

GRAPHCORE HARDWARE AND SOFTWARE



GRAPHCORE



Graphcore Confidential

GRAPHCORE'S SOLUTION

Hardware



IPU processor
designed for AI

Software



Poplar SDK and
development tools

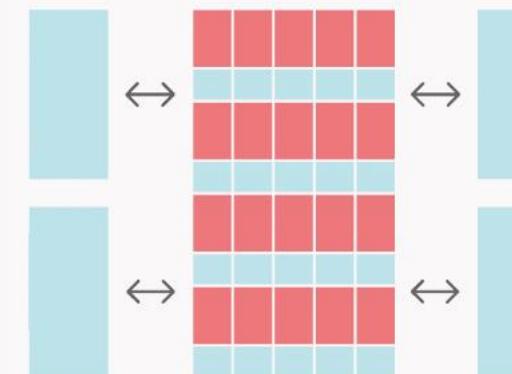
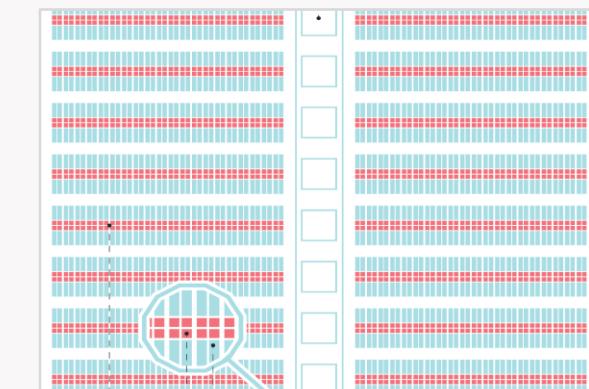
Platform



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
Processor Memory			
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)

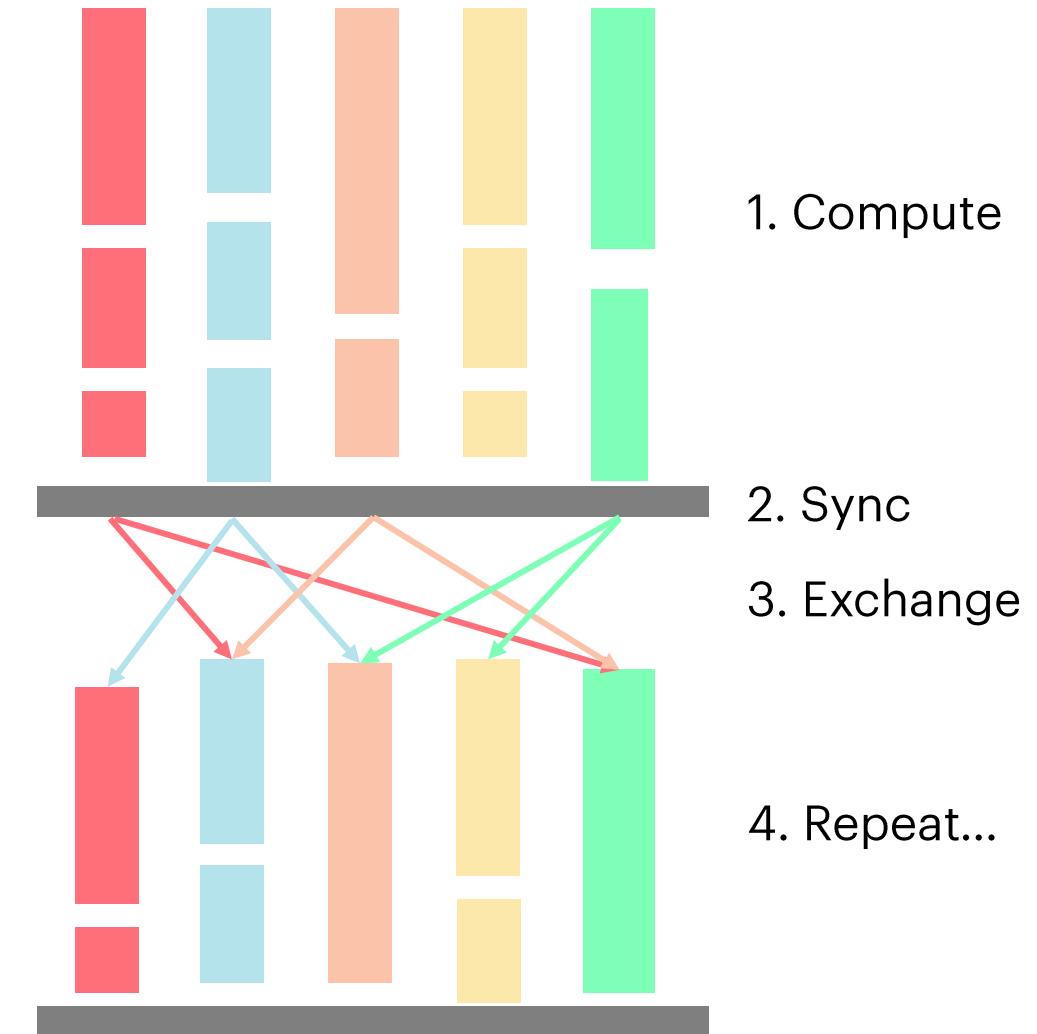
BRIDGING SOFTWARE TO PARALLEL HARDWARE

Bulk Synchronous Parallel (BSP) is the computer science solution that bridges software to parallel processors.

- IPU implements BSP in hardware.

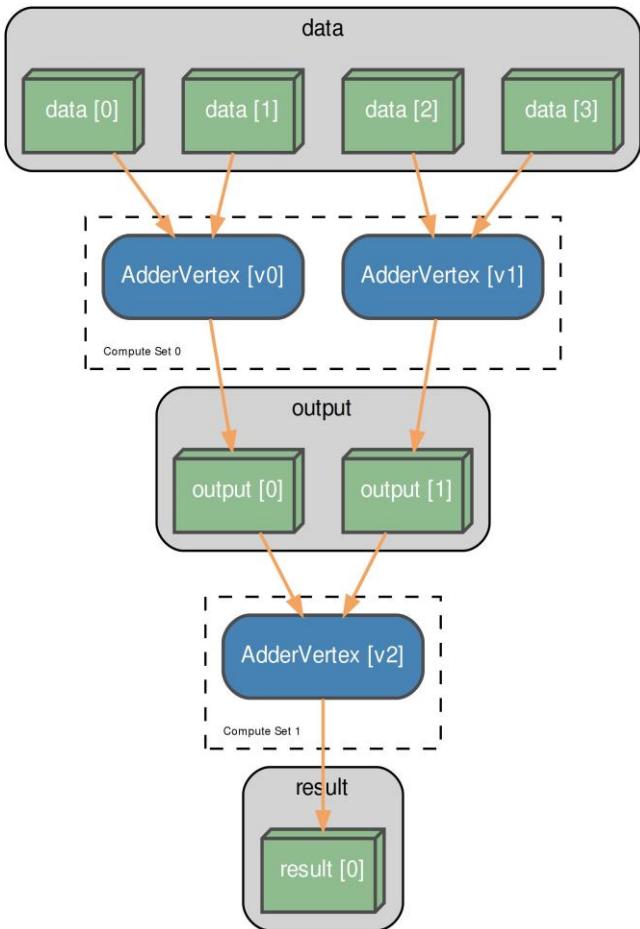
BSP solves parallel software issues:

