

IPU HANDS-ON SESSION

Minkyu Kim
Field AI Engineer

GRAPHCORE



Reinforcement Learning Neural network visualization from **POPLAR™**



Day01

1. IPU HW
2. Poplar SDK
3. Basic porting guide w/ a simple model
 - Pytorch
 - Tensorflow

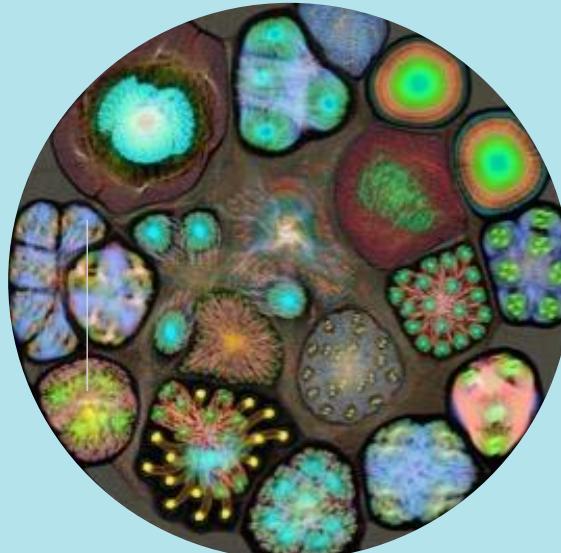
ABOUT US...

Hardware



IPU processors
designed for AI

Software



Poplar® software stack &
development tools

Platforms



M2000 and Server
IPU-POD₆₄ scale-out



The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

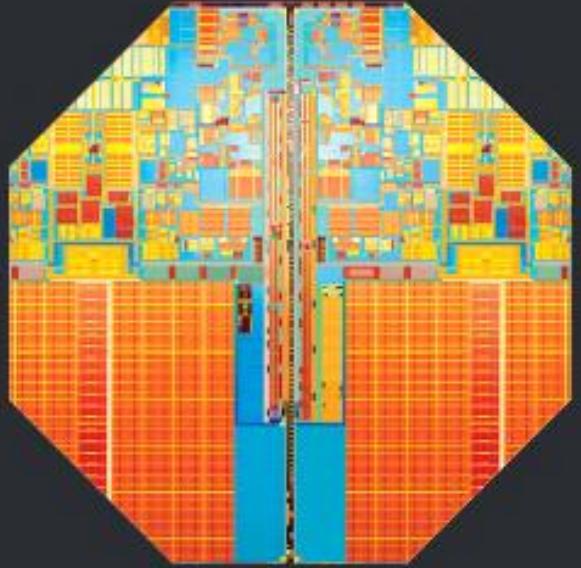
Abstract

Hardware, systems and algorithms research communities have historically had different incentive structures and fluctuating motivation to engage with each other explicitly. This historical treatment is odd given that hardware and software have frequently determined which research ideas succeed (and fail). This essay introduces the term hardware lottery to describe when a research idea wins because it is suited to the available software and hardware and *not* because the idea is superior to alternative research directions.

Examples from early computer science history illustrate how hardware lotteries can delay research progress by casting successful ideas as failures. These lessons are particularly salient given the advent of domain specialized hardware which make it increasingly costly to stray off of the beaten path of research ideas. This essay posits that the gains from progress in computing are likely to become even more uneven, with certain research directions moving into the fast-lane while progress on others is further obstructed.

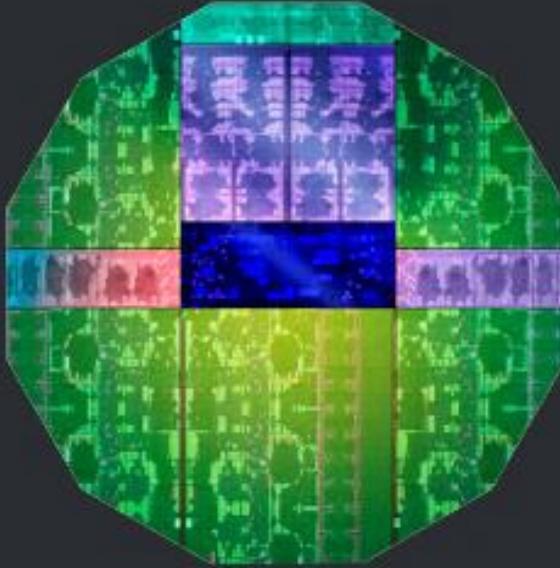
OUR IPU LETS INNOVATORS CREATE THE NEXT BREAKTHROUGHS IN MACHINE INTELLIGENCE

LEGACY PROCESSOR ARCHITECTURES HAVE BEEN REPURPOSED FOR ML



CPU

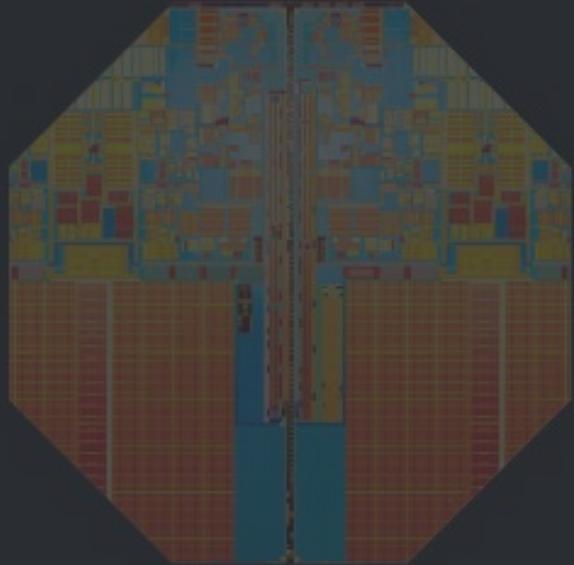
Apps and Web/
Scalar



GPU

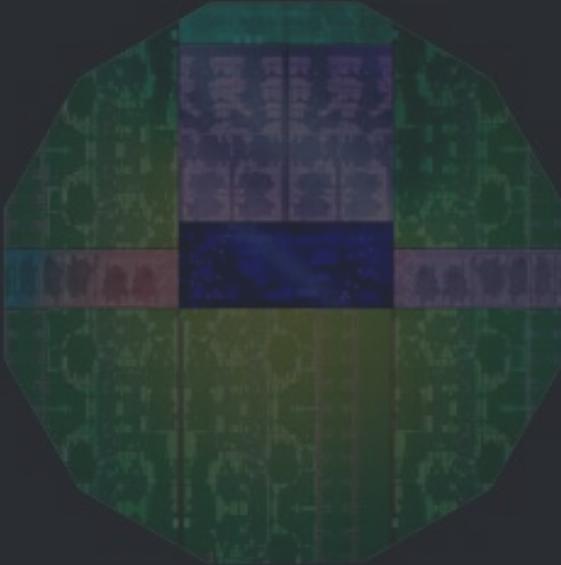
Graphics and HPC/
Vector

A NEW PROCESSOR IS REQUIRED FOR THE FUTURE



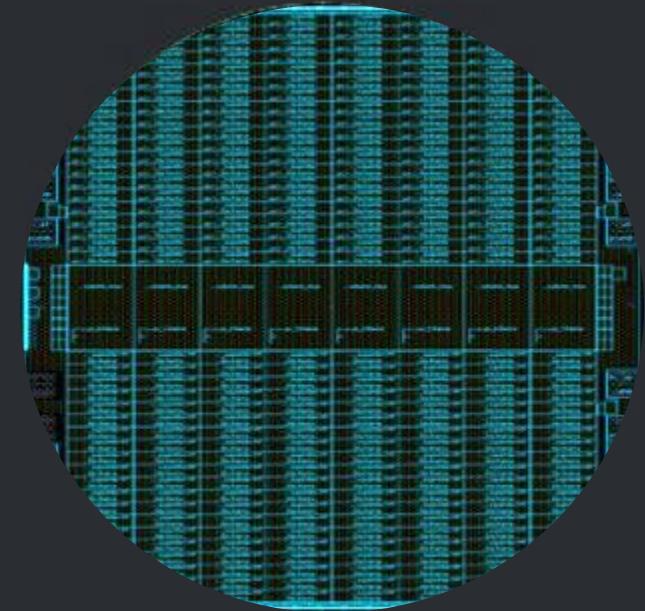
CPU

Apps and Web/
Scalar



GPU

Graphics and HPC/
Vector



IPU

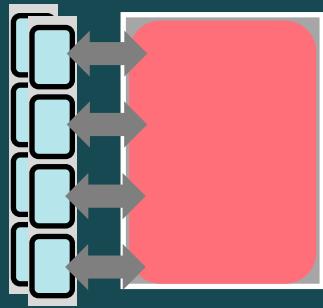
Artificial Intelligence/
Graph

MACHINE LEARNING REQUIRES 2 KINDS OF COMPUTE INNOVATION

CPU

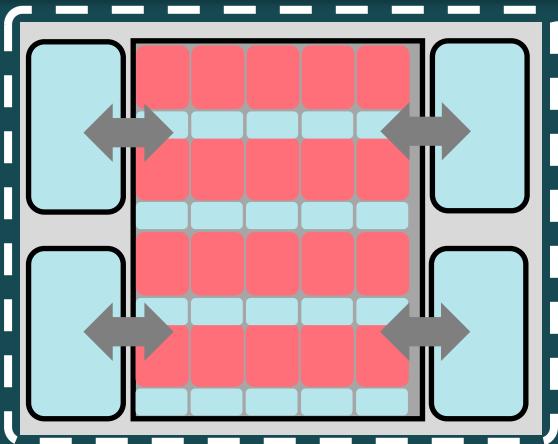
Parallelism

Suitable for scalar processes



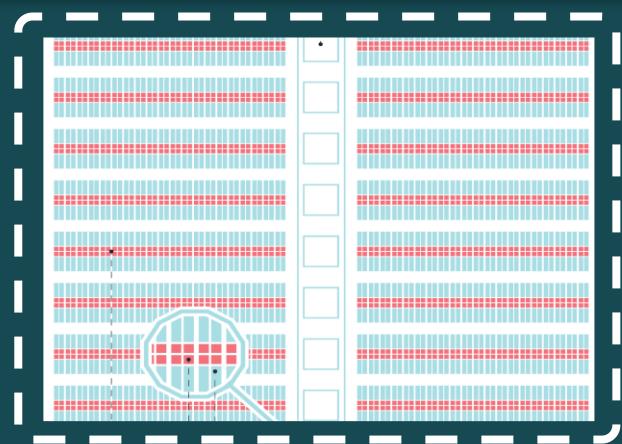
GPU

SIMD/SIMT architecture.
Suitable for large blocks of dense contiguous data



IPU

Massively parallel MIMD.
High performance/efficiency as ML trends to sparsity & small kernels



Memory Access

Off-chip memory

1x

5x – 32x

320x

Model and Data spread across off-chip and small on-chip cache and shared mem.

Model & Data in tightly coupled large locally distributed SRAM

Processor

Memory



IPU-Tiles™

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

IPU-Core™

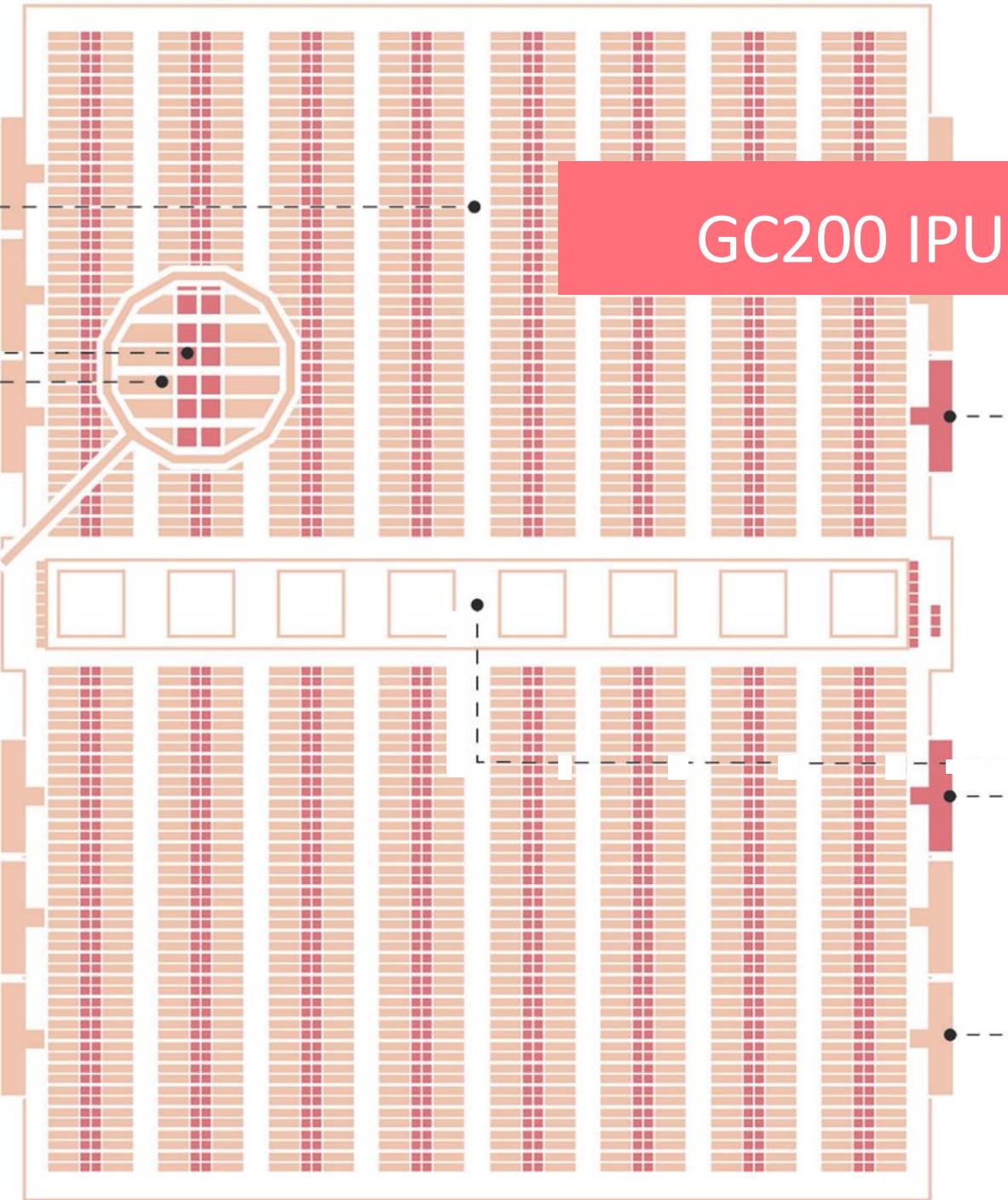
1472 independent IPU-Core™

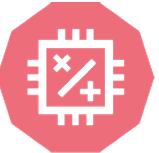
8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

