Imperial College London

Continual Learning for HEP

Marco Barbone

m.barbone19@imperial.ac.uk

Imperial College London

About me

- M.Sc. in Computer Science and Networking Scuola superiore Sant'Anna & Università di Pisa
- CERN OpenLab Summer student
- Research Software Engineer, Maxeler Technologies
- Software Engineer, Intel Corporation
- Software Engineer, Optiver
- Ph.D. Student, Custom Computing research group, Imperial College London, Supervisor Prof. Wayne Luk
- Member of CERN-Imperial College CMS group
- Associate Ph.D. Student, The Institute of Cancer Research, London
- Member of the CRUK Convergence Science Centre Training Committee

Imperial College London

Continual learning

Continual learning (CL) is a particular machine learning paradigm where the data distribution and learning objective change through time, or where all the training data and objective criteria are never available at once.

The evolution of the learning process is modeled by a sequence of learning experiences where the goal is to be able to learn new skills all along the sequence without forgetting what has been previously learned.

https://arxiv.org/pdf/1907.00182.pdf

Imperial College London

Changing Environments

There are many situations in which a Machine Learning (ML) system is deployed in an evolving environment

- This can be with changing conditions which leads to a target dataset varying significantly from the training dataset

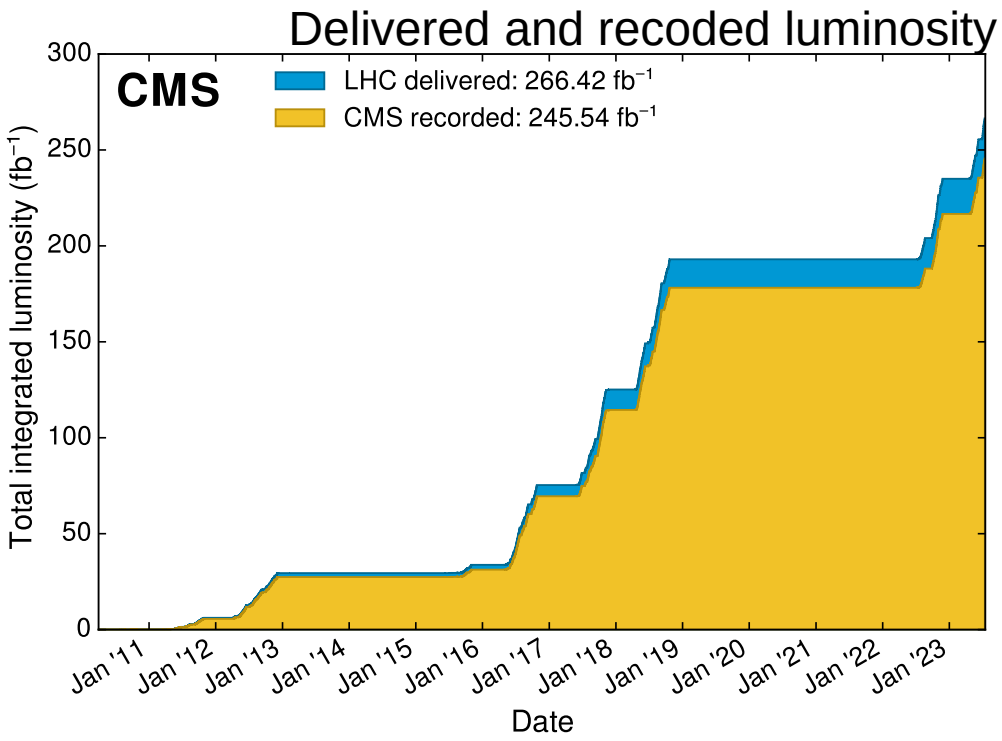
In particle physics experiments we tend to train ML systems on large Monte Carlo simulation datasets and then deploy the systems on real data which can have variable conditions

- The detector itself might not respond in the desired manner, alignment and calibrations may need to be performed on the data

The Phase-2 Level 1 Trigger (L1T) uses ML to make decisions: small, low-latency models, might not be robust to the changing detector environment

Imperial College London

CMS detector degradation



Radiation damages the sensors

https://twiki.cern.ch/twiki/bin/view/CMSPublic/LumiPublicResults

Signal-to-noise ratio trend

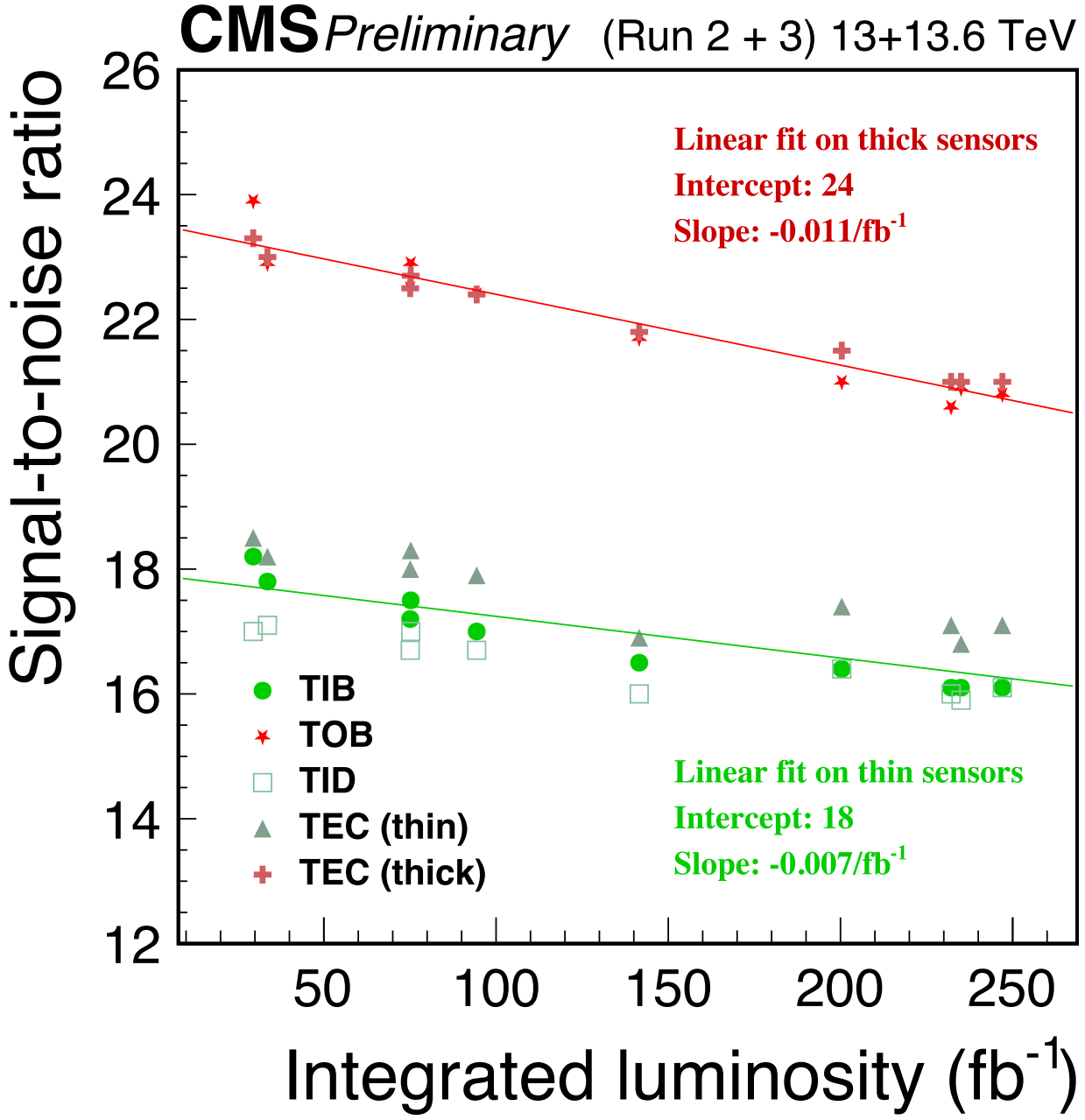


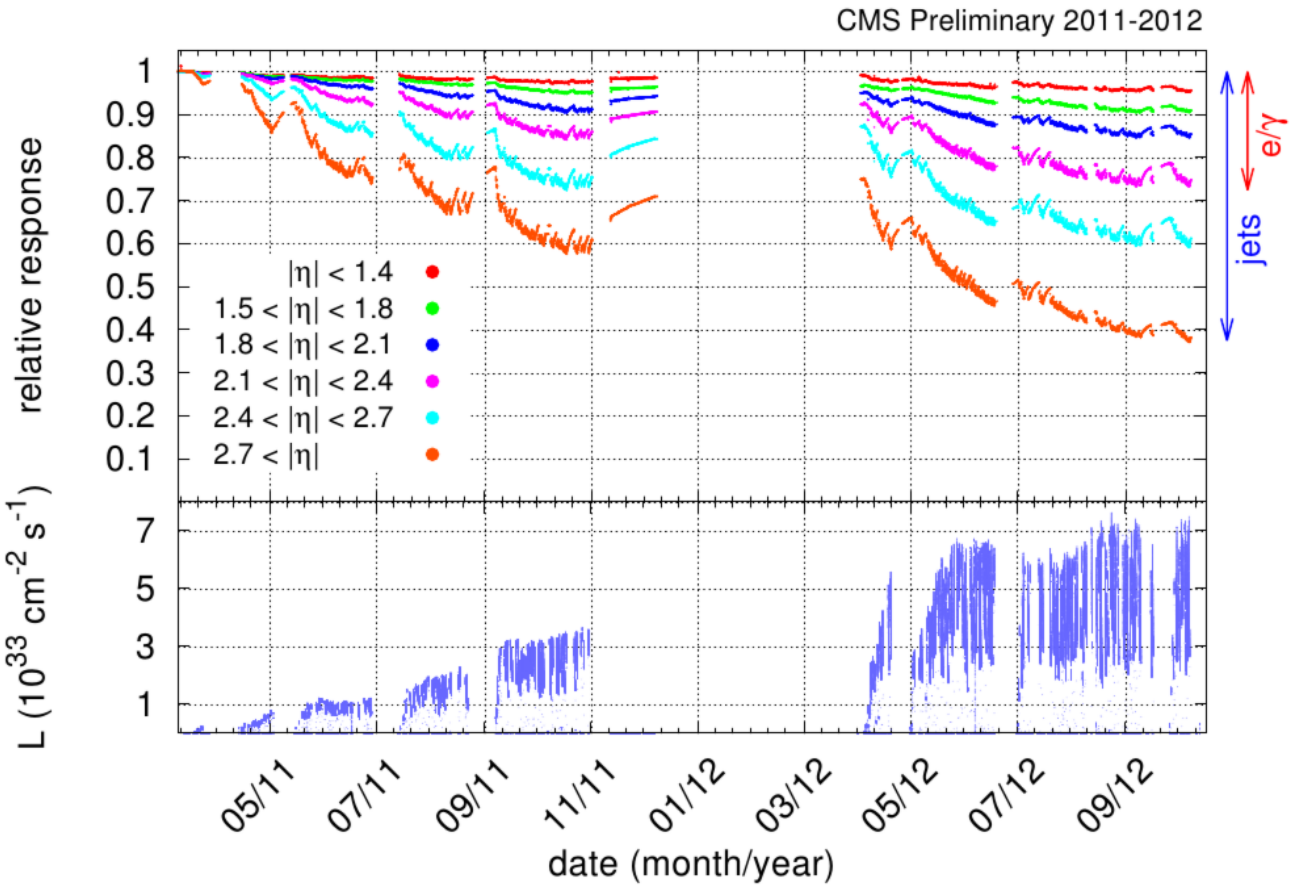
Figure-3: The signal-to-noise ratio (S/N) measured in pp collision since Run 2 is reported as a function of the integrated delivered luminosity. Measurements have been performed split per sensor thickness in the Tracker End-caps (320 μm or 500 μm). The S/N scales with integrated luminosity. The two points that lie on top of each around 75 fb^{-1} come from different years: the lower S/N is at the end of 2022 and the higher one is at the beginning of 2023.
The data were fitted with a linear function.

Source: https://cds.cern.ch/record/2865841/files/DP2023_040.pdf

Imperial College London

CMS ECAL relative response loss of response with time

- CMS ECAL relative response to laser light (440 nm) versus time, averaged over all crystals in one given pseudorapidity bin, for the 2011 and 2012 data taking.



