MicroKWS: Speech Recognition Flow for Embedded Devices

13.05.2026 AI:AT Workshop

This work has been developed in the project DI-OSVISE. DI-OSVISE is funded by the German Ministry of Research, Technology and Space (BMFTR) (reference numbers: 16ME0957). The authors are responsible for the content of this publication.

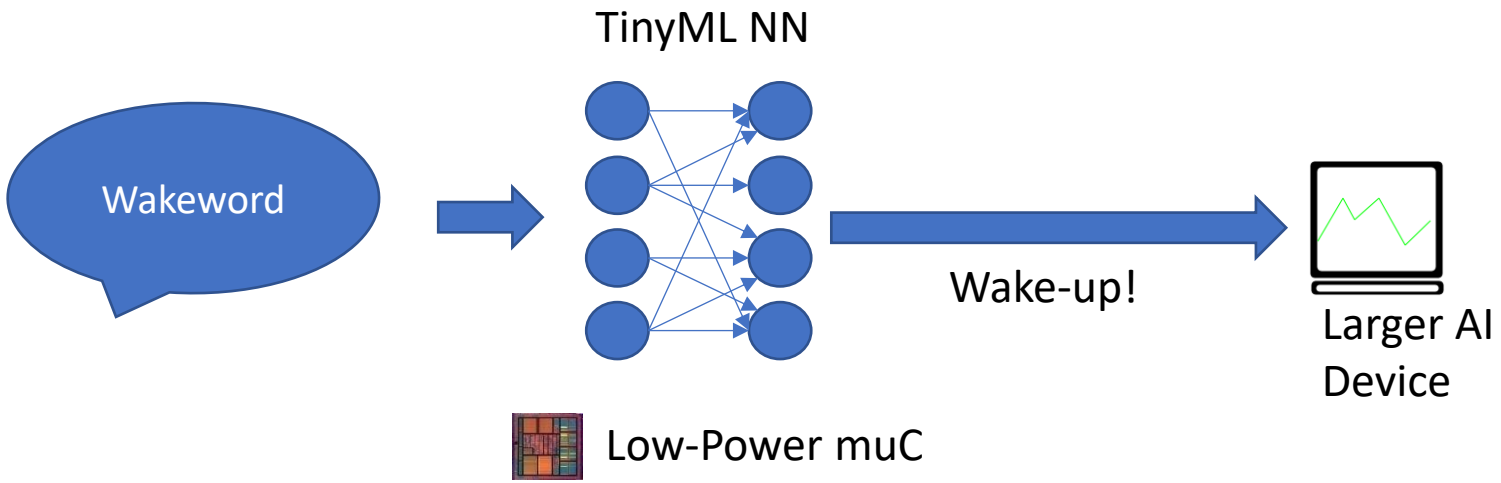
With funding from the:

Federal Ministry of Research, Technology and Space

DI-OSVISE

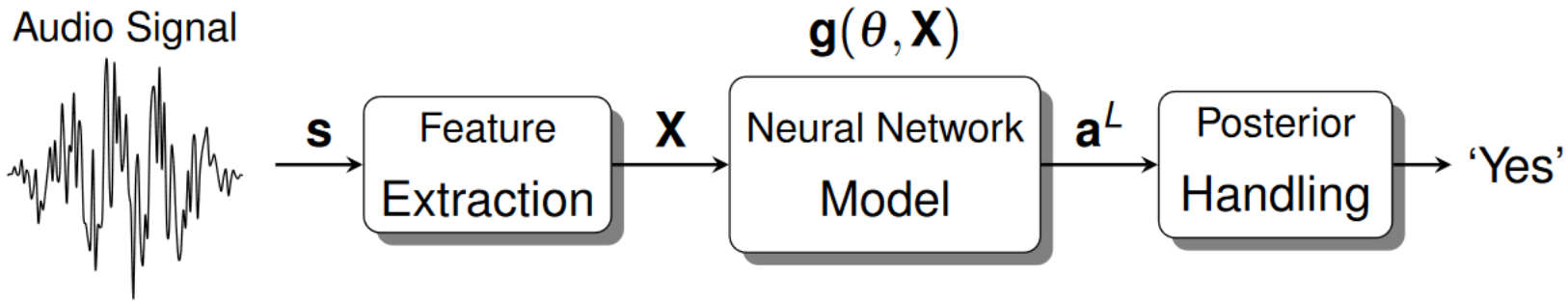
Motivation: TinyML / Edge AI

- Typical machine learning applications require a lot of memory and computational power
- Let’s develop AI affordable products for the future IoT market which
  - Are very small
  - Can be battery-powered (e.g. always on wake-up)



Application Example: Keyword Spotting (KWS)

- **Scenario:** Given a set of Keywords (e.g. `yes,no,cat,dog`), we want to design a neural network architecture which can classify these words with a good accuracy.