47.5TB/s memory bandwidth per IPU





COLOSSUS MK2

the worlds most complex processor

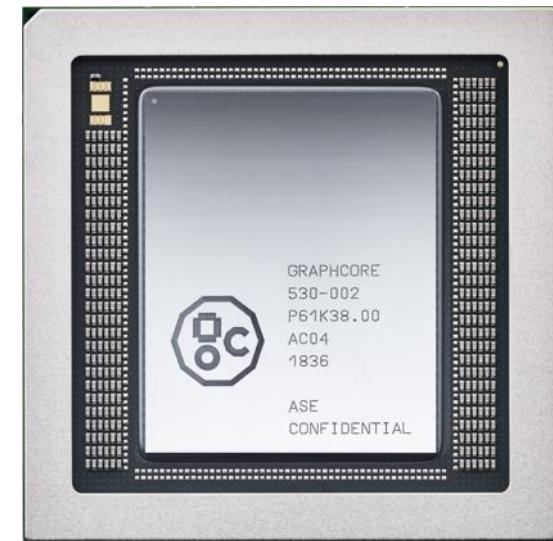
59.4Bn transistors, TSMC 7nm @ 823mm²

250TFlops AI-Float | 900MB In-Processor-Memory™

1472 independent processor cores

8832 separate parallel threads

>8x step-up in system performance vs Mk1



GC200 IPU

INTRODUCING THE BOW IPU

WORLD'S FIRST 3D WAFER-ON-WAFER PROCESSOR



3D silicon wafer stacked processor

Optimized silicon power delivery

350 TeraFLOPS AI compute

0.9 GigaByte In-Processor-Memory @ 65TB/s

1,472 independent processor cores

8,832 independent parallel programs

10x IPU-Links™ delivering 320GB/s

IPU BOW 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

Solder Bumps

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

4 x Bow 3D Wafer-on-Wafer IPUs

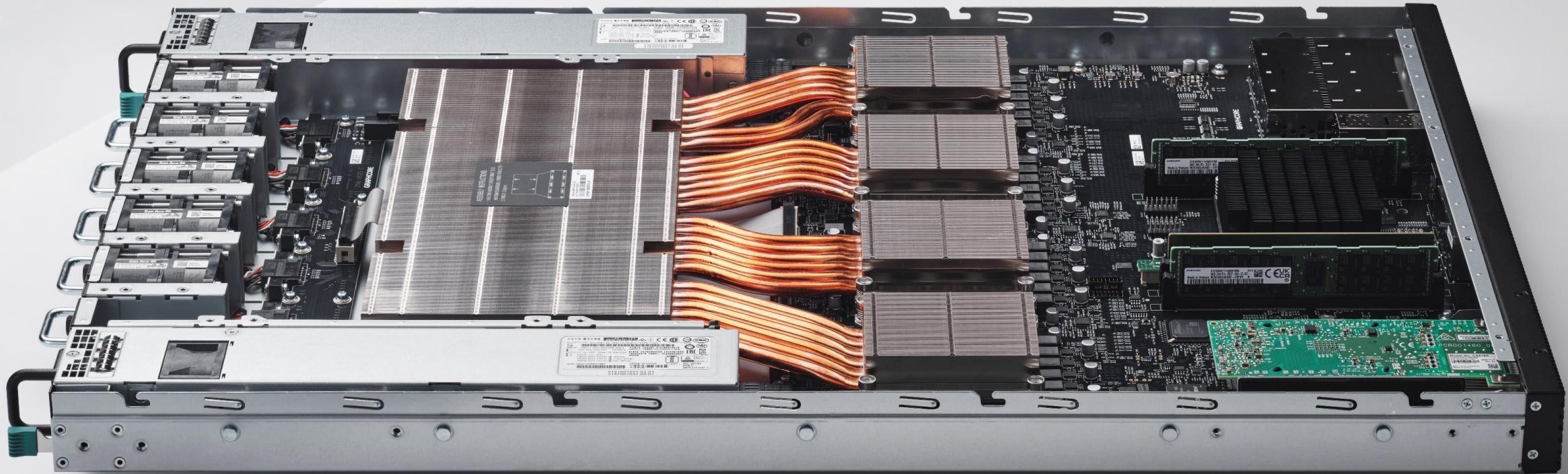
1.4 PetaFLOPS AI Compute

3.6 GB In-Processor-Memory @ 260TB/s

Up to 256 GB IPU Streaming Memory

2.8 Tbps IPU-Fabric™

Same 1U blade form factor



BOW: 3RD GENERATION IPU SYSTEMS



BOW POD₁₆

4x Bow-2000
5.6 PetaFLOPS
1 CPU server



BOW POD₃₂

8x Bow-2000
11.2 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 POD₁₀₂₄

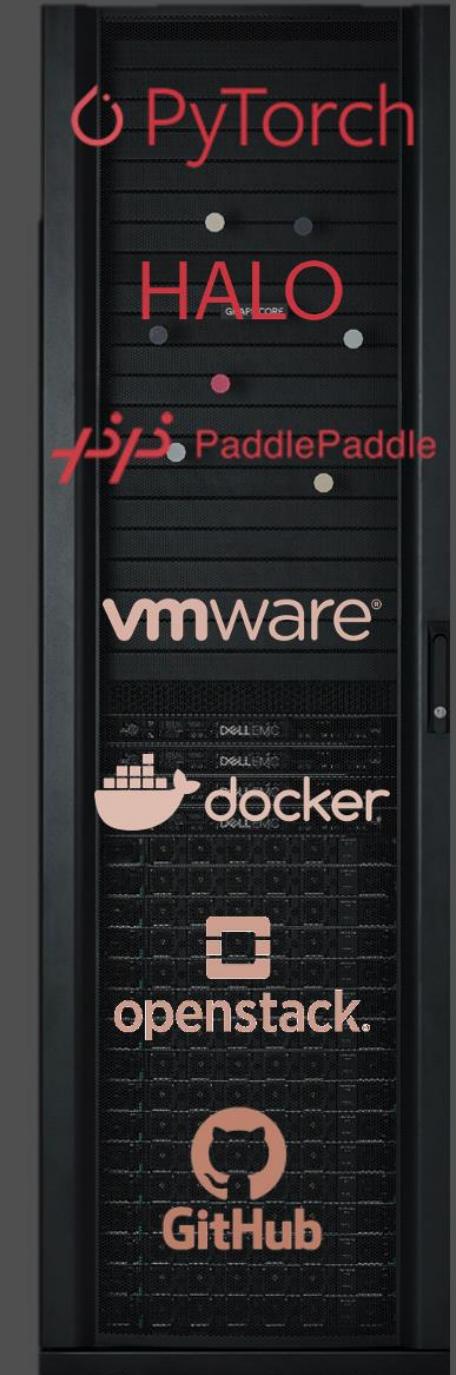
256x Bow-2000
358.4 PetaFLOPS
16 - 64 CPU server(s)
Early access

100% SOFTWARE COMPATIBLE

No code changes to run existing IPU applications

Full support in Poplar SDK and reference applications

Seamless integration into IPU software ecosystem





THE POPLAR® SDK

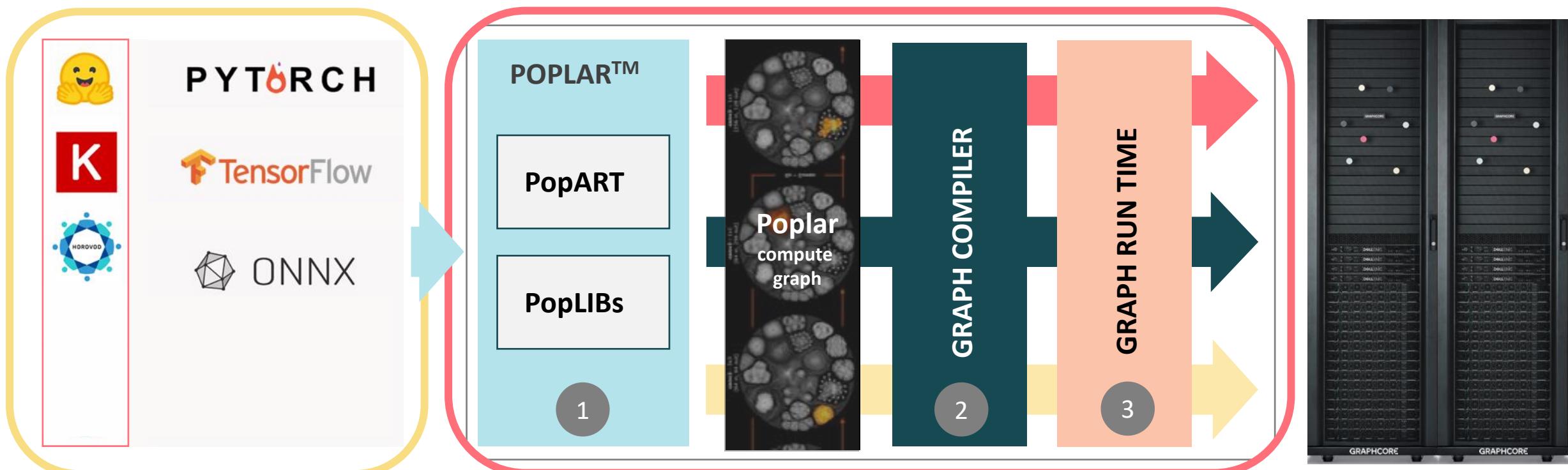


BERT LARGE – POPLAR GRAPH VISUALIZATION FROM POPART



POPLAR-SDK

유저 : 일반적인 high-level 프레임워크 사용



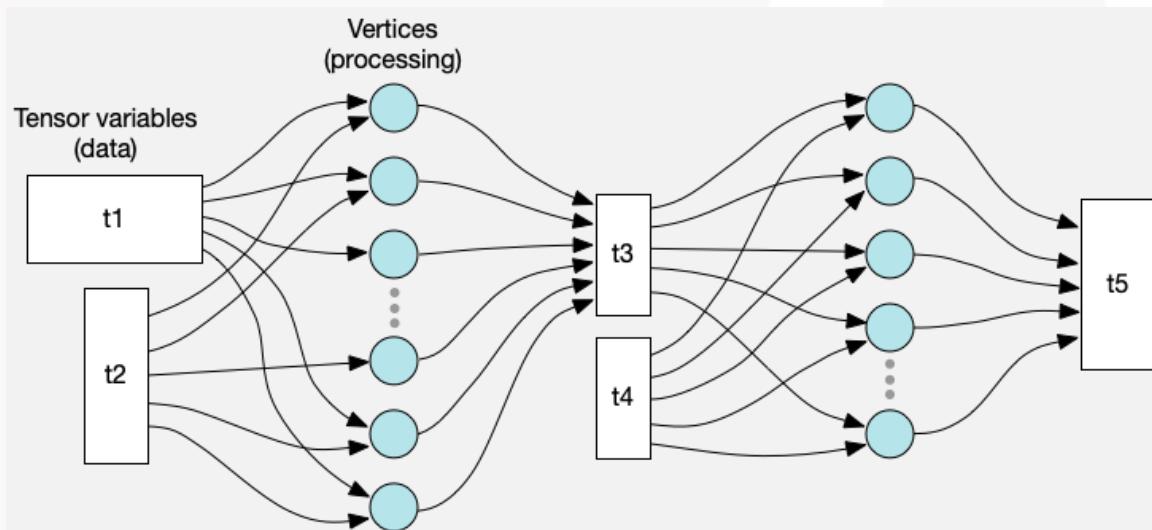
Poplar SDK : 연산 그래프 생성 + IPU 연산 계획 수립 + 실행

POPLAR 내부동작

Poplar 라이브러리 종류

Library	Description
poplin	Linear algebra functions (matrix multiplications, convolutions)
popnn	Functions used in neural networks (for example, non-linearities, pooling and loss functions)
popops	Functions for operations on tensors in control programs (elementwise functions and reductions)
poprand	Functions for populating tensors with random numbers
popsparse	Functions for operating on sparse tensors
poputil	General utility functions for building graphs

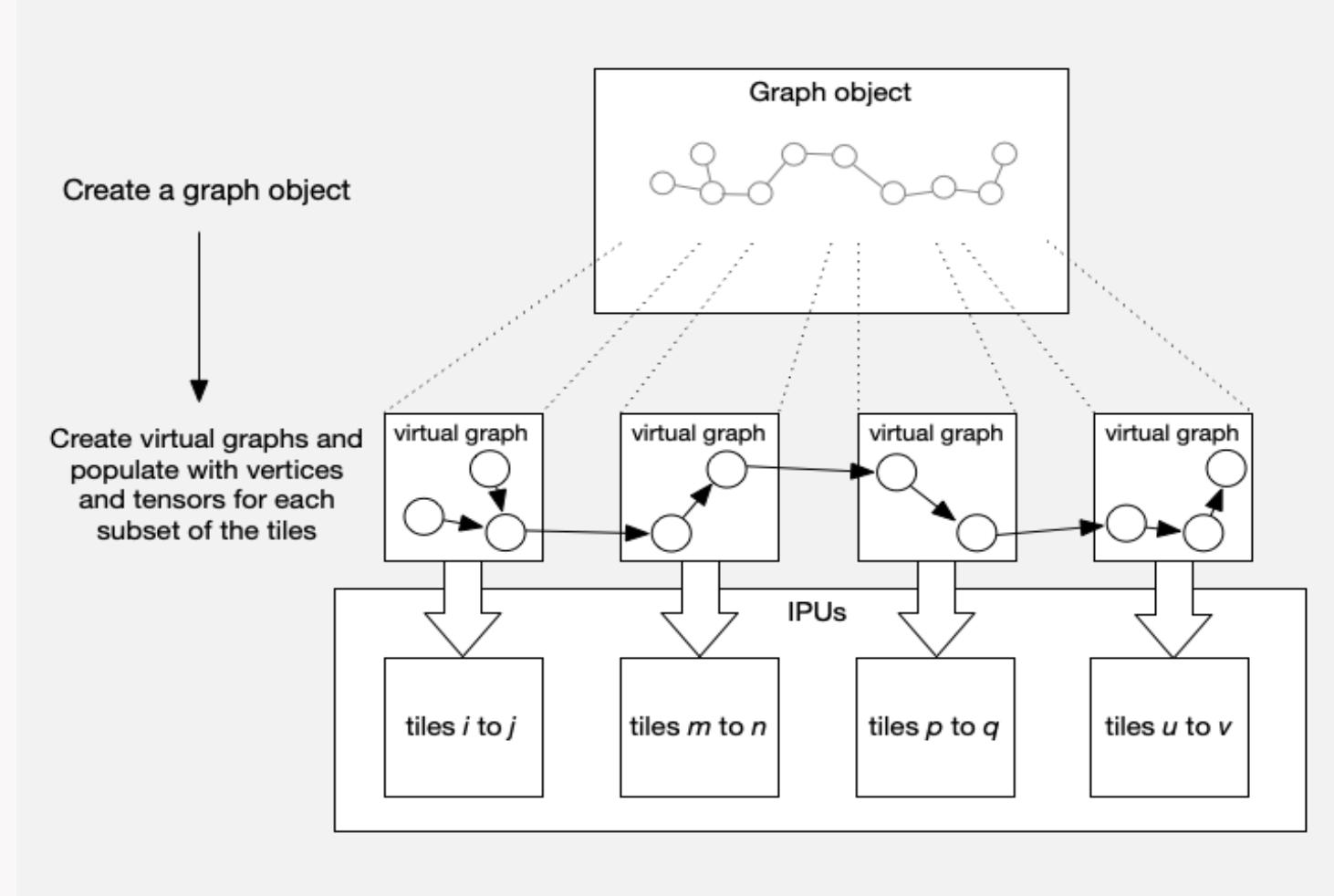
Compute Graph (연산 그래프)



- **Vertex (Codelet)** : 단일 타일에서 한 번에 수행되는 연산
- **Edge (Data)** : 로컬 타일 혹은 다른 타일의 데이터
 - **로컬 타일** : 로컬 메모리로의 Read/Write
 - **다른 타일** : 타일 간의 Exchange