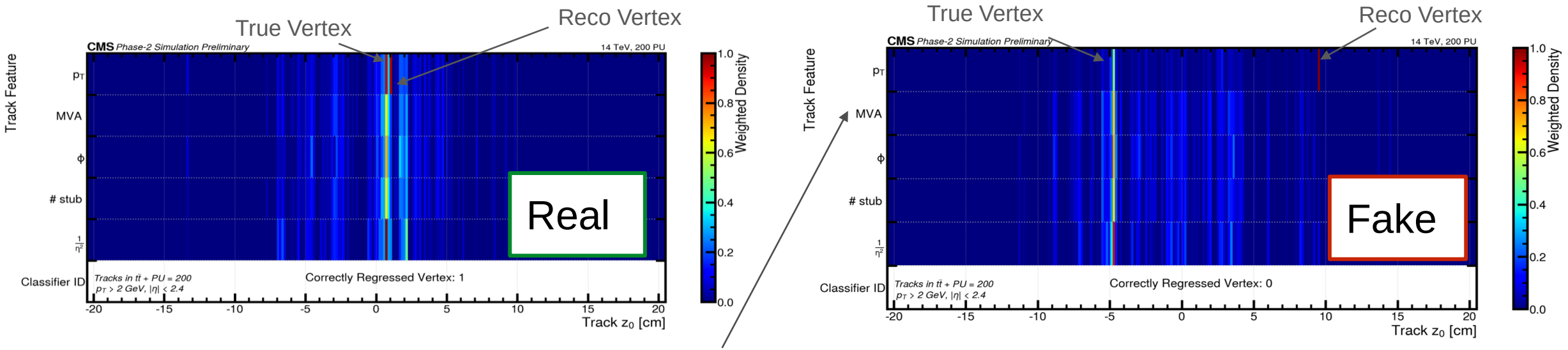
https://cds.cern.ch/record/1494093/plots

Imperial College London

Continual Learning in the CMS Phase-2 Trigger

CMS Phase-2 trigger uses tracker primitives reconstructed using the outer tracker

Vertex finding is performed by binning tracks in the z-direction upon which a sliding window passes over to find the highest track p_T position



Use histogrammed track features (MVA is from a track quality BDT) [CMS-CR-2022-236]

Imperial College London

Continual Learning in the CMS Phase-2 Trigger

Use datasets with degraded outer tracker modules

Use a CNN model that classifies real and fake vertices

1. Does the ML model performance drop with new samples?
2. Does retraining the model with the new samples improve performance?
3. Does using CL to retrain the model improve performance further?

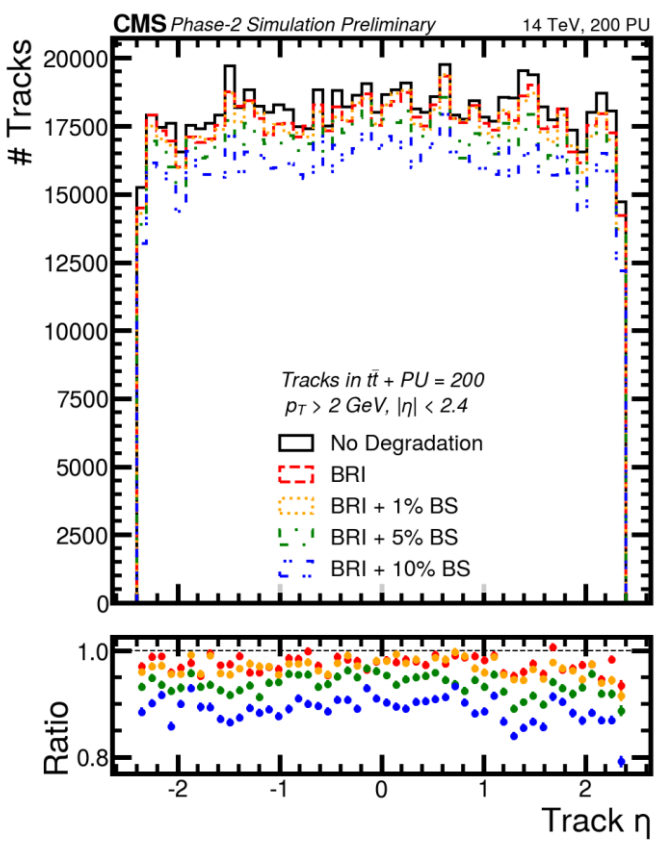
Imperial College London

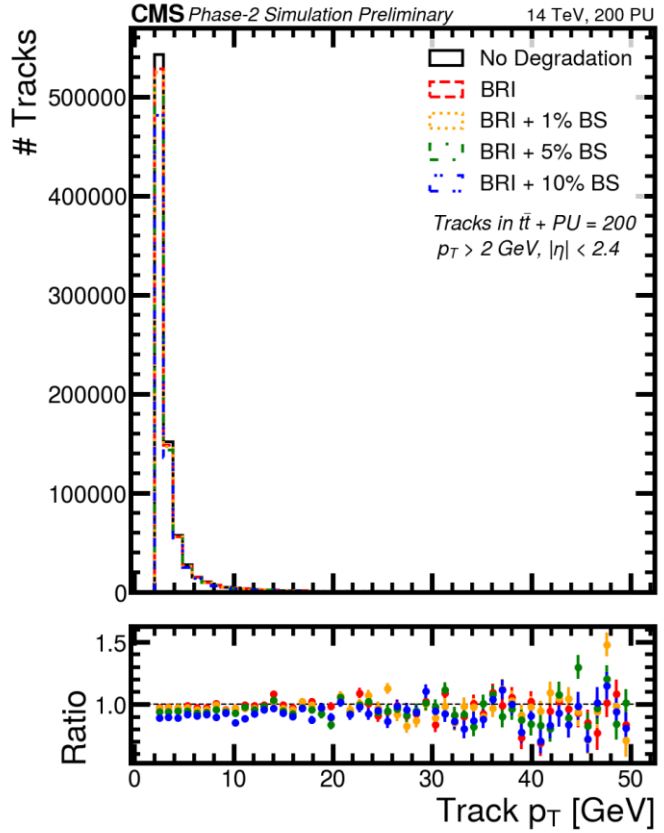
Characterising Detector Degradation

CMS Outer tracker has PS modules made up of PS-p (pixel) and PS-s (strip) submodules and 2S modules (strips only)

Have specific datasets that emulate the CMS outer tracker degrading [1] all generated from a top quark pair production sample with an additional 200 pileup:

- No Degradation: no inefficiency (reference)
- BRI: PS-p bias rails inefficiency only
- BRI + 1% BS: PS-p bias rails inefficiency + 1% bad strips in PS-s and 2S sensors
- BRI + 5% BS: PS-p bias rails inefficiency + 5% bad strips in PS-s and 2S sensors
- BRI + 10% BS: PS-p bias rails inefficiency + 10% bad strips in PS-s and 2S sensors





[1] CMS-TDR-014

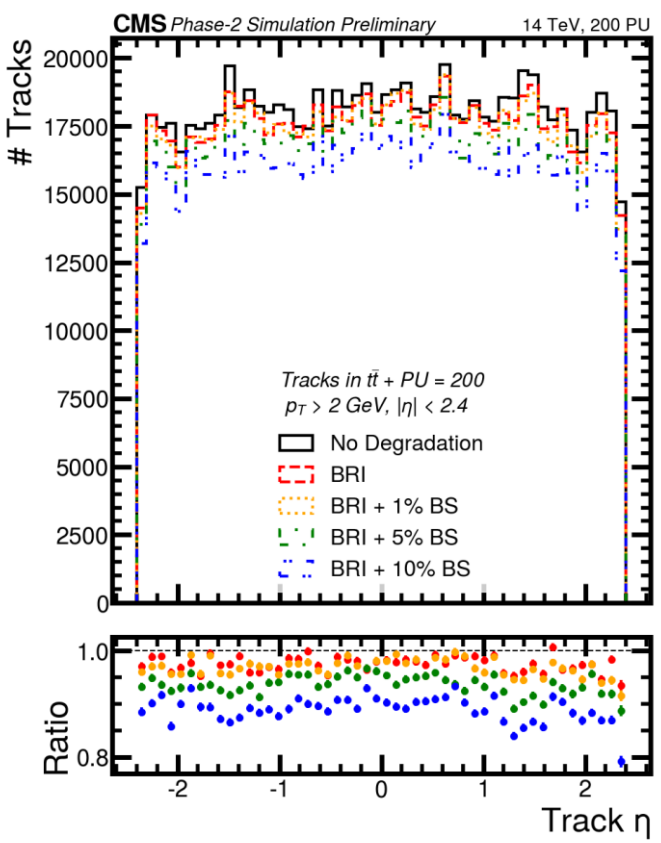
Imperial College London

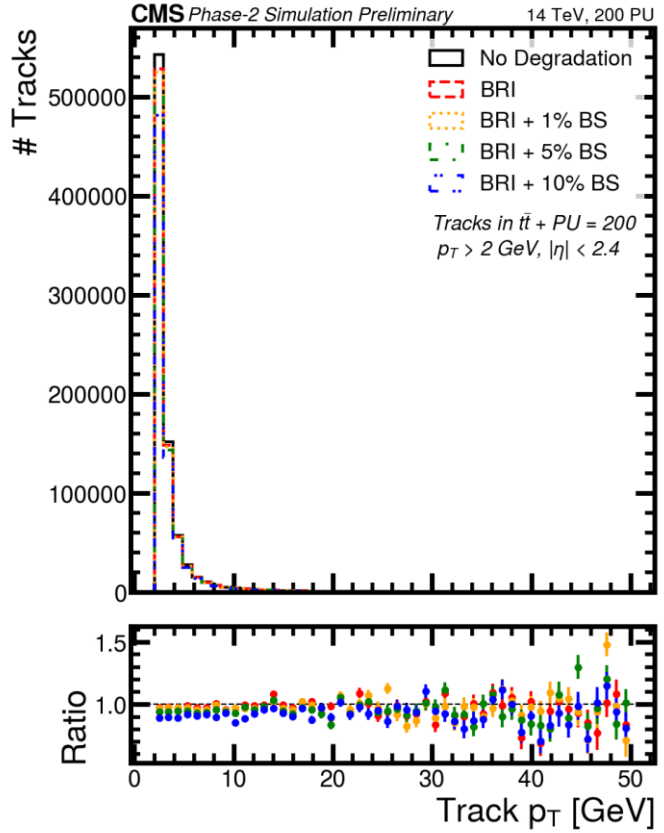
Characterising Detector Degradation

CMS Outer tracker has PS modules made up of PS-p (pixel) and PS-s (strip) submodules and 2S modules

Fewer tracks as the detector degrades especially at low p_T, quality of the tracks broadly unchanged

- No Degradation: no inefficiency (reference)
- BRI: PS-p bias rails inefficiency only
- BRI + 1% BS: PS-p bias rails inefficiency + 1% bad strips in PS-s and 2S sensors
- BRI + 5% BS: PS-p bias rails inefficiency + 5% bad strips in PS-s and 2S sensors
- BRI + 10% BS: PS-p bias rails inefficiency + 10% bad strips in PS-s and 2S sensors





[1] CMS-TDR-014

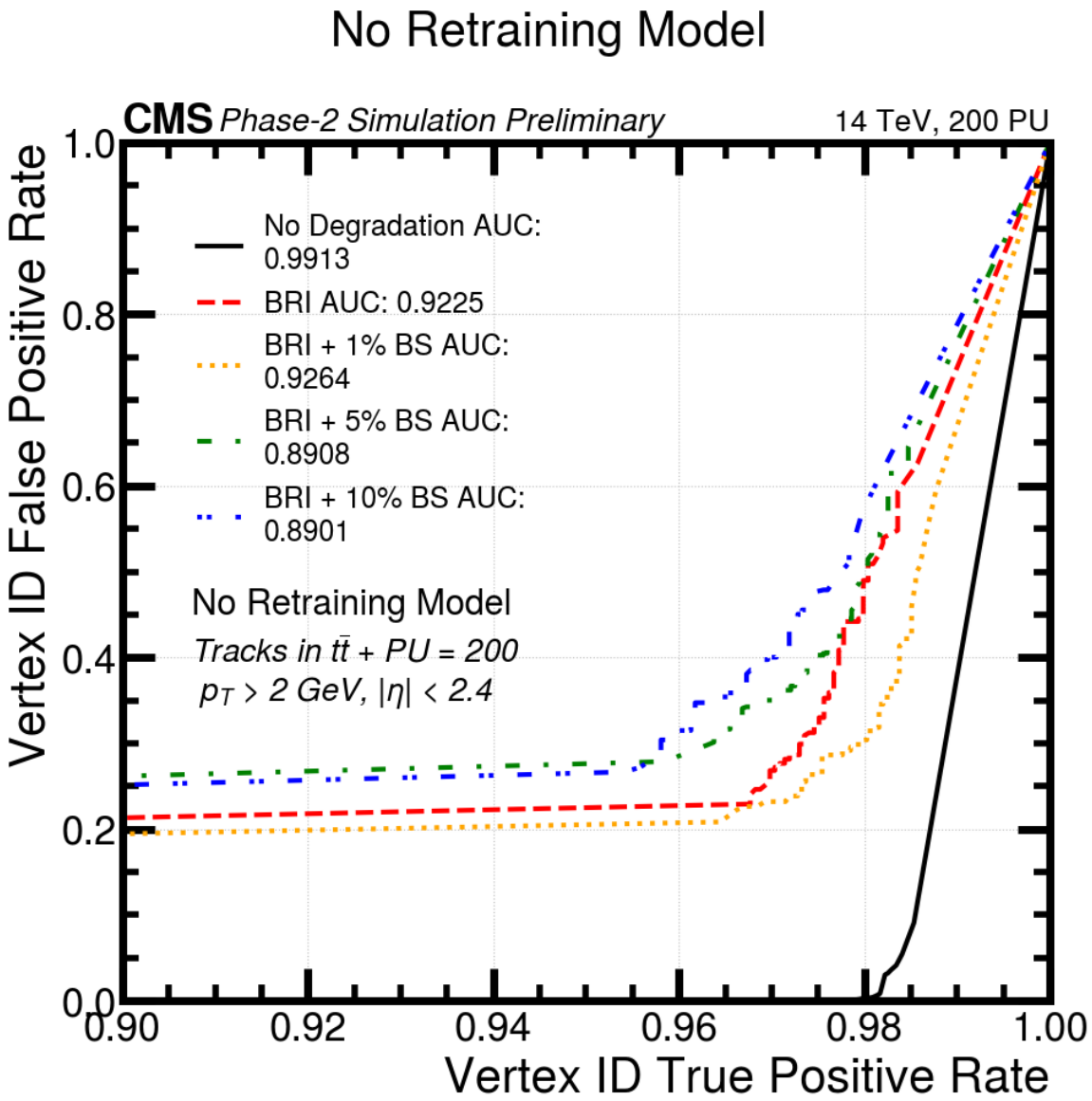
Imperial College London

No Retraining Model

No Retraining model is as if we left our ML in the trigger without changing anything