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

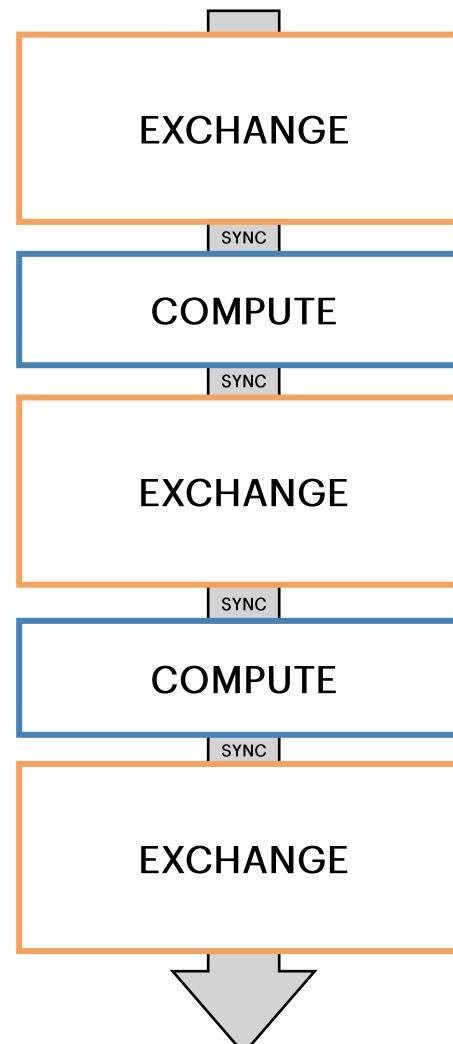


EXECUTION MODEL

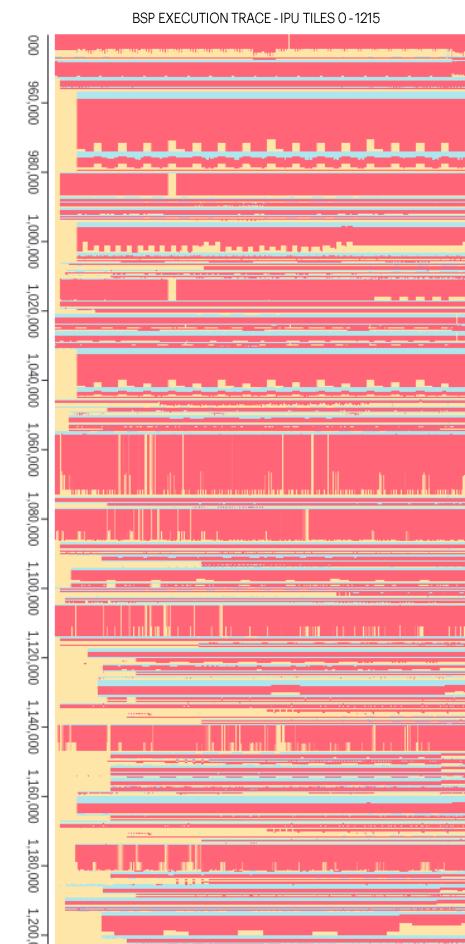
COMPUTATIONAL GRAPH



BSP SCHEDULE



OPTIMIZED IPU EXECUTION



OUTPUT FROM POPVISION GRAPH ANALYSER

GRAPHCORE

PRODUCT DETAILS



GRAPHCORE



Graphcore Confidential

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

BOW IPU PROCESSOR

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

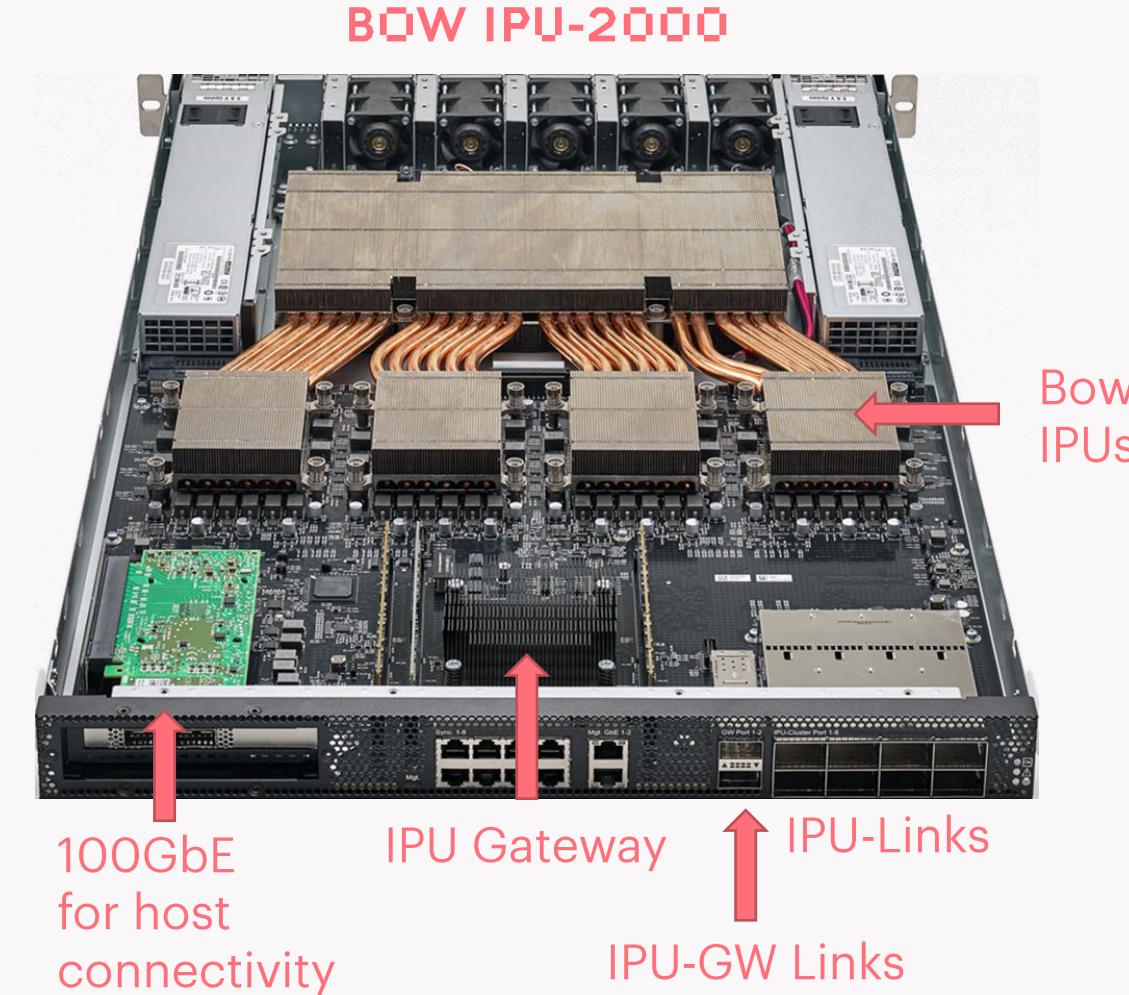
IU blade form factor delivering 1.4 PetaFLOPS AI Compute