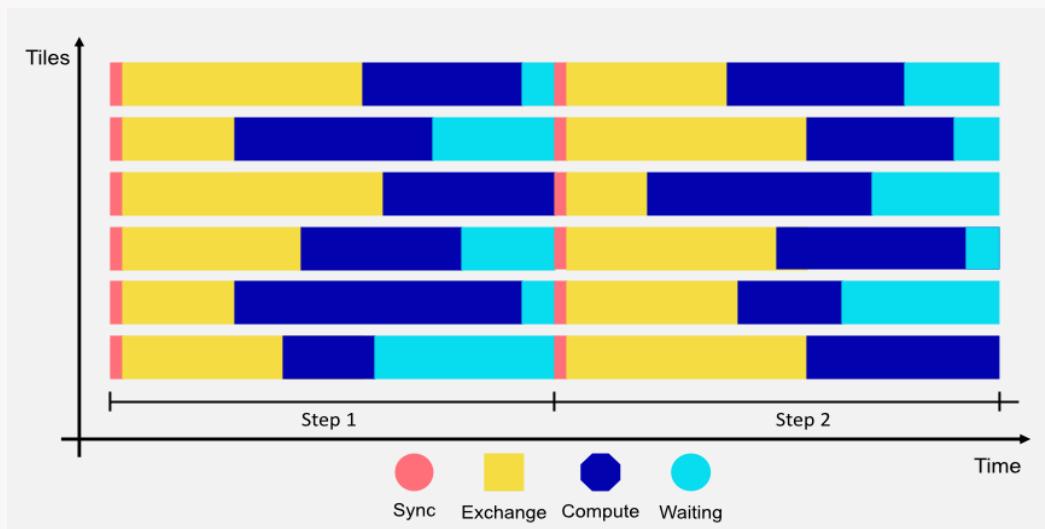
POPLAR 내부동작

- Graph compiler

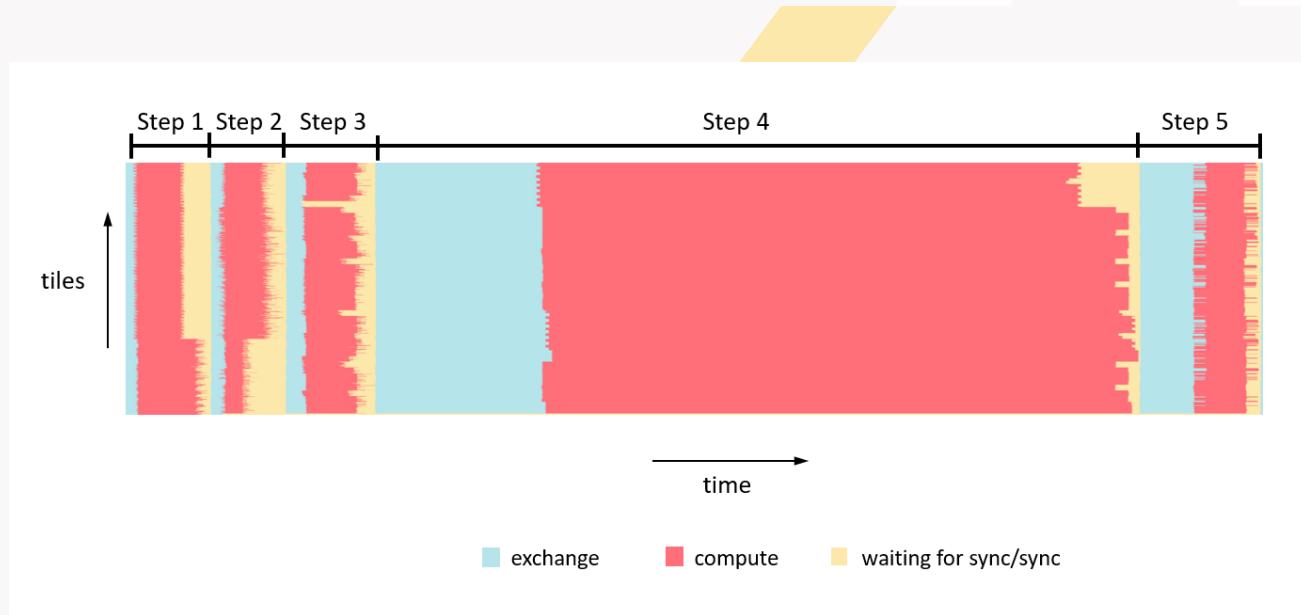


POPLAR 내부동작

BSP (Bulk Synchronous Parallel)



- Local compute : 모든 타일들의 병렬 연산
- Global synchronization : 모든 타일의 연산 종료 대기
- Data exchange : 타일 간의 데이터 전송



실제 BSP 시각화 예시

ML FRAMEWORKS SUPPORTED TODAY



TENSORFLOW FOR THE IPU

TensorFlow

- Graphcore supplies its own branch of TensorFlow that supports the IPU
- Full backend with extensive feature coverage
- Enable TensorFlow to target the IPU directly via Poplar
- TensorFlow 1.15 support
- TensorFlow 2.5 support
- Inference and training
- Full integration with PopVision analysis tools



PYTORCH FOR THE IPU

- PopTorch is a set of extensions for PyTorch
- PopTorch enables native PyTorch models to run directly on IPUs
- Minimal changes to PyTorch code
- PyTorch 1.10.0+cpu
- Inference and training
- Full integration with PopVision analysis tools

Define a model within PyTorch

Create an IPU execution wrapper around the model and run as normal

PopTorch uses the `torch.jit.trace` API to trace the model to PyTorch IR

Compile the graph in PopART and then run on one or more IPUs

POPART: ONNX SUPPORT FOR THE IPU

The Poplar Advanced Runtime

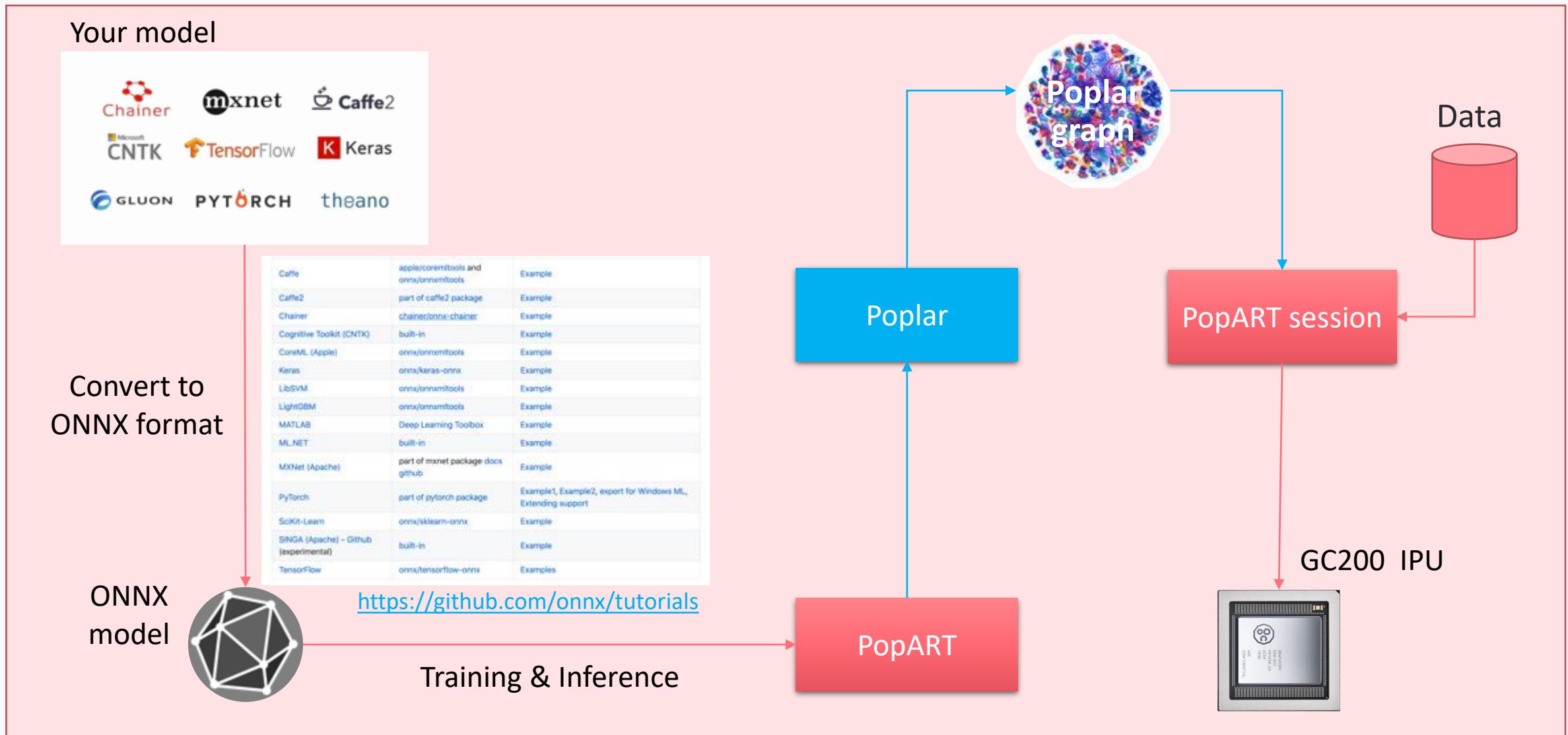
PopART has three main features:

1. It can import **ONNX** graphs into a runtime environment.
2. It provides a simple interface for constructing ONNX graphs without needing a third party.
3. It runs imported graphs in inference, evaluation or training modes, by building a Poplar engine, connecting data feeds and scheduling the execution of the Engine.

What is ONNX?

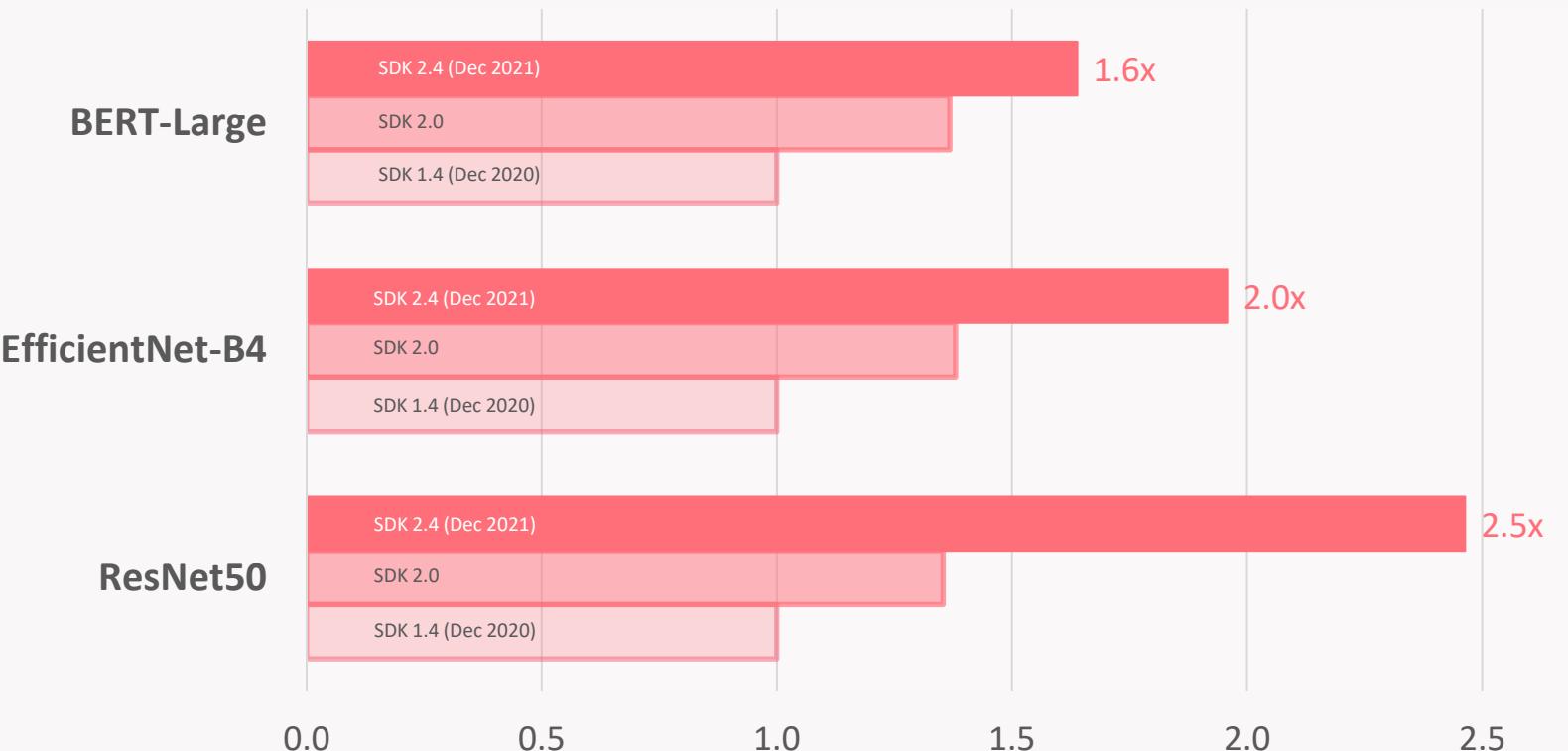
- Open Neural Network Exchange
- ONNX is a serialisation format for neural network systems that can be created and read by several frameworks including Caffe2, PyTorch and MXNet.

IMPORTING A MODEL INTO POPART VIA ONNX



SDK 2.4 PERFORMANCE IMPROVEMENT

Throughput Improvement since SDK 1.4 (Dec 2020)



BERT-Large Phase 1 Pre-Training (SL128) Throughput
G16- EfficientNet-B4 Training Throughput
ResNet50 v1.5 Training Throughput
<https://www.graphcore.ai/performance-results>



DEVELOPER ECOSYSTEM ENGAGEMENT



Graphcore Announces Production Release of PyTorch for IPU

Author: Matt Fyles, SVP Software, Graphcore

PyTorch + GRAPHCORE

Today we are introducing our first production release of PyTorch for IPU — PyTorch — combining the performance of the Graphcore IPU-M2000 system and the developer-ready accessibility of PyTorch. This will enable the fast-growing PyTorch developer community to make new breakthroughs in machine intelligence with Graphcore IPU systems, while maintaining the dynamic PyTorch experience.

GRAPHCORE



Introducing the Initial Release of PyTorch Lightning for Graphcore IPUs

David Norman · Follow · Aug 12 · 3 min read

PyTorch Lightning now supports Graphcore IPUs

We are thrilled to announce PyTorch Lightning now supports Graphcore IPUs. The team at PyTorch Lightning has been working heroically on building IPU integration over the last few months and is now making this available to the community with its 1.4 release. We really appreciate their close collaboration with the Graphcore team to help us with our mission to make IPUs easier to use for developers.

GRAPHCORE CONFIDENTIAL



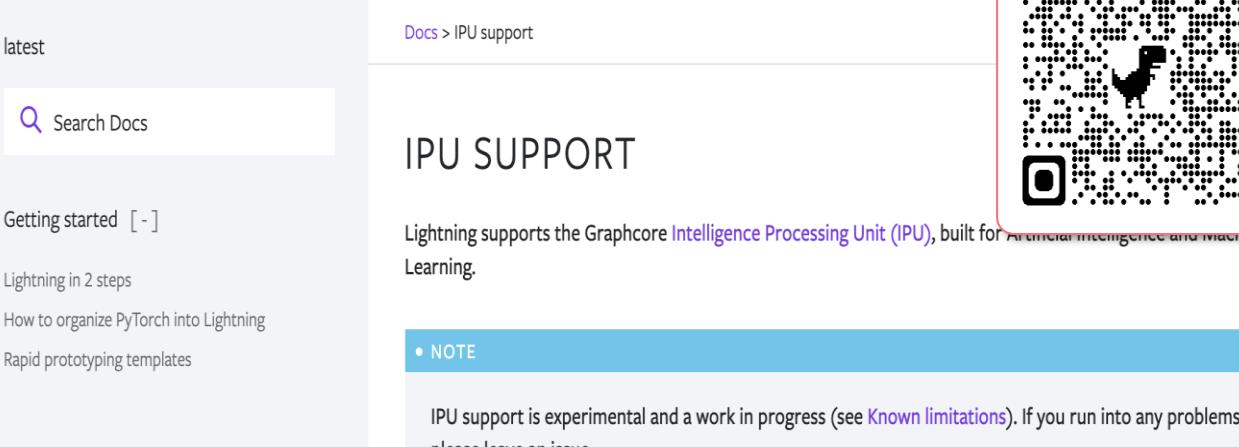
GRAPHCORE



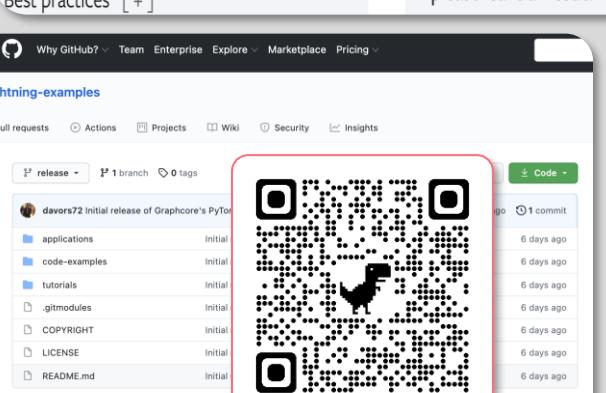
PYTORCH LIGHTNING ON IPU RESOURCES



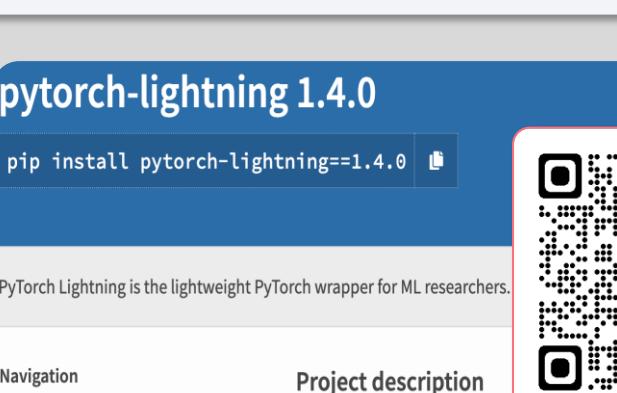
The image shows the PyTorch homepage. At the top left is the PyTorch logo. Below it is a sub-headline: "An open source machine learning framework that accelerates the path from research prototyping to production deployment." A horizontal navigation bar follows, with links: ANNOUNCEMENTS, TUTORIALS, CASE STUDIES, DEVELOPER SPOTLIGHT, RESPONSIBLE AI, ALL STORIES, VIDEO, and social media icons for Twitter and Facebook. To the right is a green button labeled "Following". Below the navigation is a large image of two circular brain scans with colorful, glowing neural activity patterns. To the right is a white box containing a QR code with a small black silhouette of a person in the center.