Performs well on what it was trained on

Performance drops of as detector degrades, not robust to the changing environment



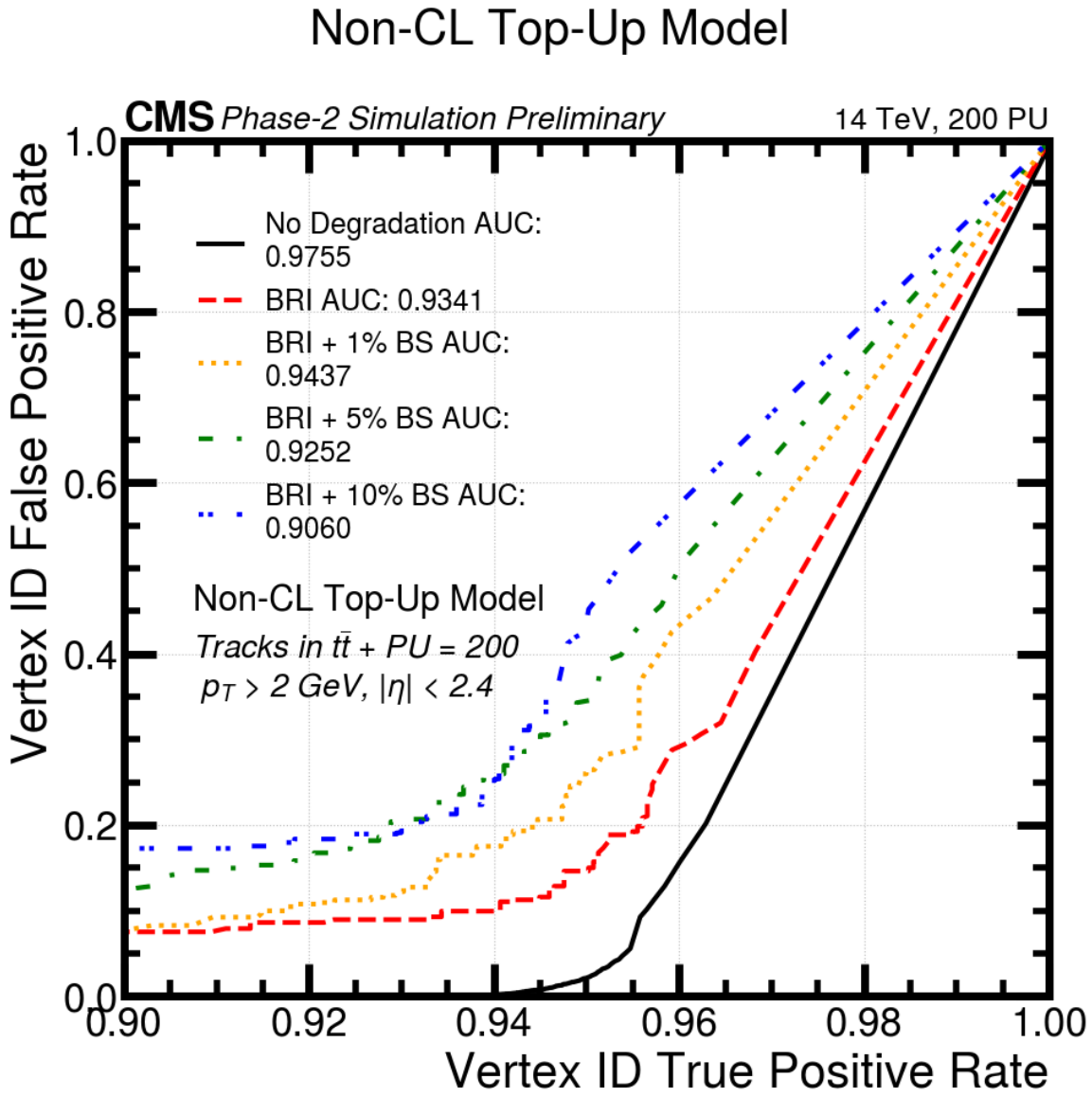
Imperial College London

Top-Up Train Model No CL

Start with no retraining model, top-up training with new datasets, 7000 training events, no fancy algorithm just same model new data, lower learning rate

Performance across the degraded samples is better, generally more robust to the degraded detector

Performance in the no degradation dataset reduced as our retraining has disrupted the original model. Causing forgetting



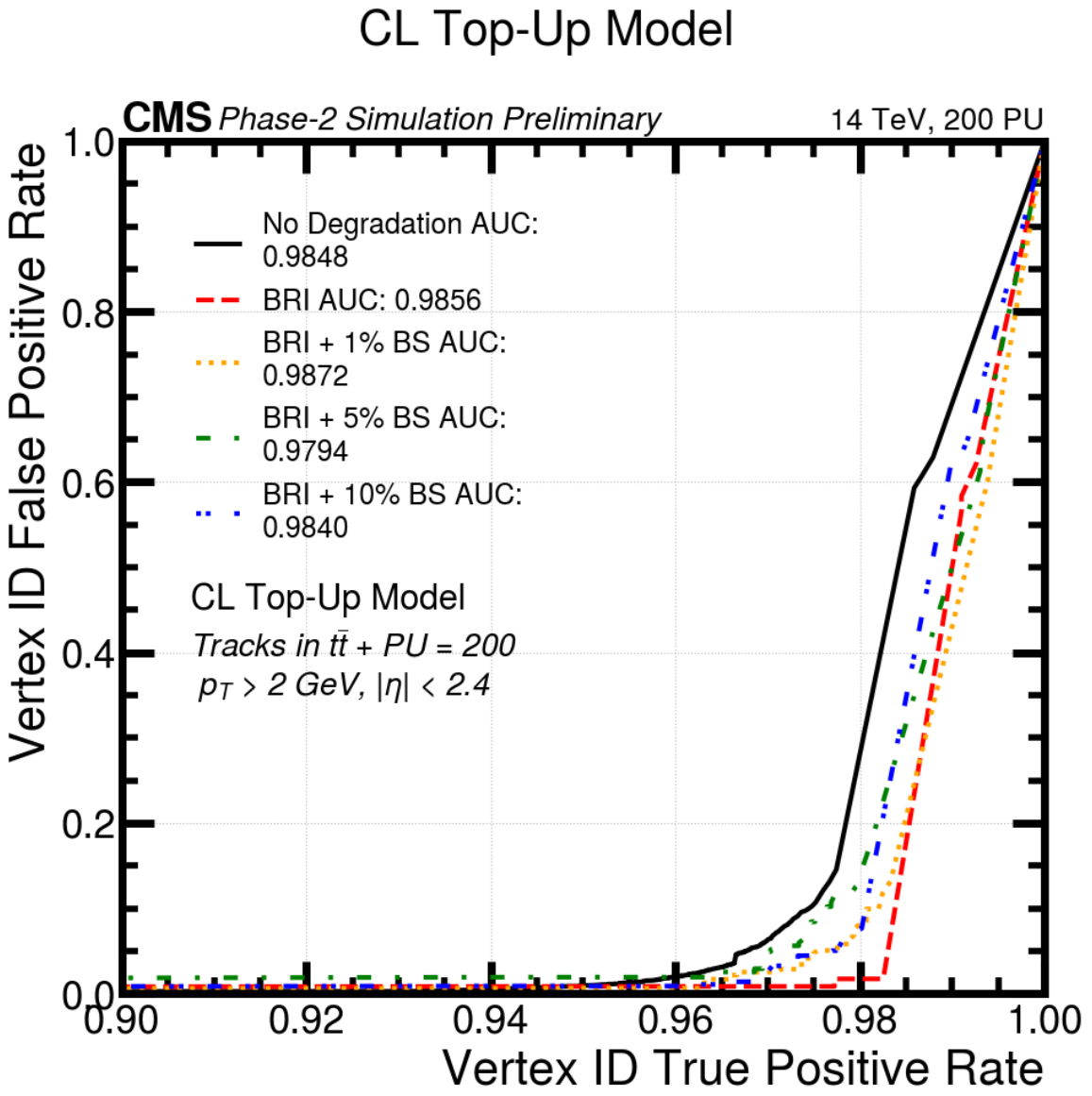
Imperial College London

CL Replay Model

Start with no retraining model, top-up train the model with new datasets in a CL manner, 7000 training events, replay algorithm, every batch has a subset of previous training datasets mixed in, always repeating old data as it trains on new

