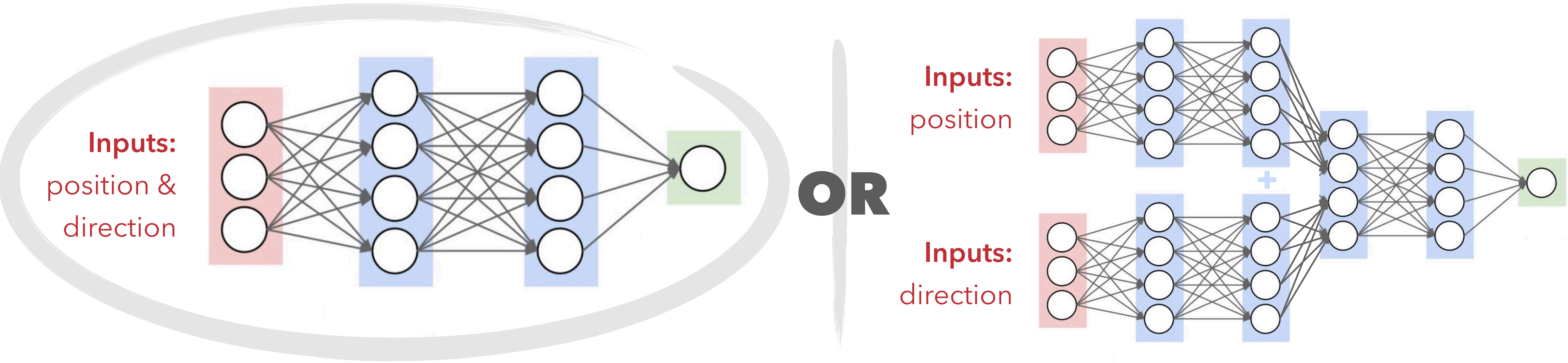
Output: Geometrically safe step length

* **Bonus:** ML architecture idea/study on backup.

ML Architecture

Baseline: fully connected layers, concatenated or split inputs

Main advantage: fast inference (compared to convolution operations)



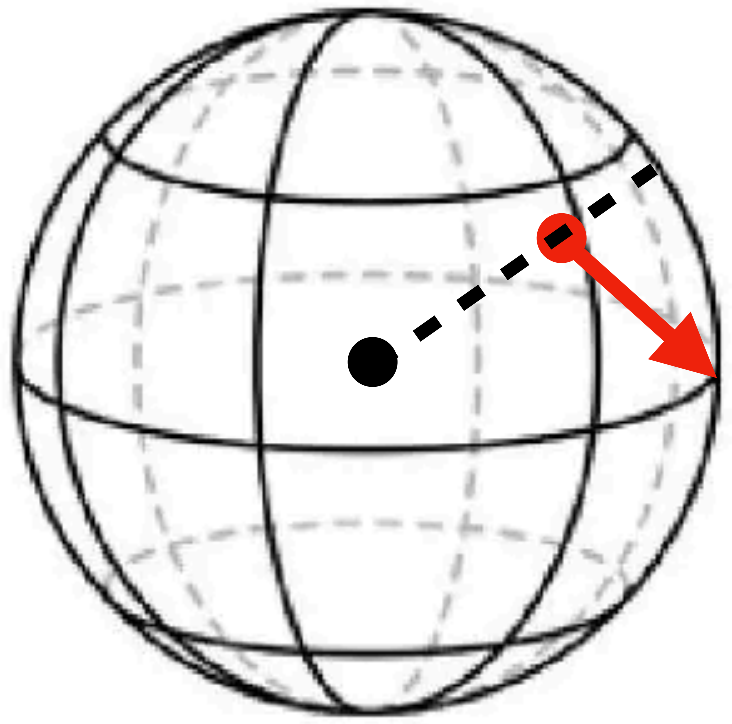
Output: Geometrically safe step length

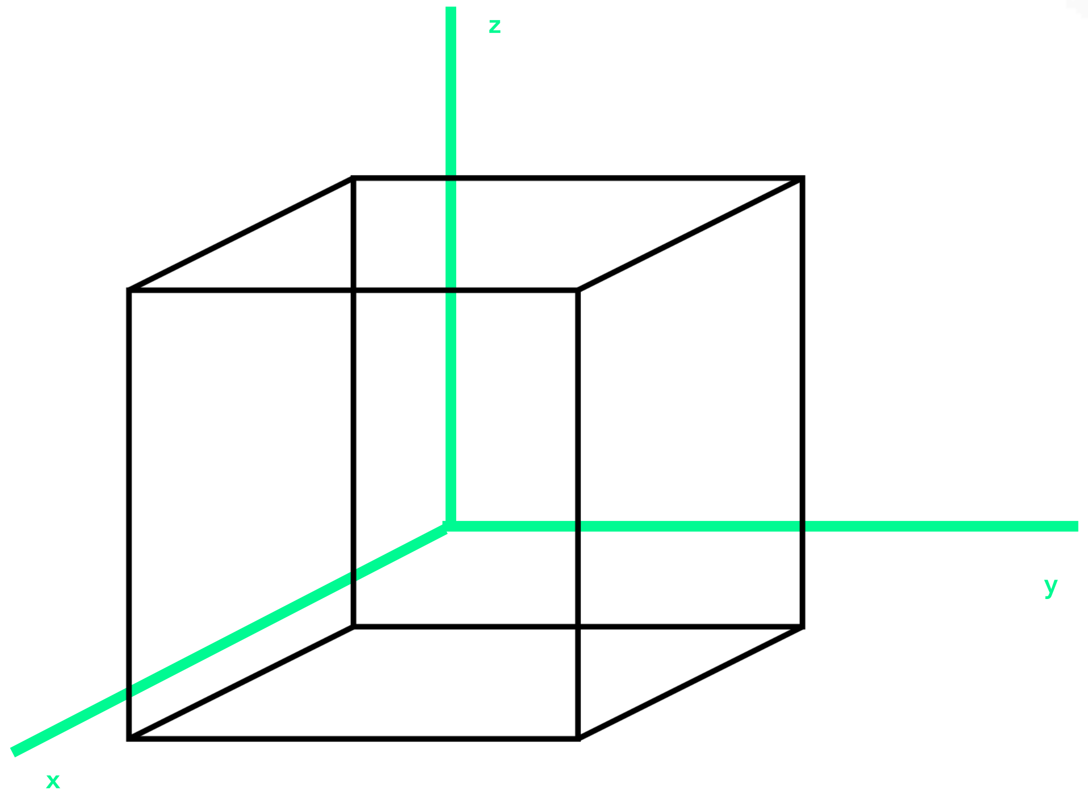
* **Bonus:** ML architecture idea/study on backup.

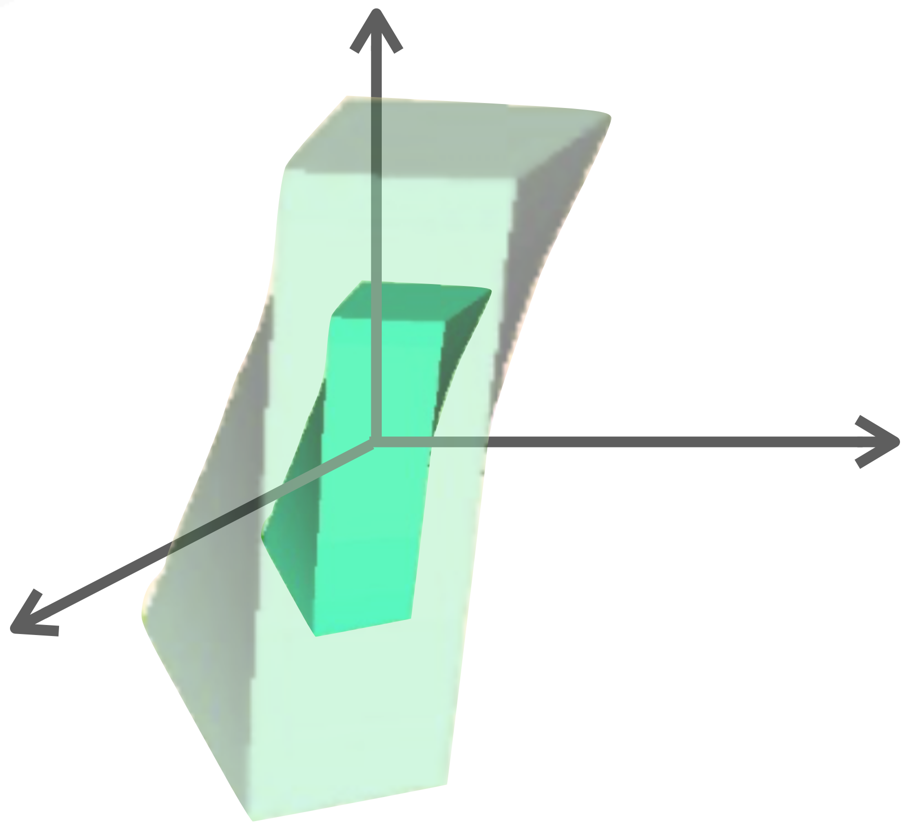
Simplified Geometries

Explore simplified geometries to study the feasibility of the surrogate modeling using ML