The screenshot shows the PyTorch Lightning website's "IPU SUPPORT" page. The header includes links for "Get Started", "Blog", "Tutorials", "Ecosystem", and "Docs". A search bar is at the top left. The main content area has a heading "IPU SUPPORT" and a note stating "Lightning supports the Graphcore Intelligence Processing Unit (IPU), built for Artificial Intelligence and Machine Learning." Below this is a "NOTE" section with the text: "IPU support is experimental and a work in progress (see [Known limitations](#)). If you run into any problems, please leave an issue." On the right side of the page is a QR code with a red border.



The screenshot shows a GitHub repository page for "graphcore/pytorch-lightning-examples". It displays a list of files and a QR code with a red border. The repository description is "Graphcore code examples and tutorials for PyTorch Lightning".



The screenshot shows the PyTorch Lightning project page on GitHub. It features a "Project description" section with the command "pip install pytorch-lightning==1.4.0" and a QR code with a red border. The page also includes sections for "Navigation", "Project description", and "Release history".

GRAPHCORE X HUGGING FACE

Hugging Face and Graphcore partner for IPU-optimized Transformers

[Update on GitHub](#)

 **sallydoherty**
Sally.Doherty guest

Speaking at the 2021 AI Hardware Summit, Hugging Face announced the launch of their new Hardware Partner Program, including device-optimized models and software integrations. Here, Graphcore - creators of the Intelligence Processing Unit (IPU) and a founding member of the program – explain how their partnership with Hugging Face will allow developers to easily accelerate their use of state-of-the-art Transformer models.

Graphcore and Hugging Face are two companies with a common goal – to make it easier for innovators to harness the power of machine intelligence.

Hugging Face's Hardware Partner Program will allow developers using Graphcore systems to deploy state-of-the-art Transformer models, optimised for our Intelligence Processing Unit

Introducing 😊 Optimum: The Optimization Toolkit for Transformers at Scale

Published September 14, 2021.

[Update on GitHub](#)


mfuntowicz Morgan Funtowicz echarlaix Ella Charlaix michaelbenayoun Michael Benayoun
jffboudier Jeff Boudier

This post is the first step of a journey for Hugging Face to democratize state-of-the-art Machine Learning production performance. To get there, we will work hand in hand with our Hardware Partners, as we have with Intel below. Join us in this journey, and follow [Optimum](#), our new open source library!

Why 😊 Optimum?

FRAMEWORKS

PyTorch Lightning TensorFlow Keras ONNX HALO PaddlePaddle

Access Graphcore's implementation of the BERT (Bidirectional Encoder Representations from Transformers) model in PyTorch for the IPU, leveraging the Hugging Face Transformers library.

Announcing our new partnership with Hugging Face allowing developers to easily accelerate their use of state-of-the-art Transformer models. Read our [guest blog](#) to learn more.

Hugging Face and Graphcore partner for IPU-optimized Transformers
SOTA Transformers meet SOTA hardware

 Graphcore Sep 14 · 4 min read

 **Hugging Face @huggingface** · 4h
Introducing 😊 Optimum

A new open source library to optimize 😊 Transformers for production performance. 🎉

Quantize, Prune, Optimize models easily, targeting hardware from our partners @intel @graphcoreai @Qualcomm! 😊


The AI community building the future

Introducing Optimum: The Optimization Toolkit for Transformers at Scale
huggingface.co/optimus

1 49 142 541 · 3 comments

 **Hugging Face** 36,436 followers 15h ·
Introducing 😊 Optimum

A new open source library to optimize 😊 Transformers for production performance. 🎉

Quantize, Prune, Optimize models easily, targeting hardware from our partners Intel Corporation Graphcore Qualcomm! 😊

<https://lnkd.in/gZEbDDs>

#ai #machinelearning #hardware

Introducing 😊 Optimum: The Optimization Toolkit for Transformers at Scale

Published September 14, 2021.

[Update on GitHub](#)


mfuntowicz Morgan Funtowicz echarlaix Ella Charlaix michaelbenayoun Michael Benayoun
jffboudier Jeff Boudier

This post is the first step of a journey for Hugging Face to democratize state-of-the-art Machine Learning production performance. To get there, we will work hand in hand with our Hardware Partners, as we have with Intel below. Join us in this journey, and follow [Optimum](#), our new open source library!

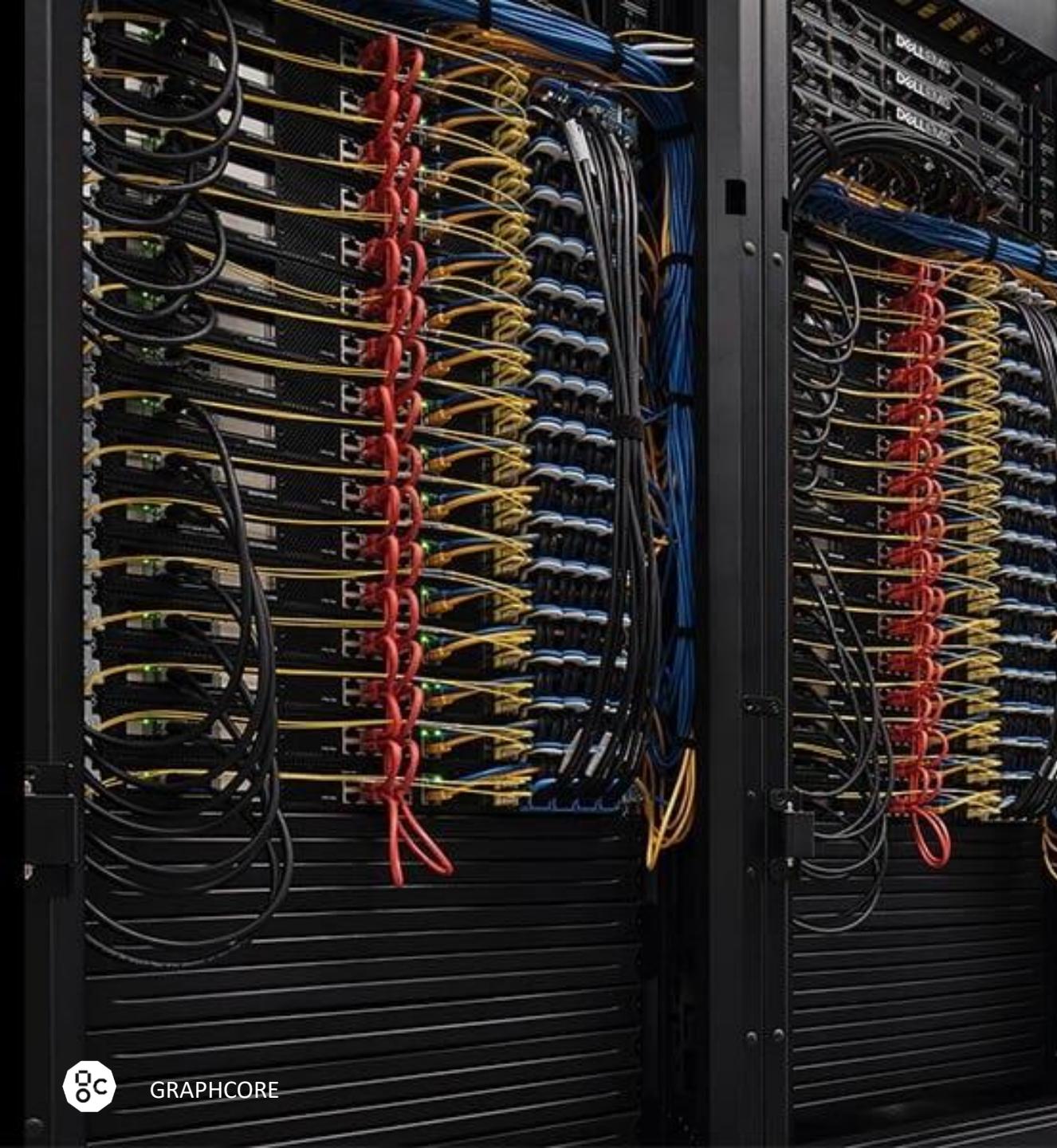
Introducing Optimum: The Optimization Toolkit for Transformers at Scale
huggingface.co/optimus · 5 min read

541 · 3 comments

3. Train

Train models faster than ever before with Graphcore Intelligence Processing Units (IPUs), the latest generation of AI dedicated hardware, leveraging the built-in IPUTrainer API to train or finetune transformers models (coming soon)

```
huggingface@hardware:~  
from optimum.graphcore import IPUTrainer  
from optimum.graphcore.bert import BertIPUConfig  
from transformers import BertForMaskedLM, BertTokenizer  
from poptorch.optim import AdamW  
  
# Allocate model and tokenizer as usual  
tokenizer = BertTokenizer.from_pretrained("bert-base-cased")  
model = BertForMaskedLM.from_pretrained("bert-base-cased")  
  
# Trainer + poptorch custom configuration optional  
ipu_config = BertIPUConfig()  
trainer = IPUTrainer(model, trainings_args, config=ipu_config)  
optimizer = AdamW(model.parameters)  
  
# This is hidden from the user, it will be handled by the Trainer  
with trainer.compile(some_data_loader) as model_f:  
    for steps in range(...):  
        outputs = trainer.step(optimizer)  
  
# Save the model and/or push to hub  
model.save_pretrained("")  
model.push_to_hub("")
```



CLOUD NATIVE & SCALEABLE

SOFTWARE DEFINED INFRASTRUCTURE

Development



Orchestration &
Scheduling



kubernetes



slurm
workload manager

Monitor &
Control



Redfish



OpenBMC

Logging &
Visualization



Provisioning &
Virtualization



openstack



Virtual-IPU



GRAPHCORE

GROWING MODEL GARDEN



POPVISION TOOLS



GRAPHCORE

MODEL GARDEN



INFERENCE MODELS

BERT-Large Inference
BERT-Large (Bidirectional Encoder Representations from Transformers) for NLP inference on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PopART](#)

BERT-Base Inference
BERT-Base (Bidirectional Encoder Representations from Transformers) for NLP inference on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PopART](#)

ResNet-50 Inference
Image classification inference on IPUs using the CNN (Convolutional Neural Network) model ResNet-50.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

ResNet-50 Inference
Image classification inference on IPUs using the CNN (Convolutional Neural Network) model ResNet-50.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

EfficientNet-B0/B4 Inference
CNN (Convolutional Neural Network) image classification inference on IPUs with EfficientNet.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

ResNeXt-101 Inference
Image classification inference on IPUs using the CNN (Convolutional Neural Network) model ResNeXt-101.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

ResNeXt-101 Inference
Image classification inference on IPUs using the CNN (Convolutional Neural Network) model ResNeXt-101.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

ResNeXt-101 Inference
Image classification inference on IPUs using the CNN (Convolutional Neural Network) model ResNeXt-101.
[View on GitHub](#) [Computer Vision](#) [PopART](#)

DIN Inference
DIN (Deep Interest Evolution Network) inference on IPUs - a recommendation model for click-through rate prediction.
[View on GitHub](#) [Other](#) [TensorFlow](#)

DIN Inference
DIN (Deep Interest Network) inference on IPUs - a recommendation model for click-through rate prediction.
[View on GitHub](#) [Other](#) [TensorFlow](#)

YOLOv3 Inference
YOLOv3 - You Only Look Once - is a convolutional neural network model that performs object detection tasks on IPUs.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

YOLOv4 Inference
YOLOv4 - You Only Look Once - is a convolutional neural network model that performs object detection tasks on IPUs.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

MobileNetV2 Inference
MobileNetV2 - Convolutional neural network inference for classification, detection and segmentation.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

MobileNetV3 Inference
MobileNetV3 - Convolutional neural network inference for classification, detection and segmentation.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

Autoencoder Inference
Custom autoencoder inference model on the IPU to perform collaborative filtering in recommender systems.
[View on GitHub](#) [Generative](#) [TensorFlow](#)

Contrastive Divergence VAE Training
Train a Variational Autoencoder / Markov Chain Monte Carlo hybrid model on IPUs.
[View on GitHub](#) [Generative](#) [TensorFlow](#)

Sales Forecasting Training
How to train a sales forecasting machine learning model on Graphcore's IPUs.
[View on GitHub](#) [Probabilistic Modeling](#) [TensorFlow](#)

Reinforcement Learning Training
How to train a deep reinforcement learning model on multiple IPUs with synchronous data parallel training.
[View on GitHub](#) [Reinforcement Learning](#) [TensorFlow](#)

UNet Industrial Training
How to run a UNet Industrial training example for image segmentation.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

TRAINING MODELS

BERT-Large Training

BERT-Large (Bidirectional Encoder Representations from Transformers) for NLP training on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [TensorFlow](#)

BERT-Base Training

BERT-Base (Bidirectional Encoder Representations from Transformers) for NLP pre-training and training on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PopART](#)

BERT-Large Training

BERT-Large (Bidirectional Encoder Representations from Transformers) for NLP training on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PyTorch](#)

BERT-Base Training

BERT-Base (Bidirectional Encoder Representations from Transformers) for NLP pre-training and training on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PyTorch](#)

BERT-Base Training

BERT-Base (Bidirectional Encoder Representations from Transformers) for NLP pre-training and training on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PopART](#)

BERT-Base Training

BERT-Base (Bidirectional Encoder Representations from Transformers) for NLP pre-training and training on IPUs.
[View on GitHub](#) [Natural Language Processing](#) [PopART](#)

ResNet-50 Training

Image classification training on IPUs using the CNN (Convolutional Neural Network) model ResNet-50.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

ResNet-50 Training

Image classification training on IPUs using the CNN (Convolutional Neural Network) model ResNet-50.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

EfficientNet-B0 Training

CNN (Convolutional Neural Network) image classification training on IPUs with EfficientNet.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

EfficientNet-B4 Training

CNN (Convolutional Neural Network) image classification training on IPUs with EfficientNet.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

ResNeXt-101 Training

Image classification training on IPUs using the CNN (Convolutional Neural Network) model ResNeXt-101.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

MCMC Training

Markov Chain Monte Carlo (MCMC) training on IPUs using standard Tensorflow Probability.
[View on GitHub](#) [Probabilistic Modeling](#) [TensorFlow](#)

Deep Voice 3 Training

Text-to-Speech training on IPUs using a Convolutional Sequence Learning technique.
[View on GitHub](#) [Speech Processing](#) [PopART](#)

DIEN Training

DIEN (Deep Interest Evolution Network) training on IPUs - a recommendation model for click-through rate prediction.
[View on GitHub](#) [Other](#) [TensorFlow](#)

DIN Training

DIN (Deep Interest Network) training on IPUs - a recommendation model for click-through rate prediction.
[View on GitHub](#) [Other](#) [TensorFlow](#)

YOLOv3 Training

YOLOv3 - You Only Look Once - is a convolutional neural network model that performs object detection tasks on IPUs.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

MobileNetV3 Training

MobileNetV3 - Convolutional neural network training for classification, detection and segmentation.
[View on GitHub](#) [Computer Vision](#) [PyTorch](#)

Autoencoder Training

Custom autoencoder model on the IPU to train collaborative filtering in recommender systems.
[View on GitHub](#) [Generative](#) [TensorFlow](#)