Far better performance across all datasets, robustness to the detector change

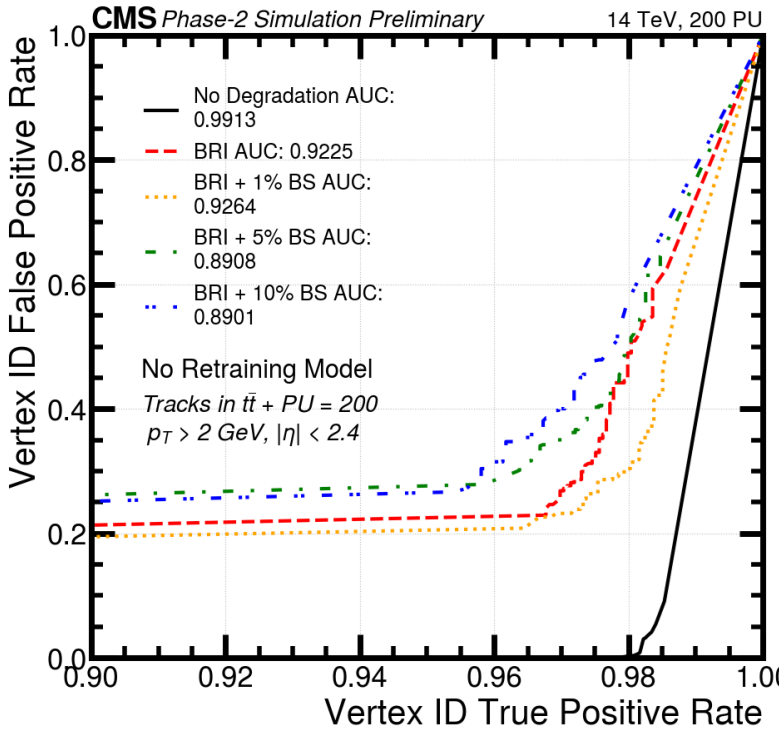
Less forgetting of original training



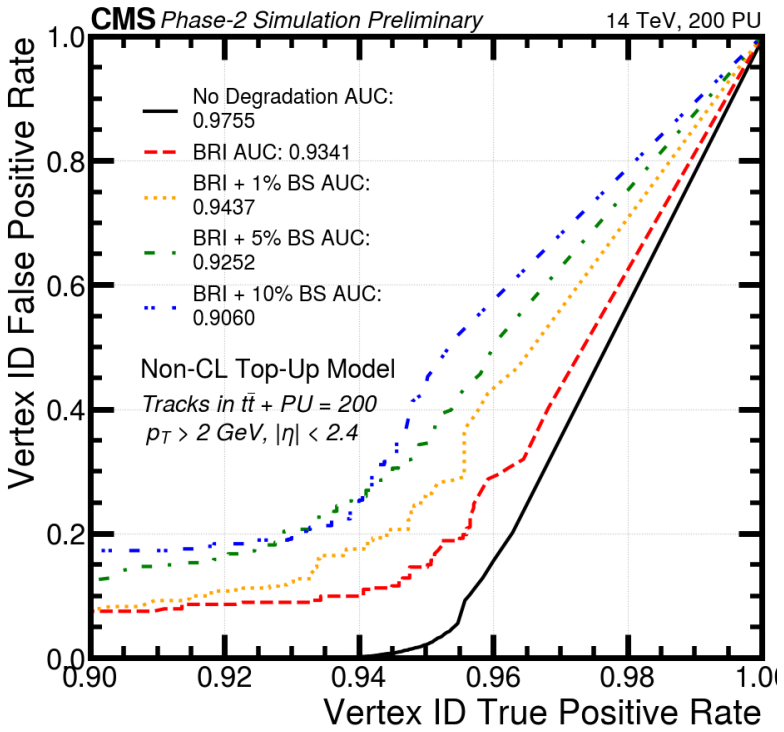
Imperial College London

Comparison

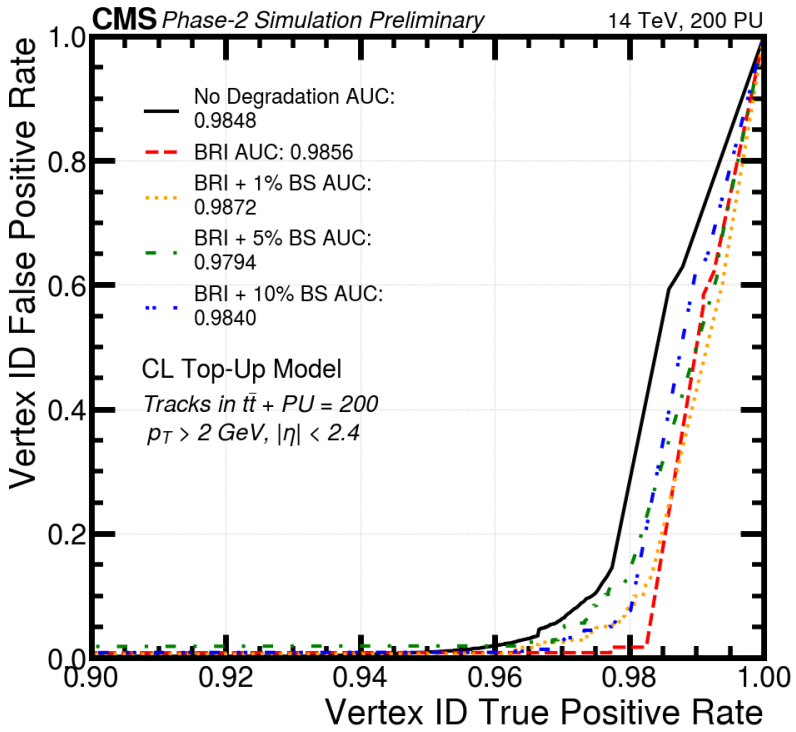
No Retraining Model



Non-CL Top-Up Model



CL Top-Up Model



Imperial College London

Continual Learning

Imperial College London

Continual learning

Continual learning (CL) is a particular machine learning paradigm where the data distribution and learning objective change through time, or where all the training data and objective criteria are never available at once.

The evolution of the learning process is modeled by a sequence of learning experiences where the goal is to be able to learn new skills all along the sequence without forgetting what has been previously learned.

https://arxiv.org/pdf/1907.00182.pdf

Imperial College London

Motivation

- Changes to the input distribution of a model can lead to a model being sub-optimal or invalid
- Having to retrain a model offline means there is a loss of accuracy until the newly retrained model is online
- Original training data might not be available anymore
- Data is generated continuously and is impossible to store
- Data cannot be generated:
  - Simulation is too computationally expensive
  - It is hard to predict the future

Imperial College London

Online Learning

Online learning represents a family of machine learning methods, where a learner attempts to tackle some predictive (or any type of decision-making) task by learning from a sequence of data instances one by one at each time

https://www.sciencedirect.com/science/article/pii/S0925231221006706

Imperial College London

Online Learning

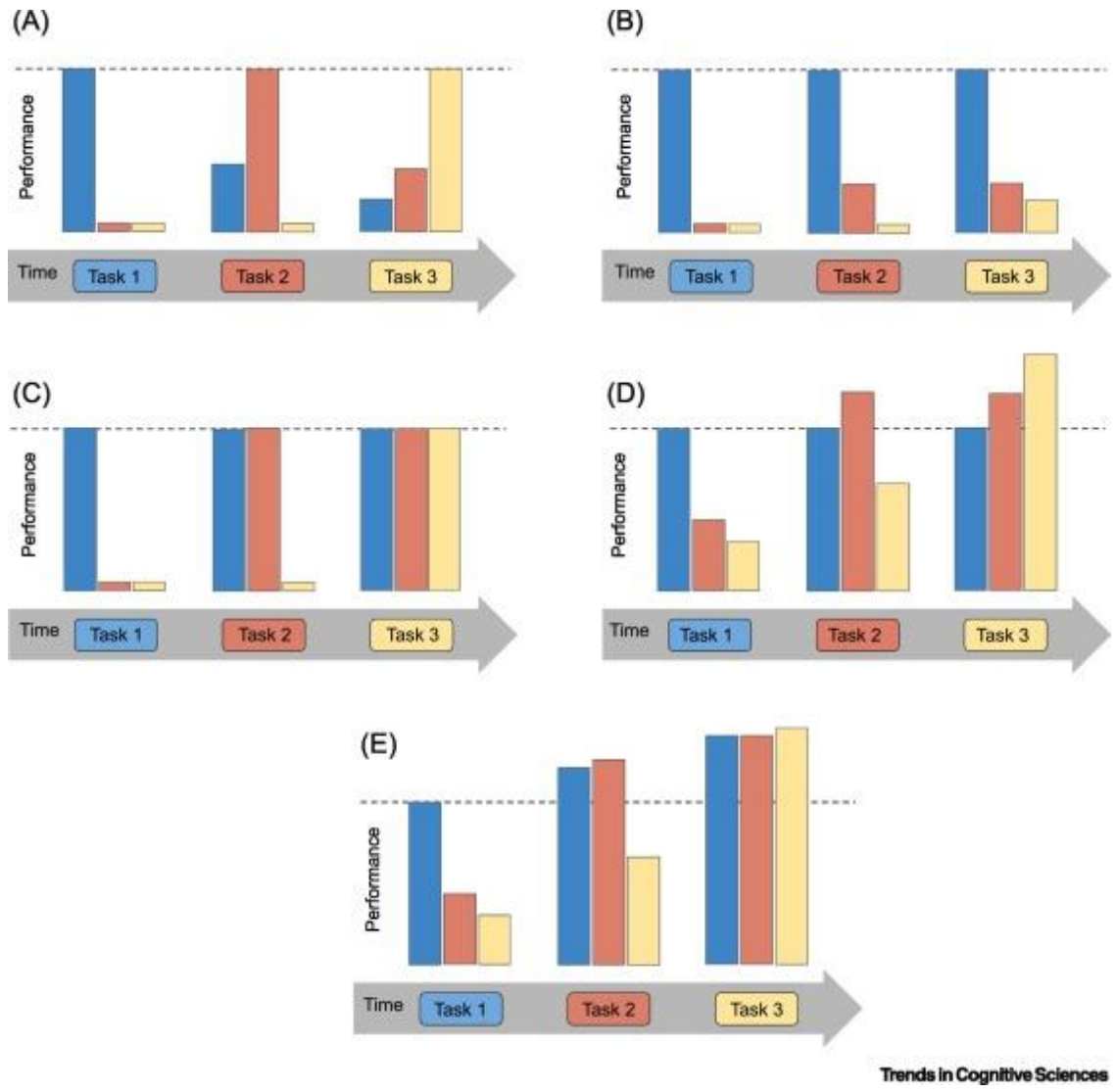
It may suffer from catastrophic forgetting.

Catastrophic forgetting: the model abruptly forgets the information of the past when trained on new tasks.

Imperial College London

CL outcomes

A. Catastrophic forgetting
B. Too little plasticity
C. No forgetting
D. No forgetting + forward transfer
E. No forgetting + forward transfer + backward transfer



https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-66132030219-9#secst0025

Imperial College London

Continual Learning (CL)

Concept: Train a model with a continuous stream of data

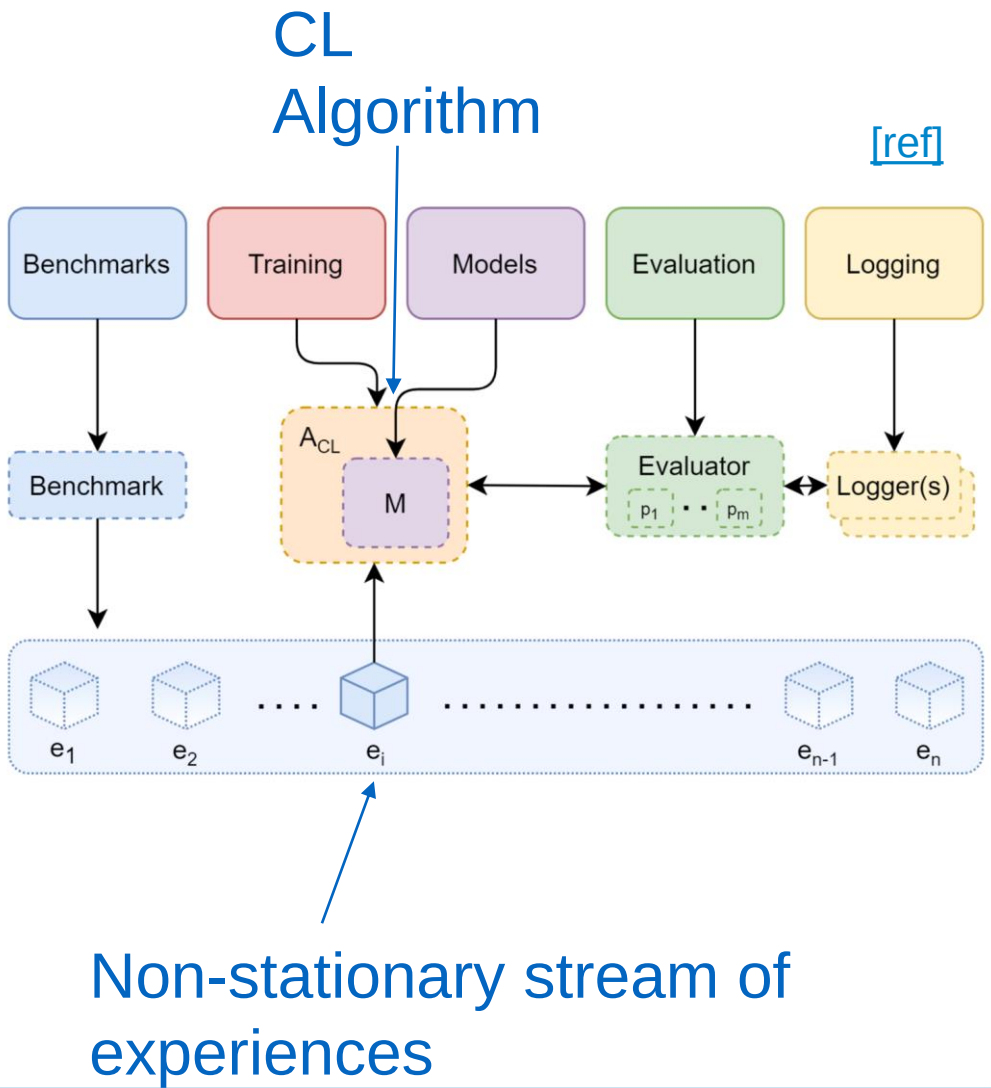
Learns from a sequence of partial experiences rather than all the data at once

Advantages:

- Avoids **catastrophic forgetting** → initial training is not disregarded
- **Adapts** to a **changing data stream** → don't need to quantify how the environment changes
- Don't need to store previous training data or retrain model

Disadvantages:

- For supervised learning need a ~continual stream of labelled data which might not be accessible
- Inference is more expensive



Imperial College London

CL strategies

- Dynamic architecture approaches:
  - Explicit architecture modification
  - Implicit architecture modification
  - Dual architectures
- Regularization Approaches:
  - Penalty computing
  - Knowledge Distillation
- Replay
- Generative Replay
- Hybrid Approaches

https://arxiv.org/pdf/1907.00182.pdf

Imperial College London

Explicit architecture modification

- Progressive NNs [ref]: for each new task to be learned, a new model is created connected to all past ones.
  - the growth of parameters is quadratic w.r.t. the number of tasks
- Growing NNs [ref]: dynamically extend layers or deepen the network
- Learning without Forgetting (LWF) [ref]: dynamically adding neurons for new tasks. Output layers can be added in order not to change output parameters from previous tasks.