Disaggregated AI/ML accelerator platform

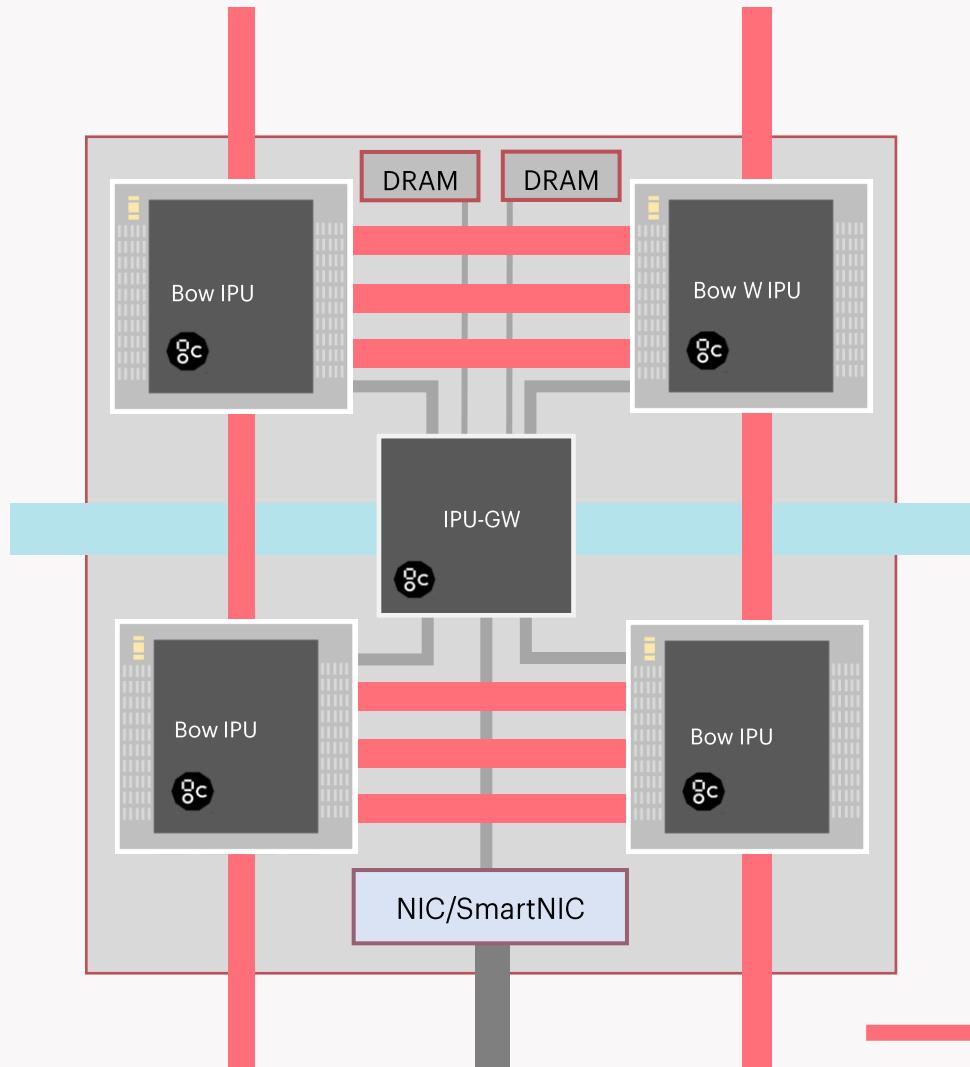
Excellent performance & TCO leveraging
In-Processor memory & IPU-Exchange

IPU-Links scale to Bow Pod64

Expansion to Bow Pod256 and beyond
with IPU-GW Links



BOW-2000: THE BUILDING BLOCK OF LARGE PODS



4x Bow IPUs

- 1.4 PFLOP₁₆ compute
- 5,888 processor cores
- > 35,000 independent parallel threads



Exchange Memory

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



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)