Contrastive Divergence VAE Training

Train a Variational Autoencoder / Markov Chain Monte Carlo hybrid model on IPUs.
[View on GitHub](#) [Generative](#) [TensorFlow](#)

Reinforcement Learning Training

How to train a deep reinforcement learning model on multiple IPUs with synchronous data parallel training.
[View on GitHub](#) [Reinforcement Learning](#) [TensorFlow](#)

Sales Forecasting Training

How to run a sales forecasting machine learning model on Graphcore's IPUs.
[View on GitHub](#) [Probabilistic Modeling](#) [TensorFlow](#)

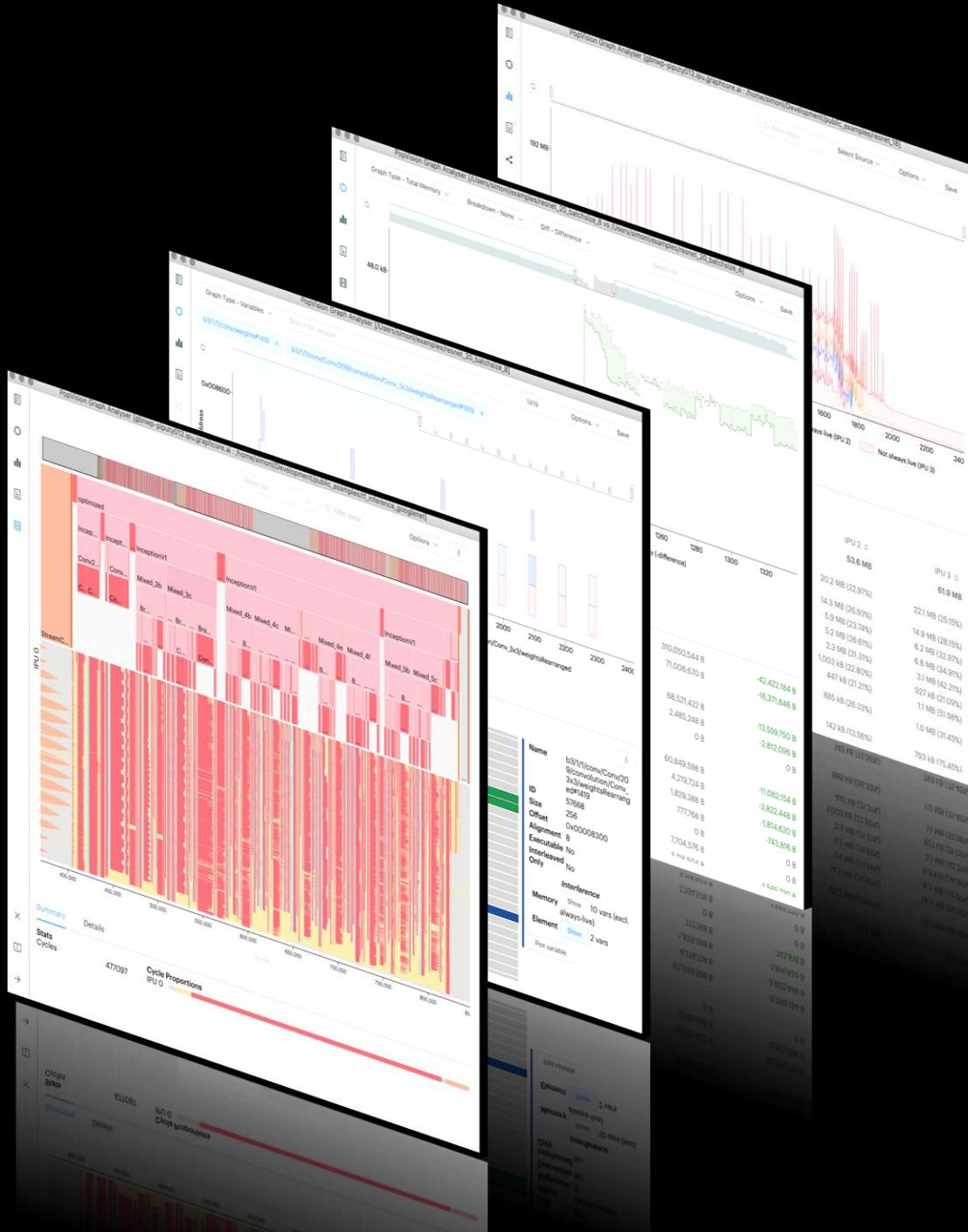
UNet Industrial Training

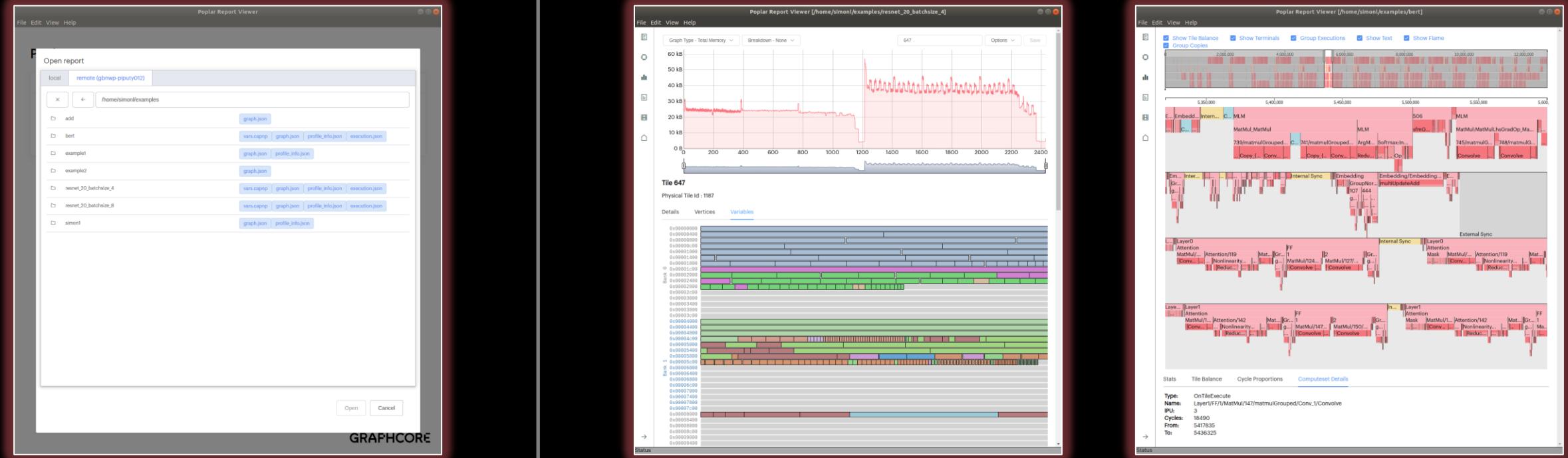
How to run a UNet Industrial training example for image segmentation.
[View on GitHub](#) [Computer Vision](#) [TensorFlow](#)

GRAPHCORE

POPVISION
ANALYSIS TOOLS

GRAPH ANALYSER 2.4
RELEASED WITH
POPLAR SDK 2.1





VERSION 1.0 SCREENSHOTS

LOCAL AND REMOTE FILE ACCESS

- SUPPORT FOR LOCAL FILE VIEWING AND ANALYSIS OF PROFILE REPORTS OFFLINE
- SUPPORT FOR REMOTE SERVER CONNECTION AND ANALYSIS OF FILES ON A SERVER CONTAINING HARDWARE

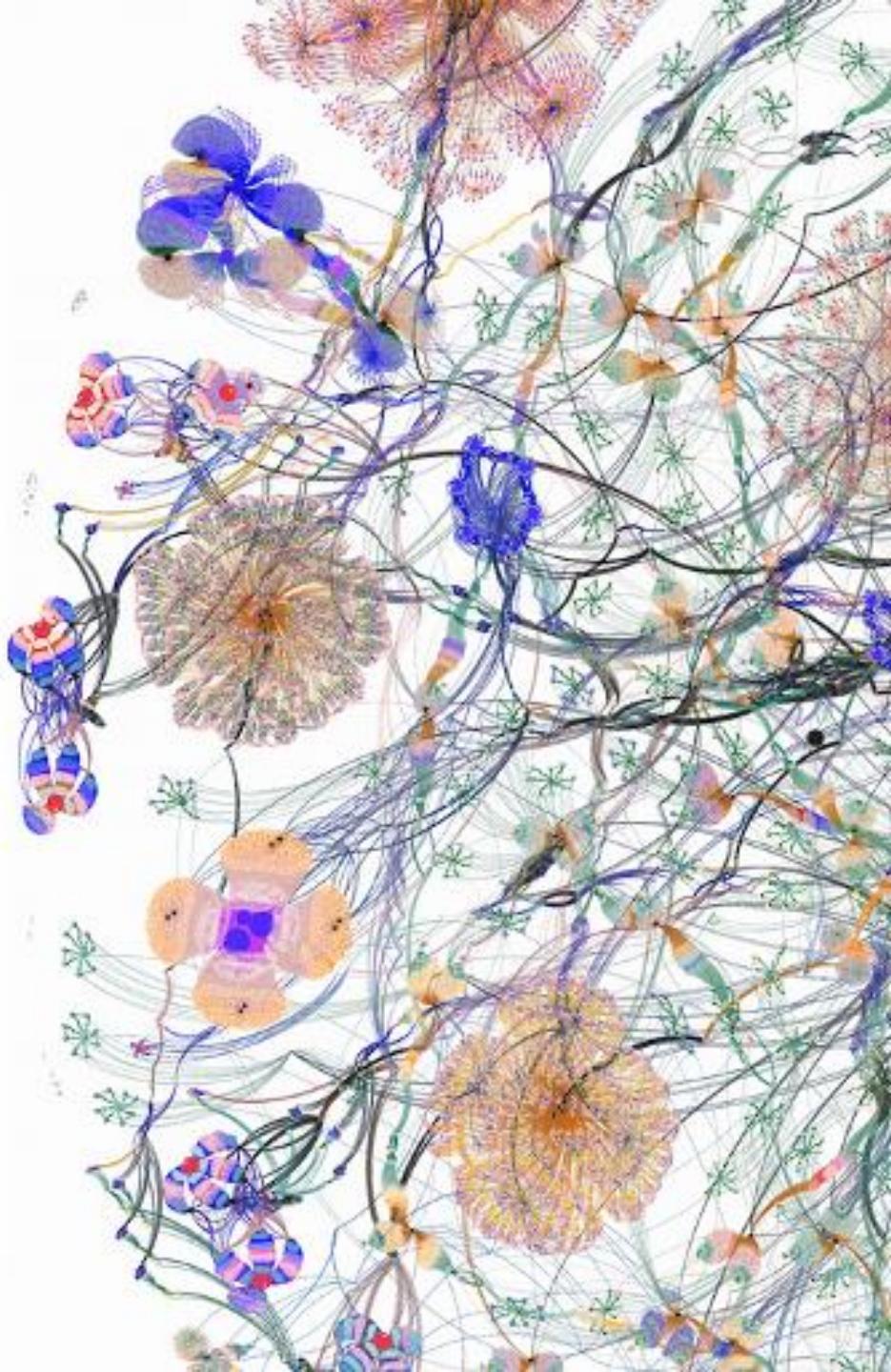
IPU APPLICATION PROFILE VISUALISATION

- PRODUCT VERSION OF GC-PERF, SUPPORTED AND DEVELOPED WITH ROADMAP FOR FUTURE FEATURES
- VISUALIZE DETAILED INFORMATION ABOUT IPU APPLICATION EXECUTION
- VIEW APPLICATION MEMORY USAGE, LIVENESS ANALYSIS, PROGRAM TREE, EXECUTION TRACE FOR IPU DEVICES
- NEW SUPPORT FROM INTERNAL TOOL TO VIEW PROGRAM SYMBOL INFORMATION



BASIC PORTING GUIDE

- PYTORCH



FROM PYTORCH TO POPTORCH

Model

```
class CustomModel(nn.Module):
    def __init__(self):
        super().__init__()

        # down-sampling layer
        self.MaxPool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.Conv2d_1_64 = nn.Conv2d(1, 64, 7, 2, 3)
        self.Conv2d_64_128_1 = nn.Conv2d(64, 128, 3, 2, 1)

    def forward(self, x):
        # down
        # print(x.shape)
        x = self.Conv2d_1_64(x)
        x = self.BatchNorm2d_64(x)
        x = self.ReLU(x)

        return x
```

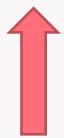
```
36   36   class CustomModel(nn.Module):
37   37   def __init__(self):
38   38       super().__init__()
39   39
40   40       # down-sampling layer
41  41 << self.loss = nn.BCELoss()
42  42
43  43       self.MaxPool = nn.MaxPool2d(kernel_size=2, stride=2)
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91  92 << def forward(self, x, targets=None):
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194  195 << if targets is not None:
195  196         return x, self.loss(x, targets)
196  197
197
```

You should place loss function inside the model definition in order to use PopTorch training wrapper

FROM PYTORCH TO POPTORCH

Model

```
5 import torch  
6 from efficientnet_pytorch import EfficientNet  
  
72     model = EfficientNet.from_pretrained('efficientnet-b0', num_classes=18)  
75     criterion = torch.nn.CrossEntropyLoss()  
78     model_with_loss = ModelWithLoss(model, criterion)
```



```
5 class ModelWithLoss(torch.nn.Module):  
6     def __init__(self, net, loss):  
7         super(ModelWithLoss, self).__init__()  
8         self.net = net  
9         self.loss = loss  
10      
11    def forward(self, x, target=None):  
12        outputs = self.net(x)  
13        if target is not None:  
14            return outputs, self.loss(outputs, target)  
15        return outputs
```

Alternatively, you can use a simple wrapper class to add loss function at the end of the model

FROM PYTORCH TO POPTORCH

Model

```
model = densenet.densenet201(num_classes=args.num_classes).eval()  
).cuda()
```

```
» 23 32 «  
33  
24 34  
25 35  
26 36
```

```
model = densenet.densenet201(num_classes=args.num_classes).eval()  
  
opts = poptorch.Options()  
opts.deviceIterations(args.device_iteration)  
opts.replicationFactor(args.replication_factor)
```

You can change how PopTorch compiles and executes models using `poptorch.Options`

```
64
```

```
poptorch_model = poptorch.inferenceModel(model, options=opts)
```

This function wraps a PyTorch model, yielding a PopTorch model that can be run on the IPU in inference mode.

FROM PYTORCH TO POPTORCH

Model

PopTorch doesn't use `torch.device` to target the IPU device.

```
net = CustomModel()

device = torch.device('cuda:' + args.gpu_num if
torch.cuda.is_available() else 'cpu')
```

```
optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
criterion = torch.nn.BCELoss()
net.to(device).train()
```

```
train = BasicDataset(dir_train, dir_train2, args.scale)
n_train = len(train)
train_loader = DataLoader(
    train, batch_size=args.batchsize, shuffle=True, drop_last=True,
    num_workers=args.num_workers, pin_memory=True)
```

```
» 9   10 «< from poptorch import optim
  10  11 << from poptorch import DataLoader
  49  52 << net = CustomModel()
  50  53
  » 51  54 «< opts = poptorch.Options()
  55
  52  56 << opts.replicationFactor(args.replication_factor)
            opts.Training.gradientAccumulation(args.gradient_accumulation)
```

```
  52  57 << optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
  » 53  58 «< poptorch_model = poptorch.trainingModel(net, options=opts,
  54
            optimizer=optimizer)
```

```
  56  60 << train = BasicDataset(dir_tr
  57  61
  58  62
  » 59  63 «< n_train = len(train)
            train_loader = DataLoader(
                opts, train, batch_size=args.batchsize, shuffle=True,
                drop_last=True,
                num_workers=args.num_workers, pin_memory=True,
                mode=poptorch.DataLoaderMode.Async
  60
  61  64
  » 62
  63  65 )
```

You can change how PopTorch compiles and executes models using `poptorch.Options`

This function wraps a PyTorch model, yielding a PopTorch model that can be run on the IPU in training mode.

