GRAPHCORE HARDWARE AND SOFTWARE

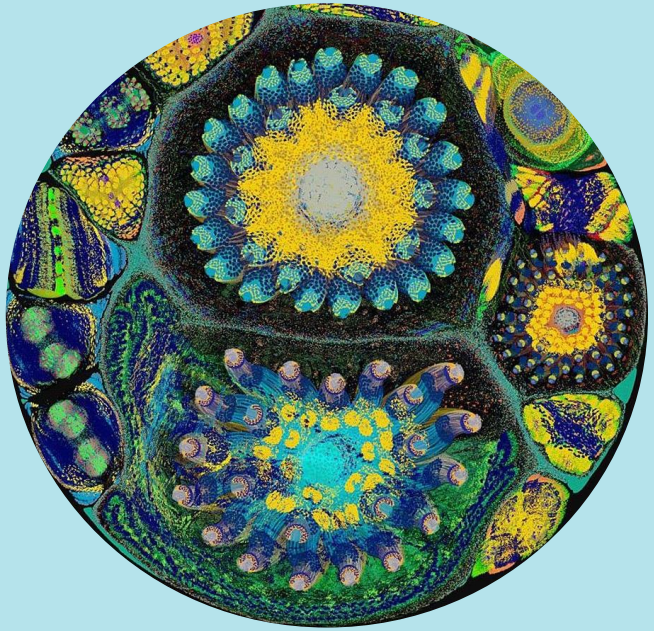
GRAPHCORE

GRAPHCORE'S SOLUTION

Hardware

IPU processor
designed for AI

Software

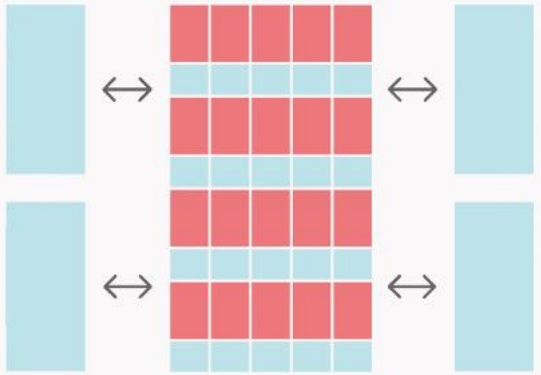
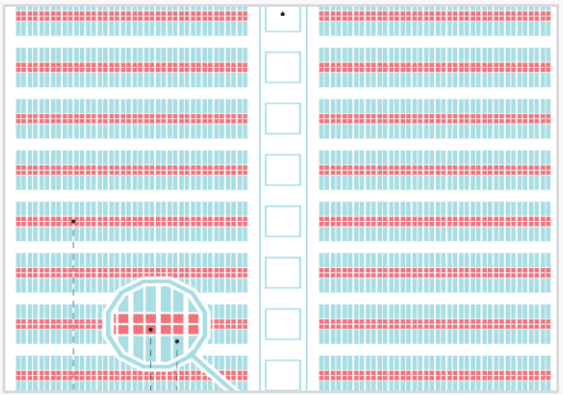


Poplar SDK and
development tools

Platform

IPU platforms
Available in the cloud

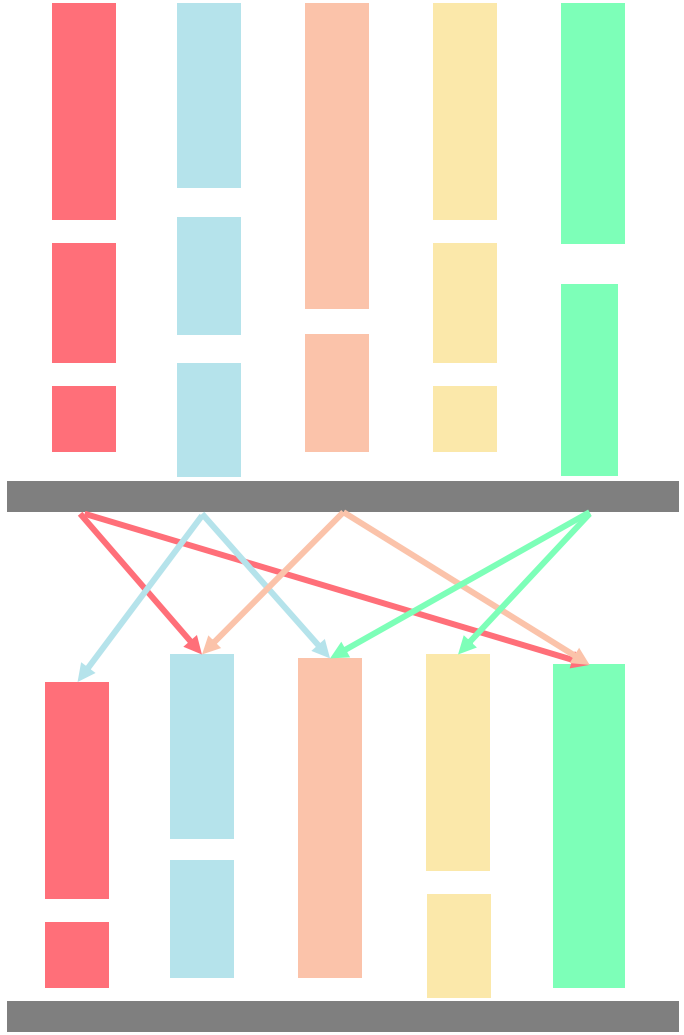
THE INTELLIGENCE PROCESSING UNIT (IPU)
WHAT MAKES IT DIFFERENT?

	CPU	GPU	IPU
Parallelism	Designed for scalar processing	SIMD/SIMT architecture. Designed for large blocks of dense contiguous data	Massively parallel MIMD architecture. High performance/efficiency for future ML trends
			
Memory Bandwidth	Off-chip memory	Model and Data spread across off-chip and small on-chip cache and shared memory (2TB/s for A100 HBM)	Main Model & Data in tightly coupled large locally distributed SRAM (~65 TB/s for Bow IPU)

BRIDGING SOFTWARE TO PARALLEL HARDWARE

Bulk Synchronous Parallel (BSP) is the computer science solution that bridges software to parallel processors.
- IPU implements BSP in hardware.

BSP solves parallel software issues:
- free of concurrency hazards
- efficient for highly parallel processors
- Poplar-SDK builds BSP compute on IPU hardware

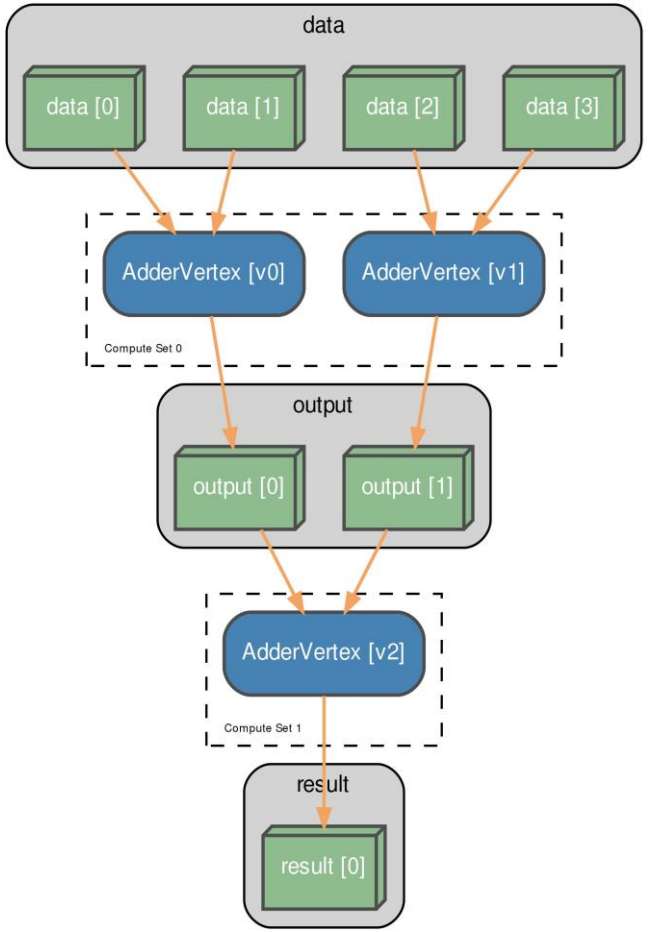


1. Compute
2. Sync
3. Exchange
4. Repeat...

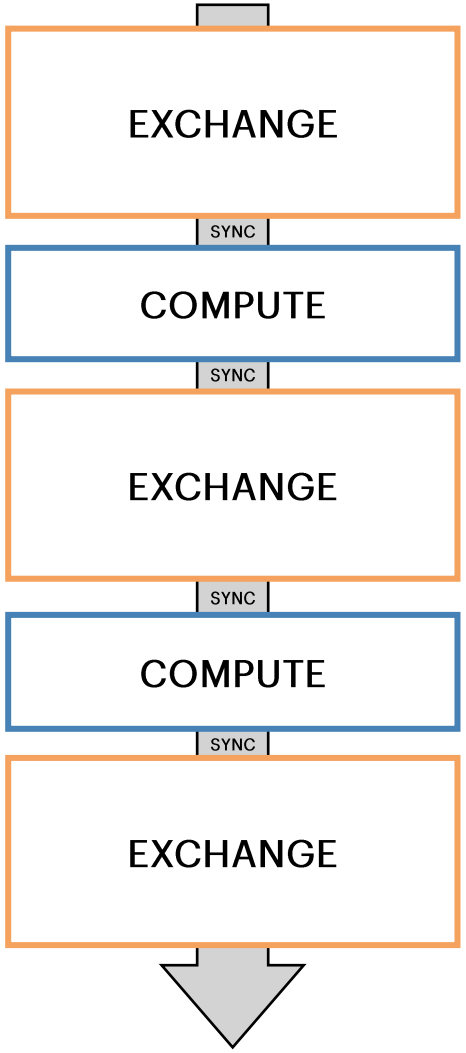
[1] Pregel: a system for large-scale graph processing, 2010; [2] Large-scale graph partitioning with Apache Giraph, 2014