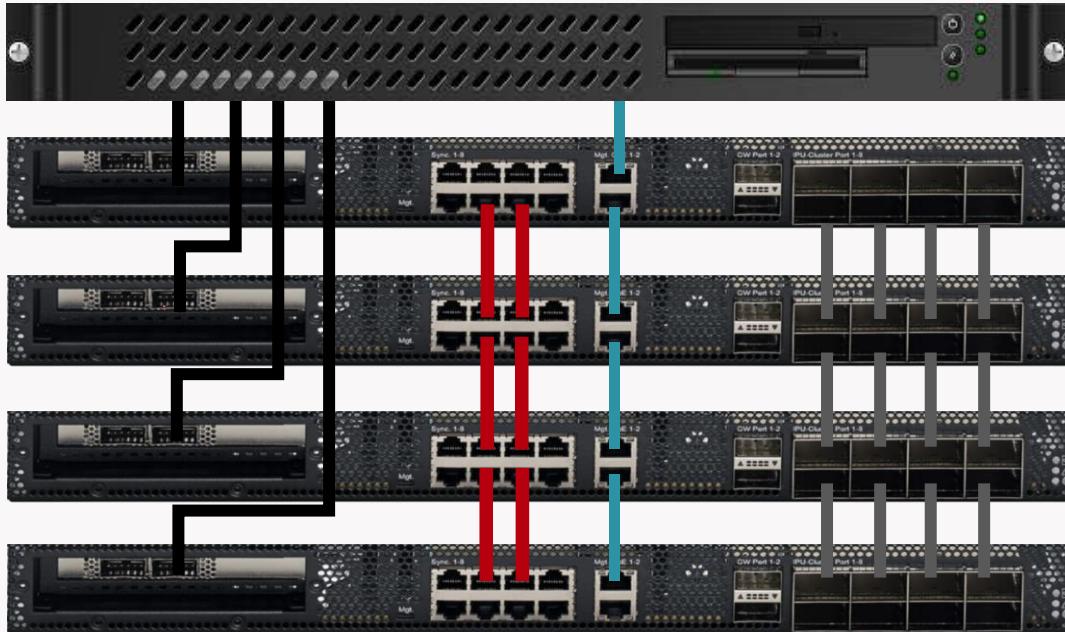


BOW POD₂₅₆

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

BOW POD16 DIRECT ATTACH

Server with x4 Bow-2000



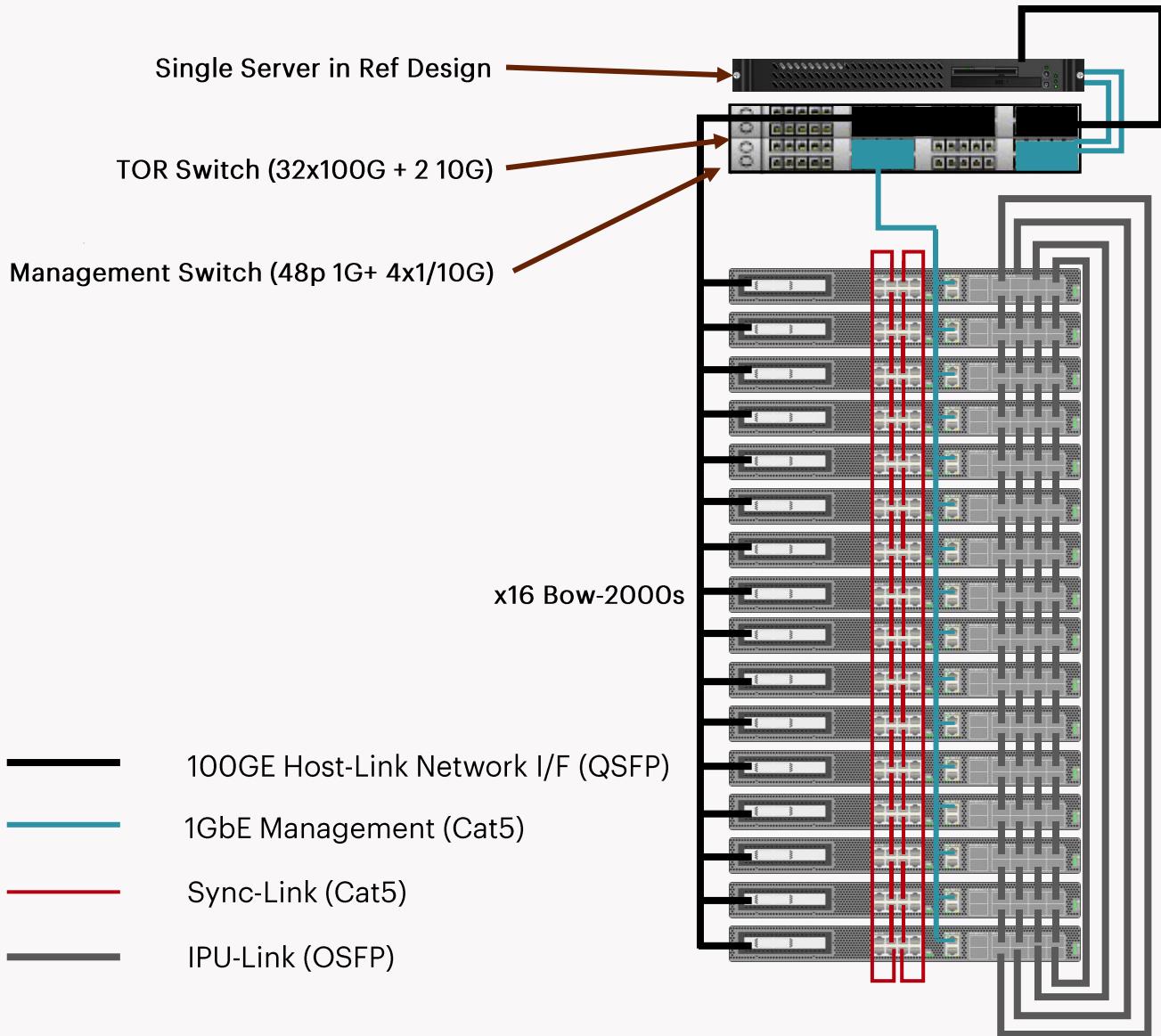
- Host-Link 100GE network interface (QSFP, 1.0m)
- 1GbE Management (Cat5, 1.5m)
- Sync-Link (Cat5, 0.15m)
- IPU-Link (OSFP, 0.3m)

- Convenient cost effective evaluation platform
- Available through Graphcore channel for on-premise or Graphcloud
- Wide range of benchmarks and examples for Bow Pod₁₆ performance evaluation
- Scale-out with Bow Pod₆₄ and beyond

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 Pod₆₄ 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



