Edge ML

- Low Latency
- Privacy & Security
- Low Power
ML Deployment on the Edge

Deployed model can have *mismatching* performance, and often there is *little clue* what this may happen.

There is a *disconnect* between model developers and app developers.
The Disconnect

Model Performance
- Accuracy
- Latency & throughput

Deployment Performance
- End-to-end latency
- Power consumption
- Memory footprint

Design choices made for training are often lost during the handoff. Design choices may conflict with heterogeneous hardware.
Example: Deploying An Image Classifier

**Input Preprocessing**

- **Channel Extraction**
  - YUV→RGB

- **Resizing**
  - 1920x1080 → 224x224

- **Numerical Conversion**
  - [0, 255], [-128, 127]
  - [0.0, 1.0], [-1.0, 1.0]

- **Orientation**
  - 0, 90, 180, 270

**Model Optimization**

- **Constant Folding**
- **Activation Fusion**
- **Quantization**
  - Batch norm, ReLU
  - Int8, Float16

**Cloud Model** → **Edge Model** → **Class output**
What could possibly go wrong?

- Preprocessing bugs

- Quantization issues

- Kernel optimization v.s. heterogeneous hardware
Challenge: How to debug?

- **Low awareness** of potential issues
- **Little visibility** into the edge black box
- **Tedious reverse engineering** to debug
ML-EXray Contributions

- **Visibility**: Instrumentation APIs for layer-level details

- **Bridging the disconnect**: Reference pipelines and data playback

- **Automated debugging**: Programming model for deployment validation

- **Awareness of deployment issues**: Uncovering deployment issues and their impact
ML-EXray Overview

- Easy-to-use API w/ low overhead
- End-to-end performance validation
- Per-layer latency measurement and output validation
- Extensibility: user-defined logging, reference pipeline, and assertions
**Customization**

**Assertion functions**: users can define custom debugging assertions to validate suspected issues, such as input channel arrangement, orientation, and normalization range.
Customization

**Log elements**: users can instrument the app at any point throughout the pipeline to log custom variables for different debugging purposes.
**Customization**

**Reference Pipelines**: users can provide alternative pipelines as references, such as a previous successful deployment pipeline on a different device.
**ML-EXray Contributions**

- **Visibility**: Instrumentation APIs for layer-level details

- **Bridging the disconnect**: Reference pipelines and data playback

- **Automated debugging**: Programming model for deployment validation

- **Awareness of deployment issues**: Uncovering deployment issues and their impact
Instrumentation APIs & Data Model

A suite of APIs in C++, Java, Python

```cpp
// (C++)
MLEXray->on_inf_start();
TfLiteStatus s = m_interpreter->Invoke();
MLEXray->on_inf_stop(&m_interpreter);
```

Data Model

- **Input / Output**
  - Model I/O
  - Per-layer I/O
  - Pre-processing I/O
  - User-defined I/O