EXECUTION MODEL

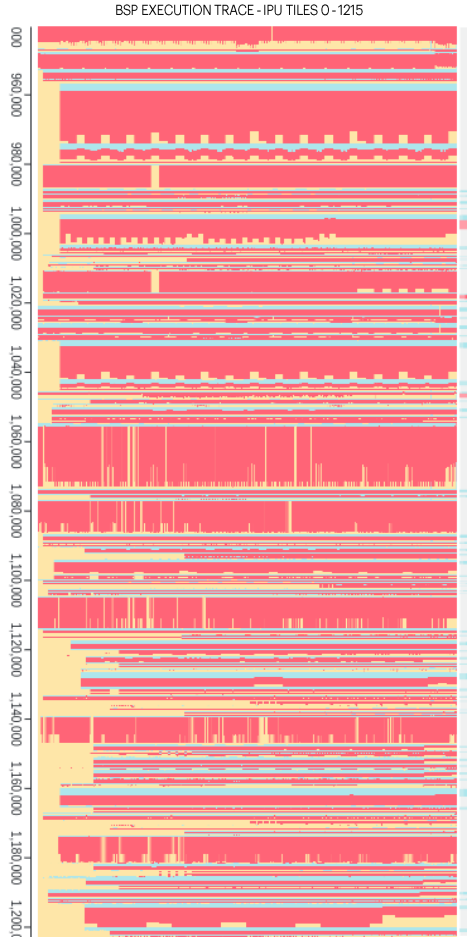
COMPUTATIONAL GRAPH



BSP SCHEDULE



OPTIMIZED IPU EXECUTION



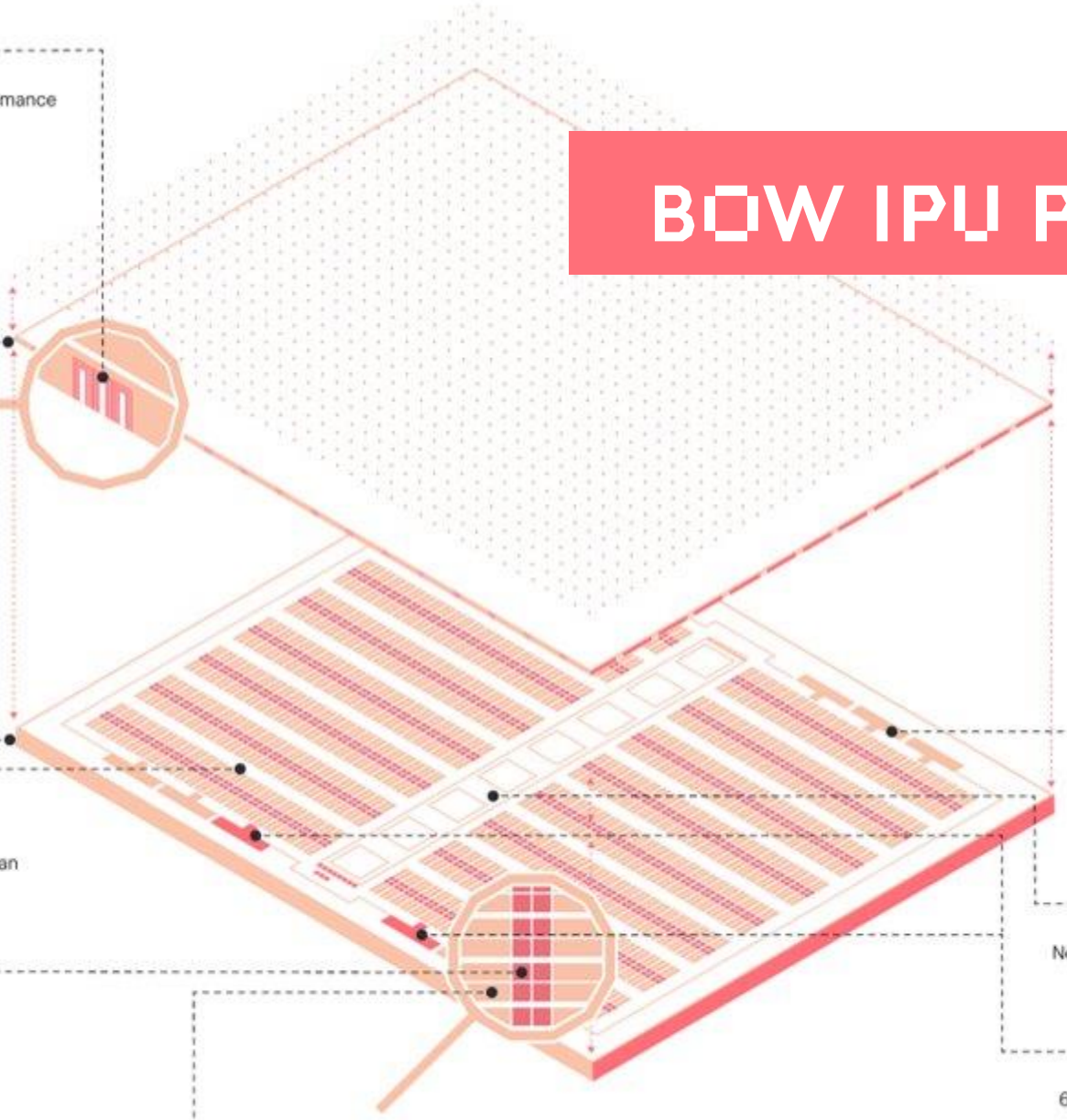
OUTPUT FROM POPVISION GRAPH ANALYSER

GRAPHCORE

PRODUCT DETAILS

GRAPHCORE

BOW IPU PROCESSOR



Deep Trench Capacitor

Efficient power delivery
Enables increase in operational performance

Wafer-On-Wafer

Advanced silicon 3D stacking technology

Closely coupled power delivery die

Higher operating frequency and enhanced overall performance

IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

1472 independent IPU-Core™

8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

65.4TB/s memory bandwidth per IPU

IPU-Links™

10x IPU-Links,
320GB/s chip to chip bandwidth

IPU-Exchange™

11 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

BOW-2000 IPU MACHINE

1U blade form factor delivering 1.4 PetaFLOPS AI Compute

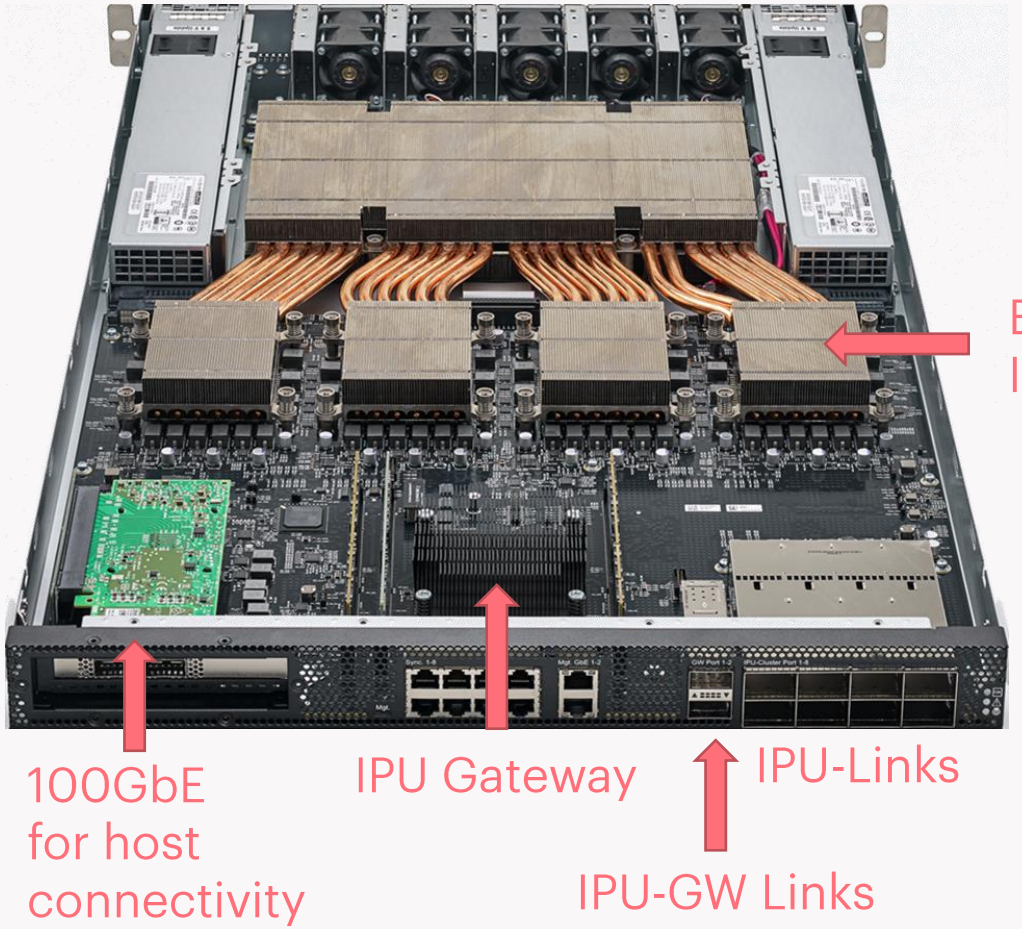
Disaggregated AI/ML accelerator platform