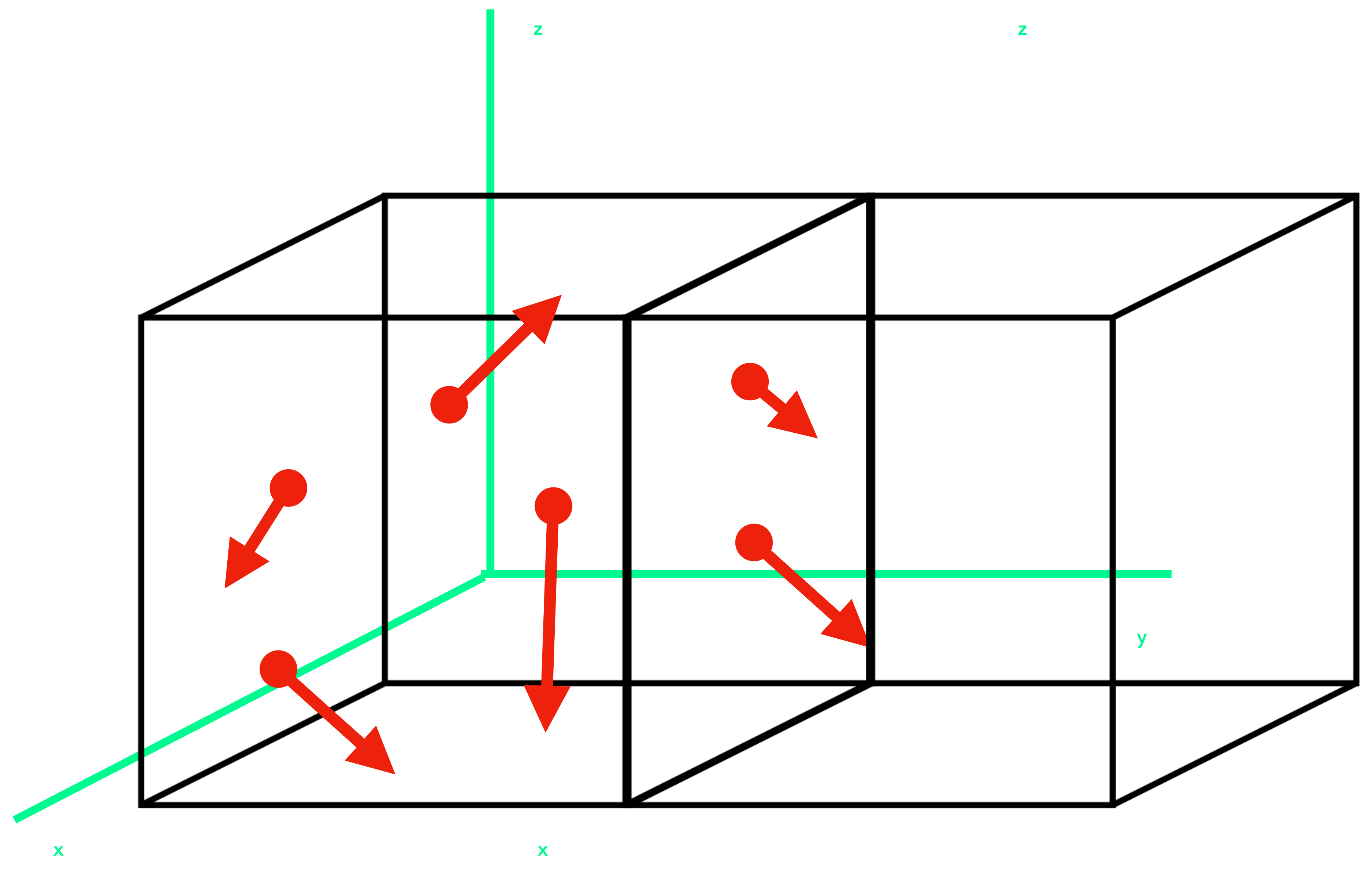
1. Sphere
2. Cube
3. (Nested) Twisted-trapezoids
4. Multi-layer calorimeter (rectangular cuboid layers)







Training/Test Data Collection



1. Sample geometry in random points & directions.
2. Geant4 application shooting *geantinos** and calculating the geometry-limited step length.
   * *Idealistic particles with no physics interaction.*
3. Write-out (csv) the position, direction and calculated length.

The usage of *geantinos* greatly speeds-up the Geant4 simulation runtime, even when using realistic geometries – 1M particles in *O*(10m).

Loss Function

Is it critical to **avoid over-predictions** of the geometrically safe step length.

Otherwise, particles can *stuck* around the boundary between geometries.

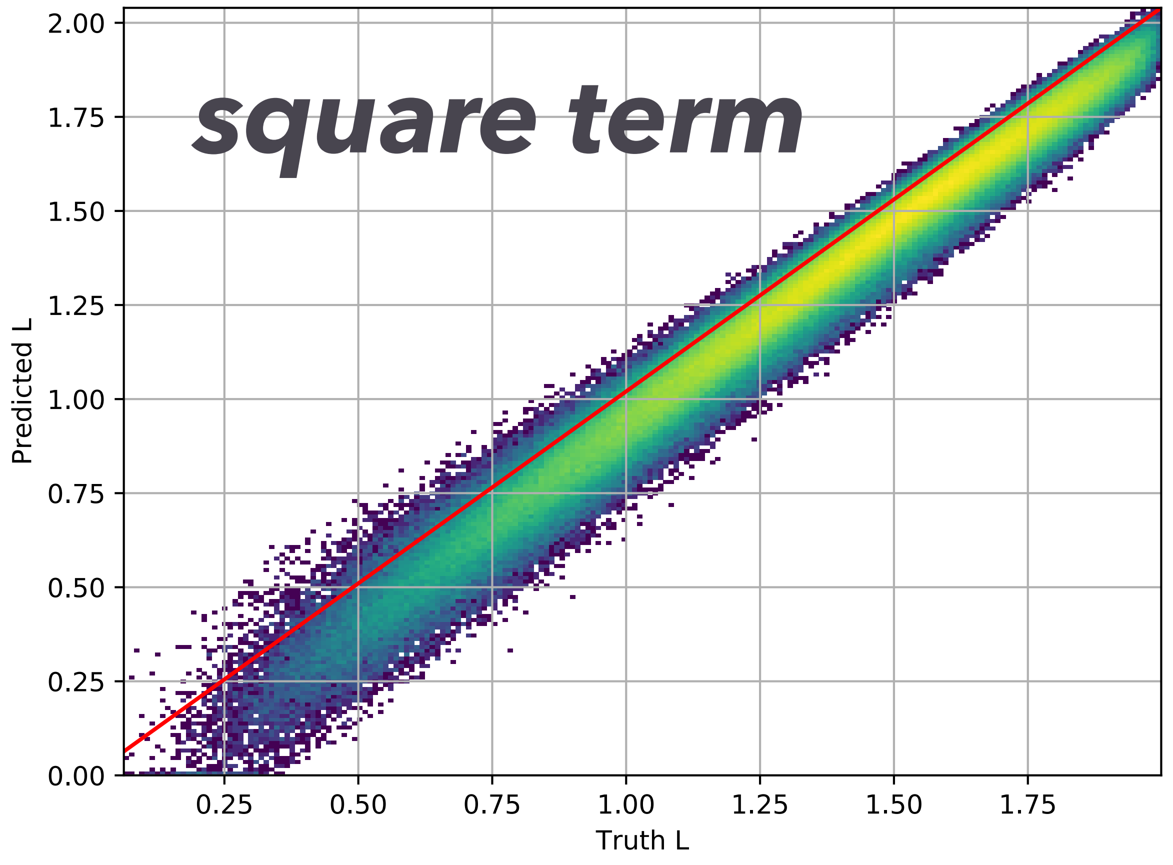
Incorporate this requirement as a additional punishment (by weight *p*) to the loss function:

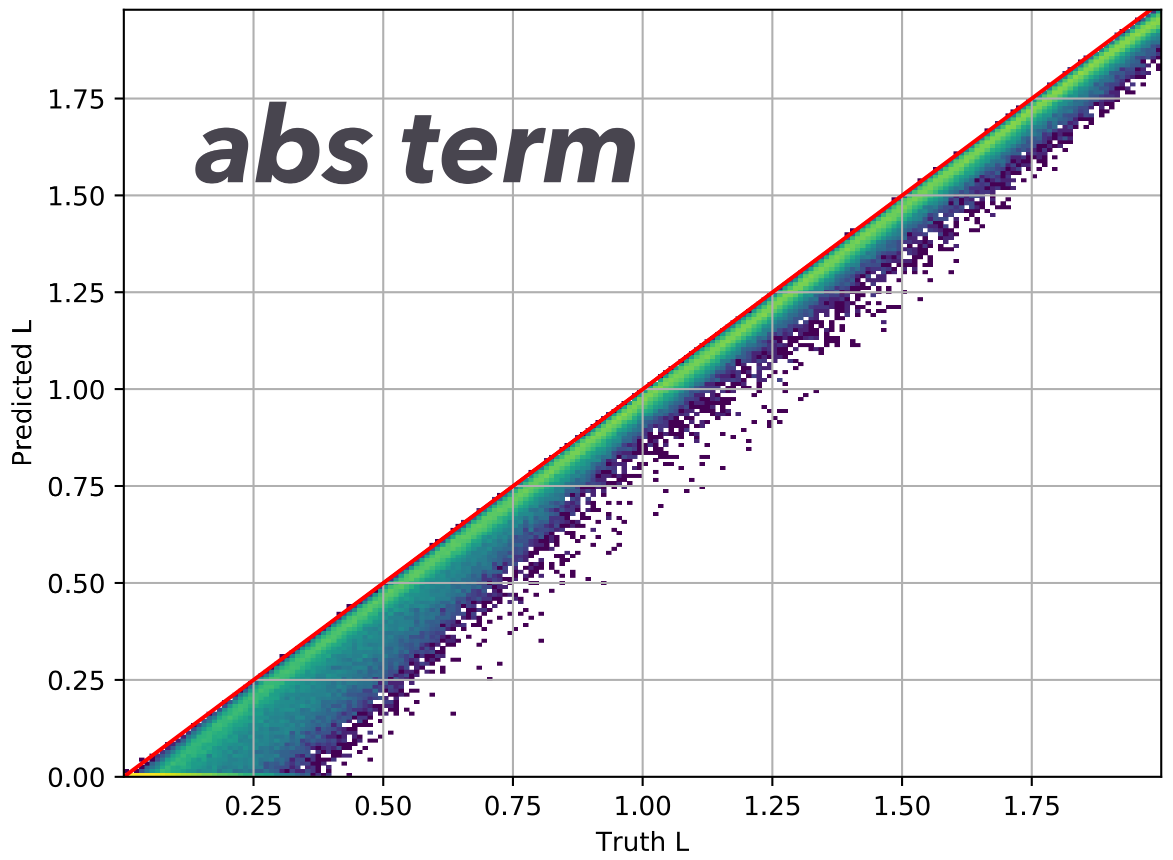
$$\text{biased-MSE} = \frac{\overbrace{\boxed{\frac{\sum_{n_i}(Y_{true} - Y_{pred})^2}{n_i}}}^{\text{when } Y_{pred} < Y_{true}} + p \times \overbrace{\boxed{\frac{\sum_{n_j}(Y_{true} - Y_{pred})^2}{n_j}}}^{\text{when } Y_{pred} > Y_{true}}}{1 + p}$$