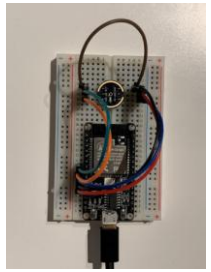


Flow Overview

- Which steps are required to build such an application?
- Demos for each step

Idea

Data Generation → Model Design & Training → Quantization → Optimization & Code Generation → Deployment → Evaluation



Prototype

Used Tools

- Used open source projects:
  - Tensorflow/Keras (open source)
  - Tensorflow Lite (open source)
  - (Micro)TVM (open source)
  - ESP-IDF
  - Various TUM tools (Speech recorder)

Model Design, Preprocessing and Training

Model Quantization

Model Optimization, Code Generation

Deployment to Target MCU

Dataset Extension

- Used hardware:
  - ESP32-C3 Microcontroller (RV32IMC ISA)
  - INMP441 digital I2S Microphone
  - RGB LED as Indicator
  - USB Serial port for debugging,…

Setup

- For simplicity we run most of the steps in a Google Colaboratory Notebook: https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb
  - Login with Google Account
  - Choose GPU-Runtime
  - Execute one cell at a time
  - Make sure that the session does not get suspended (after 10 min of inactivity)

- Deployment of the Target SW on real HW can not be done on Colab!