- **Performance**
  - End-to-end Latency
  - Per-layer Latency
  - Memory usage

- **Peripheral**
  - Orientation
  - Motion
  - Lighting

Assertion function

```python
def channel_assertion(edge_out, ref_out) {
    if not np.allclose(edge_out, ref_out):
        edge_out = cv2.cvtColor(edge_out, cv2.COLOR_BGR2RGB)
    if np.allclose(edge_out, ref_out):
        raise AssertionError('BGR->RGB')
}
ML-EXray Contributions

- **Visibility**: Instrumentation APIs for layer-level details

- **Bridging the disconnect**: Reference pipelines and data playback

- **Automated debugging**: Programming model for deployment validation

- **Awareness of deployment issues**: Uncovering deployment issues and their impact
Reference Pipelines and Playback

Input / Output
- Model I/O
- Per-layer I/O
- Pre-processing I/O
- User-defined I/O

Performance
- End-to-end Latency
- Per-layer Latency
- Memory usage

Peripheral
- Orientation
- Motion
- Lighting

Reference ML Pipeline
- Cloud model
- Correct pre-processing
- No model optimization
- Default standard kernels

Edge ML Pipeline
- Edge model
- User pre-processing code
- Optimized model
- Edge device specific kernels (e.g. optimized op resolver)
A Basic Reference Pipeline Example

### Input Preprocessing

- **Channel Extraction**
  - YUV->RGB

- **Resizing**
  - 1920x1080 -> 224x224

- **Numerical Conversion**
  - [0,255], [-128,127]

- **Orientation**
  - 0, 90, 180, 270

### Model Optimization

- **Constant Folding**
  - Batch norm

- **Activation Fusion**
  - ReLU

- **Quantization**
  - Int8, Float16

Cloud Model → Lightweight Interpreter → OpResolver

Edge Model
User-defined Reference Pipelines

**Input Preprocessing**
- Channel Extraction: YUV->RGB
- Resizing: 1920x1080 -> 224x224
- Numerical Conversion: [0,255], [-128,127]
- Orientation: [0,0.9,180,270]

**Model Optimization**
- Constant Folding
- Activation Fusion: Batch norm, ReLU
- Quantization: Int8, Float16

**Unoptimized vs optimized**
- Cloud Model
- Edge Model

**Lightweight Interpreter**
- OpResolver
User-defined Reference Pipelines

**Input Preprocessing**
- Channel Extraction: YUV→RGB
- Resizing: 1920x1080 → 224x224
- Numerical Conversion: [0,255], [-128,127]
- Orientation: 0, 90, 180, 270

**Model Optimization**
- Constant Folding
- Activation Fusion: Batch norm → ReLU
- Quantization: Int8, Float16
- Lightweight Interpreter: OpResolver

Cloud Model → Edge Model
Unquantized vs quantized
User-defined Reference Pipelines

Input Preprocessing
- Channel Extraction: YUV->RGB
- Resizing: 1920x1080 -> 224x224
- Numerical Conversion: [0,255], [-128,127]
- Orientation: 0, 90, 180, 270

Model Optimization
- Constant Folding
- Activation Fusion: Batch norm, ReLU
- Quantization: Int8, Float16

Output
- Lightweight Interpreter
- OpResolver

Reference Op vs optimized Op
User-defined Reference Pipelines

Input Preprocessing
- Channel Extraction: YUV->RGB
- Resizing: 1920x1080 -> 224x224
- Numerical Conversion: [0,255], [-128,127]
- Orientation: 0,90,180,270

Model Optimization
- Constant Folding
- Activation Fusion: Batch norm, ReLU
- Quantization: Int8, Float16

Cloud Model
- OpResolver

Edge Model
- Lightweight Interpreter

EdgeTPU vs Arduino
Android vs Ubuntu
Reference Pipelines Repo

Current reference pipelines released
• MobileNet v1
• MobileNet v2
• MobileNet v3
• Inception v3
• Densenet 121
• Resnet50 v2
• ...

Open to public pull requests
ML-EXray Contributions

- **Visibility**: Instrumentation APIs for layer-level details

- **Bridging the disconnect**: Reference pipelines and data playback

- **Automated debugging**: Programming model for deployment validation