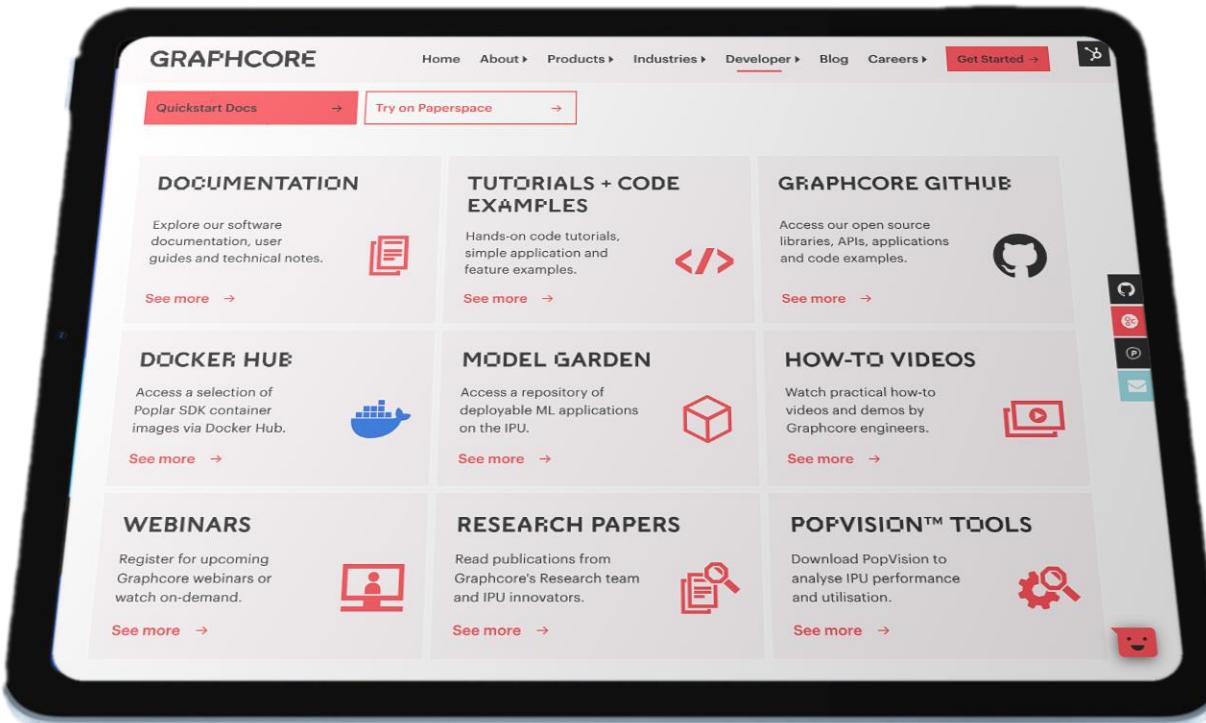
GRAPHCORE



Graphcore Confidential

GRAPHCORE SOFTWARE ECOSYSTEM

WORLD CLASS DEVELOPER RESOURCES FOR IPU USERS



WWW.GRAPHCORE.AI/DEVELOPER

A screenshot of the Graphcore Documents website, viewed on a mobile device. The top navigation bar shows 'GRAPHCORE' and 'Graphcore Documents Version: Latest'. Below this is a search bar labeled 'Search docs'. The main content area lists various document categories: Getting Started, Software Documents, Hardware Documents, Technical Notes and White Papers, Examples and Tutorials, Document Updates, and Alphabetical List of All Documents. At the bottom is a section for 'Graphcore License Agreements'.



GRAPHCORE DOCUMENTS

Getting Started

Background information and quick-start guides for Graphcloud and Pod systems

Software Documents

Documentation for the Poplar SDK and other software

Hardware Documents

Documentation for installing and using IPU-Machines and Pod systems

Technical Notes and White Papers

Technical notes and white papers on Graphcore technology

Document Updates

The latest news about new documents and examples

Examples and Tutorials

Tutorials and application examples for running on the IPU

Getting started with PyTorch for the IPU

Running a basic model for training and inference

AI Customer Engineer, Chris Bogdikiewicz introduces PyTorch for the IPU. With PopTorch™ - a simple Python wrapper for PyTorch programs, developers can easily run models, directly on Graphcore IPUs with a few lines of extra code.

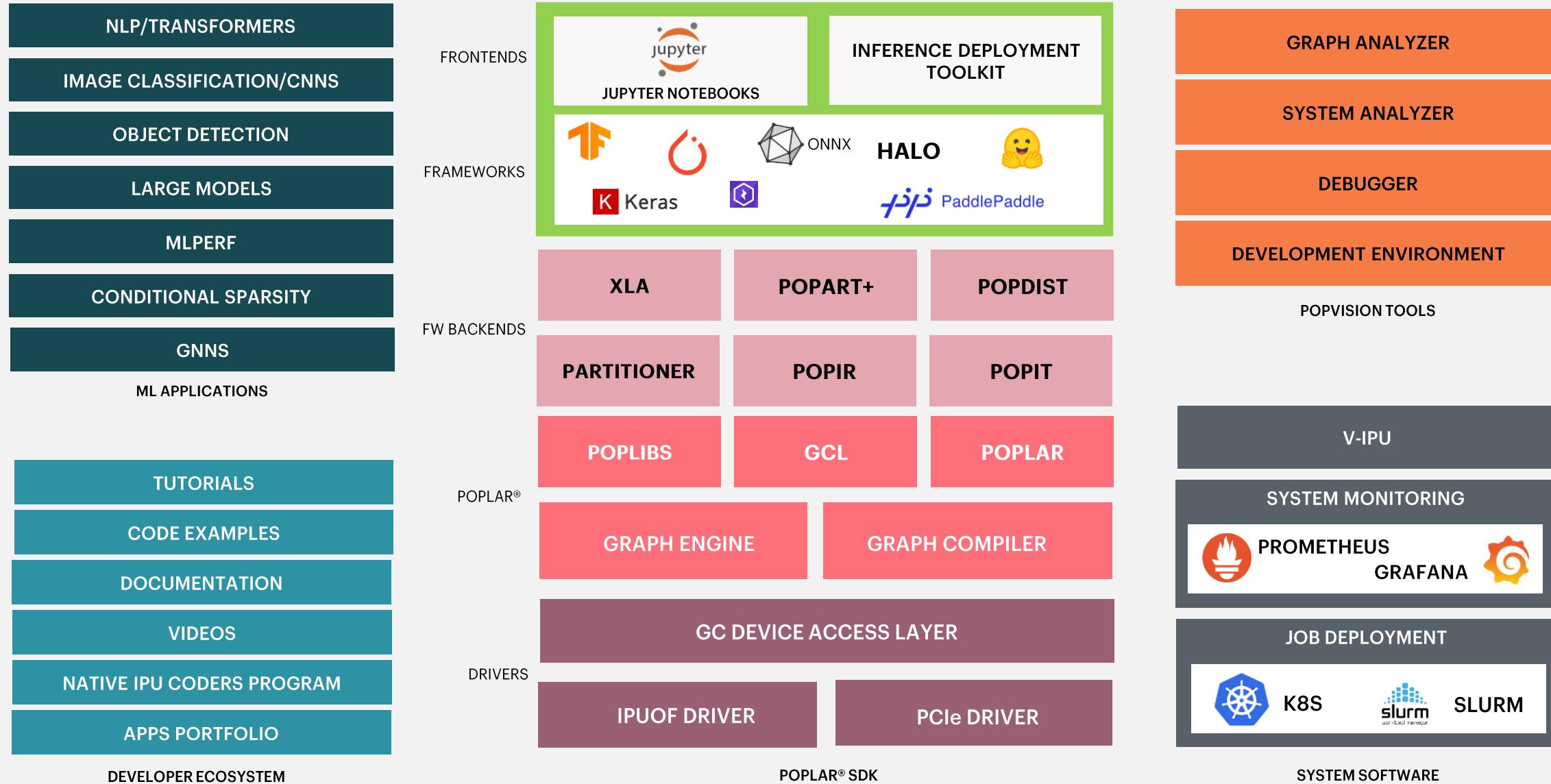
In this video, Chris provides a quick demo on running a basic model for both training and inference using a MNIST based example.