p is an additional hyperparameter to tune

Note: *alternative loss function using* ***abs*** *on the second term, although non-convex nature makes is* ***unstable/diffucult to train****.*

Spherical geometry examples





Hyperparameter Optimization

Using the DeepHyper package developed at ANL:

Neural architecture and hyperparameter search at HPC scale

Objective Definition

$$MAE = avg(|Y_{truth} - Y_{pred}|)$$

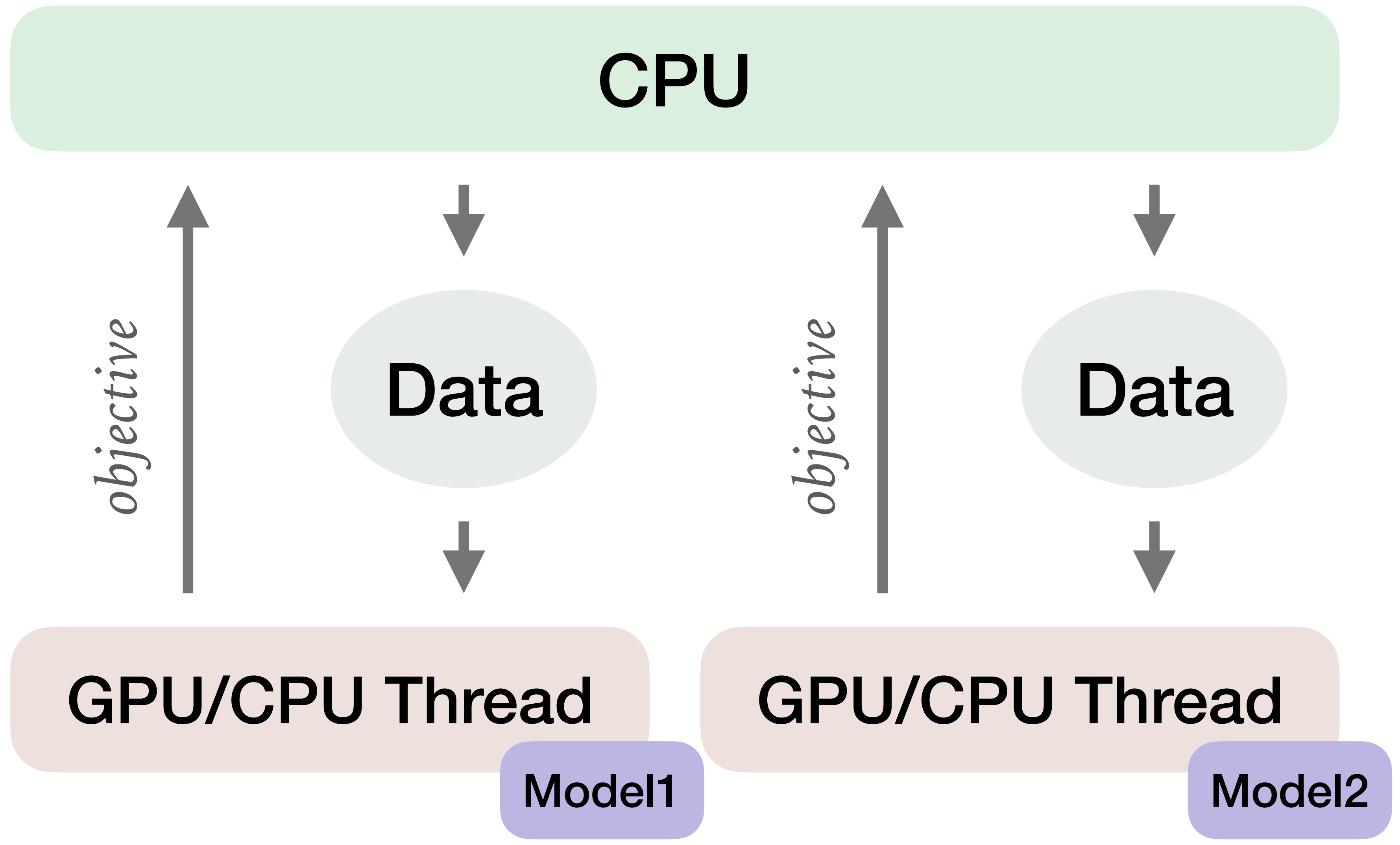
$$OPM = max(|Y^{OP}_{Pred} - Y_{Truth}|)$$

$$objective = \alpha \times MAE + (1 - \alpha) \times OPM, \text{ where } \alpha = 0.7$$