FROM PYTORCH TO POPTORCH

Optimizer

```
from torch import optim
from torch.utils.data import DataLoader

net = CustomModel()

device = torch.device('cuda:' + args.gpu_num if
torch.cuda.is_available() else 'cpu')

optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
criterion = torch.nn.BCELoss()
net.to(device).train()

train = BasicDataset(dir_train, dir_train2, args.scale)
n_train = len(train)
train_loader = DataLoader(
    train, batch_size=args.batchsize, shuffle=True, drop_last=True,
    num_workers=args.num_workers, pin_memory=True)
```

```
» 9   10 «< from poptorch import optim
10  11   from poptorch import DataLoader

net = CustomModel()

opts = poptorch.Options()
opts.replicationFactor(args.replicas)
opts.Training.gradientAccumulation(1)

optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
poptorch_model = poptorch.trainingModel(net, options=opts,
                                        optimizer=optimizer)

train = BasicDataset(dir_train, dir_train2, args.scale)
n_train = len(train)
train_loader = DataLoader(
    opts, train, batch_size=args.batchsize, shuffle=True,
    drop_last=True,
    num_workers=args.num_workers, pin_memory=True,
    mode=poptorch.DataLoaderMode.Async
) )
```

PopTorch optimizers have features to support float16 precision, such as loss scaling, velocity scaling, bias correction and accumulator types.

FROM PYTORCH TO POPTORCH

DataLoader

Poptorch provides a thin wrapper around the traditional `torch.utils.data.DataLoader` to abstract away some of the batch sizes calculations. If `poptorch.DataLoader` is used in a distributed execution environment, it will ensure that each process uses a different subset of the dataset.

```
from torch import optim
from torch.utils.data import DataLoader
net = CustomModel()
device = torch.device('cuda:' + args.gpu_num if
torch.cuda.is_available() else 'cpu')
optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
criterion = torch.nn.BCELoss()
net.to(device).train()

train = BasicDataset(dir_train, dir_train2, args.scale)
n_train = len(train)
train_loader = DataLoader(
    train, batch_size=args.batchsize, shuffle=True, drop_last=True,
    num_workers=args.num_workers, pin_memory=True)
```

```
» 9   10 << 10   11 << from poptorch import DataLoader
net = CustomModel()
opts = poptorch.Options()
opts.replicationFactor(args.replication_factor)
opts.Training.gradientAccumulation(args.gradient_accumulation)

optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
poptorch_model = poptorch.trainingModel(net, options=opts,
optimizer=optimizer)

train = BasicDataset(dir_train, dir_train2, args.scale)
n_train = len(train)
train_loader = DataLoader(
    opts, train, batch_size=args.batchsize, shuffle=True,
    drop_last=True,
    num_workers=args.num_workers, pin_memory=True,
    mode=poptorch.DataLoaderMode.Async
)» 51   52 << 51   54 << 52   53 << 52   55 << 52   56 <<
```

FROM PYTORCH TO POPTORCH

DataLoader

```
from torch import optim
from torch.utils.data import DataLoader

net = CustomModel()

device = torch.device('cuda:' + args.gpu_num if
torch.cuda.is_available() else 'cpu')

optimizer = optim.SGD(net.parameters(), args.lr, weight_decay=1e-5)
criterion = torch.nn.BCELoss()
net.to(device).train()

train = BasicDataset(dir_train, dir_train2, args.scale)
n_train = len(train)
train_loader = DataLoader(
    train, batch_size=args.batchsize, shuffle=True, drop_last=True,
    num_workers=args.num_workers, pin_memory=True)
```

```
» 9   10 « from poptorch import optim
  10  11   from poptorch import DataLoader

  49   52   net = CustomModel()
  50   53
  » 51  54 « opts = poptorch.Options()
  52   56   opts.replicationFactor(args.replication_factor)
            opts.Training.gradientAccumulation(args.gradient_accumulation)

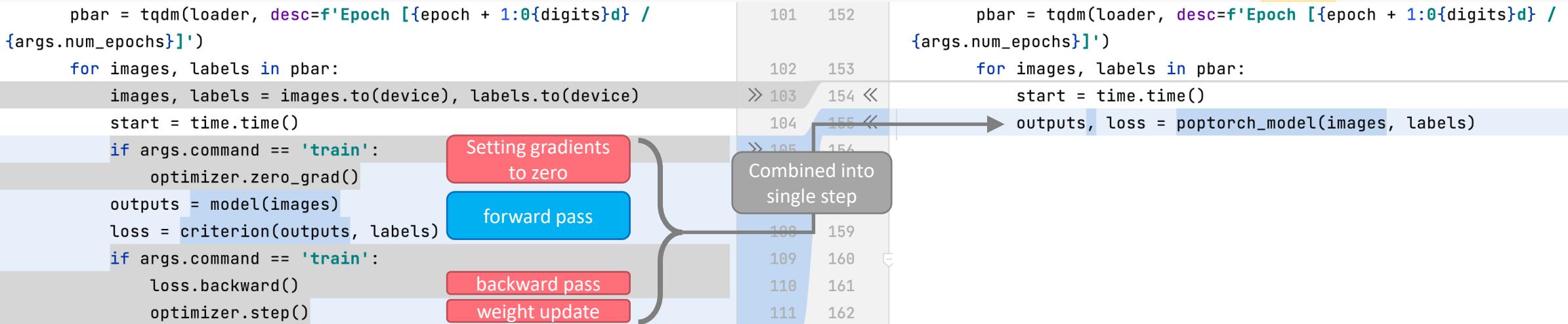
  52   57   optimizer = optim.SGD
  » 53  58 « poptorch_model = poptorch.inferenceModel(net, opts,
  54
            optimizer=optimizer)

  56   60   train = BasicDataset(dir_train, dir_train2, args.scale)
  57   61   n_train = len(train)
  58   62   train_loader = DataLoader(
  » 59  63 «     opts, train, batch_size=args.batchsize, shuffle=True,
  60
            drop_last=True,
  61   64   num_workers=args.num_workers, pin_memory=True,
  » 62  66 «     mode=poptorch.DataLoaderMode.Async
  63   65   )
```

To reduce host overhead, you can offload the data loading process to a separate thread by specifying `mode=poptorch.DataLoaderMode.Async` in the `DataLoader` constructor.

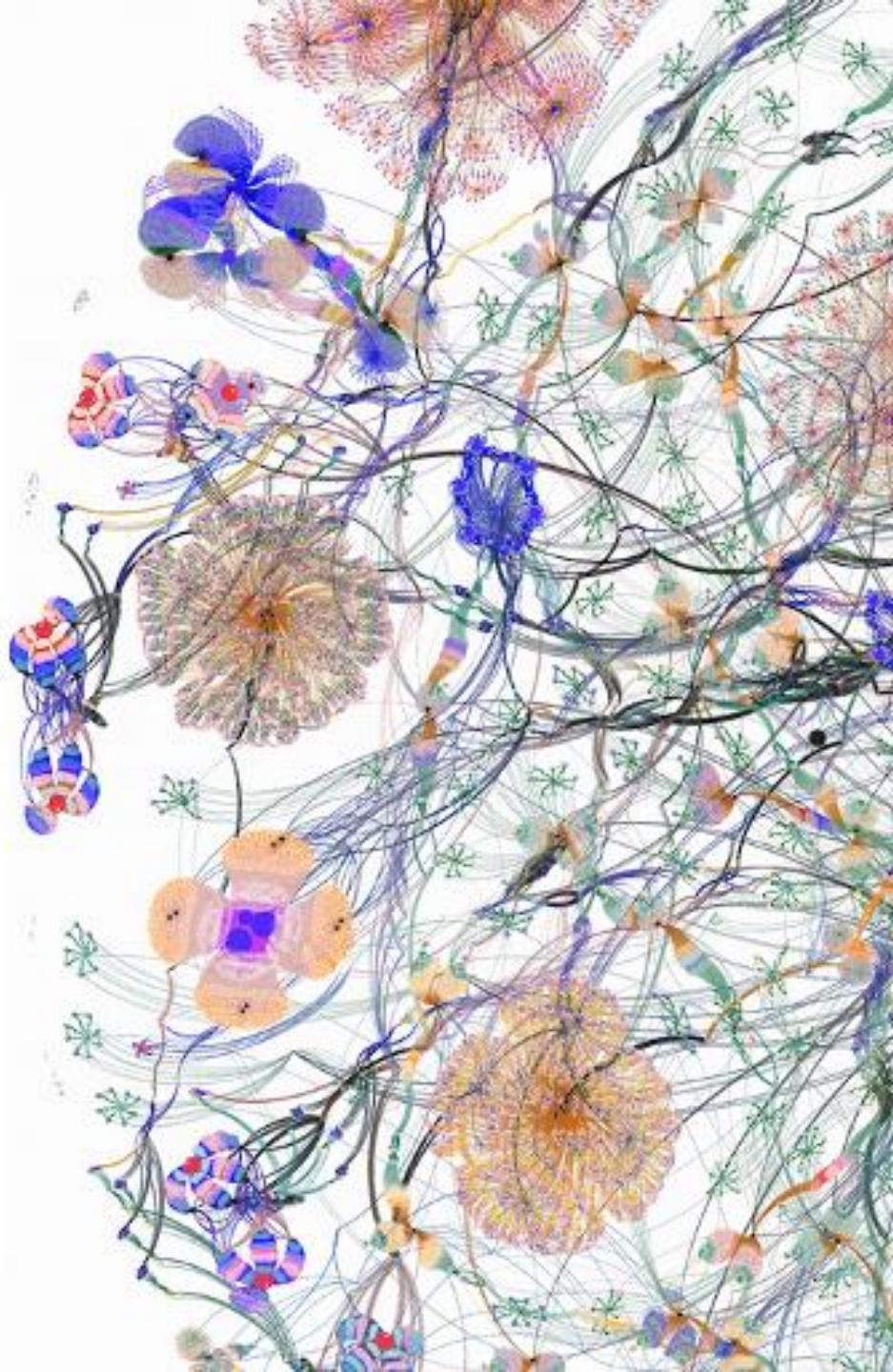
FROM PYTORCH TO POPTORCH

Training / Inference loop



BASIC PORTING GUIDE

- TENSORFLOW2



TENSORFLOW 2 FOR THE IPU

[Tensorflow 2 User Guide – 3.1. IPUStrategy](#)

Use the **IPUStrategy** context to ensure that everything within that context will be compiled for the IPU device. You should do this instead of using the `tf.device` context:

```
import tensorflow as tf
from tensorflow.python import ipu

# Configure the IPU device.
config = ipu.config.IPUConfig()
config.auto_select_ipus = 1
config.configure_ipu_system()

# Create a simple model.
def create_model():
    return tf.keras.Sequential([
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10)
    ])

# Create a dataset for the model.
def create_dataset():
    mnist = tf.keras.datasets.mnist

    (x_train, y_train), (_, _) = mnist.load_data()
    x_train = x_train / 255.0

    train_ds = tf.data.Dataset.from_tensor_slices(
        (x_train, y_train)).shuffle(10000).batch(32, drop_remainder=True)
    train_ds = train_ds.map(lambda d, l:
                           (tf.cast(d, tf.float32), tf.cast(l, tf.int32)))

    return train_ds.repeat().prefetch(16)

dataset = create_dataset()

# Create a strategy for execution on the IPU.
strategy = ipu.ipu_strategy.IPUStrategy()
with strategy.scope():
    # Create a Keras model inside the strategy.
    model = create_model()

    # Compile the model for training.
    model.compile(
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        optimizer=tf.keras.optimizers.RMSprop(),
        metrics=["accuracy"],
    )

    model.fit(dataset, epochs=2, steps_per_epoch=100)
```

TENSORFLOW2

IPUConfig : device selection을 비롯한 system configuration 세팅

```
1 from tensorflow.python import ipu
2
3 cfg = ipu.config.IPUConfig()
4 # Select multi-IPU with device ID 30, which contains all 16 IPUs of a 16-IPU system.
5 cfg.select_ipus = 30
6 cfg.configure_ipu_system()
```

- select_ipus : 특정 IPU ID 지정
- auto-select_ipus : 사용할 IPU 개수를 지정하면 자동으로 할당
- configure_ipu_system() : 세팅된 configuration 적용

TENSORFLOW2

IPUStrategy : context 내의 모델을 poplar XLA를 통해 IPU에서 동작시키는 역할

(Keras의 fit()을 사용하는 경우)

```
1  strategy = ipu.ipu_strategy.IPUStrategy()
2  with strategy.scope():
3      # Create a Keras model inside the strategy.
4      model = create_model()
5
6      # Compile the model for training.
7      model.compile(
8          loss=tf.keras.losses.SparseCategoricalCrossentropy(),
9          optimizer=tf.keras.optimizers.RMSprop(),
10         metrics=["accuracy"],
11     )
12
13     model.fit(dataset, epochs=2, steps_per_epoch=100)
```

IPUStrategy의 scope 내에서 모델 생성 및 실행

TENSORFLOW2

IPUStrategy : context 내의 모델을 poplar XLA를 통해 IPU에서 동작시키는 역할

(custom training loop을 사용하는 경우)

```
@tf.function(experimental_compile=True)
def training_step(num_iterations, iterator, in_model, optimizer):

    for _ in tf.range(num_iterations):
        features, labels = next(iterator)
        with tf.GradientTape() as tape:
            predictions = in_model(features, training=True)
            prediction_loss = tf.keras.losses.sparse_categorical_crossentropy(
                labels, predictions)
            loss = tf.reduce_mean(prediction_loss)
            grads = tape.gradient(loss, in_model.trainable_variables)
            optimizer.apply_gradients(zip(grads, in_model.trainable_variables))

strategy = ipu.ipu_strategy.IPUStrategyV1()
with strategy.scope():
    # Run the custom training loop over the dataset.
    strategy.run(training_step,
                 args=[NUM_ITERATIONS, train_iterator, model, opt])
```

IPU에서 동작시킬 부분에 @tf.function 데코레이터 사용

IPUStrategy의 scope 내에서 strategy.run()을 통해 실행

TENSORFLOW 2

IPU-optimized operations

[12.1. Image operations](#)

[12.2. Matmul serialisation](#)

[12.3. LSTM and GRU](#)

[12.4. Dropout](#)

[12.5. Embedding lookup](#)

[12.6. Group normalisation](#)

[12.7. Instance normalisation](#)

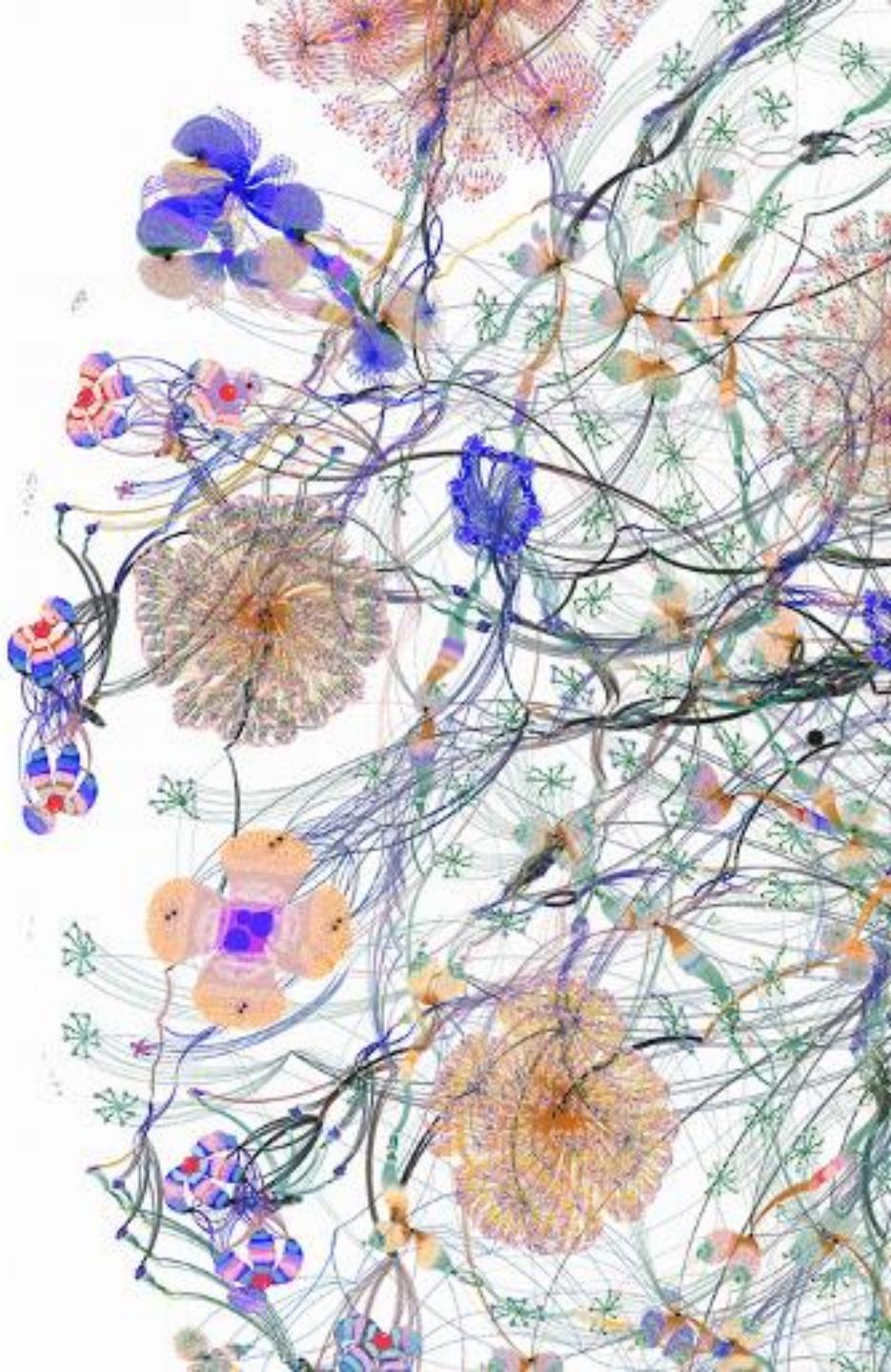
[12.8. Layer normalisation](#)

