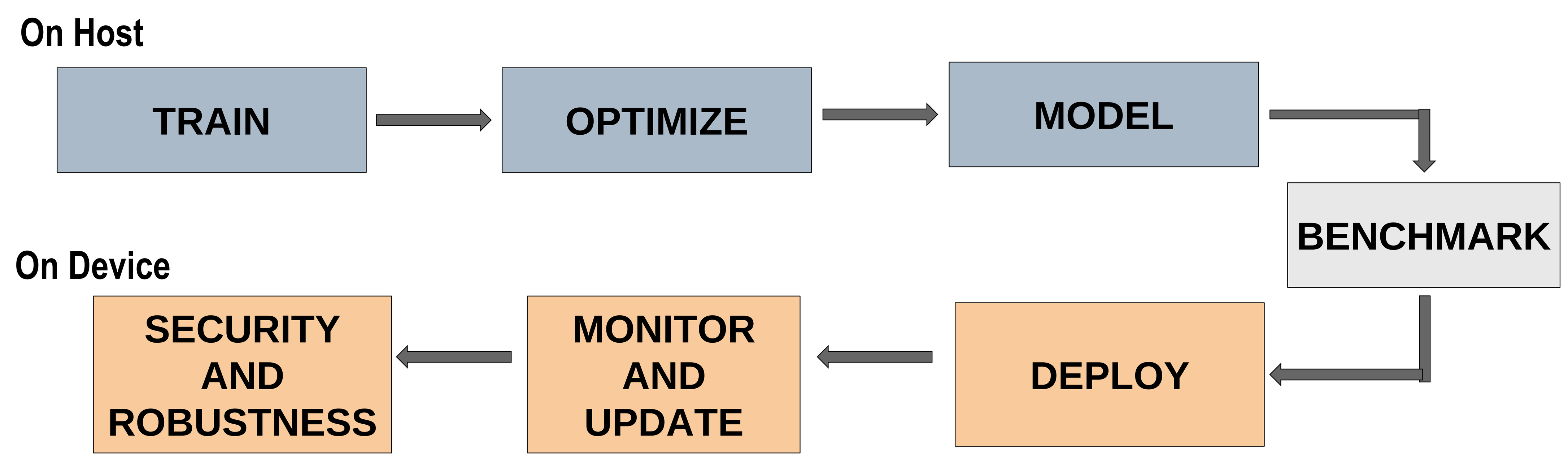
TinyMLOps: Overview, Challenges and Implementation

Avyay Sah[1,2], Soham Chatterjee[2], Archana Vaidheeswaran[2]
Purdue University[1,2]
ScaleDown Team, Singapore[2]

DIFFERENCES

MLOPS	TINYMLOPS
Models are trained and deployed on powerful devices	Models are deployed on resource constrained edge devices
Large models with multiple supported architectures and Ops	Small models with few supported hardwares and Ops
Accuracy and availability are important	Latency, throughput, power consumption are important
Containerization, CI/CD, logging and monitoring is required	No containers; logging and monitoring is difficult to do
Performance checks and model updates are done regularly	Performance checks and model updates are difficult to do
Robustness and Autoscaling to traffic	Scaling is difficult after deployment; Robustness is important

TINYMLOPS ARCHITECTURE



TinyMLOps are a set of best practices that can help you build and deploy machine learning applications on TinyML devices successfully.

Model Optimization

Challenges

Before deploying models, they need to be optimized to get the best performance from the target hardware.