Note: *Another term to promote "fast" models was also explored.*

Parallelization Strategy

single machine

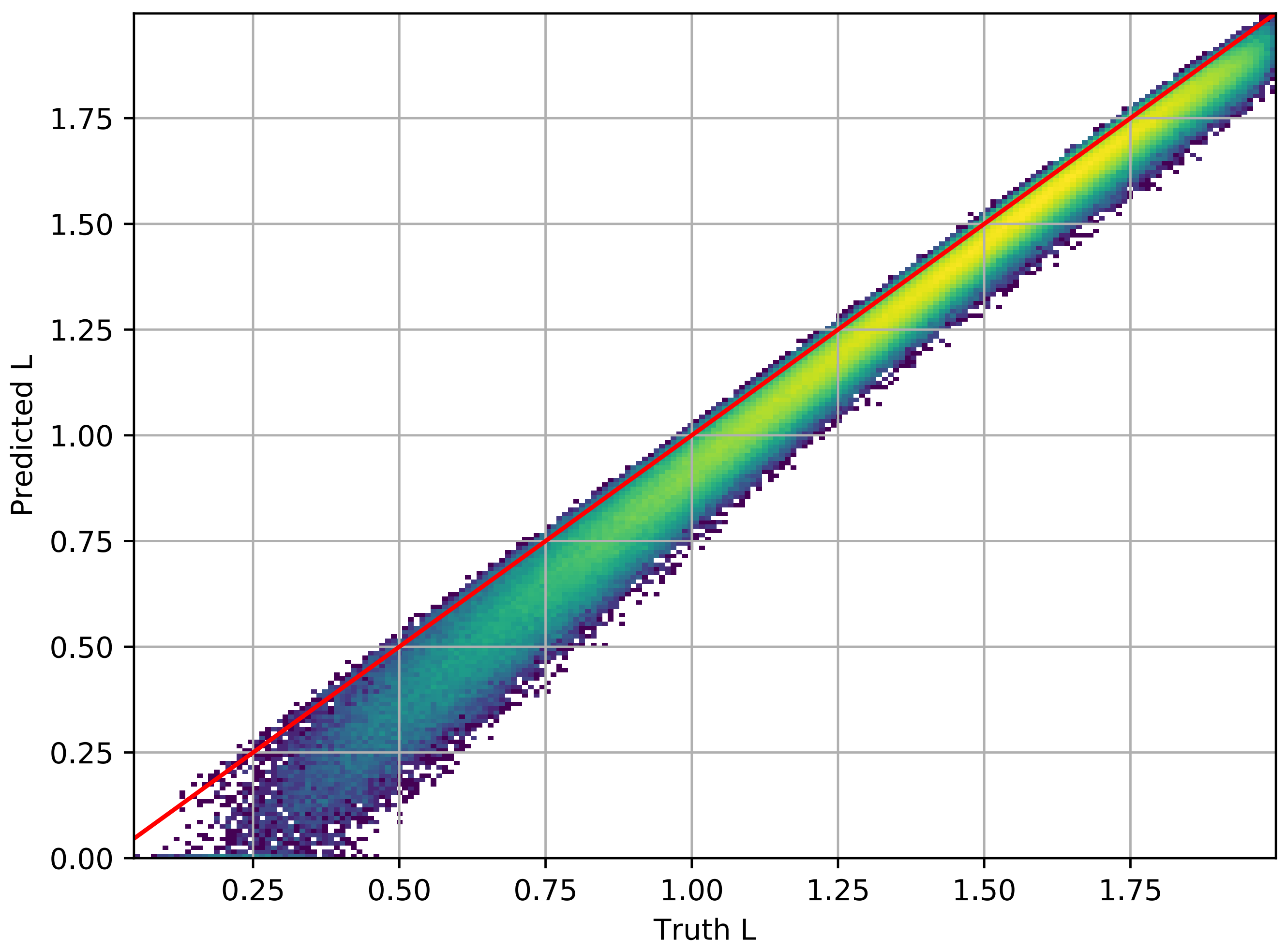


Powered by: RAY

Preliminary Results

Training and inference on a **unit cube** geometry

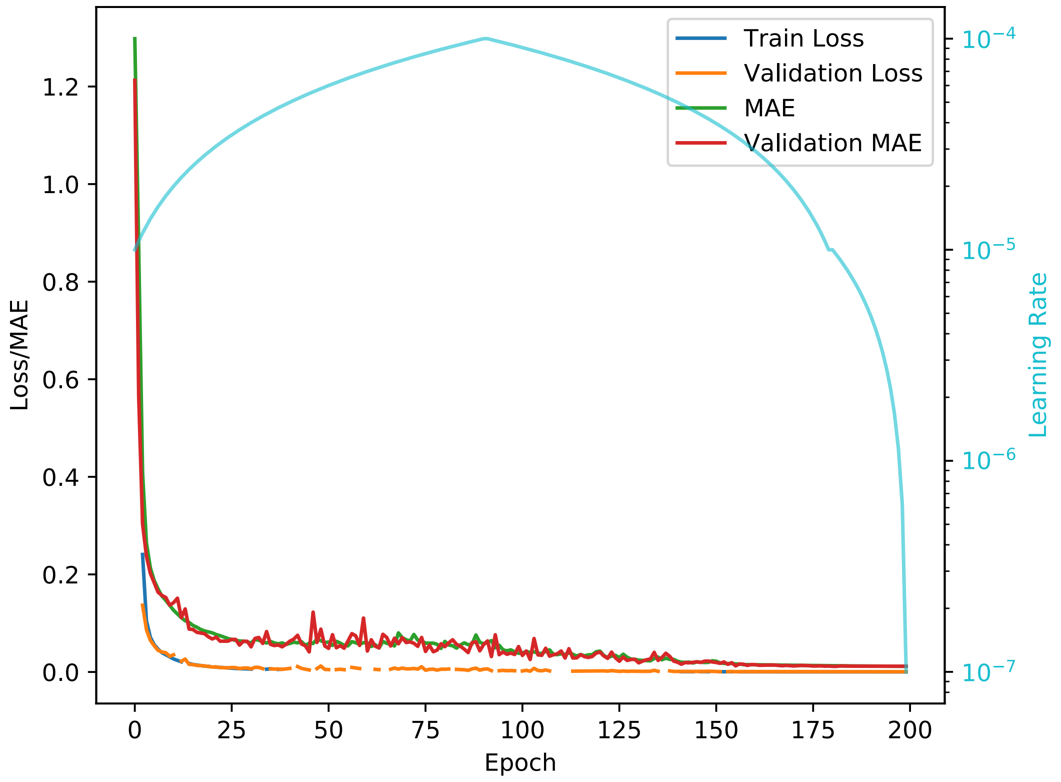
Optimization result

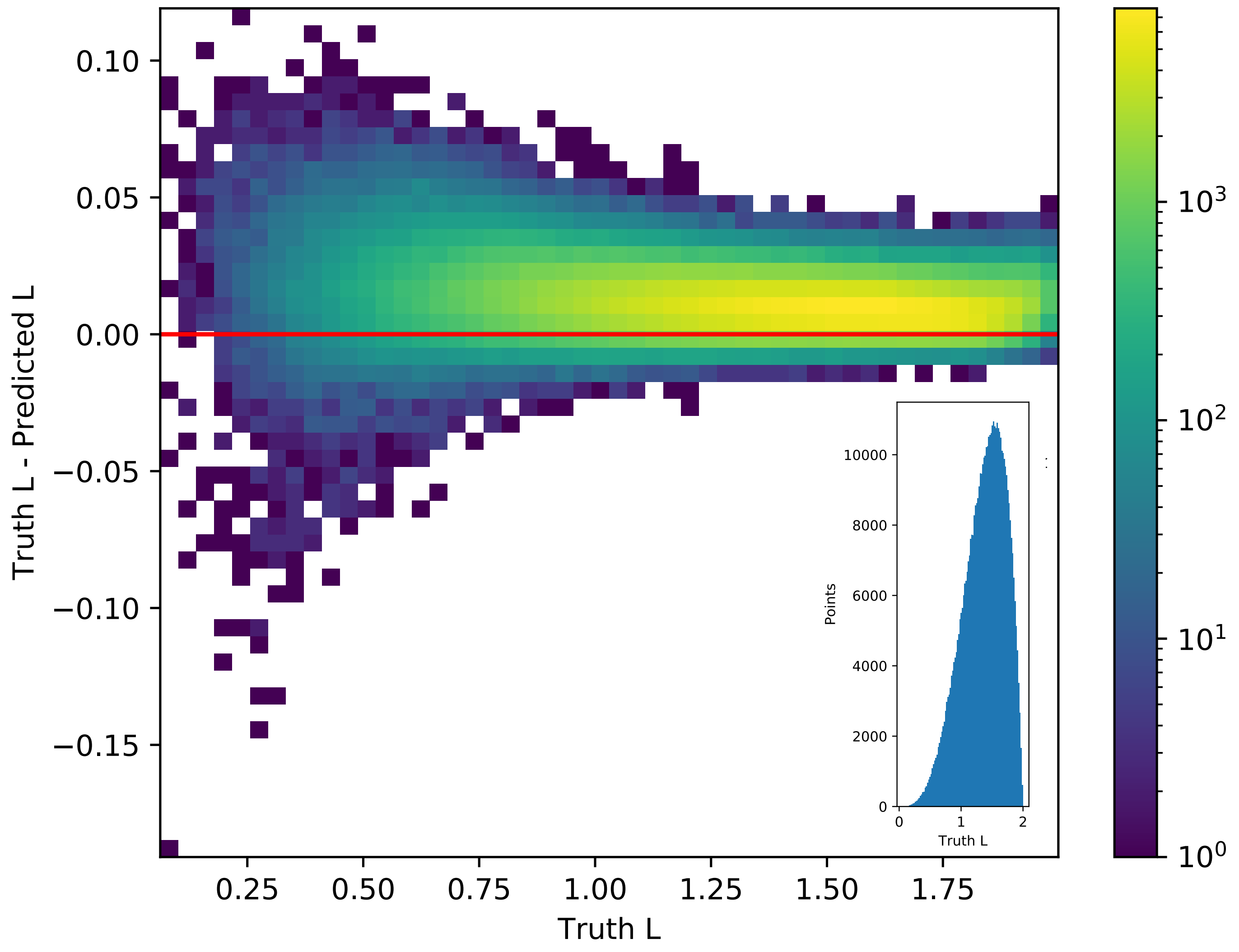


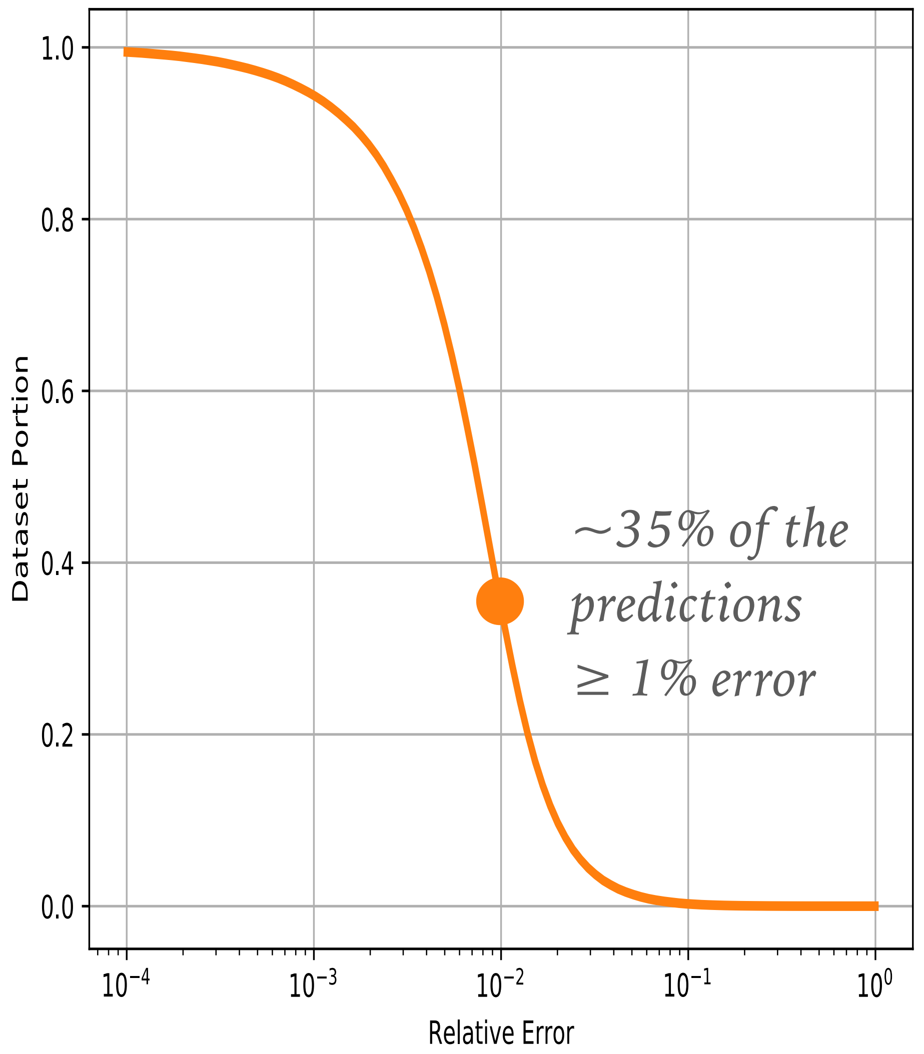
"Layers": 4,
"Nodes": 400,
"Activation": "relu",
"OutputActivation": "relu",
"negPunish": 8.0,
"Optimizer": "Adam",
"LearningRate": 1e-05,
"Batch": 3000,
"Epochs": 200