[Get the Code](#)

[Read the Guide](#)



GRAPHCORE SOFTWARE MATURITY



POPLIBS GRAPH LIBRARIES

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

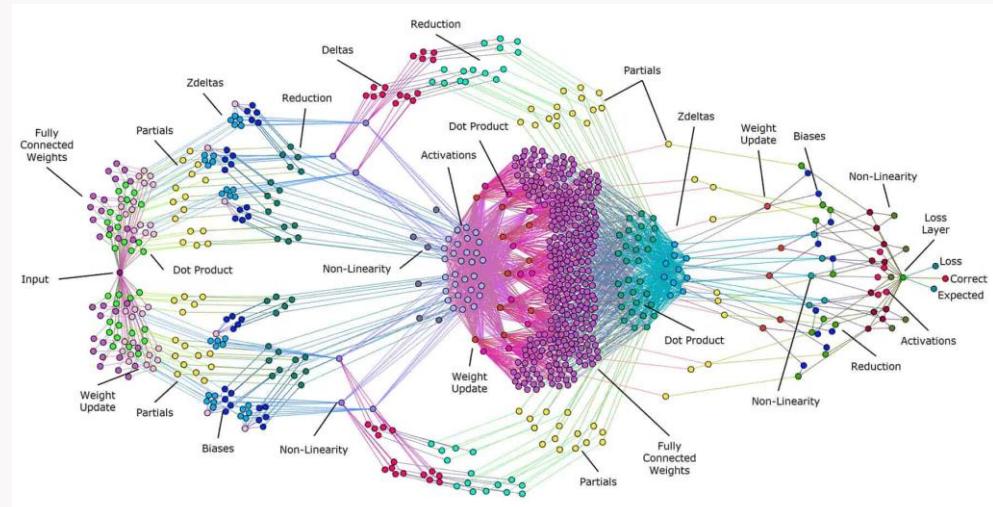
- Over 50 optimized functions for common machine learning models
- More than 750 high performance compute elements
- Simple C++ graph building API
- Implement any application
- Full control flow support

POPNN	FUNCTIONS USED IN NEURAL NETWORKS (NON-LINEARITIES, POOLING, LOSS FUNCTIONS)
POPLIN	OPTIMIZED LINEAR ALGEBRA FUNCTIONS (MATRIX MULTIPLICATION, CONVOLUTIONS)
POPOPS	FUNCTIONS FOR PERFORMING ELEMENTWISE OPERATIONS ON TENSOR DATA
POPRAND	HIGH PERFORMANCE FUNCTIONS FOR POPULATING TENSORS WITH RANDOM NUMBERS
POPUTIL	GENERAL UTILITY FUNCTIONS FOR BUILDING GRAPHS FOR IPU DEVICES
GCL	OPTIMIZED COLLECTIVES LIBRARY SUPPORTING MODEL AND DATA PARALLEL

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

POPLAR™ POPVISION TOOLS

GRAPH ANALYSER

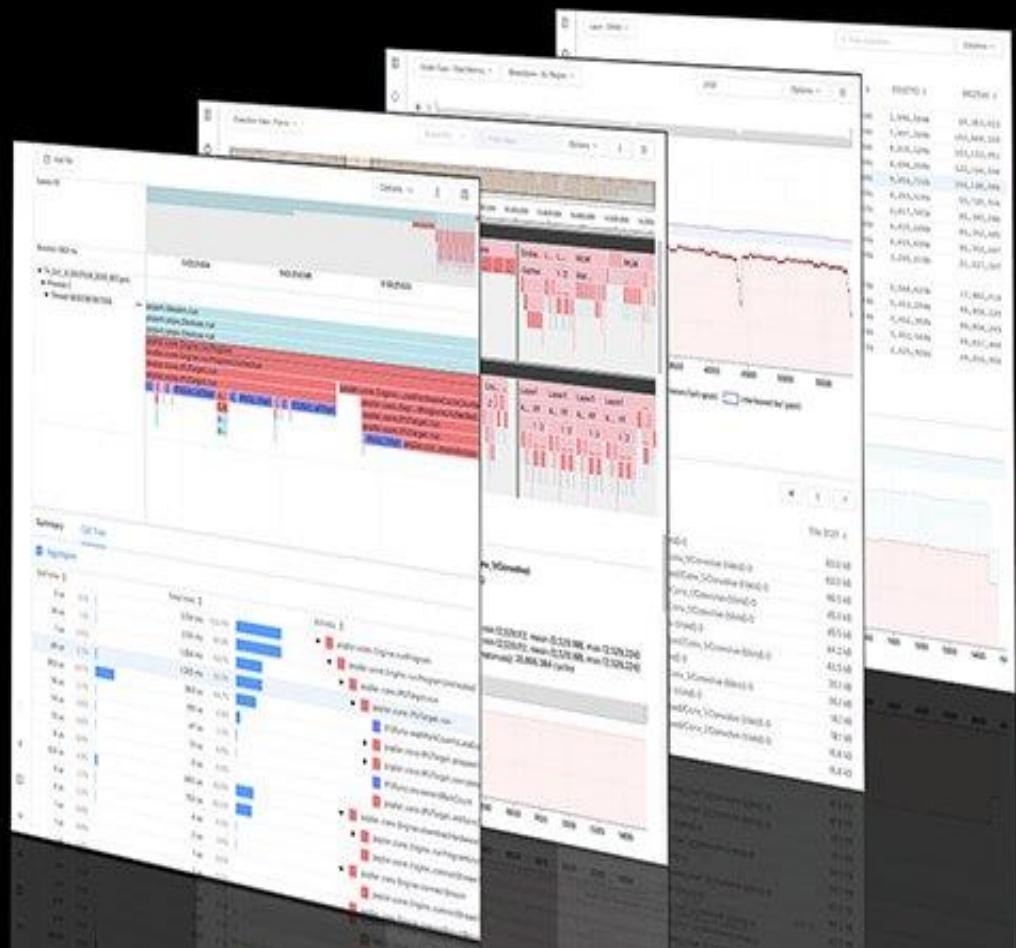
Useful for analysing and optimising the memory use and execution performance of ML models on the IPU

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.



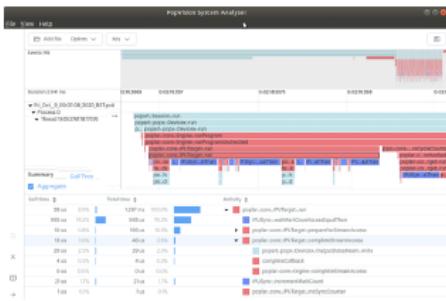
EXECUTION TRACE REPORT

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.



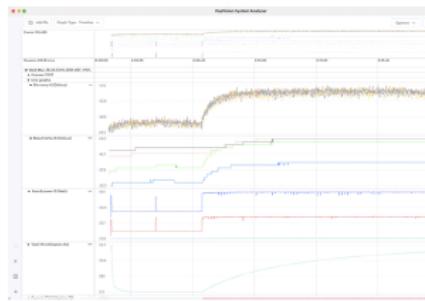
REPORT COMPARISONS

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.