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

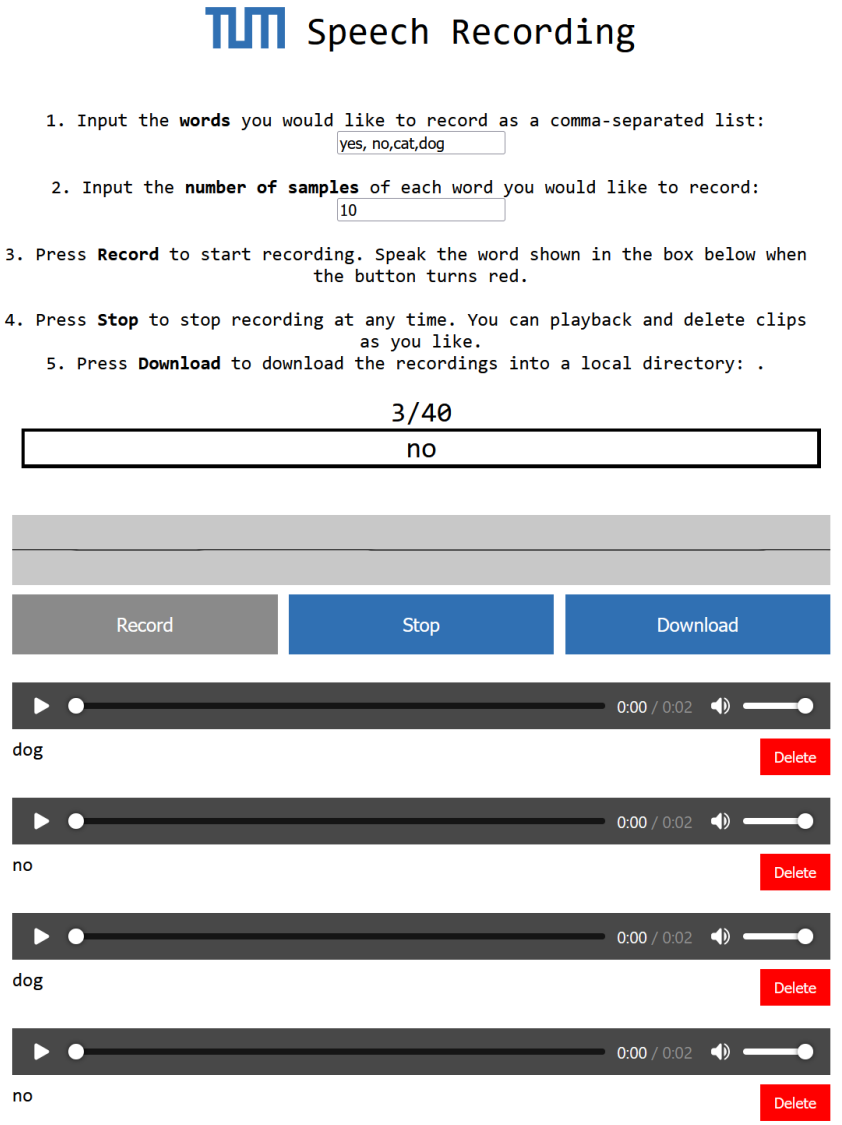
1. Data Generation

Data Generation

- Where do we get our data for training?
  - Collect yourself
    - Record on your own → A lot of data is required!
    - Search web for samples → Needs labeling!
  - Use existing data
    - Many well known datasets available online
      - Image Detection: MNIST, Cifar10, MobileNet,…
      - Speech recognition: speech_commands
  - Combination of both
    - Use existing data and add your own!

MicroKWS Sample Recorder

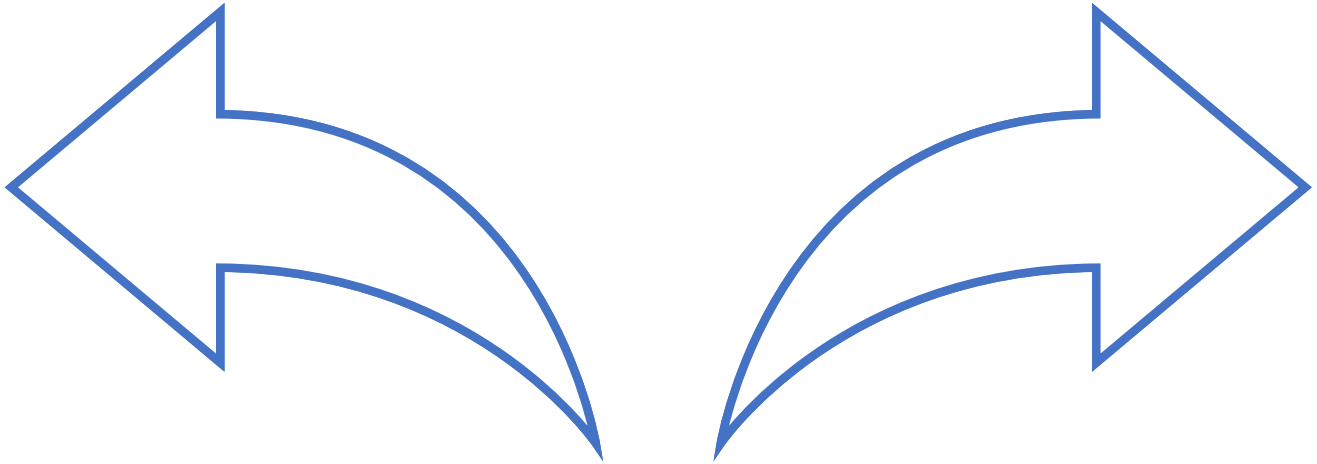
- Custom webapp for speech recording
- Generated WAV files are compatible with `speech_commands` dataset



Speech Commands Dataset

- Used Dataset: `speech_commands_v0.02`
- Audio dataset of spoken words designed to help train and evaluate keyword spotting systems.
- Background: Collected in a community project in 2017
- Size: 8.17 GiB, ~ 100k WAV files
- 34 Keywords, ~3,000 samples each
- Special labels: unknown, silence, (background noise)

1. Data Generation (I)



Record own samples?

Use existing dataset?