1-cycle LR scheduling improves performance

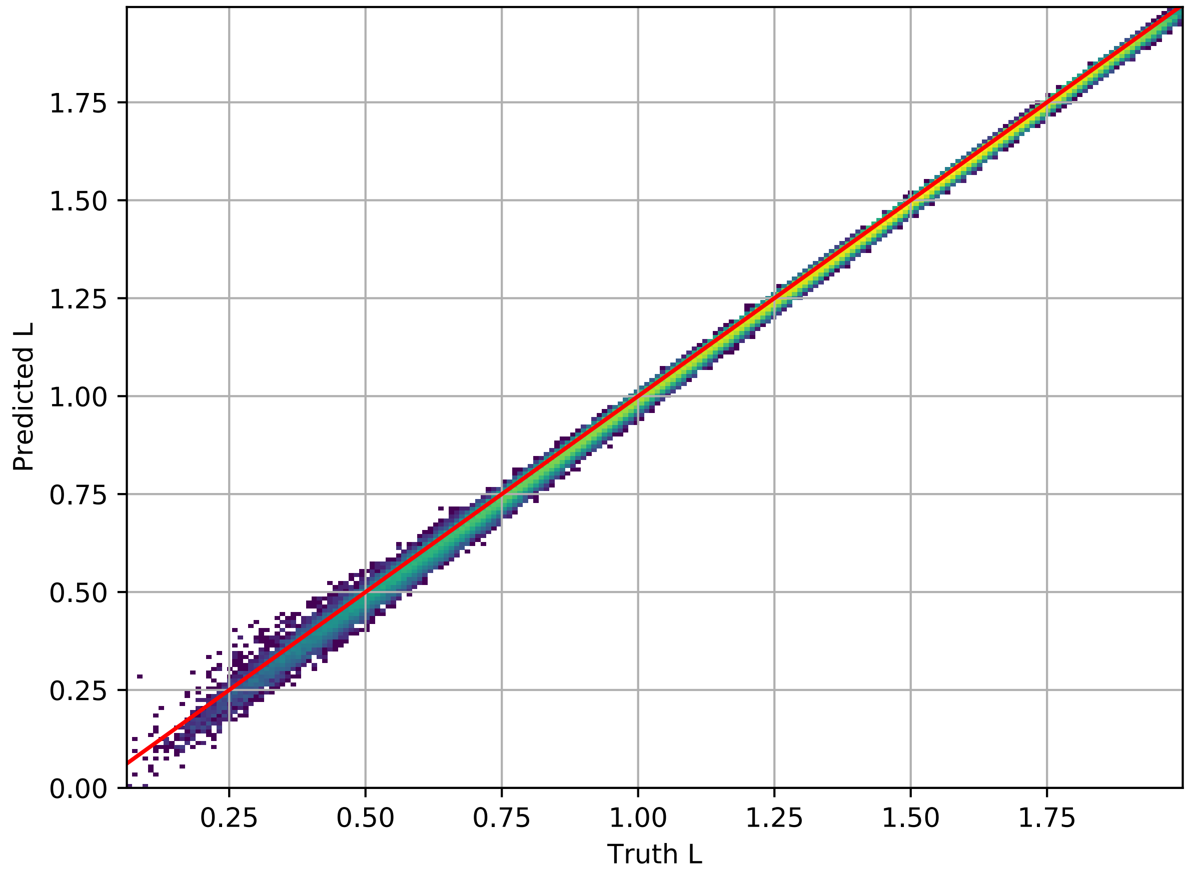
[1803.09820]







Improved model



Evaluation Time Measurements

How long Geant4 takes to explore the geometry vs ML inference?*

	Geometry	Time [μs]
Geant4	ATLAS EMEC	~ 5
	(nested) Twisted Trapezoids	50 - 100
Dense NN evaluation	Cube*	~ 1000

Remarks!

- No batch evaluation
- Evaluation on CPU | *e.g. 100 parallel evaluations on GPU ≃ 1500 μs*
- Naive model structure | *"compressed" model could accelerate inference*
- Tests in Python | *e.g. ONNX Runtime to deploy within C++ software*

* Locate position inside geometry tree and calculate distance to next boundary in order to limit step.

* Although, no strong dependence found on among simplified geometries.

Outlook

So far we trying to **speedup the *'ComputeStep'*** workload using ML surrogate modeling.

Although the **bottleneck seems to come from the number of evaluations.**

Callees	CPU Time: Total
G4SteppingManager::Stepping	100.0%
G4SteppingManager::DefinePhysicalStepLength	66.7%
G4VProcess::AlongStepGPIL	58.2%
G4Transportation::AlongStepGetPhysicalInteractionLength	51.7%
G4Navigator::ComputeStep	34.1%
G4NormalNavigation::ComputeStep	20.9%
G4VoxelNavigation::ComputeStep	10.8%

Class::Method	**Calls**
G4Transportation::AlongStepGetPhysicalInteractionLength	1,953,450
↪ G4Navigator::ComputeStep	1,853,815
↪ G4NormalNavigation::ComputeStep	1,393,472
↪ G4VoxelNavigation::ComputeStep	460,341

100 x 1 GeV photons, ATLAS End-Cap (2.10 < |η| < 2.15)

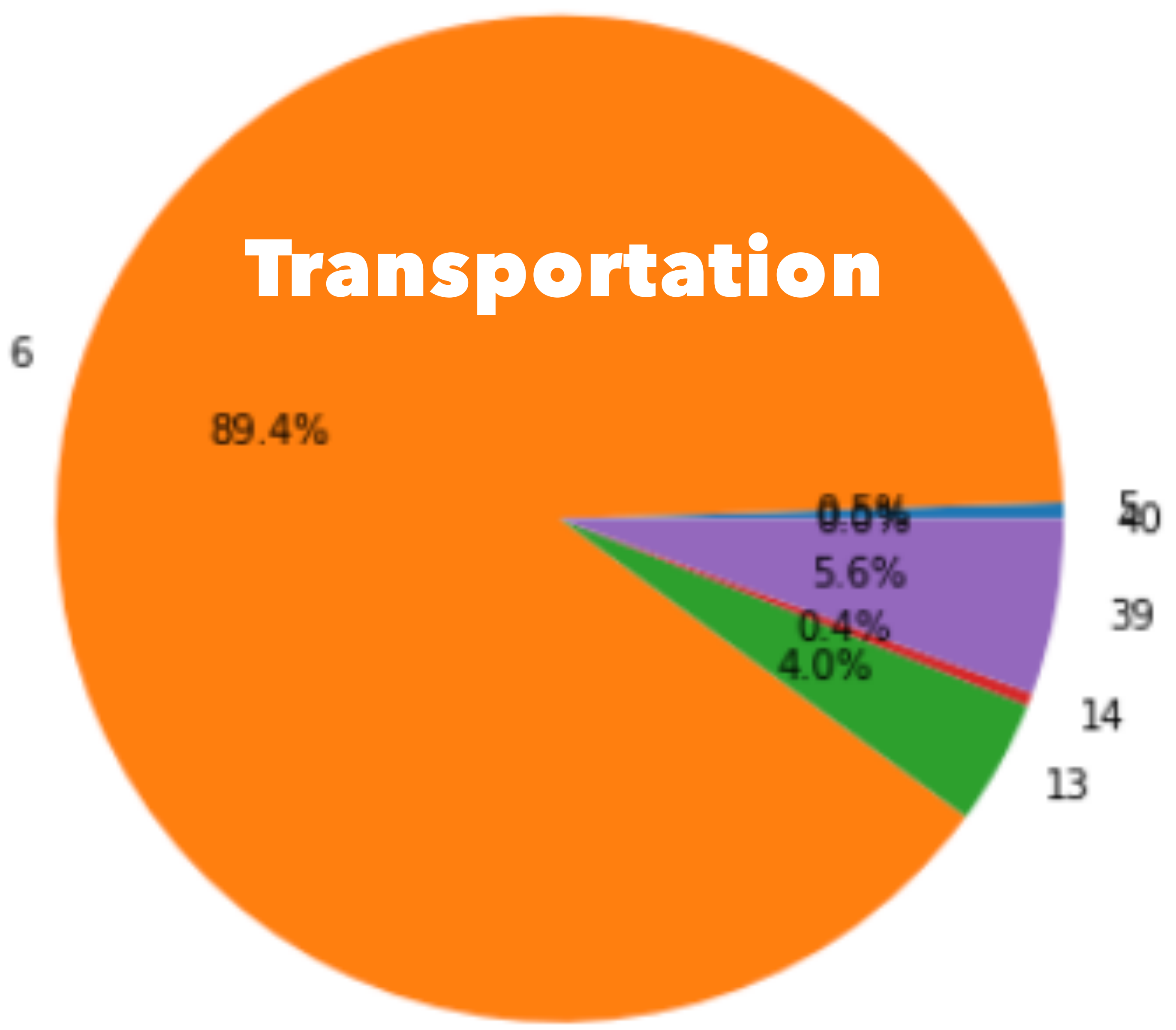
Could we reduce the number of call? What are the photons actually doing and need to explore the geometry so intensively?*