Excellent performance & TCO leveraging In-Processor memory & IPU-Exchange

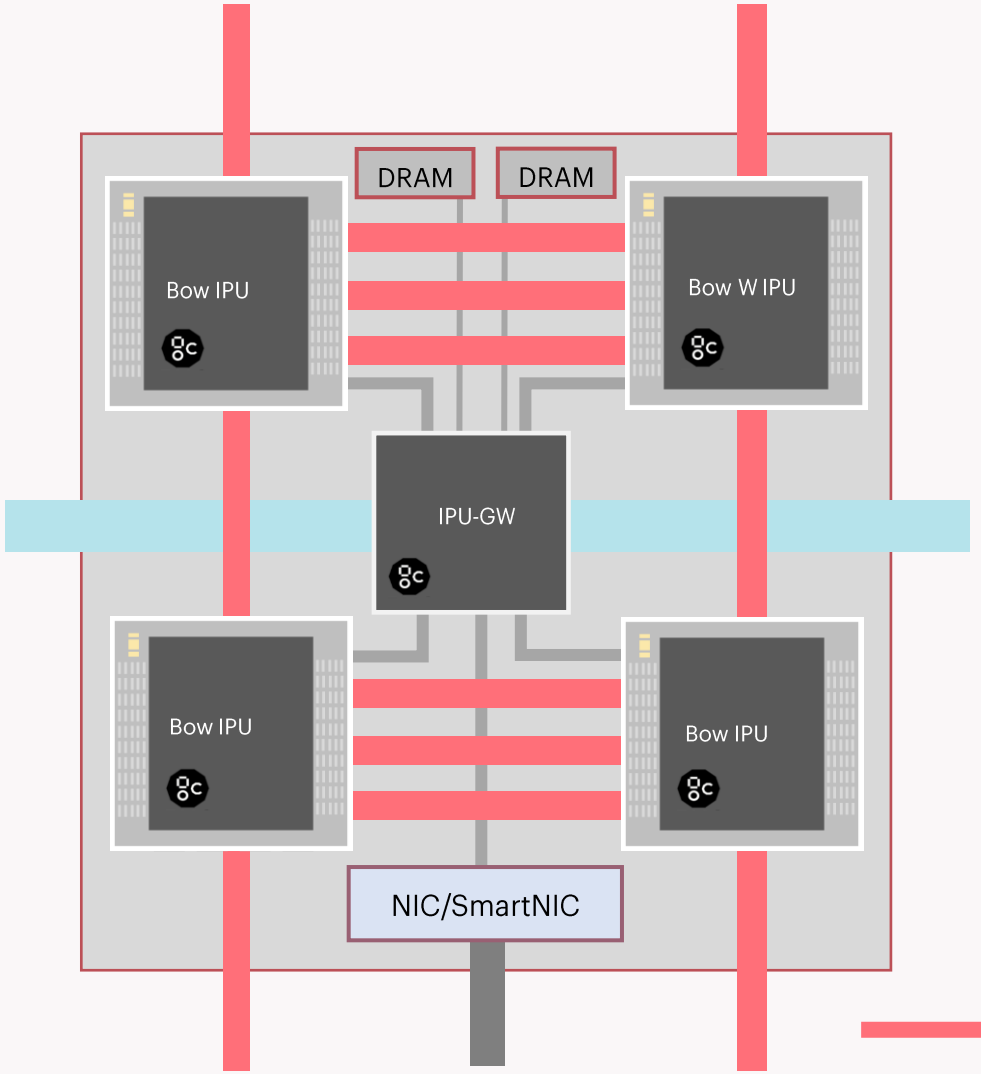
IPU-Links scale to Bow Pod64

Expansion to Bow Pod256 and beyond with IPU-GW Links

BOW IPU-2000



BOW-2000: THE BUILDING BLOCK OF LARGE PODS



COMPUTE

4x Bow IPUs

- 1.4 $PFLOP_{16}$ compute
- 5,888 processor cores
- > 35,000 independent parallel threads

DATA

Exchange Memory

- 3.6GB In-Processor-Memory @ 260 TB/s
- 128GB Streaming Memory DRAM (up to 256GB) @ 20 GB/s

COMMUNICATIONS

IPU-Fabric managed by IPU-GW

- Host-Link – 100GE to Poplar Server for standard data center networking
- IPU-Link – 2D Torus for intra-POD64 communication
- GW-Link - 2x 100Gbps Gateway-Links for rack-to-rack – flexible topology

x16 IPU-Link [64GB/s]

Host-Link Network I/F [100Gbps]

IPU-GW Link [100Gbps]

x8 PCIe G4 [32GB/s]

BOW: 3RD GENERATION IPU SYSTEMS

BOW POD16

4x Bow-2000
5.6 PetaFLOPS
1 CPU server

BOW POD64

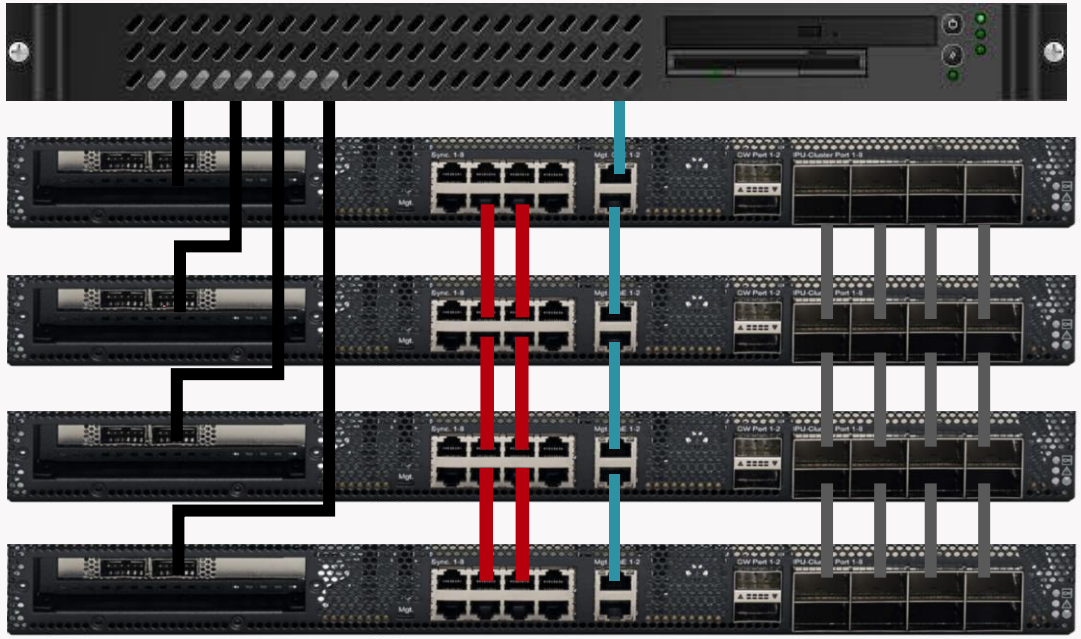
16x Bow-2000
22.4 PetaFLOPS
1-4 CPU server(s)

BOW POD256

64x Bow-2000
89.6 PetaFLOPS
4-16 CPU server(s)

BOW POD16 DIRECT ATTACH

Server with x4 Bow-2000



- Convenient cost effective evaluation platform
- Available through Graphcore channel for on-premise or Graphcloud
- Wide range of benchmarks and examples for Bow Pod_{16} performance evaluation
- Scale-out with Bow Pod_{64} and beyond