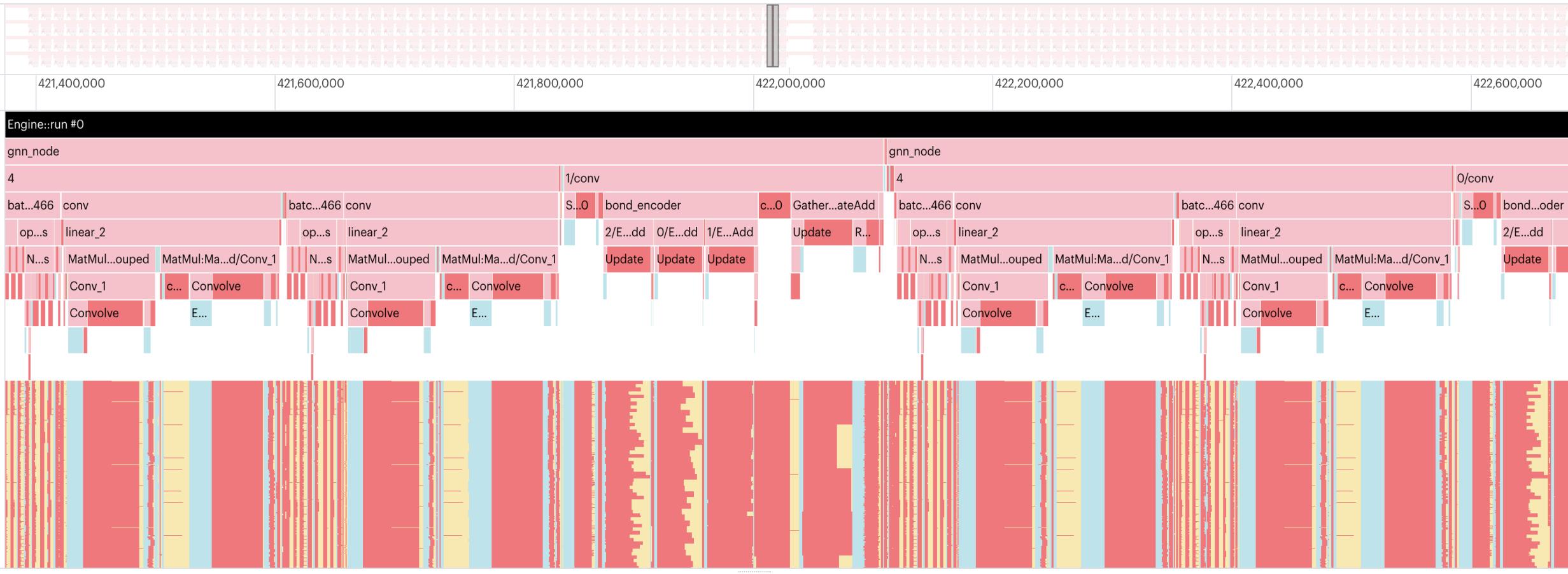
LOCAL + REMOTE REPORTS

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

Navigation 🔍 65787% 🔍 ⟲ ⟳ <



Utilization/memory map of every tile/every IPU



GRAPHCORE CONFIDENTIAL

STANDARD ML FRAMEWORK SUPPORT

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



HALO



Easy port of
high-level
framework
models



IPU-
Processor
Platforms



Keras

GPU

```
as tf
keras.layers import *
keras.utils import to_categorical
tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(64, drop_remainder=True)

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

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)
```

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 = to_categorical(y_train, 10)
 ds_train = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(64, drop_remainder=True)

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

```
_, ind = torch.max(predictions, 1)
# provide labels only for samples, where prediction is available (during the training, not
# predictions.size()[0]:]
accuracy = torch.sum(torch.eq(ind, labels)).item() / labels.size(0)

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 (de
    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 (d

args = parser.parse_args()

training_data = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('mnist_data/', train=True, download=True, tra
    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,
    transform=torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize((0.1307,), (0.3089,))
    ]))
)
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: {:.2f}%".format(sum_acc / len(test_data)))
```

IPU

```
_, ind = torch.max(predictions, 1)
# provide labels only for samples, where prediction is available (during the training, not
# labels = labels[-predictions.size()[0]:]
accuracy = torch.sum(torch.eq(ind, labels)).item() / labels.size(0)
return accuracy

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 (de
    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 (d

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, tra
    batch_size=args.batch_size, shuffle=True, drop_last=True)
)
test_data = poptorch.DataLoader(opts,
    torchvision.datasets.MNIST('mnist_data/', train=False, download=True, tra
    transform=torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize((0.1307,), (0.3089,))
    ]))
)
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: {:.2f}%".format(sum_acc / len(test_data)))
```



PyG

PyG is the ultimate library for Graph Neural Networks

Build graph learning pipelines with ease



GRAPHCORE



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

Apr 05, 2023 | Poplar, PyTorch, GNN
GRAPHCORE USERS CAN NOW BUILD AND RUN GNNs WITH PYTORCH GEOMETRIC

Written By: Blazej Banaszewski

Apr 05, 2023 | Developer, PyTorch, GNN
ACCELERATING PYG ON IPUS: UNLEASH THE POWER OF GRAPH NEURAL NETWORKS

Written By: Blazej Banaszewski, Adam S, Akash Swamy & Mihai Polc

Apr 05, 2023 | Developer, GNN, Paperspace
GETTING STARTED WITH PYTORCH GEOMETRIC (PYG) ON GRAPHCORE IPUS

Written By: Adam Sanders and Arianna Saracino

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GINConv
class GIN(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        torch.manual_seed(1234)
        self.conv = GINConv(in_channels, out_channels, add_self_loops=True)