** Reminder: photons dominate the simulation load*

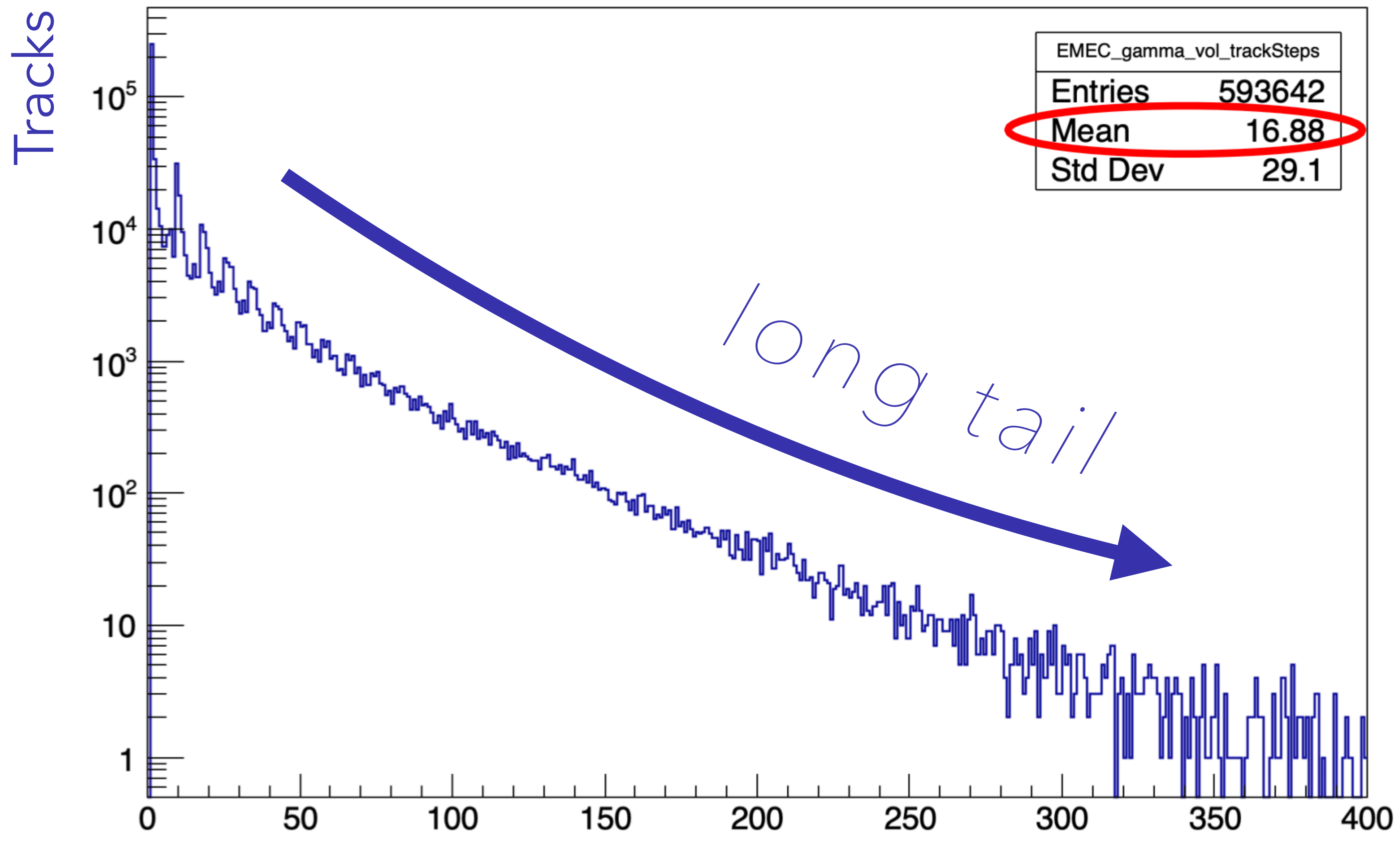
Outlook

What are **photons** doing in each simulation step?

ATLAS End-Cap



- Being neutral, they do not steadily lose energy via Coulomb interactions with atoms.
- Far more penetrating than charged particles of similar energy.
- No physics during transportation.



Track's total simulation steps

Conclusions

ML surrogate modeling within Geant4 to accelerate geometry exploration.

G4 evaluation **O(μs)** vs (prelim.) ML evaluation **O(ms)**.

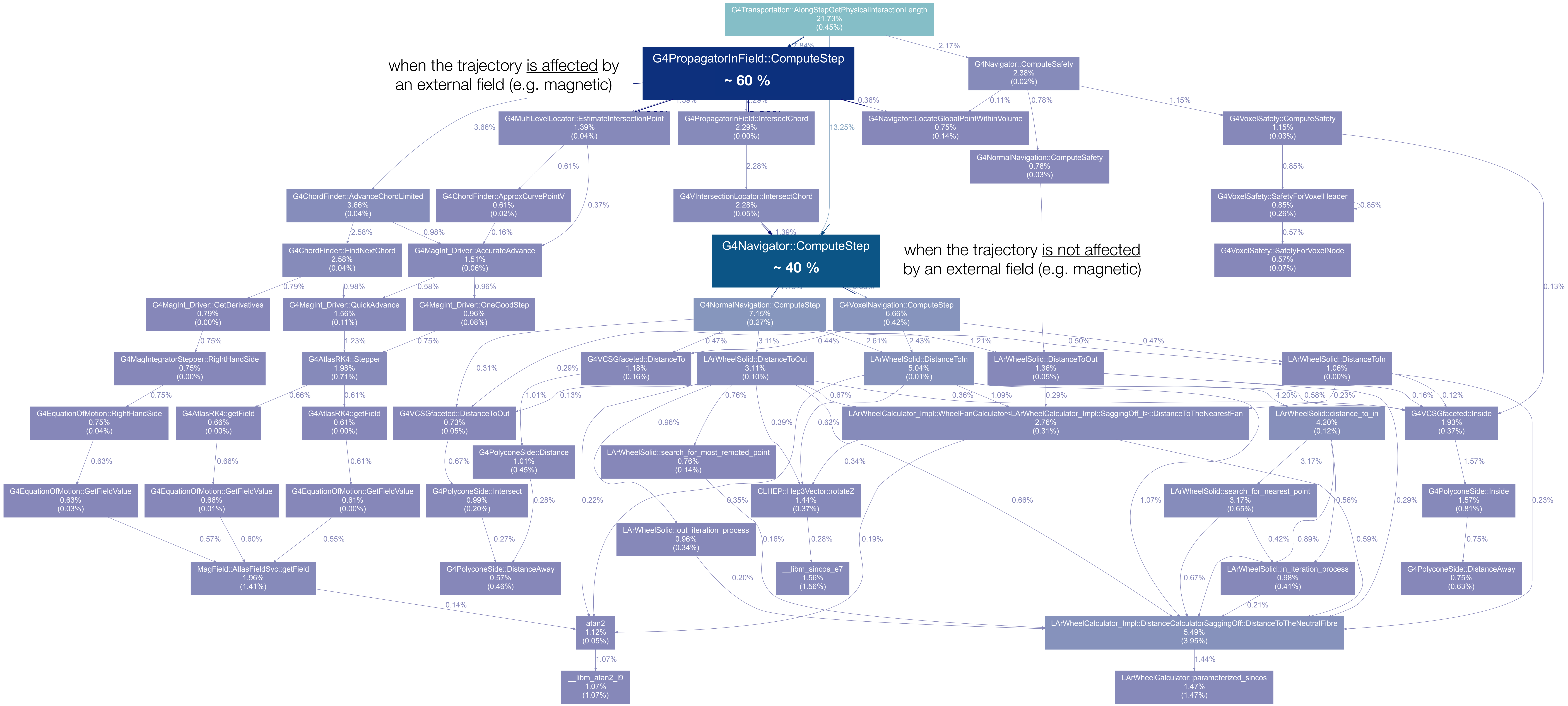
Serial nature of G4 makes it difficult to exploit batch ML inference.

R&D to **reduce the number of times** inference is needed.