- Optimization mainly reduces model size and operations
- Models can also be optimized for energy consumption, peak RAM usage, latency, throughput etc.
- Optimization can reduce accuracy and increase bias
- Optimization may not increase performance
  - Pruned models are not supported on many systems
  - INT8 operations are not fast on many systems
- Fragmented ecosystem with tools supporting limited hardware devices or optimization techniques causing interoperability issues

Tools and Techniques

- Optimizing while keeping end hardware in mind
- Algorithms
  - Quantization
  - Pruning
  - Weight Sharing and Hashing
- Tools
  - TFLite
  - Edge Impulse
  - OpenVINO
  - ONNX

ScaleDown

```
model=scaledown.load_model(model_file, model_type='tf/pytorch/openvino')
tf_model=scaledown.ConvertToTF.convert(model)
```

Model Training

When training models for TinyML, we need to keep in mind the constraints of the target hardware and account for losses during model optimization

Challenges

- Does the model use unsupported operations
- Can the model architecture be efficiently executed on the target hardware
- Is the model framework supported on the target hardware
- Can we train the model to reduce accuracy drop when optimizing

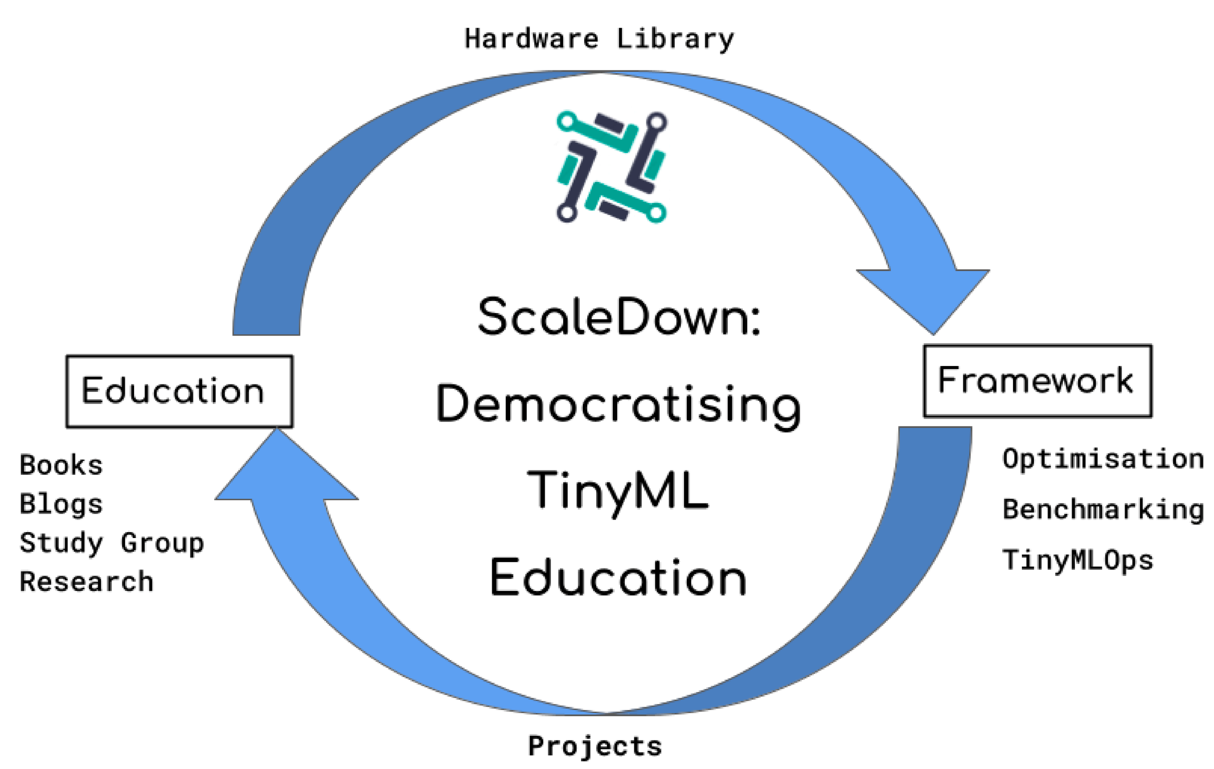
Tools and Techniques

- Architectures and Ops
  - MobileNet and EfficientNet
  - Use simple Ops and layers: Separable Convolutions, Stride, Pooling
  - Do NOT use complex data flows
- Training Frameworks
  - Support for optimization
  - Support for Architectures and Ops
  - TensorFlow, PyTorch
- Algorithms
  - Quantization Aware Training
  - Neural Architecture Search for TinyML
  - Knowledge Distillation