Imperial College London

Implicit Architecture Modification

- Dynamically freezing weights that are not changed when learning a new task
- Frozen weights are not changed by the backpropagation but are used in the forward pass
  - Difficult to balance the number of frozen weights
- Piggyback [ref]: learn binary masks to freeze weights
- PackNet [ref]: iteratively pruning unimportant weights and fine-tuning them for learning new tasks

Imperial College London

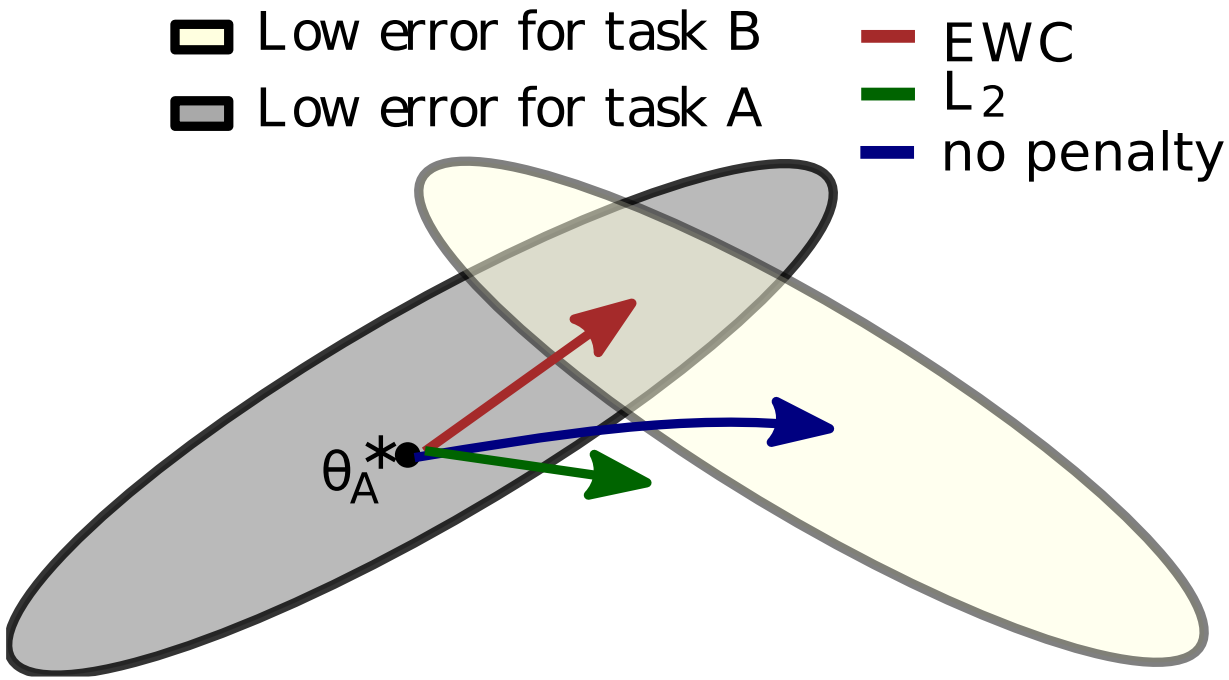
Dual networks

- Emulates the human brain:
  - Hippocampus = short term memory (STM)
  - Neocortex = long term memory (LTM)
- DGDMN [ref]:
  - If a new task is learnt an STM is assigned to it
  - When all the STM are exhausted, they are consolidated in LTM to be freed

Imperial College London

Penalty Computing

- Elastic Weight Consolidation (EWC) [ref]: searching for important weights inside the models and protect them afterwards to prevent forgetting.



Imperial College London

Penalty Computing (cont'd)

- Synaptic Intelligence (SI) [ref]:
  - endows each individual synapse with a local measure of "importance" in solving tasks the network has been trained on in the past;
  - when training on a new task we penalize changes to important parameters to avoid old memories from being overwritten.

Imperial College London

Knowledge distillation

- After A has learned to solve a task, train B to solve the same task
- Forward the same input to both A and B and impose B to have the same output as A
- Distillation should be more efficient than retraining B because A produces a soft-target that helps B to learn faster
- In continual learning, after network A learned to solve the old task, distill knowledge from A to B while B is learning a new task. B should learn how to solve both tasks.

https://arxiv.org/abs/1503.02531, https://arxiv.org/pdf/1907.00182.pdf

Imperial College London

Pure Rehearsal (replay)

- Save past samples as memory of past tasks
  - Samples needs to be carefully chosen and sorted
- Interleave previous data with new data as model is trained
- Might require large storage
- Privacy concerns

https://hal.science/hal-01951954/document

Imperial College London

Generative Replay

- Replay, but instead of storing samples, it trains a generative model on the data distribution
- One example is to use dual networks:
  - One frozen model generates samples from past experiences
  - Another learns to generate and classify actual samples in addition to the regenerated ones.
  - When a task is over, replace the frozen model by the actual one, freeze it, and initialize a new model to learn next task

https://arxiv.org/abs/1710.10368

Imperial College London

Starting point

- Replay with reservoir sampling

Imperial College London

Tools and frameworks

Imperial College London

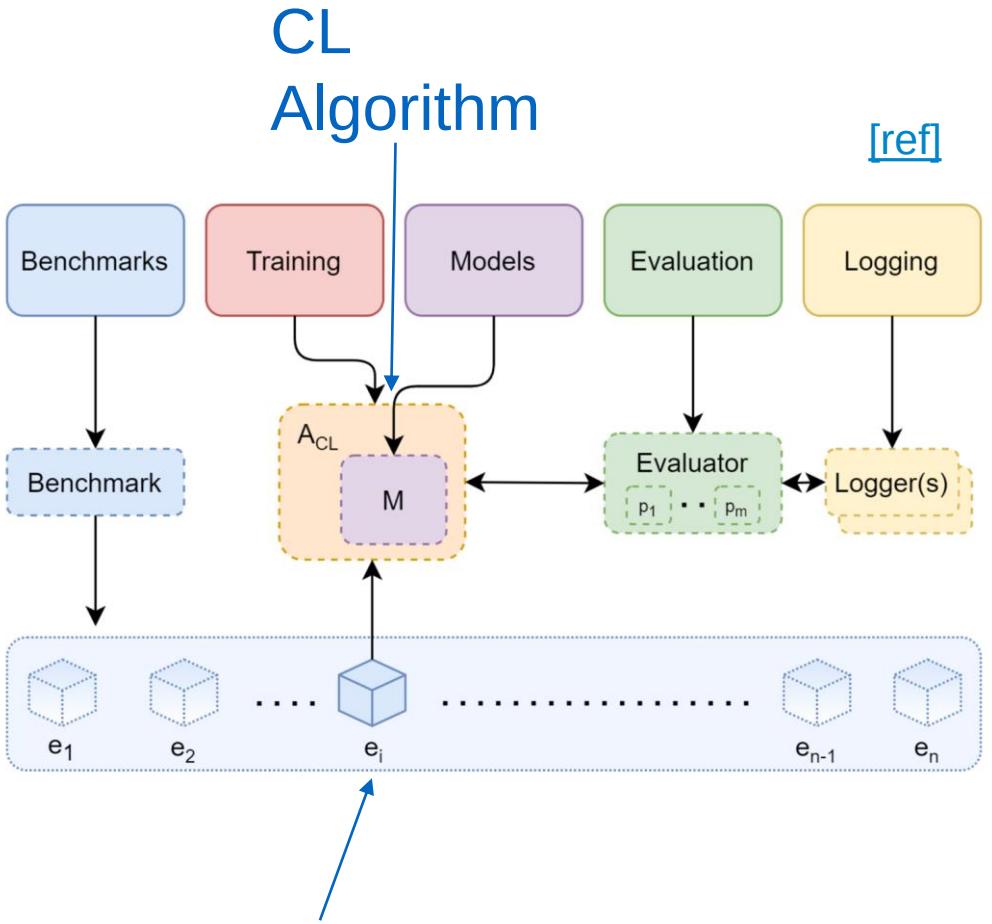
Tools and frameworks

https://pytorch.org/
https://avalanche.continualai.org/
https://arxiv.org/pdf/2104.00405.pdf

Imperial College London

Avalanche

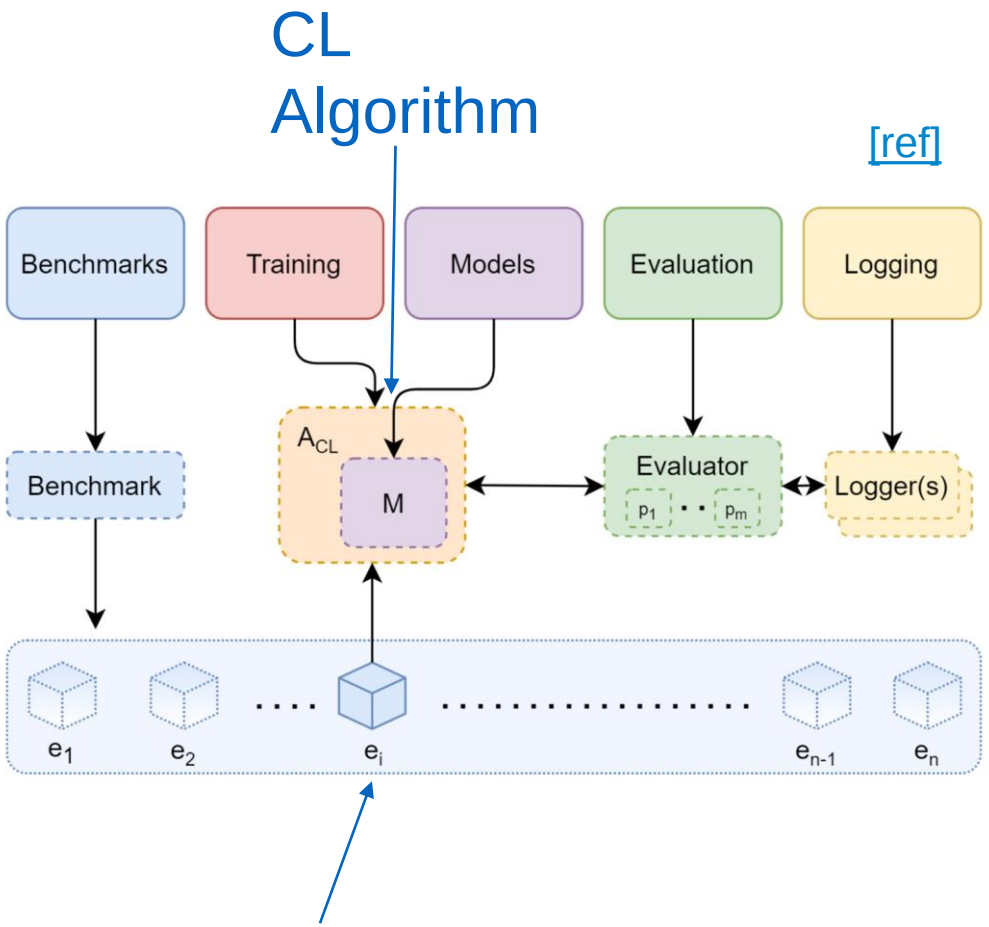
A *Benchmark* generates a stream of experiences e_i which are sequentially accessible by the continual learning algorithm A_{CL} with its internal model M.
The *Evaluator* object directly interacting with the algorithm provides a unified interface to control and compute several performance metrics (p_i), delegating results logging to the Logger(s) objects.



Imperial College London

Avalanche

- Benchmarks
- Model architecture
- CL strategy
- Metrics
- Logger



https://arxiv.org/pdf/2104.00405.pdf

Imperial College London

Benchmark

- Dataset: collection of examples
- Scenario: specificities about the continual stream of data, a continual learning algorithm will face
- Benchmark: combination of a scenario with one or multiple datasets

- Need to specify training and test experiences and task labels
- The benchmark provides train stream and test stream

Imperial College London

Benchmark

```
pattern_shape = (3, 32, 32)

# Definition of training experiences
# Experience 1
experience_1_x = torch.zeros(100, *pattern_shape)
experience_1_y = torch.zeros(100, dtype=torch.long)

# Experience 2
experience_2_x = torch.zeros(80, *pattern_shape)
experience_2_y = torch.ones(80, dtype=torch.long)

```

Imperial College London

Benchmark

```
12    # Test experience
13    # For this example we define a single test experience,
14    # but "tensors_benchmark" allows you to define even more than one!
15    test_x = torch.zeros(50, *pattern_shape)
16    test_y = torch.zeros(50, dtype=torch.long)
17
18    generic_scenario = tensors_benchmark(
19        train_tensors=[(experience_1_x, experience_1_y), (experience_2_x, experience_2_y)],
20        test_tensors=[(test_x, test_y)],
21        task_labels=[0, 0],  # Task label of each train exp
22        complete_test_set_only=True
23    )
24
```

Imperial College London

Standalone Metrics

```
1    acc_metric = Accuracy()
2    acc_metric.update(real_y, predicted_y)
3    acc = acc_metric.result()
4    print("Average Accuracy: ", acc)
5    acc_metric.reset()
6
```

Imperial College London

Plugin Metrics

```
7   eval_plugin = EvaluationPlugin(
8       accuracy_metrics(minibatch=True, epoch=True, experience=True, stream=True),
9       loss_metrics(minibatch=True, epoch=True, experience=True, stream=True),
10      timing_metrics(epoch=True),
11      forgetting_metrics(experience=True, stream=True),
12      cpu_usage_metrics(experience=True),
13      confusion_matrix_metrics(num_classes=benchmark.n_classes, save_image=False, stream=True),
14      disk_usage_metrics(minibatch=True, epoch=True, experience=True, stream=True),
15      loggers=[InteractiveLogger()],
16      strict_checks=False
17  )
18
```