Backup

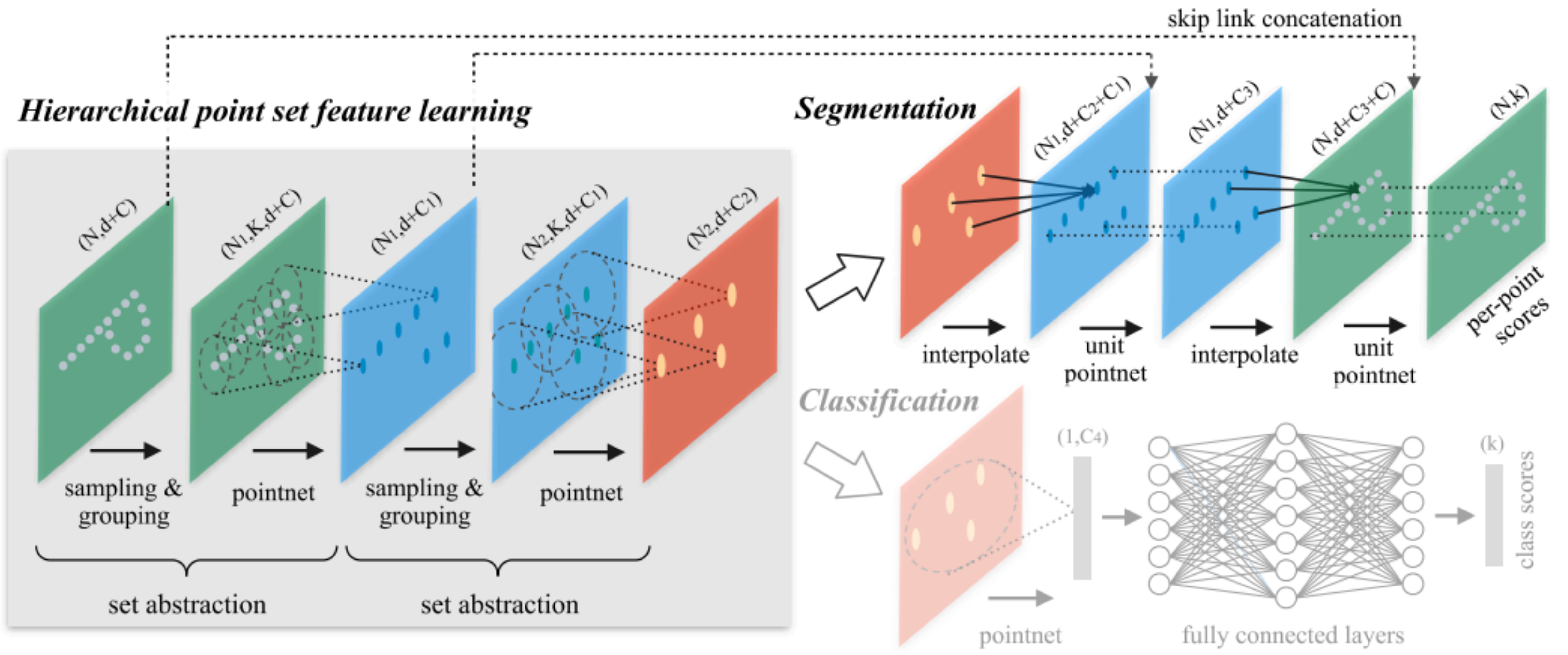
Geant4 Call Tree

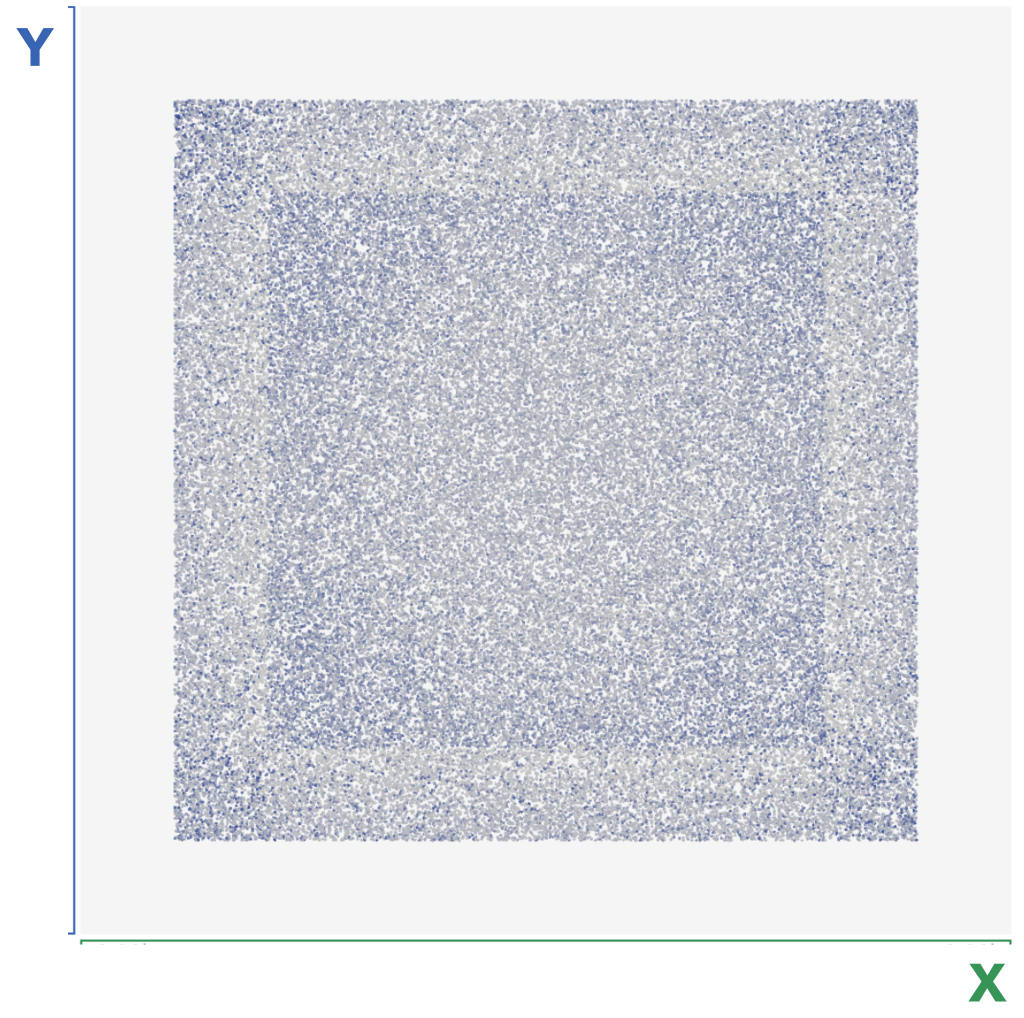
G4Transportation::AlongStepGetPhysicalInteractionLength



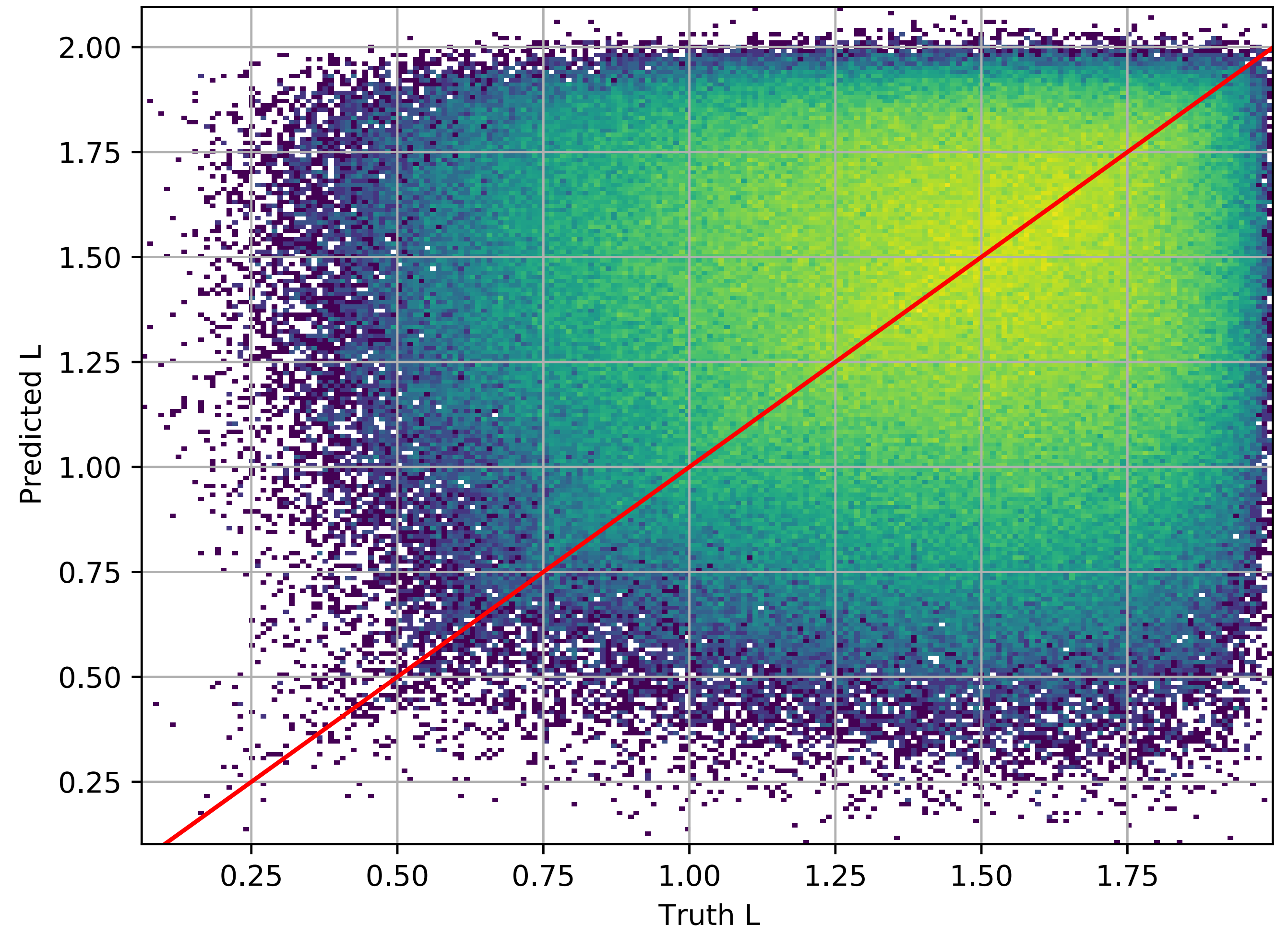
Bonus: PointNet

- Irregular and unordered point-cloud data
- Convolution on 3D points
- Local hierarchical feature learning
- Feature propagation to interpolate point-wise predictions
- Architecture robust in under-sampled regions