- **Awareness of deployment issues**: Uncovering deployment issues and their impact
Deployment Validation and Assertions

- **End-to-end Performance Benchmark**

- **Per-layer Validation**

- **Running Assertions for Root-cause**

Accuracy Validation

Per-layer Validation

Root-cause Analysis
Evaluation

• Wide applicability

• System overhead

• Pre-processing bugs and impact

• Quantization issues and impact

• Sub-optimal kernels and latency
## Wide Applicability

ML-EXray catches a wide range of deployment issues across different tasks.
**System Overhead**

**Lightweight**

- Latency: a few ms increase
- Memory: < 5 MB

**Easy to use**

- a few LoC for different validation target

---

**Image Classifier (MobileNetv2) on Android Phones**

<table>
<thead>
<tr>
<th></th>
<th>Lat (ms) CPU only</th>
<th>Lat (ms) GPU enabled</th>
<th>Mem (MB)</th>
<th>Disk (KB/Frm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel 4</td>
<td>128.2±6.1</td>
<td>16.7±0.3</td>
<td>6.42</td>
<td>-</td>
</tr>
<tr>
<td>P4(Inst)</td>
<td>129.6±5.0</td>
<td>19.1±0.6</td>
<td>10.12</td>
<td>0.41</td>
</tr>
<tr>
<td>Pixel 3</td>
<td>157.0±6.7</td>
<td>28.4±0.4</td>
<td>9.26</td>
<td>-</td>
</tr>
<tr>
<td>P3(Inst)</td>
<td>158.3±7.3</td>
<td>30.0±0.5</td>
<td>12.37</td>
<td>0.41</td>
</tr>
</tbody>
</table>

---

**Instrumentation & Assertion Line-of-Code**

<table>
<thead>
<tr>
<th>Debugging Target</th>
<th>Line of Code</th>
<th>W/ ML-EXRAY</th>
<th>W/O ML-EXRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inst Asrt</td>
<td>Total</td>
<td>Inst Asrt</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>1 3</td>
<td>4</td>
<td>18 7</td>
</tr>
<tr>
<td>Quantization</td>
<td>4 9</td>
<td>13</td>
<td>82 183</td>
</tr>
<tr>
<td>Lat. &amp; Mem.</td>
<td>4 4</td>
<td>8</td>
<td>14 8</td>
</tr>
<tr>
<td>Per-layer Lat.</td>
<td>2 6</td>
<td>8</td>
<td>14 90</td>
</tr>
</tbody>
</table>
Eradicating these issues, ML-EXray can correct model performance by 5 - 40%
Localizing Quantization Issues

Quantized MobileNet not working well

Per-layer validation can easily localize the issue
# Sub-optimal Kernels and Latency

Latency by layer type (MobileNet v2)

<table>
<thead>
<tr>
<th>Layer Type (Count)</th>
<th>Mobile (ms)</th>
<th>Mobile Quant (ms)</th>
<th>Mobile Quant Ref (ms)</th>
<th>Emulator (x86) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-Conv(17)</td>
<td>95.4</td>
<td>22.7</td>
<td>2885.2</td>
<td>120.0</td>
</tr>
<tr>
<td>Conv(35)</td>
<td>23.5</td>
<td>32.3</td>
<td><strong>18662.3</strong></td>
<td><strong>1409.8</strong></td>
</tr>
<tr>
<td>FC(1)</td>
<td>7.4</td>
<td>7.1</td>
<td>7.0</td>
<td>71.2</td>
</tr>
<tr>
<td>Mean(1)</td>
<td>6.1</td>
<td>5.6</td>
<td>5.0</td>
<td>2.5</td>
</tr>
<tr>
<td>Pad(4)</td>
<td>1.6</td>
<td>18.7</td>
<td>60.8</td>
<td>104.8</td>
</tr>
<tr>
<td>Add(10)</td>
<td>1.5</td>
<td>7.7</td>
<td>99.8</td>
<td>7.0</td>
</tr>
<tr>
<td>Softmax(1)</td>
<td>0.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>Quantize(1)</td>
<td>-</td>
<td>3.3</td>
<td>0.7</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>136.26</strong></td>
<td><strong>97.816</strong></td>
<td><strong>21721.2</strong></td>
<td><strong>1715.7</strong></td>
</tr>
</tbody>
</table>

Reference kernels may be more stable but much slower

Emulator is slow on convolution layers
Debug Your Edge ML

GitHub
https://github.com/hangqiu/ML-EXray

arXiv
https://arxiv.org/abs/2111.04779

hangqiu@stanford.edu

MLSys | 2022
ML-EXray: Visibility into ML Execution on the Edge
Thank you!
ML-EXray: Visibility into ML Deployment on the Edge

Hang Qiu, Ioanna Vavelidou, Jian Li, Evgenya Pergament,
Pete Warden, Sandeep Chinchali, Zain Asgar, Sachin Katti