Imperial College London

Strategy

They usually take only three parameters:

- Model: this must be a torch.nn.Module.
- Optimizer: torch.optim.Optimizer already initialized on your model.
- Loss: a loss function such as those in torch.nn.functional.
- Evaluator: computes performance metrics

```
cl_strategy = Naive(
    model, SGD(model.parameters(), lr=0.001, momentum=0.9),
    CrossEntropyLoss(), train_mb_size=500, train_epochs=1, eval_mb_size=100,
    evaluator=eval_plugin)
```

Imperial College London

Training loop

```
print('Starting experiment...')
results = []
for experience in benchmark.train_stream:
    print("Start of experience: ", experience.current_experience)
    print("Current Classes: ", experience.classes_in_this_experience)

    cl_strategy.train(experience)
    print('Training completed')

    print('Computing accuracy on the whole test set')
    results.append(cl_strategy.eval(benchmark.test_stream))
```

Imperial College London

Full example – Metrics

```
benchmark = create_benchmark()

# MODEL CREATION
model = create_model()

# choose some metrics and evaluation method
interactive_logger = InteractiveLogger()

eval_plugin = EvaluationPlugin(
    accuracy_metrics(minibatch=True, epoch=True, experience=True, stream=True),
    loss_metrics(minibatch=True, epoch=True, experience=True, stream=True),
    forgetting_metrics(experience=True),
    loggers=[interactive_logger],
)
```

Imperial College London

Full example – CL strategy

```
16  cl_strategy = Replay(
17      model,
18      torch.optim.Adam(model.parameters(), lr=0.1),
19      CrossEntropyLoss(),
20      train_passes=1,
21      train_mb_size=10,
22      eval_mb_size=32,
23      device=device,
24      evaluator=eval_plugin
25  )
26
```

Imperial College London

Full example – training loop

```
27  # TRAINING LOOP
28  print("Starting experiment...")
29  results = []
30
31  # Create online benchmark
32  batch_streams = benchmark.streams.values()
33  # ocl_benchmark = OnlineCLScenario(batch_streams)
34  for i, exp in enumerate(benchmark.train_stream):
35      # Create online scenario from experience exp
36      ocl_benchmark = OnlineCLScenario(
37          original_streams=batch_streams, experiences=exp, experience_size=10
38      )
39      # Train on the online train stream of the scenario
40      cl_strategy.train(ocl_benchmark.train_stream)
41      results.append(cl_strategy.eval(benchmark.test_stream))
42
```

Imperial College London

Full example – Naïve baseline

```
cl_strategy = OnlineNaive(
    model,
    Adam(model.parameters(), lr=0.1),
    CrossEntropyLoss(),
    train_passes=1,
    train_mb_size=10,
    eval_mb_size=32,
    device=device,
    evaluator=eval_plugin,
)
```

Imperial College London

Full example – Joint

```python
# Joint training strategy
joint_train = JointTraining(
    model,
    optimizer,
    criterion,
    train_mb_size=32,
    train_epochs=1,
    eval_mb_size=32,
    device=device,
)
```

```python
# train and test loop
# Differently from other
# avalanche strategies,
# you NEED to call train
# on the entire stream.
joint_train.train(train_stream)
print(joint_train.eval(test_stream))
```

Imperial College London

Alternating Minimization

Imperial College London

Training

- It is computationally expensive
- Usually done on GPUs
- Embedded systems, constrained environments do not have accelerators

Imperial College London

Goal

Find an alternative to **stochastic gradient descend** (SGD) to allow CL on embedded system

Imperial College London

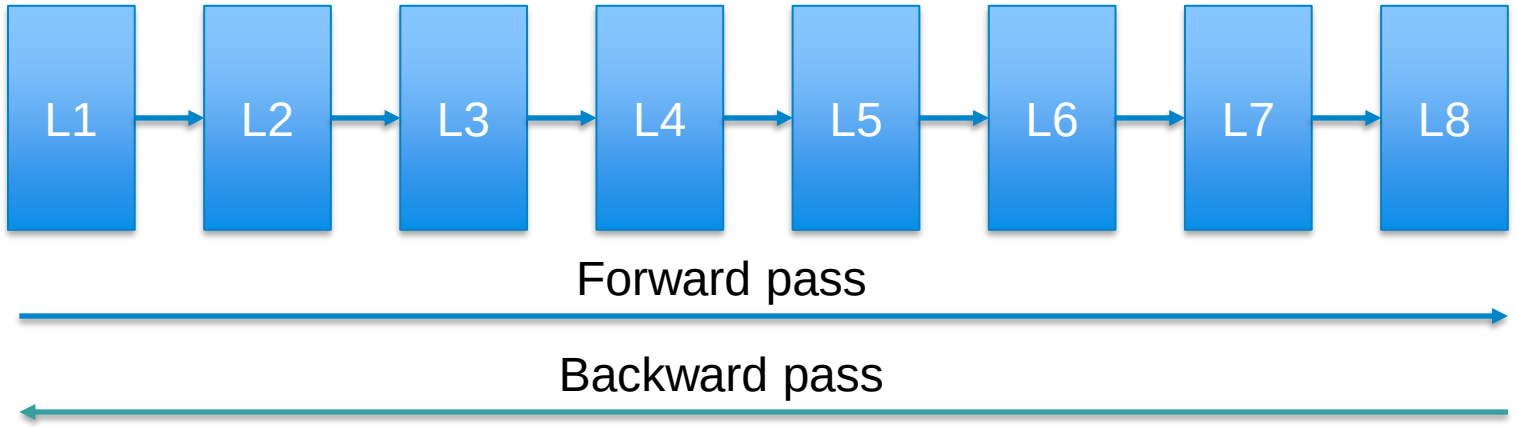
Alternating Minimisation (Altmin)

- An alternative to SGD
- Open source
- Proof of concept
- Experiments are reproducible

Imperial College London

Altmin vs SGD

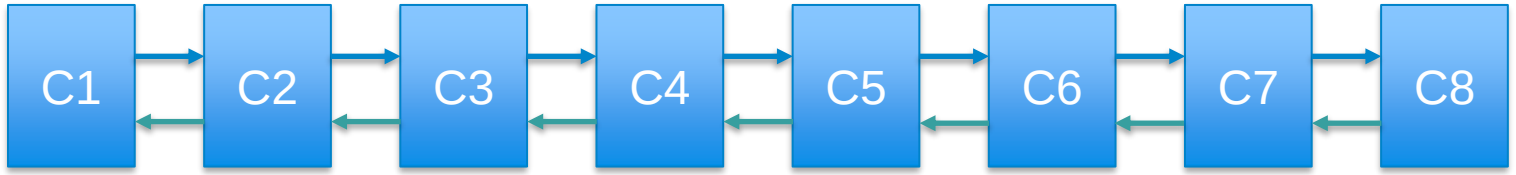
AltMin drops one order of complexity



L1 cost function depends on all the previous layers

SGD

Altmin



- Forward pass populates "codes"
- Cost function depends only on the previous layer **(No chain rule)**

Imperial College London

Altmin vs SGD

	Backpropagation	Alternating Minimisation
Vanishing and exploading gradients	Yes	No
Biologically implausible	Yes	Yes (but closer to plausible)
Allows for parallel weight updates	No	Yes
Gives good accuracy on benchmark datasets	Yes	Yes
Speed of initial convergence	Slightly slower	Slightly quicker
Smoothness of convergence	High	Medium

Imperial College London

Dataset

Modeled a "typical" High Luminosity LHC event that emulates the CMS phase-2 upgraded detector

Dataset created from a top quark pair production sample generated in Pythia with an additional 200 soft proton-proton interactions overlayed on top

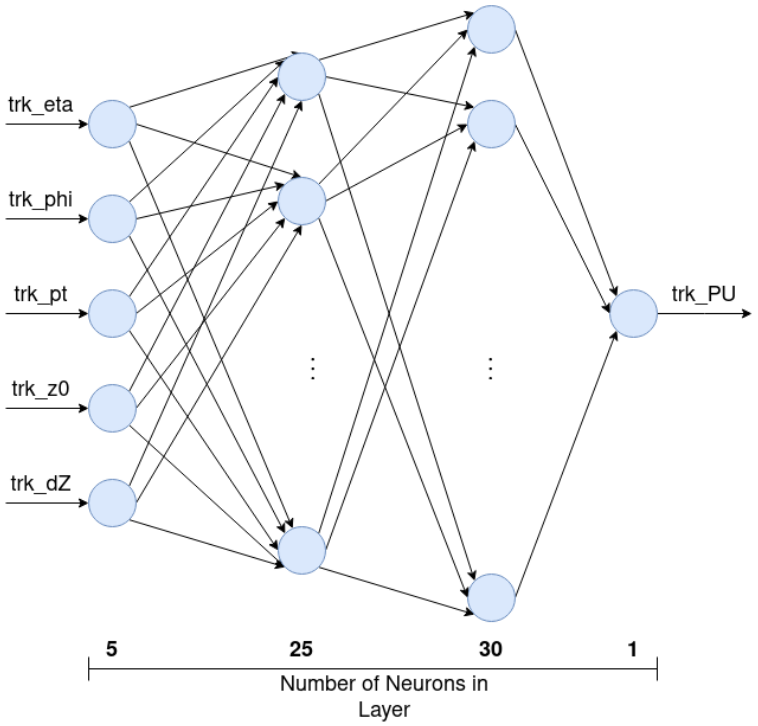
Tracks were generated using Delphes running a simulation of the high lumi CMS detector, tracks kinematically constrained to pT > 2 GeV, |eta| < 2.4 and |z0| < 15 cm to emulate a CMS level-1 tracking set of tracks

Tracks were then reprocessed in python to generate two datasets, an unsmeared dataset of 10,000 events, taken directly from the Delphes output and another 10,000 events that were smeared using a gaussian smear on each track parameter. This smearing was gradually increased across a set of 10 separate smearings to give a set of 10 individual experiences for the CL algo. Smearings emulate a worsening of detector resolution over time

Imperial College London

Network

- The model takes 5 features as input
- There are two hidden layers with 25 and 30 neurons respectively
- Track eta, phi, pt and z0 are track helix parameters taken from the delphes simulation and smeared using a guassian smear for the smeared datasets
- Track dZ is the distance between the track z0 and a primary vertex found in the event using a simple histogram based vertex algorithm
- The target is whether the track originated from the underlying top quark pair production or from the additional soft pileup



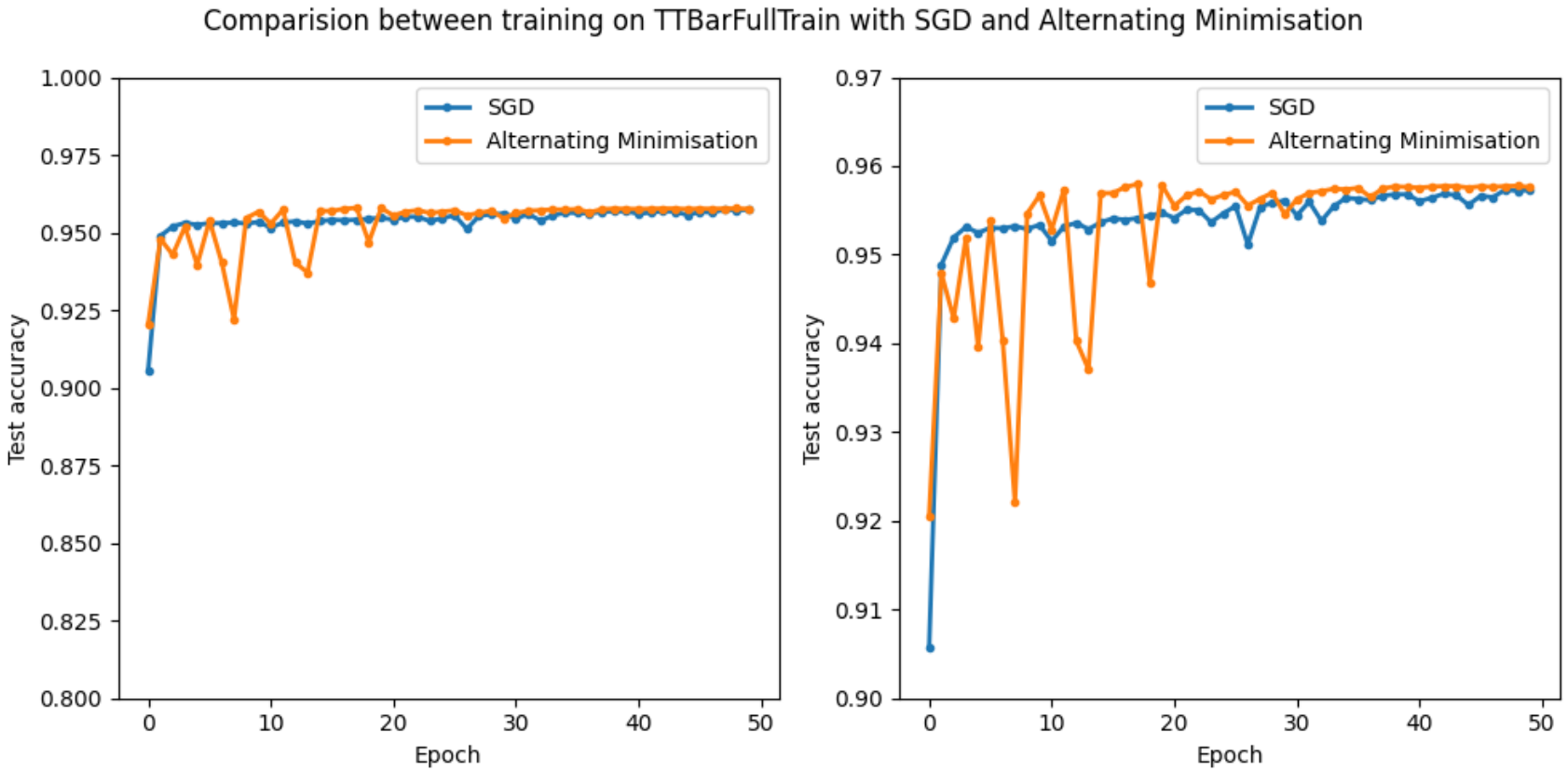
Imperial College London

Experimental setup

- Trained on non smeared data
- Executed the CL model on smeared data
- Compared against SGD