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

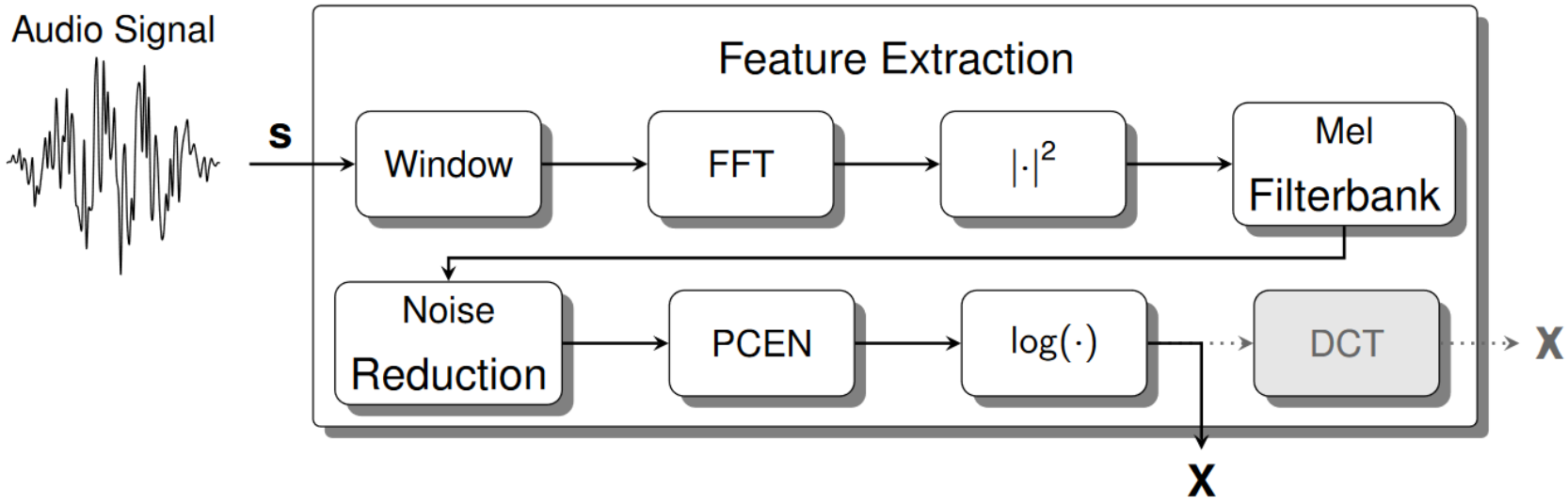
Demo: Speech Commands Dataset

Data Preprocessing

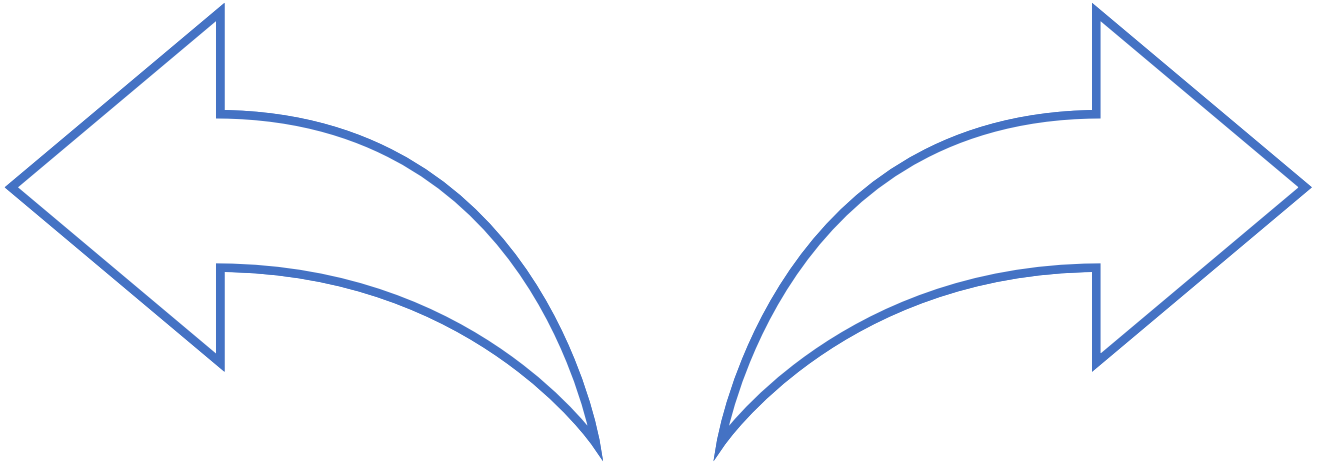
- How to prepare our data for training
  - Use raw samples
    - Easy to implement
    - Problems: large input, will not learn to detect shifted samples,…
  - Custom preprocessing
    - Data Augmentation → Add noise & shifts to data to generalize better
    - Normalization → Adjust volume, center samples,…
    - Domain Transformations → time domain vs. frequency domain