Host-Link 100GE network interface (QSFP, 1.0m)

1GbE Management (Cat5, 1.5m)

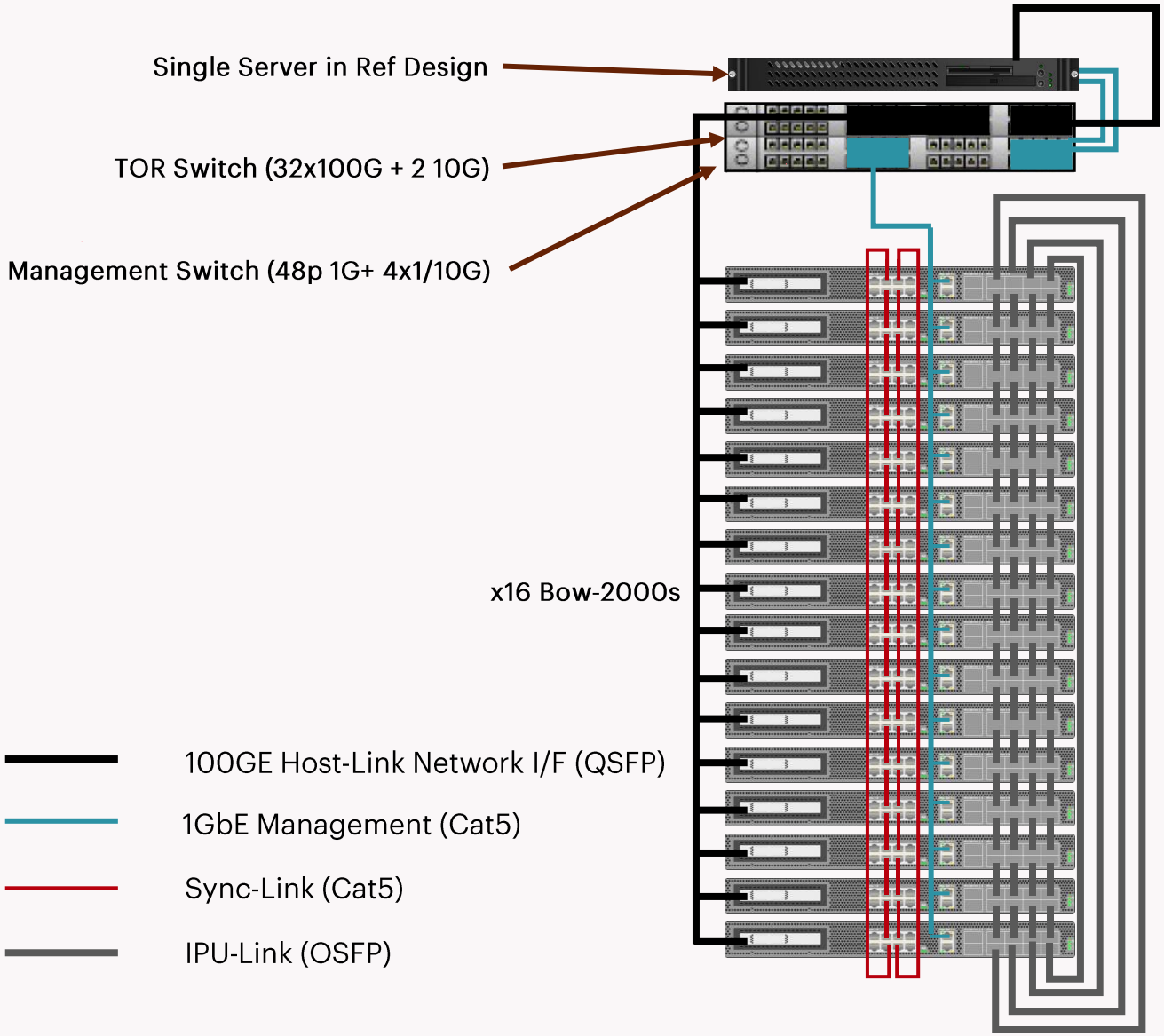
Sync-Link (Cat5, 0.15m)

IPU-Link (OSFP, 0.3m)

BOW POD64 REFERENCE DESIGN

Pre-Qualified 64-IPU Design with Reference Server and Switches

- Up to 16 Bow-2000 platforms
- Reference architecture supports different server requirements based on workload
- Bow Pod_{64} Configuration:
  - 64 IPUs
  - 22.4 PFLOPs @ FP16.16
  - ~58GB IPU In-Processor memory
  - ~7TB Streaming Memory
- Bow Pod Host disaggregation
  - Flexibly connect required host server compute over fabric
- 2D-Torus topology
  - Maximizes bandwidth across IPU-Links
  - All-Reduce 2x faster than mesh topology
- Scalable to 64K Bow IPUs