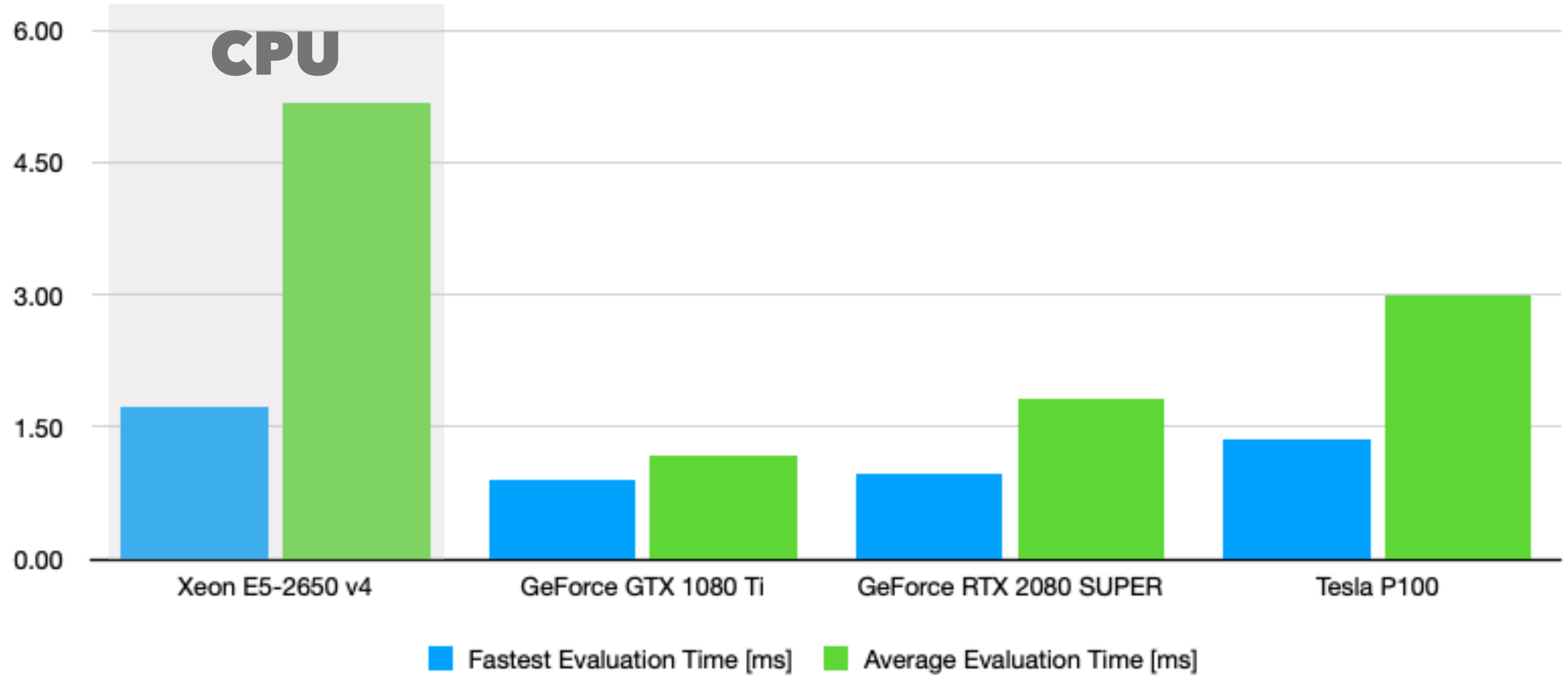


sphere



Evaluation Time

How long does it take to evaluate the neural network?

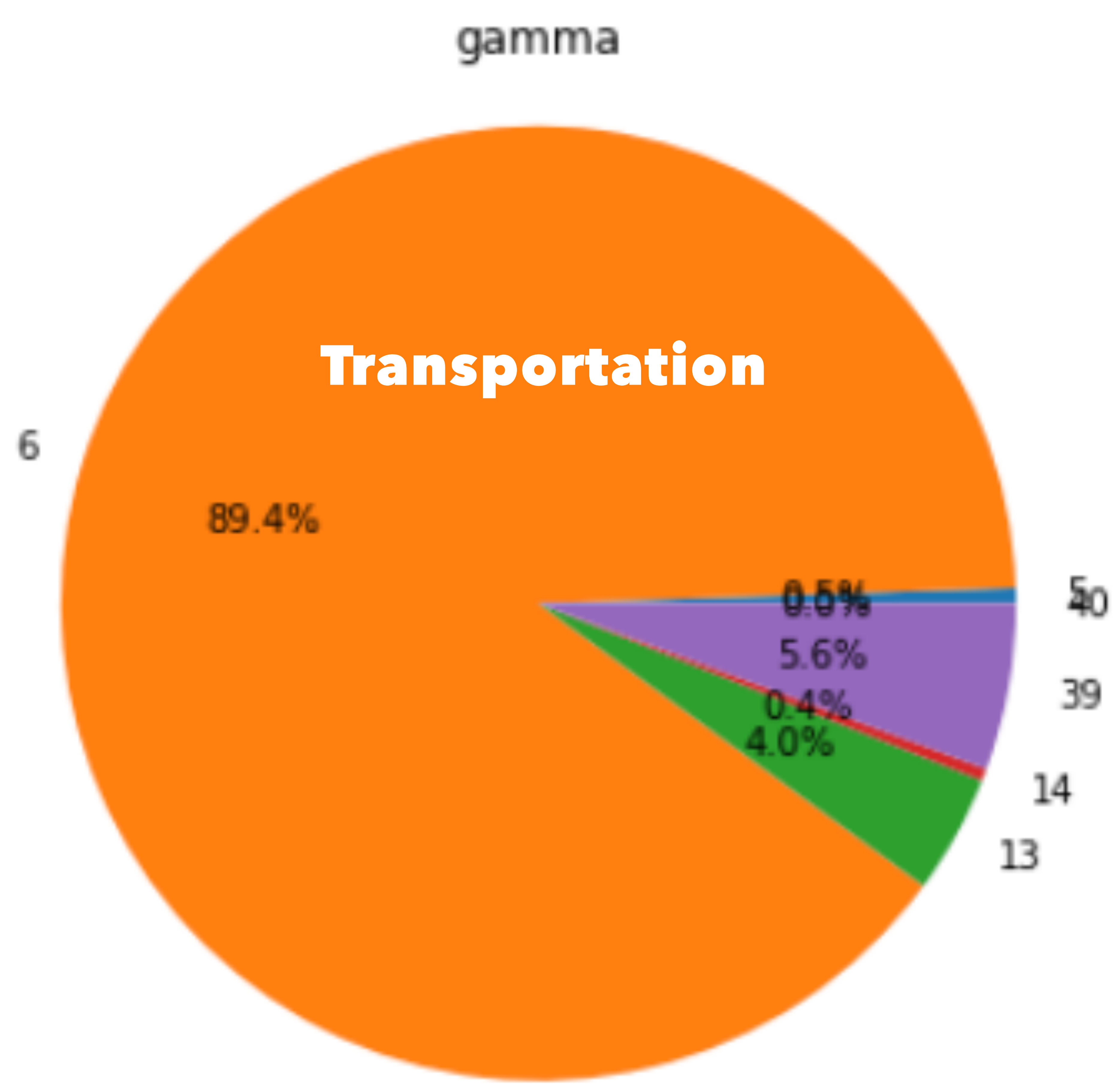


	Fastest Evaluation Time [ms]	Average Evaluation Time [ms]
Xeon E5-2650 v4	1.74	5.17
GeForce GTX 1080 Ti	0.91	1.19
GeForce RTX 2080 SUPER	0.96	1.83
Tesla P100	1.37	3.00

Caveats

- No batch processing (= Parallelism)
- ~ 1M parameter model
- Tests in Python

Photon Processes




```
m_prcNameMap["Unknown"] = 0;
m_prcNameMap["CoulombScat"] = 1;
m_prcNameMap["Decay"] = 2;
m_prcNameMap["G4FastSimulationManagerProcess"] = 3;
m_prcNameMap["He3Inelastic"] = 4;
m_prcNameMap["Rayl"] = 5;
m_prcNameMap["Transportation"] = 6;
m_prcNameMap["alphaInelastic"] = 7;
m_prcNameMap["annihil"] = 8;
m_prcNameMap["anti-lambdaInelastic"] = 9;
m_prcNameMap["anti_neutronInelastic"] = 10;
m_prcNameMap["anti_protonInelastic"] = 11;
m_prcNameMap["anti_sigma-Inelastic"] = 12;
m_prcNameMap["compt"] = 13;
m_prcNameMap["conv"] = 14;
m_prcNameMap["dInelastic"] = 15;
m_prcNameMap["eBrem"] = 16;
m_prcNameMap["eIoni"] = 17;
m_prcNameMap["electronNuclear"] = 18;
m_prcNameMap["hBertiniCaptureAtRest"] = 19;
m_prcNameMap["hBrems"] = 20;
m_prcNameMap["hFritiofCaptureAtRest"] = 21;
m_prcNameMap["hIoni"] = 22;
m_prcNameMap["hPairProd"] = 23;
m_prcNameMap["hadElastic"] = 24;
m_prcNameMap["ionIoni"] = 25;
m_prcNameMap["kaon+Inelastic"] = 26;
m_prcNameMap["kaon-Inelastic"] = 27;
m_prcNameMap["kaon0LInelastic"] = 28;
m_prcNameMap["kaon0SInelastic"] = 29;
m_prcNameMap["lambdaInelastic"] = 30;
m_prcNameMap["msc"] = 31;
m_prcNameMap["muBrems"] = 32;
m_prcNameMap["muIoni"] = 33;
m_prcNameMap["muMinusCaptureAtRest"] = 34;
m_prcNameMap["muPairProd"] = 35;
m_prcNameMap["nCapture"] = 36;
m_prcNameMap["nKiller"] = 37;
m_prcNameMap["neutronInelastic"] = 38;
m_prcNameMap["phot"] = 39;
m_prcNameMap["photonNuclear"] = 40;
m_prcNameMap["pi+Inelastic"] = 41;
m_prcNameMap["pi-Inelastic"] = 42;
m_prcNameMap["positronNuclear"] = 43;
m_prcNameMap["protonInelastic"] = 44;
m_prcNameMap["sigma+Inelastic"] = 45;
m_prcNameMap["sigma-Inelastic"] = 46;
m_prcNameMap["tInelastic"] = 47;
m_prcNameMap["xi0Inelastic"] = 48;
```