Towards a Serverless Platform for Edge AI

Thomas Rausch
Waldemar Hummer
Vinod Muthusamy
Alexander Rashed
Schahram Dustdar

@thrauat
Drone

Microsoft Build 2018 // Vision Keynote: https://www.youtube.com/watch?v=rd0R68w3FZ0
Edge AI Accelerators

- Google Edge TPU
- NVIDIA Jetson
- Intel Neural Compute Stick
- Baidu Kunlun
- Microsoft Project BrainWave
- Huawei Atlas
AI Operationalization

ModelOps Platform

Hummer et al., ModelOps: Cloud-based Lifecycle Management for Reliable and Trusted AI. IC2E’19.
Serverless Model

Event (Request) → Trigger → Function

```
def handle(req):
    s3 = boto3.client('s3')
    with open(tmpfile, 'wb') as f:
        s3.download_fileobj('bucket', req['obj'], f)
    data = numpy.load(f)
    m = train_model(data, req['train_params'])
    s3.upload_fileobj(serialize(m), 'bucket', 'model')
    # ...
```
def handle(req):
    s3 = boto3.client('s3')
    with open(tmpfile, 'wb') as f:
        s3.download_fileobj('bucket', req['obj'], f)
    data = numpy.load(f)
    m = train_model(data, req['train_params'])
    s3.upload_fileobj(serialize(m), 'bucket', 'model')
    # ...
A Serverless Platform for Edge AI

AI Workflow Programming Model
- Data and Models as first-class citizens
- Model Selectors
- Policies
- Gates

Execution Platform
- Deviceless function scheduling
- Policy enactment
- Context awareness
- Data locality awareness
@consumes.data(
    selector={'urn': 'mnist:data'},
    holdout=0.2)
@produce.model(
    type='classifier',
    urn='mnist:model')
def train(data: Data, request) -> Model:
    arr = data.to_ndarray()
    return Model(train_model(arr))

def inference(model: Model, request):
    data = request['input']
    # data prep tasks
    prediction = model.estimate(data)

@policy.deadline('2s')
@policy.fn(node='user_device',
            capability='gpu')
@policy.data(network=['company_network'],
             strict=True)

@consumes.model(selector={
    'type': 'image_classifier',
    'data_tags': ['machine_x'],
    'accuracy': '>=0.88'})
def inference(model: Model, request):
    data = request['input']
    # data prep tasks
    prediction = model.estimate(data)

@gate.bias(attribute='age',
            predicate='<0.8')
@gate.drift(metric='confidence',
             predicate='<0.2')
```python
@consumes.model(selector={'urn': 'model:base'})
@consumes.data(batch = 100, selector=...)
@produces.model(type='regressor', urn='model:user:{usr}')</p>
@policy.fn(node = 'local')
@policy.data(network = 'local', strict=True)
def refine(model: Model, data: Data):
    ndarr = data.to_ndarray() # data artifact API
    # transfer learning code
    return refined_model
```
Data Locality Tradeoffs

Deploy the container image to the edge?
OR
Send the data to the cloud?

Cluster Middleware

Data

Container Image

proximity

Edge
Preprocess \[\lambda\] Train \[\lambda\] Inference

Scheduler + Simulator: https://git.dsg.tuwien.ac.at/serverless-edge-ai/sched-sim
def handle(req):
    s3 = boto3.client('s3')
    with open(tmpfile, 'wb') as f:
        s3.download_fileobj('bucket', req['obj'], f)
    data = numpy.load(f)
    m = train_model(data, req['train_params'])
    s3.upload_fileobj(serialize(m), 'bucket', 'model')
    # ...
Discussion

i Feedback

- Correct level of abstraction?
- API/SDK features?
- Validation criteria?

ii Controversial points

- Deviceless model (does it work?)
- Transparent data management

iii Open issues

- Request routing architecture
- Proximity and bandwidth monitoring
- Learning optimal placements

iv Failure risks

- Model too high-level for scheduler
- “Bring-your-own-device” will fail
Backup Slides
Capability-Aware Pipeline Execution

Cluster Middleware

- Process
- Train
- Compress
- Validate
- Deploy

execute

HPC

General Purpose Computing

Embedded AI (NVIDIA Jetson)
Hierarchical Model Deployments

ML Model
- Unfiltered data
- Anonymized data
- Base model deployment

Data centers
Base model training

Edge Resources

IoT

Model refinement

Personal Assistant

Source: Medtronic Sugar.IQ IBM
Platform Perspective

Source: Google Cloud IoT Edge
# Cognitive Assistance Applications


## Table 1. Example wearable cognitive assistance applications.

<table>
<thead>
<tr>
<th>App name</th>
<th>Example input video frame</th>
<th>App description</th>
<th>Symbolic representation</th>
<th>Example guidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td><img src="image" alt="Face Image" /></td>
<td>Jogs user’s memory of a familiar face whose name cannot be recalled. Detects and extracts a tightly cropped image of each face, then applies popular open source face recognizer OpenFace (cmusatyalab.github.io/openface), which is based on a deep neural network (DNN) algorithm. Whispers name of person. Can be used in combination with mood detection algorithms to offer conversational hints.</td>
<td>ASCII text of name</td>
<td>Whispers “Barack Obama”</td>
</tr>
<tr>
<td>Pool</td>
<td><img src="image" alt="Pool Image" /></td>
<td>Helps novice pool player aim correctly. Gives continuous visual feedback (left arrow, right arrow, or thumbs up) as user turns cue stick. Correct shot angle is calculated based on widely used fractional aiming system. Uses color, line, contour, and shape detection. Symbolic representation describes positions of cue ball, object ball, target pocket, and top and bottom of cue stick.</td>
<td><code>&lt;Pocket, object ball, cue ball, cue top, cue bottom&gt;</code></td>
<td></td>
</tr>
<tr>
<td>Ping-Pong</td>
<td><img src="image" alt="Ping-Pong Image" /></td>
<td>Tells novice player to hit ball to left or right, depending on which is more likely to beat opponent. Uses color, line, and optical-flow-based motion detection to detect ball, table, and free space.</td>
<td><code>&lt;InRally, ball position, opponent position&gt;</code></td>
<td>Whispers “Left”</td>
</tr>
</tbody>
</table>