ScaleDown

```
kd=KnowledgeDistillation(teacher, student, optimizer, distillation_loss, student_loss)
loss=kd.train_step(data)
```

SCALEDOWN: AN OPEN SOURCE FRAMEWORK FOR TINYMLOPS



PyTorch ⇄ TensorFlow ⇄ OpenVINO ⇄ TensorRT

Optimization Techniques	Frameworks
Quantization	Pytorch
Pruning	Tensorflow
Knowledge Distillation	OpenVino

- ScaleDown is an open-source project that provides open training and resources as well as fosters collaboration and innovation in TinyML
- Our mission at ScaleDown is to educate students and early career professionals about TinyML and to build tools to make developing TinyML applications easier
- TinyML engineers need to learn not only about machine learning, but also electronics and embedded systems so that they can optimise and deploy models on microcontrollers.
- However, there is a wide gap in the community for learning resources and tools to help beginners and early career professionals learn about TinyML, experiment and make products to build a portfolio to get jobs. To help people better understand this field, we do community work like hosting workshops, study groups and talks. We also create free learning resources like books and courses.

BENCHMARKING

Challenges

Proxy Metrics vs Perceptible Metrics

Proxy Metrics can be determined from the model without deploying: Memory, FLOPs, Accuracy

Perceptible Metrics are device dependent: Latency, Throughput, Energy, Peak RAM Usage

Proxy Metrics do NOT correlate with runtime metrics

This can be due to architectural features of the target device are not used optimally

This results in misleading optimization or NAS

Tools and Techniques

Benchmarks should be done both on-host and on-device to get real runtime metrics

You can get close approximations with services like Renode

Performance metrics should be prioritized, especially for battery systems

Benchmarking tools are giving by

- TFLite
- OpenVINO
- DL Workbench

ScaleDown

```
model.get_report()
```

Model Deployment

- Deployment Architectures
  - Multi-tenancy: Concurrent, Model Placement, Fleetwise
  - Cascade
- How do we package applications for TinyML?
  - Containers are being developed by Hammer of the Gods (HOTG)
- How to reduce battery power consumption
  - Sleep mode
  - Reduce logging and transmitting data
  - Neuromorphic Sensors
- Many Cloud Platforms have tools for TinyML

ScaleDown

```
model.create_deployment_package(target='pi')
```

Help us by

1. Donate Hardware
2. Create learning material
3. Collaborate on projects or research
4. Contribute to our framework
5. Use our library and framework

Roadmap at ScaleDown

Q1:2022: ScaleDown support for other Optimization Techniques, Support for TFLite, OpenVINO, ONNX and PyTorch, Starting Hardware Library to 1 city

Q2:2022: 1st cycle of Hardware Library Completed, ScaleDown support for deployment, benchmarking and model conversion

Q3:2022: Publish 1 research paper, ScaleDown support for TinyMLOps, monitoring, security and model updates, Grow Hardware Library to 2 cities, Complete 2 cycles of Hardware Library

Q4:2022: Publish 3 research papers, ScaleDown support for research, Complete 4 cycles of Hardware Library

Monitor and Update

We need to be able to monitor deployed devices to check for failures, damages, low battery:

- Device to cloud, or Device to Gateway to Cloud
- Monitor using cellular or wifi connection
- Mesh systems when no network connectivity
- TinyML devices need updates for
  - Deploying retrained models
  - Updating firmware
  - Adding functionality
- Updating TinyML devices:
  - Federated Learning and On device training
  - Use Gateways to receive updates and manage a small fleet of TinyML Devices
  - Partial updates should be possible to reduce battery consumption
- Security from Side-Channel and Fault Injection Attacks
- Robustness
  - To sensor failure
  - Data Drift and Environmental Conditions
  - Redundancy