[12.9. GeLU activation](#)

[12.10. Sequence slice](#)

[12.11. Histogram](#)

HANDS-ON TUTORIAL



GRAPHCORE COMMAND LINE TOOLS

- gc-monitor : 현재 파티션 내 ipu 하드웨어 정보 및 ipu 프로세스 확인

gc-monitor								
Partition: 'pt-minkyuk-16-ipus' has 16 reconfigurable IPUs								
IPU-M	Serial	ICU FW	Type	Server version	ID	PCIe ID	Routing	GWSW
10.1.5.21	0046.0002.8204521	2.4.4	M2000	1.9.0	0	3	DNC	2.4.2
10.1.5.21	0046.0002.8204521	2.4.4	M2000	1.9.0	1	2	DNC	2.4.2
10.1.5.21	0046.0001.8204521	2.4.4	M2000	1.9.0	2	1	DNC	2.4.2
10.1.5.21	0046.0001.8204521	2.4.4	M2000	1.9.0	3	0	DNC	2.4.2
10.1.5.22	0012.0002.8204521	2.4.4	M2000	1.9.0	4	3	DNC	2.4.2
10.1.5.22	0012.0002.8204521	2.4.4	M2000	1.9.0	5	2	DNC	2.4.2
10.1.5.22	0012.0001.8204521	2.4.4	M2000	1.9.0	6	1	DNC	2.4.2
10.1.5.22	0012.0001.8204521	2.4.4	M2000	1.9.0	7	0	DNC	2.4.2
10.1.5.23	0040.0002.8204521	2.4.4	M2000	1.9.0	8	3	DNC	2.4.2
10.1.5.23	0040.0002.8204521	2.4.4	M2000	1.9.0	9	2	DNC	2.4.2
10.1.5.23	0040.0001.8204521	2.4.4	M2000	1.9.0	10	1	DNC	2.4.2
10.1.5.23	0040.0001.8204521	2.4.4	M2000	1.9.0	11	0	DNC	2.4.2
10.1.5.24	0050.0002.8204521	2.4.4	M2000	1.9.0	12	3	DNC	2.4.2
10.1.5.24	0050.0002.8204521	2.4.4	M2000	1.9.0	13	2	DNC	2.4.2
10.1.5.24	0050.0001.8204521	2.4.4	M2000	1.9.0	14	1	DNC	2.4.2
10.1.5.24	0050.0001.8204521	2.4.4	M2000	1.9.0	15	0	DNC	2.4.2
No attached processes								

- vipu-admin list partitions : IPU-POD 내 파티션 정보 확인

SETUP

1. Poplar SDK 활성화

- source SDK_PATH/poplar*/enable.sh
- source SDK_PATH/popart*/enable.sh

2. 가상환경 생성 및 활성화

- virtualenv -p python3 venv_tutorial
- source venv_tutorial/bin/activate

3. 패키지 설치

- (pytorch)pip install SDK_PATH/poptorch*.whl
- (tensorflow)pip install SDK_PATH/tensorflow-2*amd*.whl (tf1/tf2, amd/intel 지정)

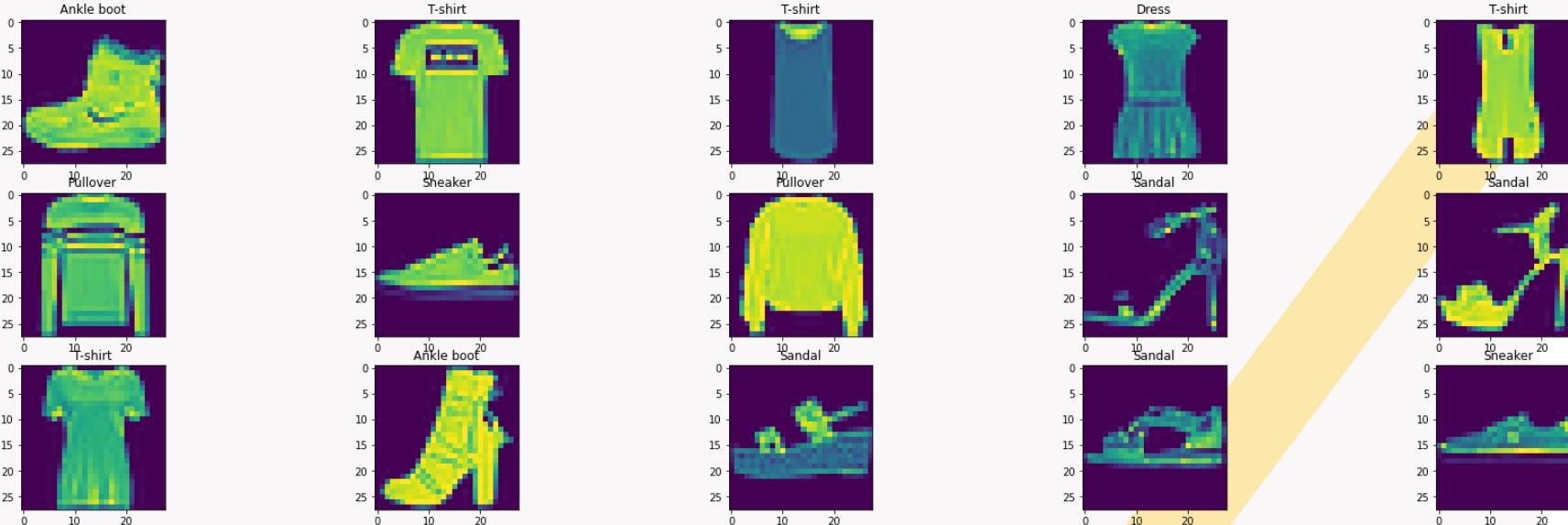
Package	Version
dataclasses	0.8
importlib-resources	5.4.0
pip	21.3.1
pkg_resources	0.0.0
poptorch	2.4.0+40669
setuptools	59.6.0
torch	1.10.0+cpu
tqdm	4.64.0
typing_extensions	4.1.1
wheel	0.37.1
zipp	3.6.0

poptorch 설치 시 호환되는 torch 버전 자동으로 함께 설치

```
(venv_tutorial) minkyuk@pod642:~/projects/NHN/hands_on_tutorials/day01$ ls ~/sdks/2.4.0/tensorflow-
tensorflow-1.15.5+gc2.4.0+139618+803bd5d97f3+amd_znver1-cp36-cp36m-linux_x86_64.whl      tensorflow-2.4.4+gc2.4.0+139613+8debb698097+amd_znver1-cp36-cp36m-linux_x86_64.whl
tensorflow-1.15.5+gc2.4.0+139620+803bd5d97f3+intel_skylake512-cp36-cp36m-linux_x86_64.whl  tensorflow-2.4.4+gc2.4.0+139619+8debb698097+intel_skylake512-cp36-cp36m-linux_x86_64.whl
```

PYTORCH TUTORIAL

FASHION MNIST EXAMPLE



- pytorch_gpu.py : GPU 코드 (IPU-POD에서 실행 시 CPU로 동작)
- pytorch_ipu.py : 최소한의 코드 변환으로 IPU 동작하는 코드
- requirements.txt : pip install -r requirements.txt

PYTORCH TUTORIAL

FASHION MNIST EXAMPLE

- 패키지 import

```
3 import torch
4 import torchvision
5 import torch.nn as nn
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8 from sklearn.metrics import accuracy_score, confusion_matrix
9 from torch.utils import data as torch_data
```

```
3 import torch
4 import torchvision
5 import torch.nn as nn
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8 from sklearn.metrics import accuracy_score, confusion_matrix
9+ import poptorch
```

- Training 시에는 Loss를 함께 리턴하는 wrapper class 정의

```
33+
34+class ModelwithLoss(nn.Module):
35+    def __init__(self, model, criterion):
36+        super().__init__()
37+        self.model = model
38+        self.criterion = criterion
39+
40+    def forward(self, x, labels=None):
41+        output = self.model(x)
42+        if labels is not None:
43+            loss = self.criterion(output, labels)
44+            return output, poptorch.identity_loss(loss, reduction='sum')
45+
46+
47+
```

PYTORCH TUTORIAL

FASHION MNIST EXAMPLE

- Device를 cuda로 세팅하는 부분 제거

```
33 if __name__ == '__main__':
34     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
35     print(device)
36
52     model = ClassificationModel()
53     model.train()
54     model = model.to(device)
48 if __name__ == '__main__':
67     model = ClassificationModel()
68     model.train()
```

- IPU 옵션 세팅

```
64+     opts = poptorch.Options()
65+     opts.enableExecutableCaching('cache')
66+
```

- Poptorch의 DataLoader / Optimizer 사용

```
56     train_dataloader = torch_data.DataLoader(train_dataset,
57                                                 batch_size=16,
58                                                 shuffle=True)
59
60     optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
70+
71+     train_dataloader = poptorch.DataLoader(opts,
72                                             train_dataset,
73                                             batch_size=16,
74                                             shuffle=True)
75+     optimizer = poptorch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

PYTORCH TUTORIAL

FASHION MNIST EXAMPLE

- Pyptorch의 trainingModel / inferenceModel 생성

```
78+     model = ModelwithLoss(model, criterion)
79+     poptorch_model = poptorch.trainingModel(model,
80+                                              options=opts,
81+                                              optimizer=optimizer)
82+
```

- Training Loop 불필요한 부분 제거

```
64     for epoch in range(1, epochs + 1):
65         bar = tqdm(train_dataloader, total=len(train_dataloader))
66         bar.set_description(f'[Epoch {epoch:02d}]')
67         for data, labels in bar:
68             data = data.to(device)
69             labels = labels.to(device)
70
71             optimizer.zero_grad()
72             output = model(data)
73             loss = criterion(output, labels)
74             loss.backward()
75             optimizer.step()
76
77             bar.set_postfix({"Loss": torch.mean(loss).item()})
```

```
84     for epoch in range(1, epochs + 1):
85         bar = tqdm(train_dataloader, total=len(train_dataloader))
86         bar.set_description(f'[Epoch {epoch:02d}]')
87         for data, labels in bar:
88             output, loss = poptorch_model(data, labels)
89             bar.set_postfix({"Loss": torch.mean(loss).item()})
```

PYTORCH TUTORIAL

FASHION MNIST EXAMPLE

■ `poptorch.identity_loss()`

```
class ModelwithLoss(nn.Module):
    def __init__(self, model, criterion):
        super().__init__()
        self.model = model
        self.criterion = criterion

    def forward(self, x, labels=None):
        output = self.model(x)
        if labels is not None:
            loss = self.criterion(output, labels)
            return output, poptorch.identity_loss(loss, reduction='sum')
        return output
```

- `poptorch`의 `backward()`에서 자동으로 인식하기 위해 해당 텐서가 Loss임을 명시
- 일반적인 `torch`의 Loss 객체를 바로 사용하는 경우 불필요하지만 Custom Loss의 경우 필수

`poptorch.identity_loss(x, reduction)`

Marks a tensor as being part of the loss calculation and, as such, will backpropagate through it in the PopTorch autograd.

This function should be called on the (final) loss of a model so that it is used as the start of backpropagation. This is equivalent to calling `x.backward()` on a tensor `x` when running on the CPU.

This function is necessary to combine multiple losses into a custom loss. It ensures that the tensor is part of the loss calculation and, as such, should be part of the backpropagation in PopTorch autograd.

Multiple calls to `identity_loss` can be made inside the same model provided they are all dependant: all marked losses must be traceable into a single final tensor itself marked by a call to `identity_loss` otherwise an error is raised.

Parameters:

- `x` (`torch.Tensor`) – The calculated loss.
- `reduction` (`str`) – Reduce the loss output as per PyTorch loss semantics. Supported values are:
 - "sum": Sum the losses.
 - "mean": Take the mean of the losses.
 - "none": Don't reduce the losses.

Returns: The loss tensor with the specified reduction applied.

Return type: `torch.Tensor`

PYTORCH TUTORIAL

FASHION MNIST EXAMPLE

- enableExecutableCaching()

```
opts = poptorch.Options()
opts.enableExecutableCaching('cache')
```

- 컴파일된 연산 그래프를 캐싱하고 이후 동일한 연산 그래프 호출 시 컴파일 없이 바로 로드하여 사용

enableExecutableCaching(*path*)

Load/save Poplar executables to the specified *path*, using it as a cache, to avoid recompiling identical graphs.

Parameters: *path* (*str* ) – File path for Poplar executable cache store; setting *path* to None`` disables executable caching.

Return type: *poptorch.Options*

TENSORFLOW2 TUTORIAL

MNIST EXAMPLE



0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

- tensorflow_gpu.py : GPU 코드 (IPU-POD에서 실행 시 CPU로 동작)
- tensorflow_ipu.py : 최소한의 코드 변환으로 IPU 동작하는 코드
- tensorflow_ipu_custom_training.py : custom training loop 예시 코드

TENSORFLOW2 TUTORIAL

MNIST EXAMPLE

■ IPUOutfeedQueue()

```
#  
# The host side queue  
#  
outfeed_queue = ipu.ipu_outfeed_queue.IPUOutfeedQueue()  
  
