“Embedded ML research at TUM: Moving NN Inference to the Extreme Edge”

Rafael Stahl
[German Area Group] - October 7, 2020
tinyML Talks Sponsors

Additional Sponsorships available – contact Bette@tinyML.org for info
WE USE AI TO MAKE OTHER AI FASTER, SMALLER AND MORE POWER EFFICIENT

Automatically compress SOTA models like MobileNet to <200KB with little to no drop in accuracy for inference on resource-limited MCUs.

Reduce model optimization trial & error from weeks to days using Deeplite's design space exploration.

Deploy more models to your device without sacrificing performance or battery life with our easy-to-use software.

VISIT bit.ly/Deeplite FOR MORE INFO
TinyML for all developers

Get your free account at http://edgeimpulse.com
Health sensors measure PPG and ECG signals critical to understanding vital signs. Signal chain products enable measuring even the most sensitive signals.

The biggest (3MB flash and 1MB SRAM) and the smallest (256KB flash and 96KB SRAM) Cortex M4 microcontrollers enable algorithms and neural networks to run at wearable power levels.

AI inferences at a cost and power point that makes sense for the edge. Computation capability to give vision to the IoT, without the power cables. **Coming soon!**
Qeexo AutoML for Embedded AI
Automated Machine Learning Platform that builds tinyML solutions for the Edge using sensor data

Key Features

- Wide range of ML methods: GBM, XGBoost, Random Forest, Logistic Regression, Decision Tree, SVM, CNN, RNN, CRNN, ANN, Local Outlier Factor, and Isolation Forest
- Easy-to-use interface for labeling, recording, validating, and visualizing time-series sensor data
- On-device inference optimized for low latency, low power consumption, and a small memory footprint
- Supports Arm® Cortex™- M0 to M4 class MCUs
- Automates complex and labor-intensive processes of a typical ML workflow – no coding or ML expertise required!

Target Markets/Applications

- Industrial Predictive Maintenance
- Automotive
- Smart Home
- Mobile
- Wearables
- IoT

Qeexo AutoML: End-to-End Machine Learning Platform

For a limited time, sign up to use Qeexo AutoML at automl.qeexo.com for FREE to bring intelligence to your devices!
Reality AI Tools® software
- Automated Feature Exploration and Model Generation
- Bill-of-Materials Optimization
- Automated Data Assessment
- Edge AI / TinyML code for the smallest MCUs

Reality AI solutions
- Automotive sound recognition & localization
- Indoor/outdoor sound event recognition
- RealityCheck™ voice anti-spoofing

is for building products

https://reality.ai   info@reality.ai   @SensorAI   Reality AI
SynSense (formerly known as aiCTX) builds ultra-low-power (sub-mW) sensing and inference hardware for embedded, mobile and edge devices. We design systems for real-time always-on smart sensing, for audio, vision, bio-signals and more.

https://SynSense.ai
## Next tinyML Talks

<table>
<thead>
<tr>
<th>Date</th>
<th>Presenter</th>
<th>Topic / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuesday, October 13</td>
<td><strong>Rajen Bhatt</strong>&lt;br&gt;Director of Engineering, Machine Learning, Qeexo Co</td>
<td>Qeexo’s Runtime-Free Architecture for Efficient Deployment of Neural Networks on Embedded Targets</td>
</tr>
<tr>
<td></td>
<td><strong>Chandrasekar Vuppalapati</strong>&lt;br&gt;Senior Vice President Products &amp; Programs,*&lt;br&gt;*Hanumayamma Innovations and Technologies Inc.</td>
<td>Democratization of Artificial Intelligence (AI) to Small Scale Farmers - a framework to deploy AI Models to Tiny IoT Edges that operate in constrained environments</td>
</tr>
</tbody>
</table>

**Webcast start time is 8 am Pacific time**<br>Each presentation is approximately 30 minutes in length

Please contact [talks@tinyml.org](mailto:talks@tinyml.org) if you are interested in presenting
Local German Committee

Alexis Veynachter,
Master Degree in Control Engineering, Senior Field Application Engineer
Infineon 32bits MCUs for Sensors, Fusion & Control

Carlos Hernandez-Vaquero
Software Project Manager, IoT devices
Robert Bosch

Prof. Dr. Daniel Mueller-Gritschneder
Interim Head - Chair of Real-time Computer Systems
Group Leader ESL - Chair of Electronic Design Automation
Technical University of Munich
Rafael Stahl

Rafael Stahl is a doctoral candidate at the Technical University of Munich at the Chair of Electronic Design Automation in his third year. He received his Bachelor and Master in “Electrical Engineering and Information Technology” from TU Munich.

He gathered work experience at the Fraunhofer Institute for Embedded Systems and Communication Technology, prototyping demos of connected car systems. Software reverse engineering is a long-time passion of his, that gained him in-depth programming and debugging experience.