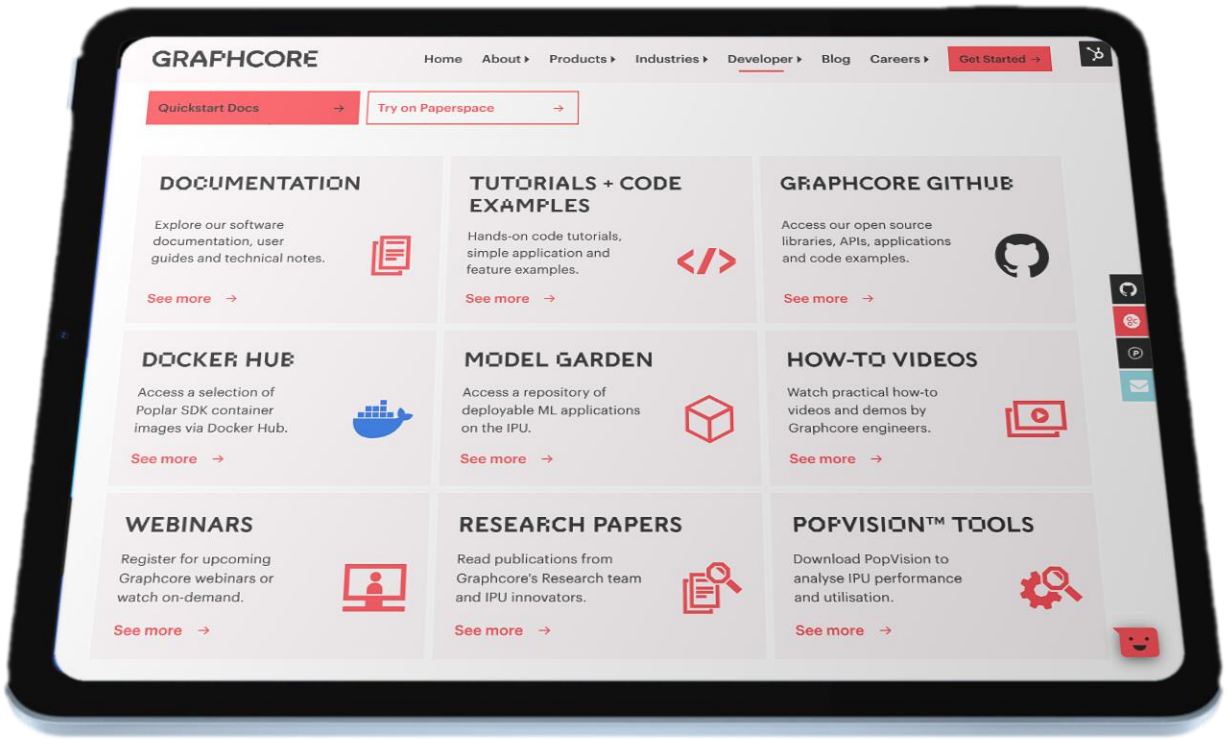


MODELS AND SOFTWARE

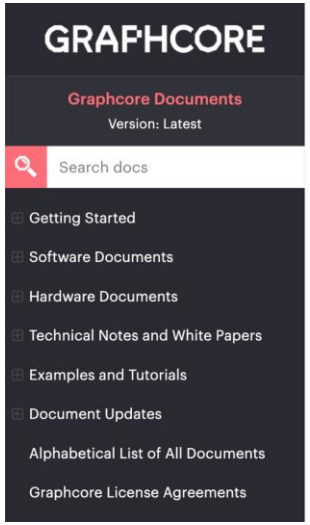
GRAPHCORE

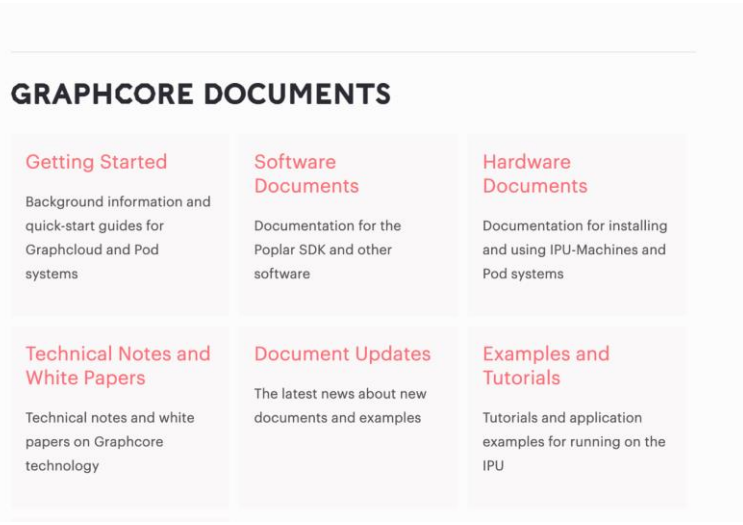
GRAPHCORE SOFTWARE ECOSYSTEM

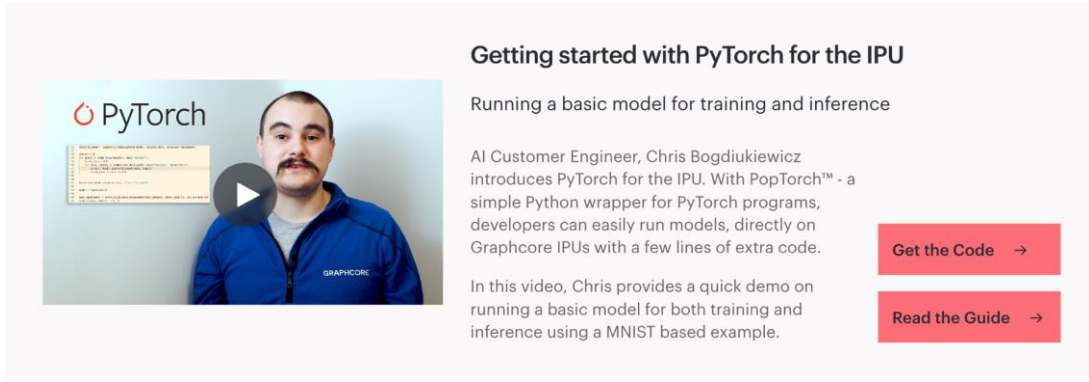
WORLD CLASS DEVELOPER RESOURCES FOR IPU USERS



WWW.GRAPHCORE.AI/DEVELOPER







GRAPHCORE SOFTWARE MATURITY

NLP/TRANSFORMERS
IMAGE CLASSIFICATION/CNNS
OBJECT DETECTION
LARGE MODELS
MLPERF
CONDITIONAL SPARSITY
GNNS

ML APPLICATIONS

TUTORIALS
CODE EXAMPLES
DOCUMENTATION
VIDEOS
NATIVE IPU CODERS PROGRAM
APPS PORTFOLIO

DEVELOPER ECOSYSTEM

FRONTENDS
JUPYTER NOTEBOOKS
INFERENCE DEPLOYMENT TOOLKIT

FRAMEWORKS
ONNX
HALO
Keras
PaddlePaddle

FW BACKENDS
XLA
POPART+
POPDIST
PARTITIONER
POPIR
POPIT

POPLAR®
POPLIBS
GCL
POPLAR
GRAPH ENGINE
GRAPH COMPILER

DRIVERS
GC DEVICE ACCESS LAYER
IPUOF DRIVER
PCIe DRIVER

POPLAR® SDK

GRAPH ANALYZER
SYSTEM ANALYZER
DEBUGGER
DEVELOPMENT ENVIRONMENT

POPVISION TOOLS

V-IPU

SYSTEM MONITORING
PROMETHEUS
GRAFANA

JOB DEPLOYMENT
K8S
SLURM

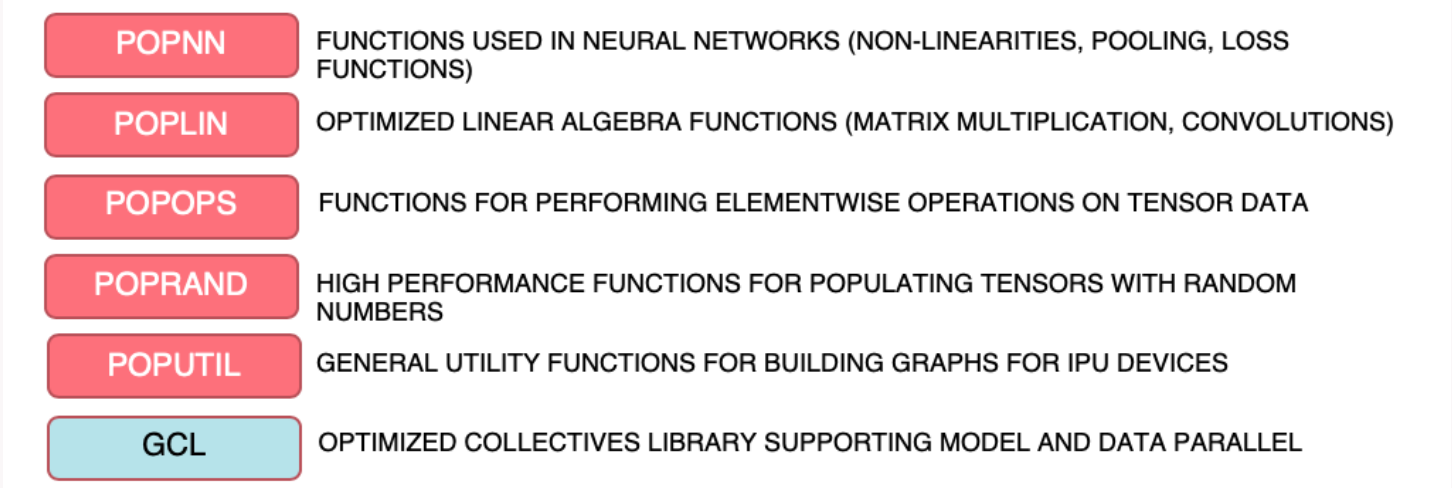
SYSTEM SOFTWARE

GRAPHCORE

POPLIBS GRAPH LIBRARIES

Complete set of open-source libraries of Machine Learning primitives and building blocks

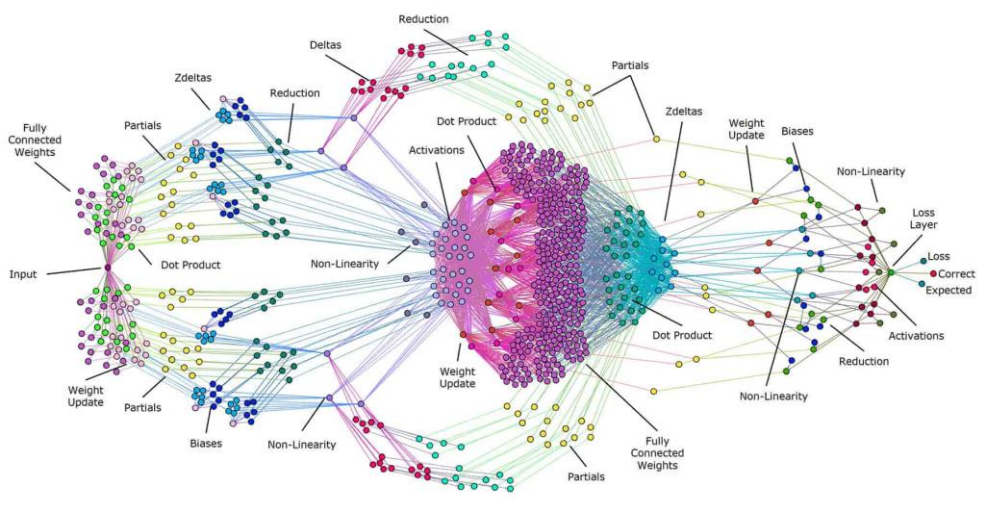
- Over 50 optimized functions for common machine learning models
- More than 750 high performance compute elements
- Simple C++ graph building API
- Implement any application
- Full control flow support



GRAPH COMPILER

State of the art compiler for simplified IPU programming

- Handling the scheduling and work partitioning of large parallel programs including memory control:
- Optimized execution of the entire application model to run efficiently on IPU platforms
- Alleviates the burden on developers to manage data or model parallelism
- Code generation using standard LLVM