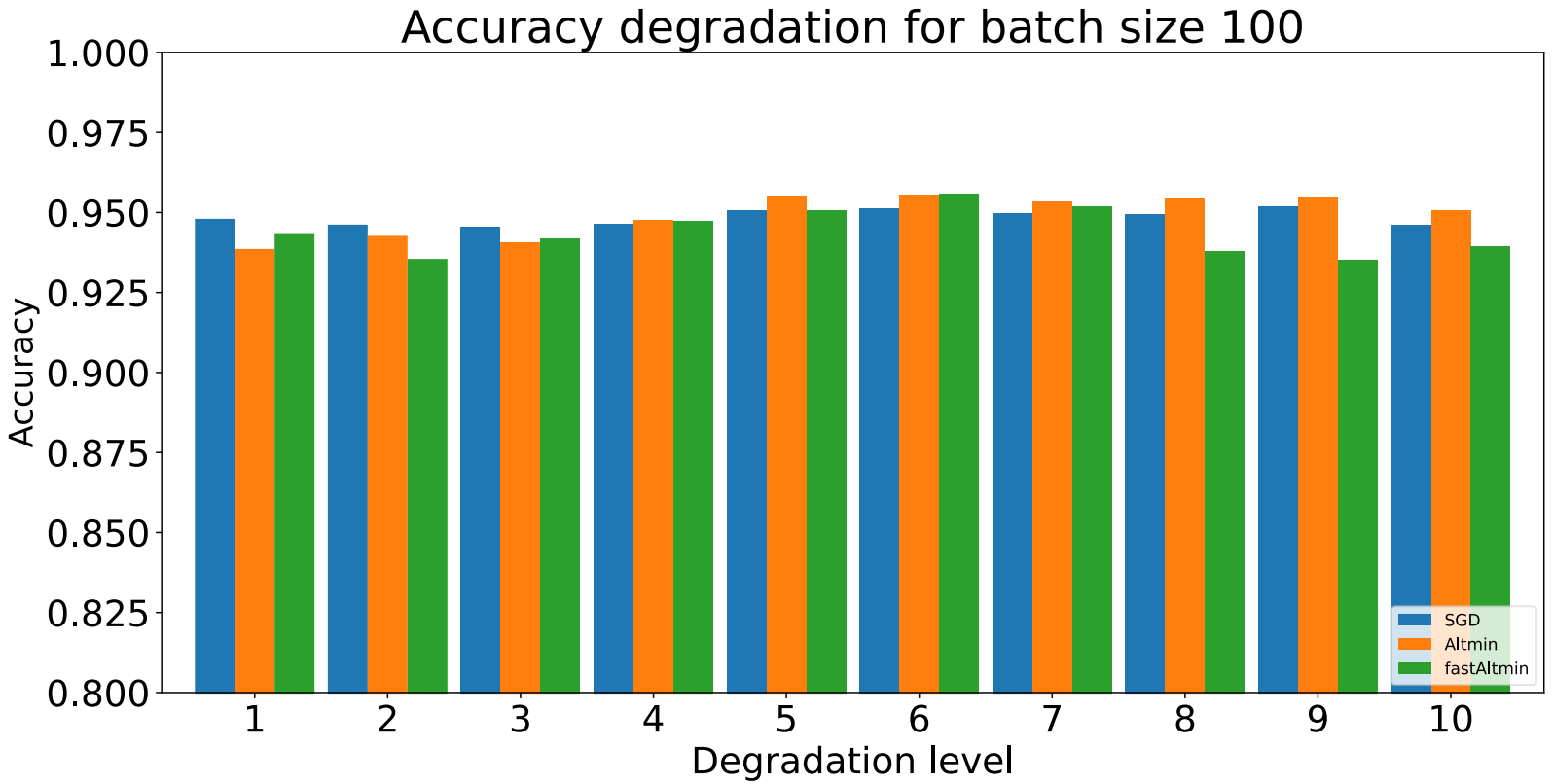
Imperial College London

Training convergence



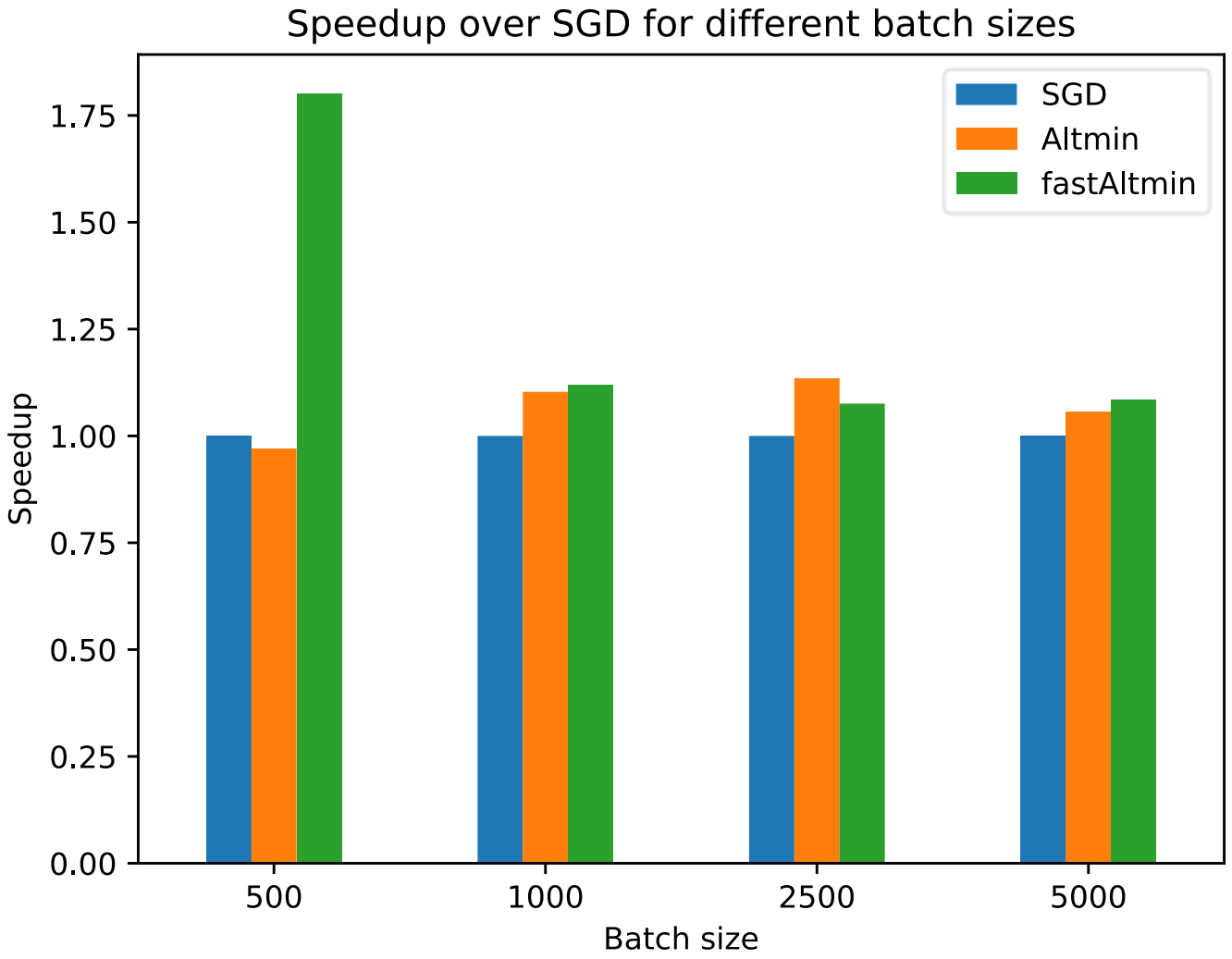
Imperial College London

Network accuracy (CL)