#  
# A custom training loop  
#  
@tf.function(experimental_compile=True)  
def training_step(steps_per_execution, iterator, in_model, optimizer):  
    for _ in tf.range(steps_per_execution):  
        features, labels = next(iterator)  
        with tf.GradientTape() as tape:  
            predictions = in_model(features, training=True)  
            prediction_loss = tf.keras.losses.sparse_categorical_crossentropy(  
                labels, predictions)  
            loss = tf.reduce_mean(prediction_loss)  
            grads = tape.gradient(loss, in_model.trainable_variables)  
            optimizer.apply_gradients(zip(grads, in_model.trainable_variables))  
  
        outfeed_queue.enqueue(loss)
```

```
class tensorflow.python.ipu.ipu_outfeed_queue.IPUOutfeedQueue(outfeed_mode=None,  
device_ordinal=None, buffer_depth=1, optimise_latency=False)
```

Generates and adds outfeed enqueue/dequeue operations to the graph.

An outfeed is the counterpart to an infeed and manages the transfer of data (like tensors, tuples or dictionaries of tensors) from the IPU graph to the host.

The queue has two modes of operation - outfeed all or outfeed last. In outfeed all mode every element that is enqueued will be stored for a subsequent dequeue. All of the enqueued elements will be returned when the dequeue operation is run. This is the default behaviour.

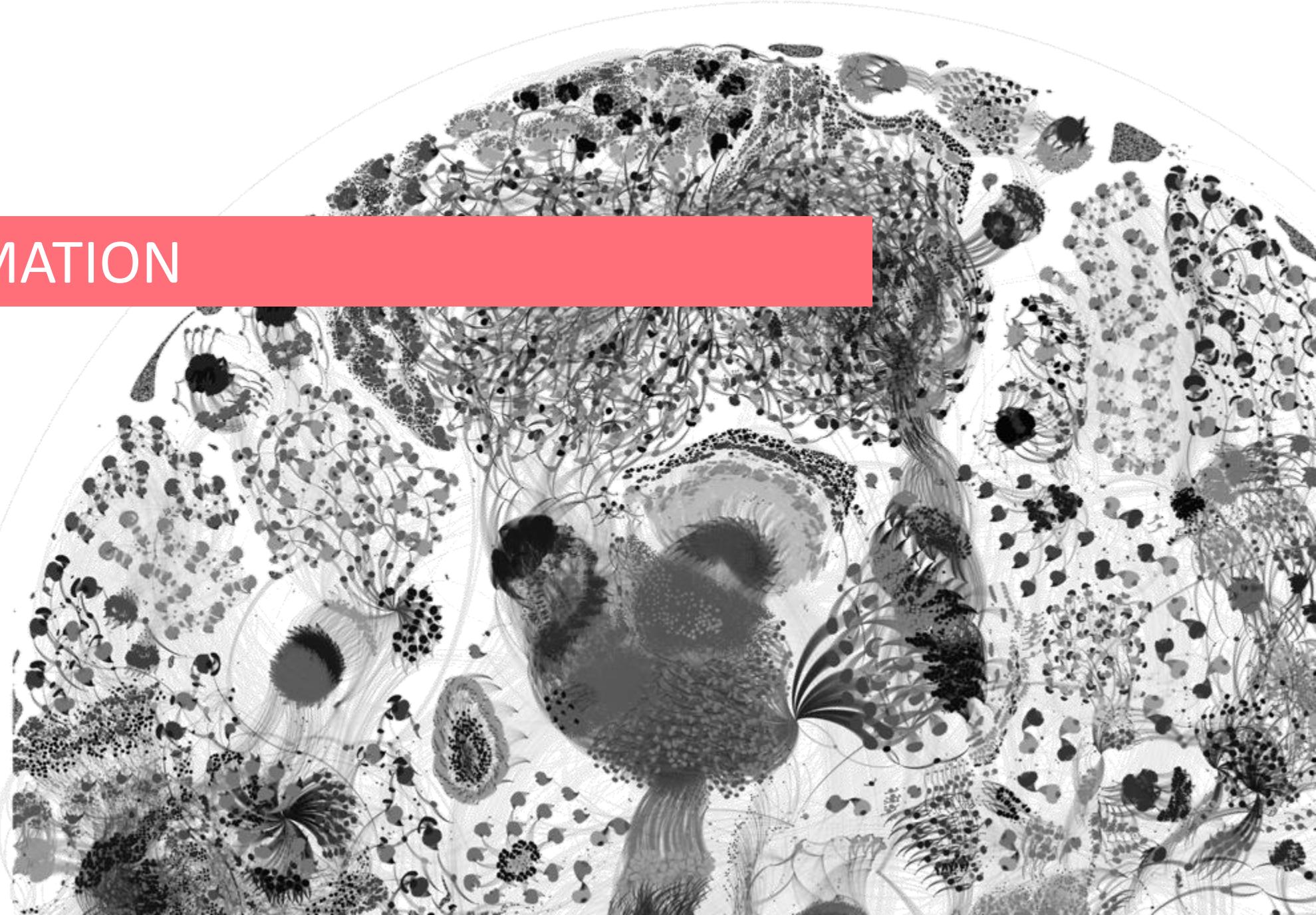
In outfeed last mode only the last enqueued element is stored. The dequeue operation will in this case return a single element.

```
__init__(outfeed_mode=None, device_ordinal=None, buffer_depth=1, optimise_latency=False)
```

Creates an IPUOutfeedQueue object.

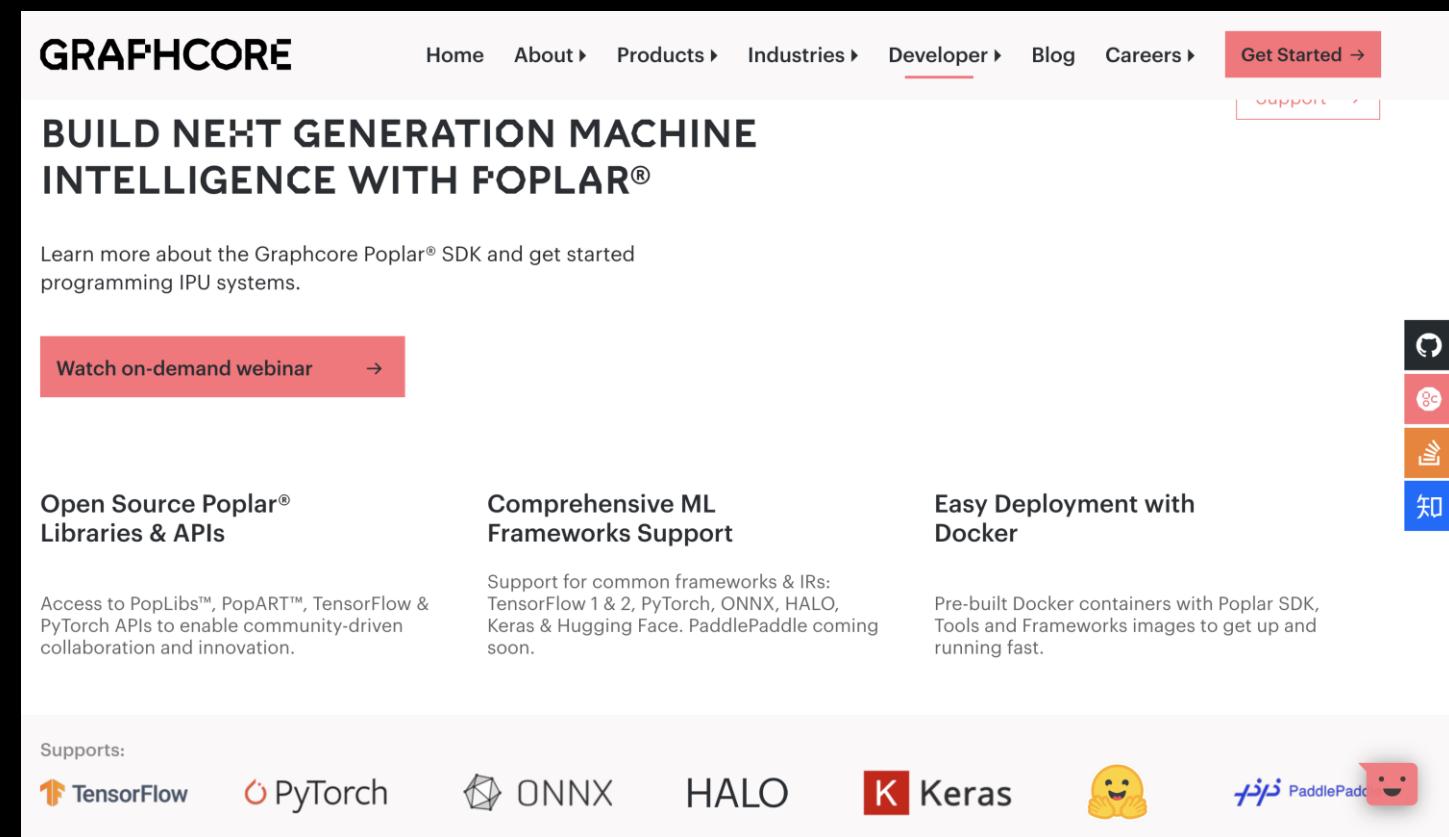
- IPU 연산의 결과값을 호스트로 리턴하는 경우
- IPUOutfeedQueue를 활용

MORE INFORMATION



DEVELOPER PORTAL

<https://graphcore.ai/developer>



The screenshot shows the Graphcore Developer Portal homepage. At the top, there's a navigation bar with links: Home, About ▾, Products ▾, Industries ▾, **Developer** (underlined), Blog, Careers ▾, and Get Started ▾. Below the navigation is a large header with the text "GRAPHCORE" and "BUILD NEXT GENERATION MACHINE INTELLIGENCE WITH POPLAR®". A sub-header below it says "Learn more about the Graphcore Poplar® SDK and get started programming IPU systems." There's a red button labeled "Watch on-demand webinar" with a right-pointing arrow. To the right of the main content area, there are three columns: "Open Source Poplar® Libraries & APIs" (describing access to PopLibs™, PopART™, TensorFlow & PyTorch APIs), "Comprehensive ML Frameworks Support" (mentioning support for TensorFlow 1 & 2, PyTorch, ONNX, HALO, Keras & Hugging Face, with PaddlePaddle coming soon), and "Easy Deployment with Docker" (describing pre-built Docker containers). At the bottom, there's a "Supports:" section with icons for TensorFlow, PyTorch, ONNX, HALO, Keras, and PaddlePaddle, each accompanied by a small smiley face icon.

GRAPHCORE

Home About ▾ Products ▾ Industries ▾ **Developer** Developer Blog Careers ▾ Get Started ▾

BUILD NEXT GENERATION MACHINE INTELLIGENCE WITH POPLAR®

Learn more about the Graphcore Poplar® SDK and get started programming IPU systems.

Watch on-demand webinar →

Open Source Poplar® Libraries & APIs

Access to PopLibs™, PopART™, TensorFlow & PyTorch APIs to enable community-driven collaboration and innovation.

Comprehensive ML Frameworks Support

Support for common frameworks & IRs: TensorFlow 1 & 2, PyTorch, ONNX, HALO, Keras & Hugging Face. PaddlePaddle coming soon.

Easy Deployment with Docker

Pre-built Docker containers with Poplar SDK, Tools and Frameworks images to get up and running fast.

Supports:

TensorFlow PyTorch ONNX HALO Keras PaddlePaddle

GRAPHCORE DOCUMENTS

<https://docs.graphcore.ai>



The screenshot shows a dark-themed documentation site for Graphcore. At the top, the word "GRAPHCORE" is written in large, bold, white capital letters. Below it, a dark sidebar contains the text "Graphcore Documents" and "Version: latest". A "Search docs" input field is also present. The main content area lists several categories: "Software Documents", "Hardware Documents", "Examples and Tutorials", "Document Updates", and "Graphcore End User License Agreement". At the bottom of the page, there are links for "Read the Docs" and a dropdown menu showing "v: latest". To the right of the screenshot, a vertical list of document types is displayed.

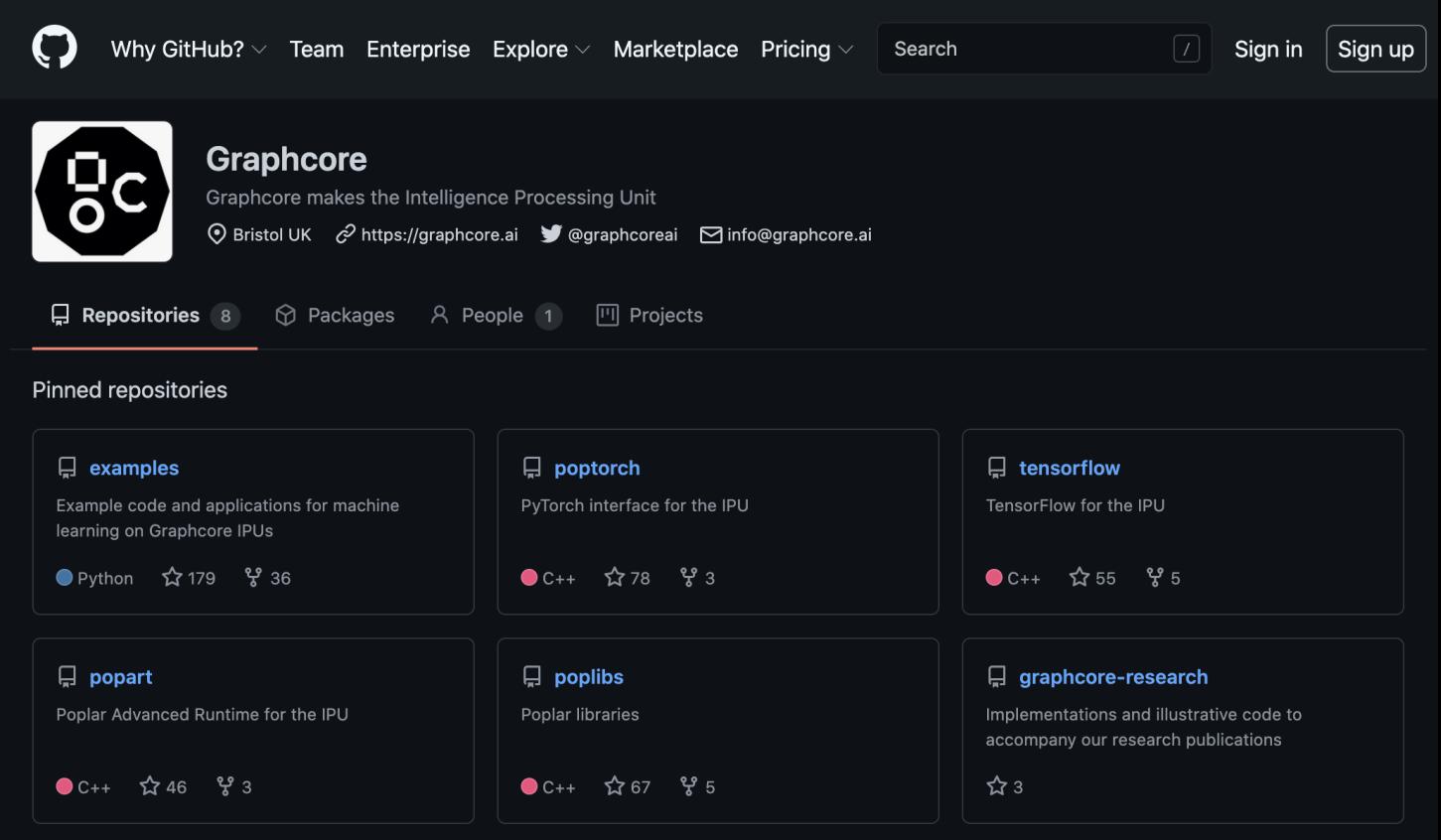
GRAPHCORE DOCUMENTS

- Software Documents
 - Getting Started
 - TensorFlow
 - PyTorch
 - PopART
 - Poplar Graph Programming Framework
 - Technical Notes
 - Profiling and Debugging
 - Open Source Software
 - License Agreements
- Hardware Documents
 - Getting Started
 - IPU-POD Systems
 - Supporting Tools
- Examples and Tutorials
 - Simple examples
 - Application examples
 - Tutorials
- Document Updates
 - April 2021
 - March 2021
 - November 2020
- Graphcore End User License Agreement



GRAPHCORE GITHUB

<https://github.com/graphcore>



The screenshot shows the GitHub profile page for Graphcore. At the top, there's a navigation bar with links for Why GitHub?, Team, Enterprise, Explore, Marketplace, Pricing, a search bar, and options to Sign in or Sign up. Below the navigation is the Graphcore logo, which is a black octagon containing a white 'bc' monogram. The repository name 'Graphcore' is displayed in bold capital letters, followed by the tagline 'Graphcore makes the Intelligence Processing Unit'. Below this, it shows the location 'Bristol UK', the website 'https://graphcore.ai', and social media links for Twitter (@graphcoreai) and email (info@graphcore.ai). A navigation bar below the bio includes 'Repositories' (8), 'Packages', 'People' (1), and 'Projects'. The main content area is titled 'Pinned repositories' and lists six repositories:

- examples**: Example code and applications for machine learning on Graphcore IPUs. Language: Python. Stars: 179. Forks: 36.
- poptorch**: PyTorch interface for the IPU. Language: C++. Stars: 78. Forks: 3.
- tensorflow**: TensorFlow for the IPU. Language: C++. Stars: 55. Forks: 5.
- popart**: Poplar Advanced Runtime for the IPU. Language: C++. Stars: 46. Forks: 3.
- poplibs**: Poplar libraries. Language: C++. Stars: 67. Forks: 5.
- graphcore-research**: Implementations and illustrative code to accompany our research publications. Stars: 3.



THANK YOU

Minkyu Kim

minkyuk@graphcore.ai