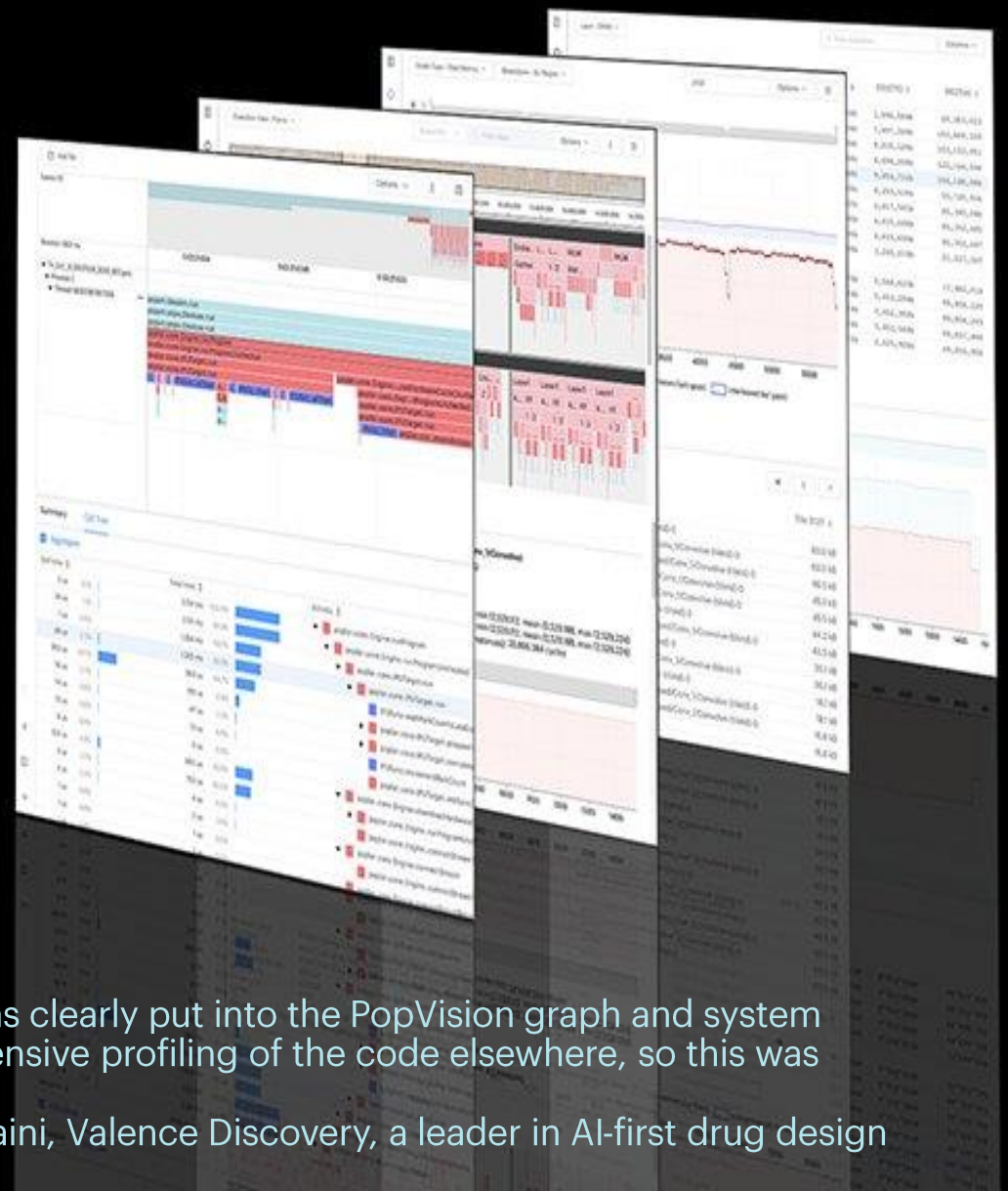
POPVISION™ TOOLS

GRAPH ANALYSER

Useful for analysing and optimising the memory use and execution performance of ML models on the IPU

SYSTEM ANALYSER

Graphical view of the timeline of host-side application execution steps



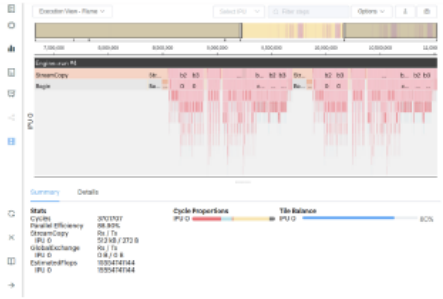
"Our team was very impressed by the care and effort Graphcore has clearly put into the PopVision graph and system analysers. It's hard to imagine getting such a helpful and comprehensive profiling of the code elsewhere, so this was really a standout feature in our IPU experience."

Dominique Beaini, Valence Discovery, a leader in AI-first drug design

POPVISION TOOLS

IPU MEMORY ANALYSIS

Capture memory information from your ML models when executed on IPUs. Inspect variable placement, size and liveness throughout the execution.

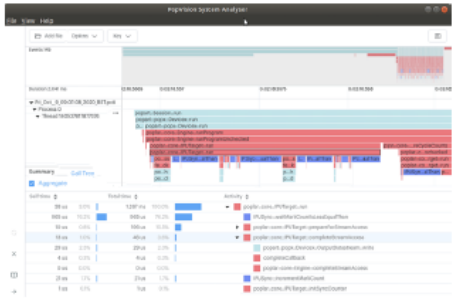


EXECUTION TRACE REPORT

View the output of instrumenting a Poplar program, capturing cycle counts for each step. See execution statistics, tile balance, cycle proportions and compute-set details.

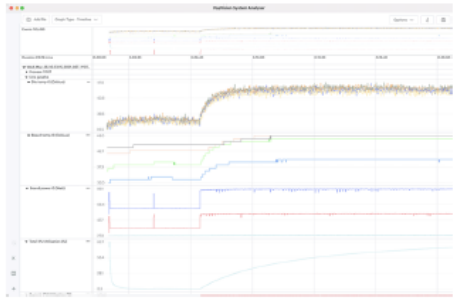
REPORT COMPARISONS

Open two reports at once to compare their memory, execution, liveness and operations. Visualise where efficiencies can be made with different model parameters.



HOST EXECUTION ANALYSIS

Understand the execution of IPU-targeted software on your host system processors. Identify any bottlenecks between CPUs and IPUs across a visual interactive timeline.



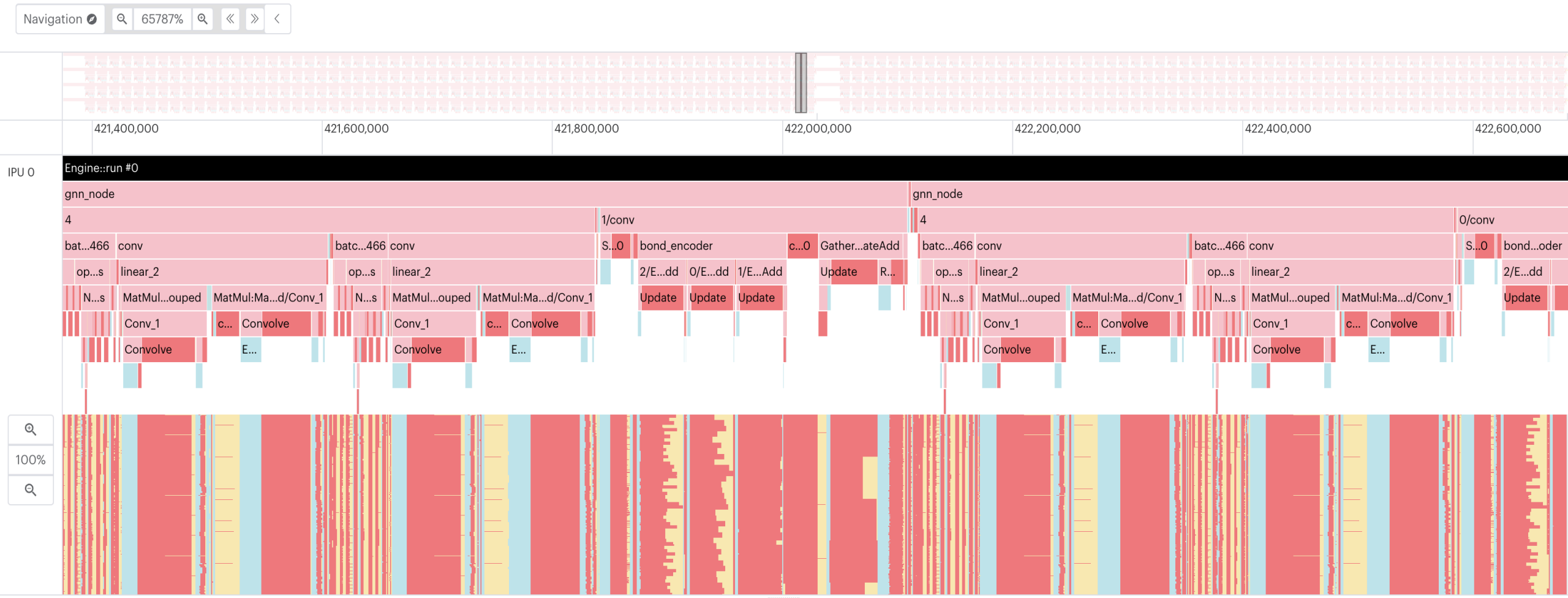
GRAPH DATA

Plot graph data of any numerical data points from the host or IPU processor systems, such as board temperature, power consumption and IPU utilisation.

LOCAL + REMOTE REPORTS

Ability to open reports either on your local machine, or remotely on the host machine. The Graph Analyser also supports local and remote report access.