Currently he is looking to improve neural network inference through target-aware methods with the wider goal of reducing the memory footprint of embedded software.

He received the Best Paper Award at SiPS 2019.
Embedded ML research at TUM: Moving NN Inference to the Extreme Edge

Rafael Stahl
Technical University of Munich
Department of Electrical and Computer Engineering
Chair of Electronic Design Automation
Online, 7th of October 2020
NN Inference at the Extreme Edge (TinyML)

- Edge devices
  - Large amounts of data from sensing and collection
  - Limited resources: Computation power, memory

- Neural networks
  - State-of-the-art solution for many non-linear problems
  - Transform data to more reasonable size or enable local decision
    - Reduced communication demand
    - Reduced latency
    - Higher availability
    - Privacy
TinyML Applications

- **Audio**
  - Keyword Spotting (KWS) / Audio Wakeup
- **Vision**
  - Video Wakeup (Face Detection)
  - Image Classification
- **Radar**
  - Gesture Recognition
- **Accelerometer**
  - Activity Detection

Hello, TinyML

Low-Power muC → Wake-up! → Larger AI Device

Image → Hello, TinyML

"Person"
Collaborators

EDA Chair @ TUM
- Design Methods for Embedded Systems and ASIC Design
- ~20 researchers; 4 researchers and ~5 students on TinyML

DeeperThings
Distributed CNN Inference
Andreas Gerstlauer (UT Austin)
Zhuoran Zhao (UT Austin)

TF Lite Micro Compiler
Static Model Code Generator
Andrew Stevens (Infineon)
Christof Petig

Scale4Edge
EDA Chair @ TUM
- Design Methods for Embedded Systems and ASIC Design
- ~20 researchers; 4 researchers and ~5 students on TinyML

DeeperThings
Distributed CNN Inference
Andreas Gerstlauer (UT Austin)
Zhuoran Zhao (UT Austin)

TF Lite Micro Compiler
Static Model Code Generator
Andrew Stevens (Infineon)
Christof Petig
Distributed Inference

• Problem
  – Memory and compute constrained devices: IoT, MCUs, mobile phones
  – Cannot store or compute full CNN

• Cloud backend
  – privacy concerns
  – latency, availability

• Goal
  – Combine memory and computation resources of several small devices
  – Use idle time of other devices by letting them cooperate
  – Optimize introduced communication demand
CNN Structure

- Common structure:
  - Feature-dominated early layers
  - Weight-dominated later layers

Layer Operation
- $X \times Y \times C$ Feature Map
- $K \times K \times C \times N$ Filter Bank

Rafael Stahl (TUM)
DeeperThings

Target: Several devices act as one device with combined computation, ROM and RAM resources

Feature-dominated partitioning
Method: Fused Tile Partitioning (FTP)
Target: Combine RAM resources

Weight-dominated partitioning
Method: Optimized Weight Partitioning (OWP)
Target: Combine ROM resources

DeeperThings FTP+OWP
DeeperThings Flow

CNN

DeeperThings Flow

Fused Tile Partitioning (FTP)

ILP

Feature-dominated Partial CNN

Optimized Weight Partitioning (OWP)

Weight-dominated Partial CNN

Decide where to switch from feature to weight partitioning

Partitioned Early Layer

Partitioned Early Layer

Partitioned Early Layer

Partitioned Early Layer

ILP

ILP: Integer Mixed Programming

Partitioned Later Layer

Partitioned Later Layer

Partitioned Later Layer

Partitioned Later Layer
Feature Partitioning – Convolutional Layers

- Distributes input/output data dominated CNN layers
Fused Tile Partitioning (FTP)

- Exploits locality of convolution by splitting feature maps into NxN grid
- Fuses corresponding tiles across layers into independent stacks
- Trade-off: Calculate overlaps multiple times $\leftrightarrow$ communicate it

Optimized Weight Partitioning (OWP)

• Distributes weight dominated CNN layers and fully-connected layers

Optimized Weight Partitioning (OWP)

• Illustrated for fully connected layers
Fully-connected Layer

\[ b = f(W \cdot a) \]
Fully-connected Network - Example

- C: Communication demand
- |W|: Number of weights

\[ |W| = 4 \times 8 = 32 \]

\[ |W| = 128 \]

\[ |W| = 64 \]

\[ |W| = 16 \]

\[ |W| = 240 \]

\[ C = 0 \]
Distributed Inference

- Simple approach: Pipelining / Sequential Layer Mapping
  - Distribute layer operations to different devices
  - Unbalanced partitions
  - For mostly idle workload: No parallelization

\[ |W| = 4 \times 8 + 8 \times 16 = 160 \]
\[ |W| = 16 \times 4 + 4 \times 4 = 80 \]
\[ C = 20 \]
Balanced Partitioning
Layer Output Partitioning (LOP)
LOP Example