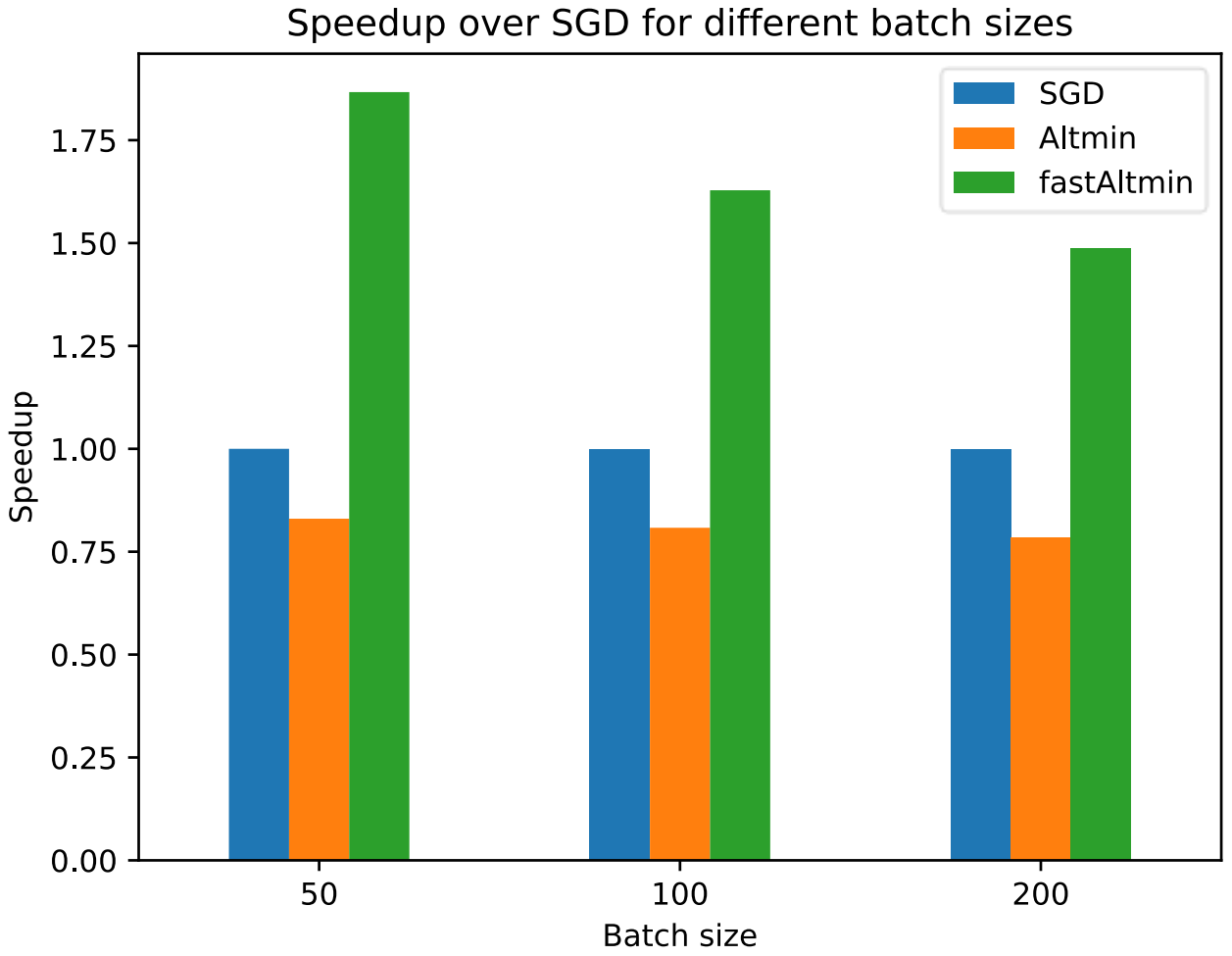
Imperial College London

Training Speedup



Imperial College London

Speedup CL scenario



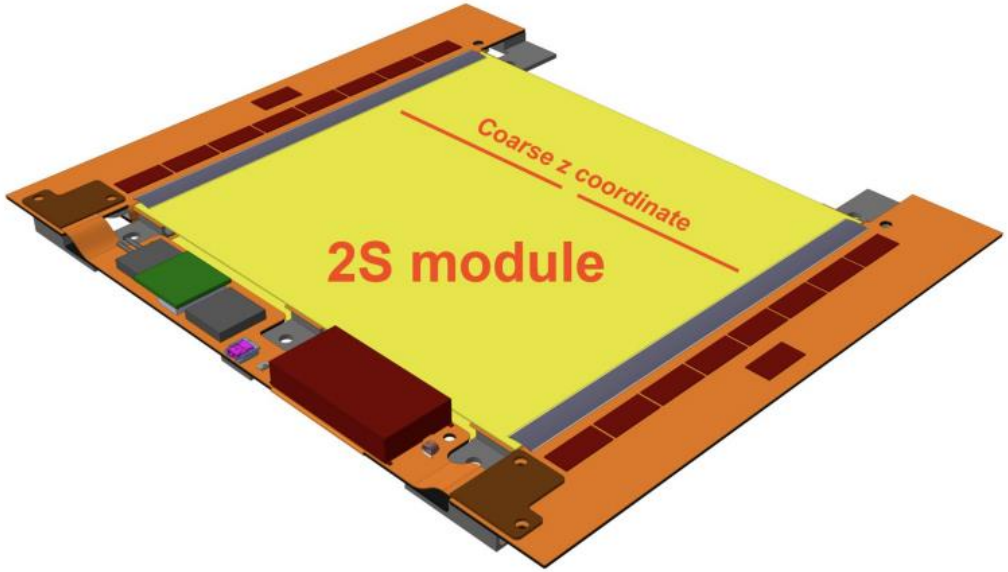
Imperial College London

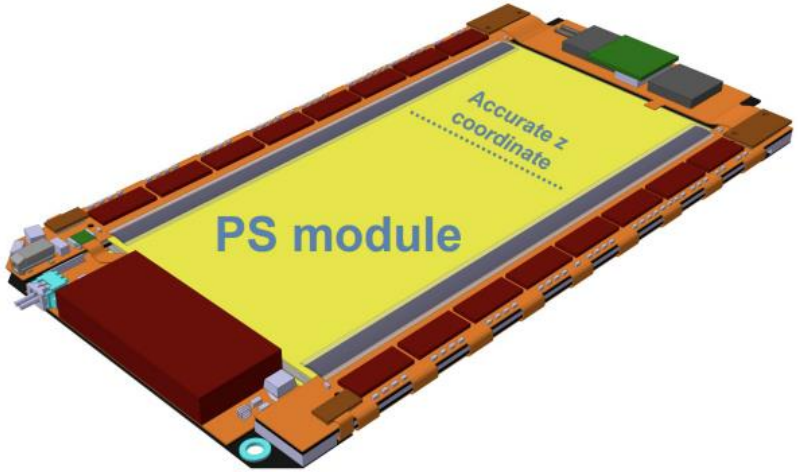
Recap

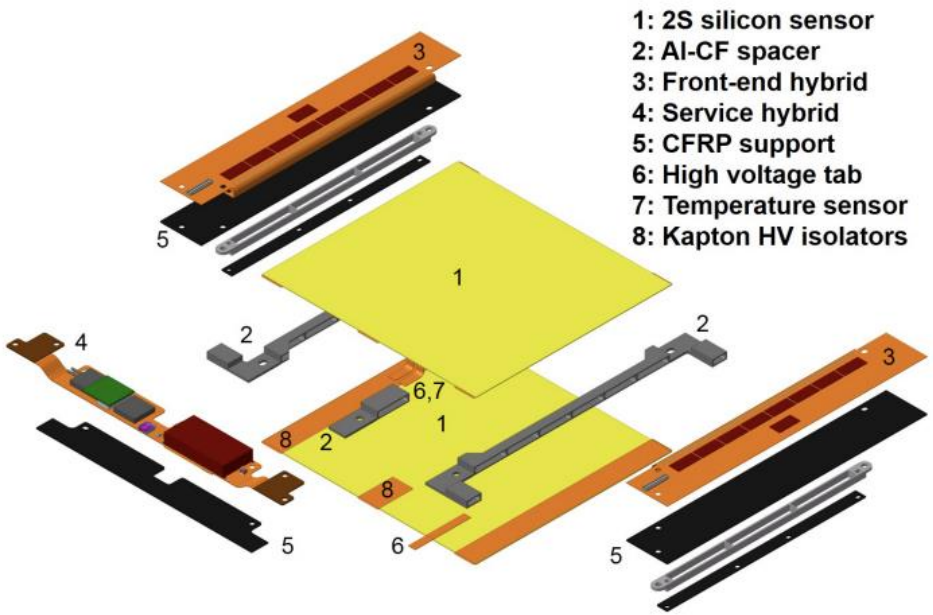
- CL can outperform traditional ML in HEP
- Mature framework & tools, easy to use
- CL can be cheaper to train
- Recent developments in the field of beyond backpropagation is making CL deployable in resource constrained environments

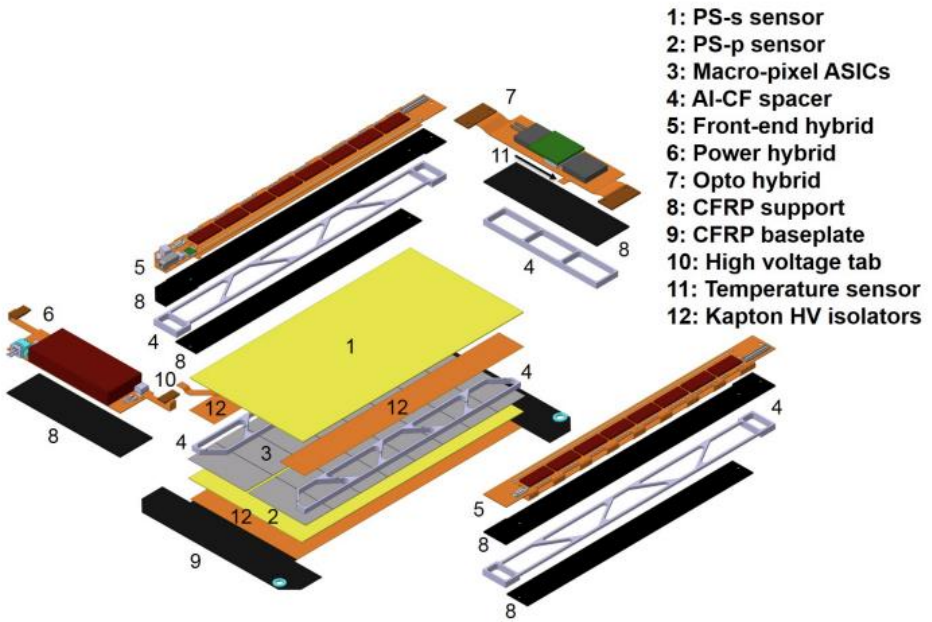
Imperial College London

Backup


Coarse z coordinate
2S module


Accurate z coordinate
PS module


1: 2S silicon sensor
2: Al-CF spacer
3: Front-end hybrid
4: Service hybrid
5: CFRP support
6: High voltage tab
7: Temperature sensor
8: Kapton HV isolators


1: PS-s sensor
2: PS-p sensor
3: Macro-pixel ASICs
4: Al-CF spacer
5: Front-end hybrid
6: Power hybrid
7: Opto hybrid
8: CFRP support
9: CFRP baseplate
10: High voltage tab
11: Temperature sensor
12: Kapton HV isolators

Imperial College London

Timescales for CMS

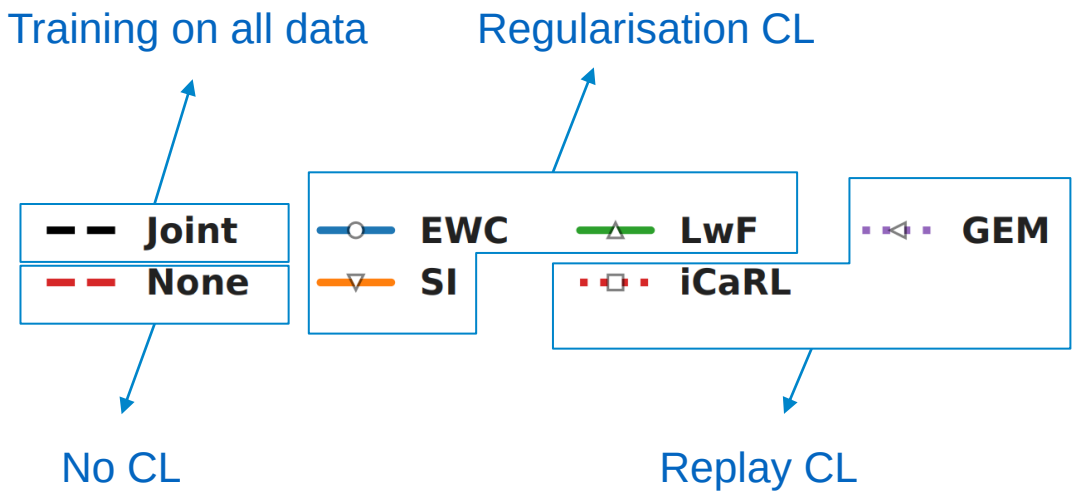
Seconds	Days	Months
Beam fluctuations	Beam conditions or small degradation	Significant detector changes
No time to create training data or retrain a model L1 can monitor ML robustness Online CL could be explored	Time to collect data directly from detector e.g. scouting or full reconstruction Between fill calibration [1][2]	Time to accurately emulate detector performance in large MC campaigns

[1] CMS-DP-2022-042 [2] CMS-DP-2022-068

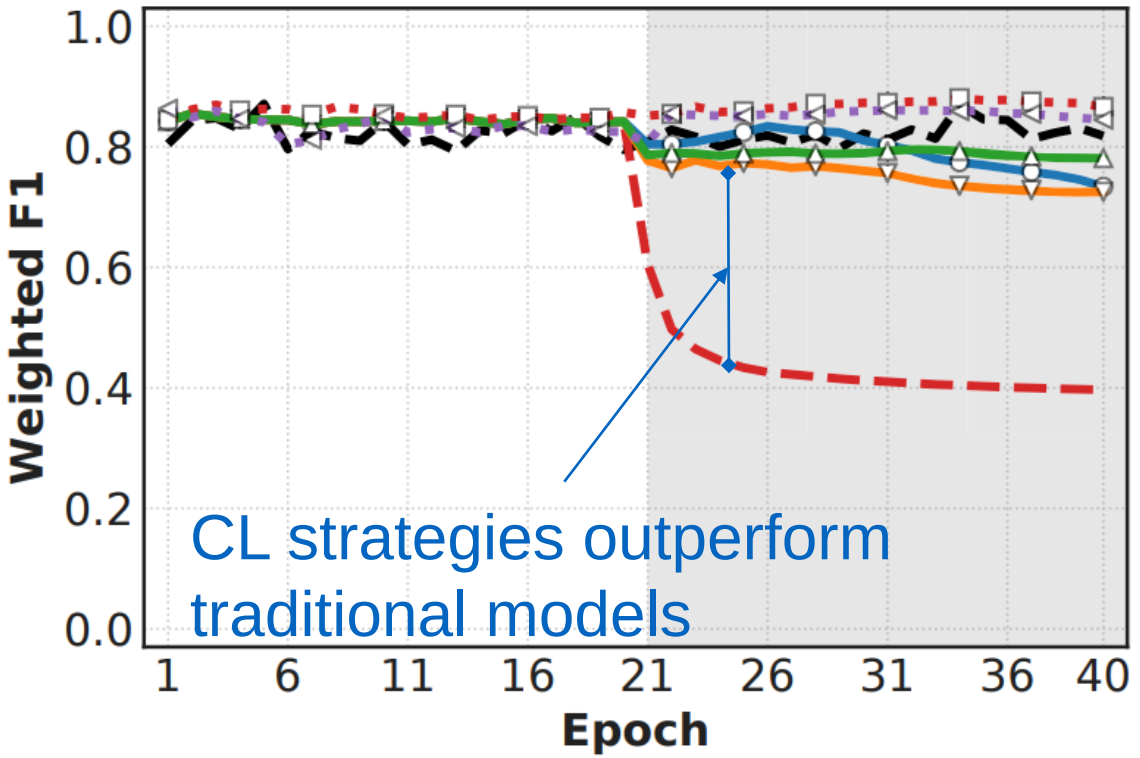
Imperial College London

Embedded Continual Learning

https://arxiv.org/ftp/arxiv/papers/2110/2110.13290.pdf



Epoch 21, starts using real-time data



Imperial College London

Regularization

- Regularization is a process of introducing additional information to prevent overfitting. I.e. weight sparsification, dropout, and early stopping.

- In the CL context there are more involved techniques.