POPVISION PERFORMANCE ANALYSER



Utilization/memory map of every tile/every IPU

STANDARD ML FRAMEWORK SUPPORT

Develop models using standard high-level frameworks or port existing models

PyTorch

TensorFlow

PyTorch Lightning

HUGGING FACE

PyG

Keras

PaddlePaddle

HALO

Existing models on alternative platforms

Easy port of high-level framework models

POPLAR®

IPU-Processor Platforms

gpu_cnn_keras.py ↔ ipu_cnn_keras.py tf_keras

Keras

GPU

```
as tf
ras.layers import *
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
ds_train = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(64, drop_remainde

model = tf.keras.Sequential([
    Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]),
    Activation('relu'),
    Conv2D(32, (3, 3)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), padding='same'),
    Activation('relu'),
    Conv2D(32, (3, 3)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(512),
    Activation('relu'),
    Dropout(0.5),
    Dense(10),
    Activation('softmax')
    ])

model.compile(loss='categorical_crossentropy',
    optimizer=tf.optimizers.SGD(learning_rate=0.016),
    metrics=['accuracy'])

model.fit(ds_train, epochs=40)
```

IPU

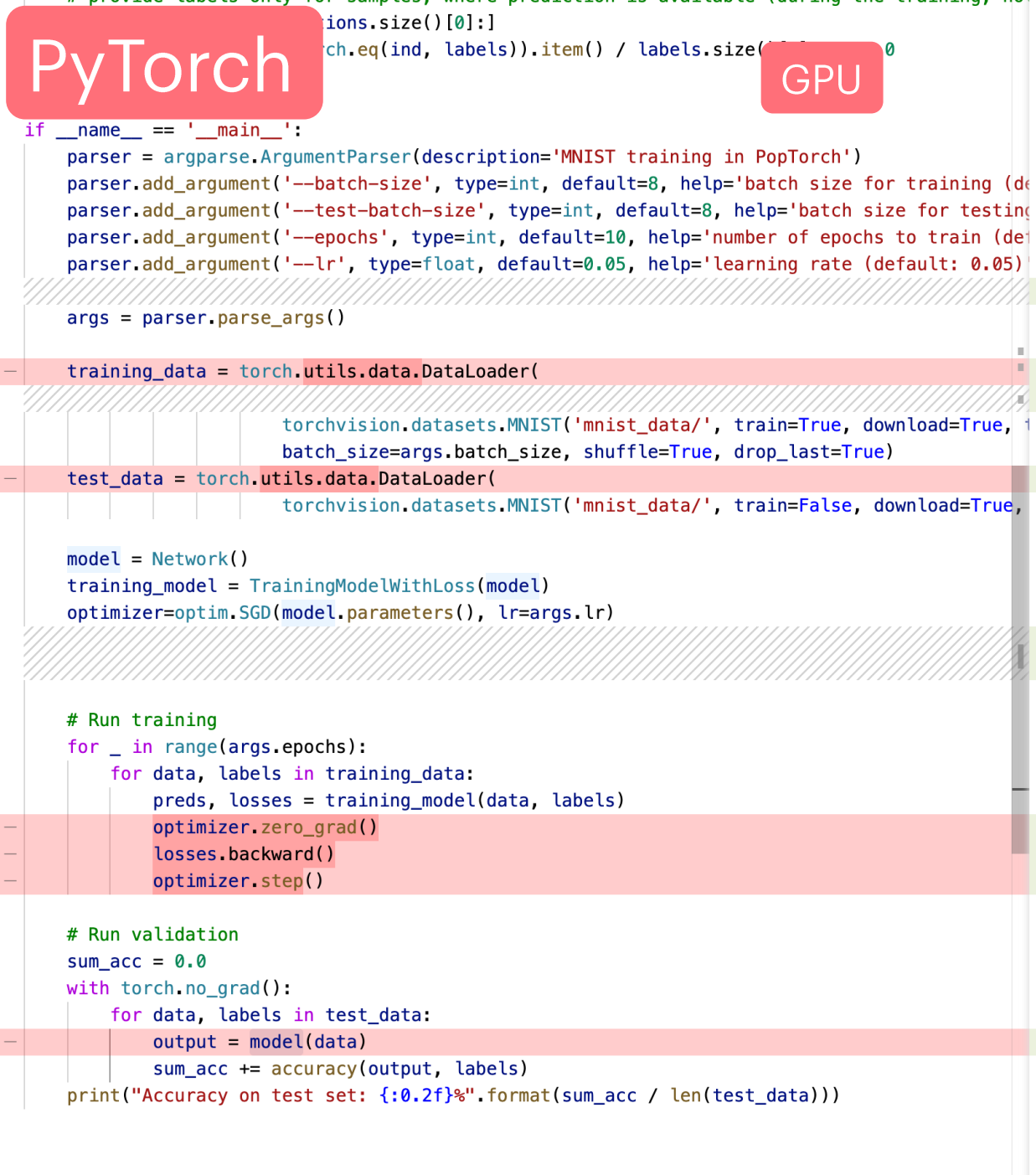
```
import tensorflow as tf
from tensorflow.keras.layers import *
+ from tensorflow.python import ipu

+ cfg = ipu.config.IPUConfig()
+ cfg.auto_select_ipus = 1
+ cfg.configure_ipu_system()
+ with ipu.ipu_strategy.IPUStrategy().scope():
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
    x_train = x_train.astype('float32') / 255.0
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    ds_train = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(64, drop_rema

    model = tf.keras.Sequential([
        Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]),
        Activation('relu'),
        Conv2D(32, (3, 3)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Conv2D(64, (3, 3), padding='same'),
        Activation('relu'),
        Conv2D(32, (3, 3)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Flatten(),
        Dense(512),
        Activation('relu'),
        Dropout(0.5),
        Dense(10),
        Activation('softmax')
        ])

    model.compile(loss='categorical_crossentropy',
        optimizer=tf.optimizers.SGD(learning_rate=0.016),
        metrics=['accuracy'])

    model.fit(ds_train, epochs=40)
```


PyTorch
GPU
if __name__ == '__main__':
parser = argparse.ArgumentParser(description='MNIST training in PopTorch')
parser.add_argument('--batch-size', type=int, default=8, help='batch size for training (de
parser.add_argument('--test-batch-size', type=int, default=8, help='batch size for testing
parser.add_argument('--epochs', type=int, default=10, help='number of epochs to train (def
parser.add_argument('--lr', type=float, default=0.05, help='learning rate (default: 0.05)
args = parser.parse_args()
training_data = torch.utils.data.DataLoader(
torchvision.datasets.MNIST('mnist_data/', train=True, download=True,
batch_size=args.batch_size, shuffle=True, drop_last=True)
test_data = torch.utils.data.DataLoader(
torchvision.datasets.MNIST('mnist_data/', train=False, download=True,
model = Network()
training_model = TrainingModelWithLoss(model)
optimizer=optim.SGD(model.parameters(), lr=args.lr)
# Run training
for _ in range(args.epochs):
for data, labels in training_data:
preds, losses = training_model(data, labels)
optimizer.zero_grad()
losses.backward()
optimizer.step()
# Run validation
sum_acc = 0.0
with torch.no_grad():
for data, labels in test_data:
output = model(data)
sum_acc += accuracy(output, labels)
print("Accuracy on test set: {:0.2f}%".format(sum_acc / len(test_data)))


IPU
if __name__ == '__main__':
parser = argparse.ArgumentParser(description='MNIST training in PopTorch')
parser.add_argument('--batch-size', type=int, default=8, help='batch size for training (de
parser.add_argument('--test-batch-size', type=int, default=8, help='batch size for testing
parser.add_argument('--epochs', type=int, default=10, help='number of epochs to train (def
parser.add_argument('--lr', type=float, default=0.05, help='learning rate (default: 0.05)
parser.add_argument('--device-iterations', type=int, default=50, help='device iterations
args = parser.parse_args()
opts = poptorch.Options().deviceIterations(args.device_iterations)
training_data = poptorch.DataLoader(opts,
torchvision.datasets.MNIST('mnist_data/', train=True, download=True,
batch_size=args.batch_size, shuffle=True, drop_last=True)
test_data = poptorch.DataLoader(opts,
torchvision.datasets.MNIST('mnist_data/', train=False, download=True,
model = Network()
training_model = TrainingModelWithLoss(model)
optimizer=optim.SGD(model.parameters(), lr=args.lr)
training_model = poptorch.trainingModel(training_model, opts, optimizer=optimizer)
inference_model = poptorch.inferenceModel(model)
# Run training
for _ in range(args.epochs):
for data, labels in training_data:
preds, losses = training_model(data, labels)
# Detach the training model so that the same IPU could be used for validation
training_model.detachFromDevice()
# Run validation
sum_acc = 0.0
with torch.no_grad():
for data, labels in test_data:
output = inference_model(data)
sum_acc += accuracy(output, labels)
print("Accuracy on test set: {:0.2f}%".format(sum_acc / len(test_data)))