- Devices exchange partition output with each other
- Equal memory and compute share
- Low communication demand

\[ |W| = 240 \]

\[ |W| = 120 \]

\[ |W| = 120 \]

\[ C = 34 \]
Layer Input Partitioning (LIP)

Layer 2, Partition 1 (L2P1)

Layer 2, Partition 2 (L2P2)

Layer 2, Merge (L2M)
Fused Layer Partitioning (FUSE)
FUSE Example

- Naïve fusing results in solution with high communication demand
  - Use LOP for non-fused layers
- Targeted fusing to reduce communication demand
  - Use LOP for non-fused layers

\[ \text{Device 1} \rightarrow \text{L1P1} \rightarrow \text{L1M} \rightarrow \text{L1C} \rightarrow \text{L2P1} \rightarrow \text{L2P2} \rightarrow \text{L3P1} \rightarrow \text{L3P2} \rightarrow \text{L4P1} \rightarrow \text{L4P2} \rightarrow \text{Device 2} \]

- Communication demand for naïve fusing:
  - \( |W| = 120 \)
  - \( C = 40 \)

- Communication demand for targeted fusing:
  - \( |W| = 120 \)
  - \( C = 22 \)
Optimized Weight Partitioning (OWP)

• LOP, LIP and Fusing also works for Convolutional Layers
Convolutional Layer Fusion

Two consecutive weight-dominated layers can be fused as well.
Experimental Evaluation

Evaluated models:
- YOLOv2 (displayed)
- AlexNet
- GoogLeNet
- VGG-16

Executed on cluster of Raspberry Pi 3

Memory usage divided by number of devices

Communication demand reduced by 28.8%
Collaborators

EDA Chair @ TUM
- Design Methods for Embedded Systems and ASIC Design
- ~20 researchers; 4 researchers and ~5 students on TinyML

DeeperThings
Distributed CNN Inference
Andreas Gerstlauer (UT Austin)
Zhuoran Zhao (UT Austin)

TF Lite Micro Compiler
Static Model Code Generator
Andrew Stevens (Infineon)
Christof Petig
Motivation

Deploying TinyML models on microcontrollers
• Existing libraries cause overhead
• Peak buffer sizes limit memory
• NAS and network optimization not tailored to target hardware

TF Lite for Microcontrollers:
• On tiny microcontrollers, overhead of the TFLM interpreter is significant
• Opaque buffers and high code volume hinder functional safety verification
• Scheduling and allocation must be kept simple
TF Lite Micro Flow

- TensorFlow API
  - TF.Keras
  - Low Level API
    - TF.Keras Model
    - Saved Model (.pb)
    - Concrete Functions
    - TF Lite Converter
    - TF Lite Flat Buffer (.tflite)

Server

Client

- TF Lite Micro Interpreter (C++)
- Online Exec. Planning
- TF Lite Micro OPS (C++)
- muC
TF Lite Micro Compiler

• Uses a generator approach for static code
  ➢ Removes interpreter from target code
  ➢ Write concrete, typed buffers for safety evaluation
  ➢ Offline scheduling and allocation

• Started as parallel effort of Christof Petig and TUM and merged into a single project
• Available open Source @ https://github.com/cpetig/tflite_micro_compiler
• RFC Design Document submitted to TF SIGMICRO mailing list
  • https://docs.google.com/document/d/1wDqC50sjCaWyQxsSn_Y-XAGh8-ozlgm2HDzX_b9Dlyo/edit?usp=sharing
Experimental Results

- Model: “hello_world” from TF Lite for Microcontrollers
- Target: RV32IMAC - ILP32 - Custom Pulpino startfiles
- Compiler: gcc-9.2.0

<table>
<thead>
<tr>
<th>Binary File</th>
<th>.text + .rodata (ROM) [kB]</th>
<th>.data + .bss (RAM) [kB]</th>
<th>Total [kB]</th>
<th>Number of Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted</td>
<td>48.1</td>
<td>4.7</td>
<td>52.8</td>
<td>61488</td>
</tr>
<tr>
<td>TF Lite Micro Compiler</td>
<td>22.2</td>
<td>3.7</td>
<td>25.9</td>
<td>14937</td>
</tr>
</tbody>
</table>
Outlook: Planned TUM TinyML Flow

All upcoming work published as Open Source @https://github.com/tum-ei-eda
Summary

- DeeperThings
  - Distributed CNN Inference
  - Balanced partitioning
  - Minimizing for memory and communication

- TensorFlow Lite Micro Compiler
  - Generating Static Model Code
  - Reduced memory and computation overhead
  - Transparency for verification
Thank you for your attention
Copyright Notice

This presentation in this publication was presented as a tinyML® Talks webcast. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyML.org