    def forward(self, x, edge_index, edge_weight=None):
        x = self.conv(x, edge_index, edge_weight=edge_weight, training=True)
        x = F.relu(x)
        return x
model = GIN(dataset.num_features, dataset.num_classes)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
print("Training on CPU...")
for epoch in range(100):
    optimizer.zero_grad()
    loss =
```

RUN GNN MODELS IN PYG ON PAPERSPACE JUPYTER NOTEBOOKS

Training Dynamic Graphs
TGN
Training



Training Large Graphs
Cluster-GCN
Training



Predicting molecular properties
SchNet



Predicting molecular properties
GIN



Link Prediction training
NBFNet
Training



ENHANCED MODEL GARDEN

The screenshot shows the Enhanced Model Garden interface. On the left, there's a sidebar with navigation links like 'Resources > Model Garden' and sections for 'MODEL GARDEN' (with a back arrow), 'LIBRARY' (with a search bar and filters for Type, Framework, and Category), and a detailed view of a model card.

FEATURED MODELS

- GPS++ INFERENCE**: A hybrid GNN/Transformer for Molecular Property Prediction inference using IPUs trained on the PCQM4Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- DISTRIBUTED KGE - TRANSE (256) TRAINING**: Knowledge graph embedding (KGE) for link-prediction training on IPUs using Poplar with the WikiKG90Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- GPT-J 6B FINE-TUNING**: GPT-J 6B fine-tuned using the GLUE MNLI dataset leveraging the Hugging Face Transformer library.
View Repository

LIBRARY

Search:

Type:

- Paperspace
- New
- Benchmarked
- Training
- Inference

Framework:

- PyTorch
- TensorFlow 1
- TensorFlow 2
- Hugging Face
- PopART
- PaddlePaddle
- Poplar

Category:

- Natural Language Processing >
- Computer Vision >
- Speech Processing >
- GNN
- Multimodal
- AI for Simulation
- Recommender
- Probabilistic Modelling
- Reinforcement Learning
- Other

Model Cards (partial list):

- STABLE DIFFUSION TEXT-TO-IMAGE INFERENCE**: The popular latent diffusion model for generative AI with support for text-to-image on IPUs using Hugging Face Optimum.
View Repository
- STABLE DIFFUSION IMAGE-TO-IMAGE INFERENCE**: The popular latent diffusion model for generative AI with support for image-to-image on IPUs using Hugging Face Optimum.
View Repository
- STABLE DIFFUSION INPAINTING INFERENCE**: The popular latent diffusion model for generative AI with support for inpainting on IPUs using Hugging Face Optimum.
View Repository
- GPS++ TRAINING**: A hybrid GNN/Transformer for training Molecular Property Prediction using IPUs on the PCQM4Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- GPS++ INFERENCE**: A hybrid GNN/Transformer for Molecular Property Prediction inference using IPUs trained on the PCQM4Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- DISTRIBUTED KGE - TRANSE (256) TRAINING**: Knowledge graph embedding (KGE) for link-prediction training on IPUs using Poplar with the WikiKG90Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- DISTRIBUTED KGE - TRANSE (256) INFERENCE**: Knowledge graph embedding (KGE) for link-prediction inference on IPUs using Poplar with the WikiKG90Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- DISTRIBUTED KGE - TRANSE (256) TRAINING**: Knowledge graph embedding (KGE) for link-prediction training on IPUs using PyTorch with the WikiKG90Mv2 dataset. Winner of the Open Graph Benchmark Large-Scale Challenge.
View Repository
- GPT-J 6B FINE-TUNING**: GPT-J 6B fine-tuned using the GLUE MNLI dataset leveraging the Hugging Face Transformer library.
View Repository
- DISTILBERT TRAINING**: DistilBERT is a small, fast, cheap and light
View Repository
- MAE TRAINING**: Implementation of MAE computer vision model in
View Repository
- FROZEN IN TIME TRAINING**: Implementation of Frozen in Time on the IPU in
View Repository

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

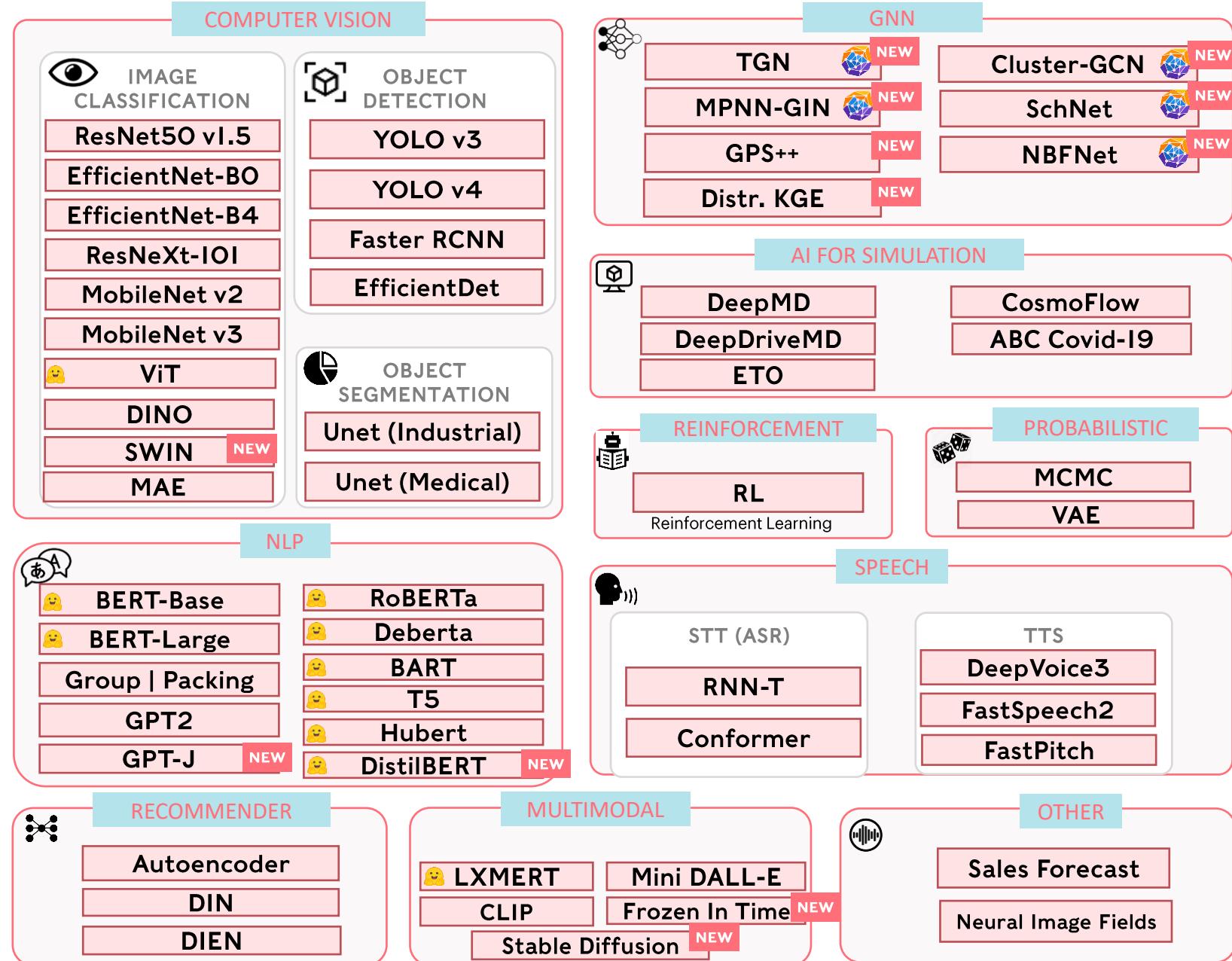
NEW FILTER/SEARCH CAPABILITY

DIRECT ACCESS TO GITHUB

PAPERSPACE NOTEBOOK LINKS



MODEL GARDEN COVERAGE



PyTorch

TensorFlow

Hugging Face

Keras

PaddlePaddle

POPART

THANK YOU

Chad Martin

chadm@graphcore.ai



Graphcore Confidential