PyG

PyG is the ultimate library for Graph Neural Networks

Build graph learning pipelines with ease

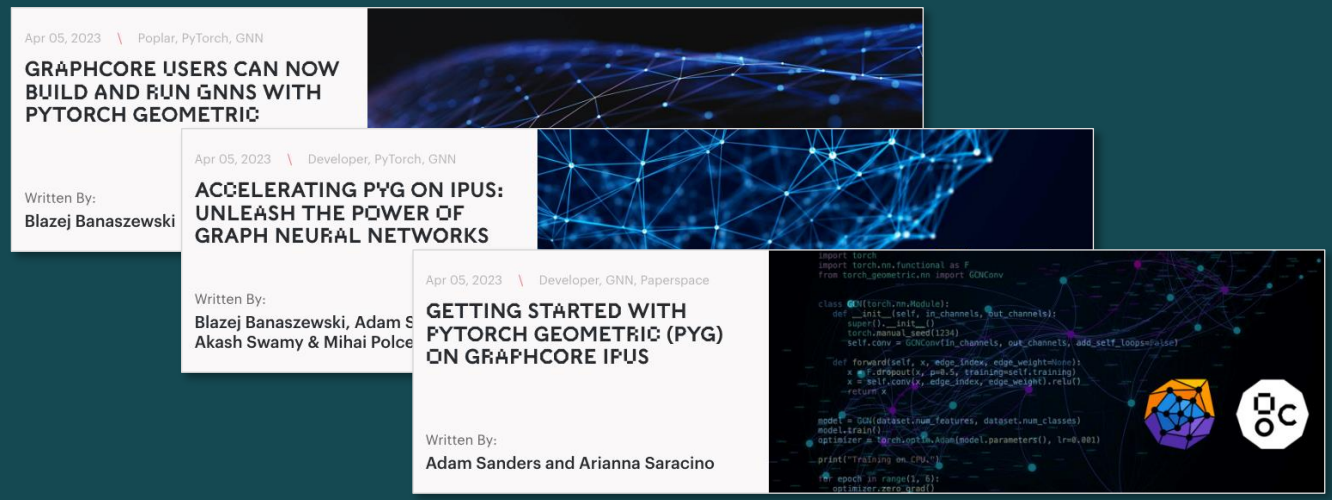
kumo GRAPHCORE HARVARD MEDICAL SCHOOL

pyg.org

"The suitability of IPUs for running GNNs and the kind of performance advantage that Graphcore and its customers have demonstrated is really helping to accelerate the uptake of this exciting model class"
Matthias Fey – PyG creator & founder of Kumo.ai

PYTORCH GEOMETRIC FOR IPU

ANNOUNCEMENT | TECHNICAL BLOG | GETTING STARTED



RUN GNN MODELS IN PYG ON PAPERSPACE JUPYTER NOTEBOOKS

Training Dynamic Graphs
TGN
Training
Run on Gradient

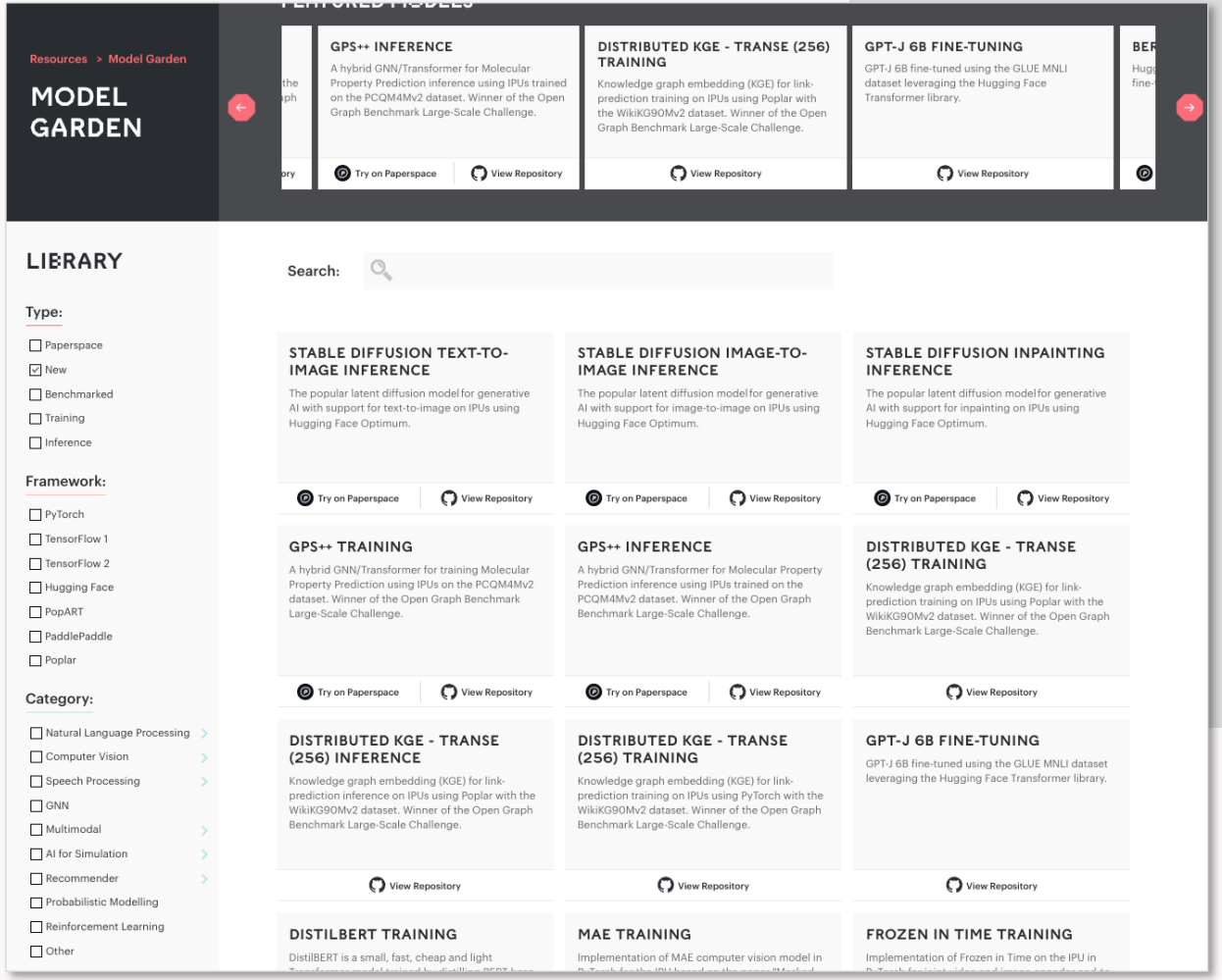
Training Large Graphs
Cluster-GCN
Training
Run on Gradient

Predicting molecular properties
SchNet
Run on Gradient

Predicting molecular properties
GIN
Run on Gradient

Link Prediction training
NBFNet
Training
Run on Gradient

ENHANCED MODEL GARDEN



PUBLIC ACCESS TO WIDE VARIETY OF MODELS, READY TO RUN ON IPU

NEW FILTER/SEARCH CAPABILITY

DIRECT ACCESS TO GITHUB

PAPERSPACE NOTEBOOK LINKS

https://www.graphcore.ai/resources/model-garden

MODEL GARDEN COVERAGE

COMPUTER VISION

IMAGE CLASSIFICATION

- ResNet50 v1.5
- EfficientNet-B0
- EfficientNet-B4
- ResNeXt-101
- MobileNet v2
- MobileNet v3
- ViT
- DINO
- SWIN NEW
- MAE

OBJECT DETECTION

- YOLO v3
- YOLO v4
- Faster RCNN
- EfficientDet

OBJECT SEGMENTATION

- Unet (Industrial)
- Unet (Medical)

GNN

- TGN NEW
- MPNN-GIN NEW
- GPS++ NEW
- Distr. KGE NEW
- Cluster-GCN NEW
- SchNet NEW
- NBFNet NEW

AI FOR SIMULATION

- DeepMD
- DeepDriveMD
- ETO
- CosmoFlow
- ABC Covid-19

REINFORCEMENT

- RL
  Reinforcement Learning

PROBABILISTIC

- MCMC
- VAE

NLP

- BERT-Base
- BERT-Large
- Group | Packing
- GPT2
- GPT-J NEW
- RoBERTa
- Deberta
- BART
- T5
- Hubert
- DistilBERT NEW

SPEECH

STT (ASR)

- RNN-T
- Conformer

TTS

- DeepVoice3
- FastSpeech2
- FastPitch

RECOMMENDER

- Autoencoder
- DIN
- DIEN

MULTIMODAL

- LXMERT
- CLIP
- Stable Diffusion NEW
- Mini DALL-E
- Frozen In Time NEW

OTHER

- Sales Forecast
- Neural Image Fields

PyG

PyTorch

TensorFlow

Hugging Face

Keras

PaddlePaddle

POPART

THANK YOU

Chad Martin
chadm@graphcore.ai