# Practical guide on PyTorch inference using AWS Inferentia

**Keita Watanabe**  
Senior Solutions Architect, AI/ML Frameworks  
Amazon Webservices Japan

## Abstract

In this session, we will go through step-by-step how to conduct the inference process of machine learning models using Inferentia. In addition, we compare the inference performance with GPU and discuss the cost advantage. In the later part of the session, we will also cover model deployment on Kubernetes.

### AWS Inferentia

- 4 Neuron Cores with up to 128 TOPS
- Two-stage memory hierarchy: Large on-chip cache + 8 GB DRAM
- Supports FP16, BF16, INT8 data types with mixed precision
- 1 to 16 Inferentia cores per instance with high-speed interconnect

### Amazon EC2 Inf1 Instances

<table>
<thead>
<tr>
<th>Instance Size</th>
<th>vCPUs</th>
<th>Memory (GiB)</th>
<th>Storage</th>
<th>Network Bandwidth</th>
<th>EBS Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf1.xlarge</td>
<td>4</td>
<td>8</td>
<td>1 EBS</td>
<td>Up to 24 Gbps</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>inf1.2xlarge</td>
<td>8</td>
<td>16</td>
<td>1 EBS</td>
<td>Up to 4.75 Gbps</td>
<td>Up to 4.75 Gbps</td>
</tr>
<tr>
<td>inf1.6xlarge</td>
<td>24</td>
<td>48</td>
<td>4 EBS</td>
<td>1 Gbps</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>inf1.24xlarge</td>
<td>96</td>
<td>192</td>
<td>16 EBS</td>
<td>1 Gbps</td>
<td>100 Gbps</td>
</tr>
</tbody>
</table>

## PyTorch Neuron Model Tracing

```python
import torch.nn as nn
import torch

class NeuronModel(nn.Module):
    def __init__(self):
        super(NeuronModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 5)
        self.conv2 = nn.Conv2d(64, 64, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 12 * 12, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 12 * 12)
        x = self.fc1(x)
        return x
```

1. Load the model and set it to evaluation mode
2. Compile a Neuron model (torch.nn.ModelTrace(model, model_size=inf1.xlarge))
3. Compile a model with NeuronCorePipeline (NeuronCore.pipeline(model, model_size=inf1.xlarge))

## Data Parallel Inference

Data Parallelism is a form of parallelization across multiple devices or cores, referred to as nodes. Each node contains the same model and parameters, but data is distributed across the different nodes.

## Neuron SDK

- Neuron compiler
- Neuron runtime
- Profiling tools

### Neuron Core Pipeline

NeuronCore Pipeline refers to the process of sharding a compute-graph across multiple NeuronCores, caching the model parameters in each core’s on-chip memory (cache), and then streaming inference requests across the cores in a pipelined manner.

## Cost Performance Comparison

<table>
<thead>
<tr>
<th>Model</th>
<th>Bert-Large</th>
<th>Bert-Base</th>
<th>Yolov5</th>
<th>Resnet50</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS G4dn.xl</td>
<td>$0.000</td>
<td>$0.025</td>
<td>$0.050</td>
<td>$0.075</td>
</tr>
<tr>
<td>AWS G5.xl</td>
<td>$0.000</td>
<td>$0.025</td>
<td>$0.050</td>
<td>$0.075</td>
</tr>
<tr>
<td>AWS Inf1.xl</td>
<td>$0.300</td>
<td>$0.600</td>
<td>$0.900</td>
<td>$0.300</td>
</tr>
</tbody>
</table>

## EKS Deployment

- AWS Deep Learning Containers
- AWS Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)

Documentation, examples, and support: [github.com/aws/aws-neuron-sdk](https://github.com/aws/aws-neuron-sdk)