Pre-processing Pipeline

- Required preprocessing steps for KWS:
  1. Labeling of voice samples
  2. Normalize dataset and filter invalid samples
  3. Split dataset in the classes: Used words, Unkown words, silence
  4. Extract features:



1. Data Generation (II)



Raw samples?

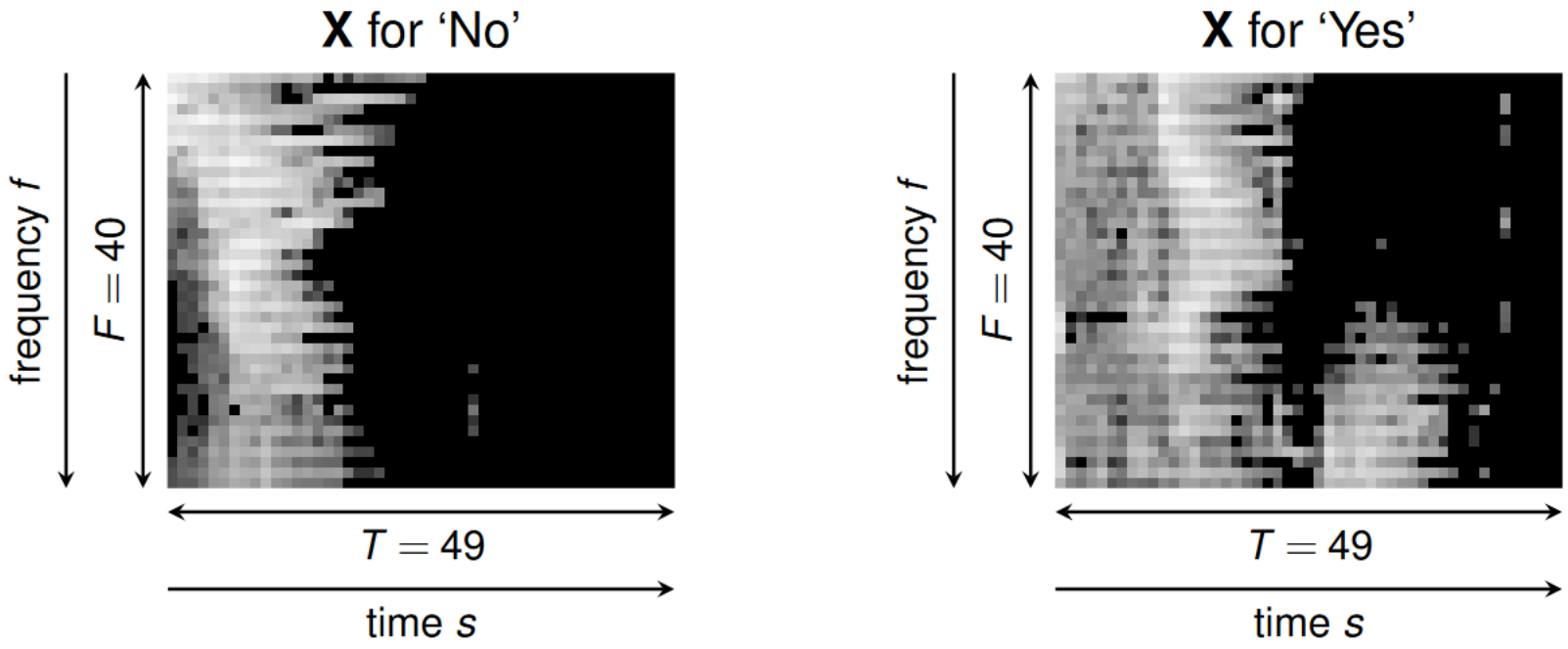
Custom preprocessing?

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

Demo: Data Preprocessing

Preprocessed Samples

- 2D images of size 49 x 40 px

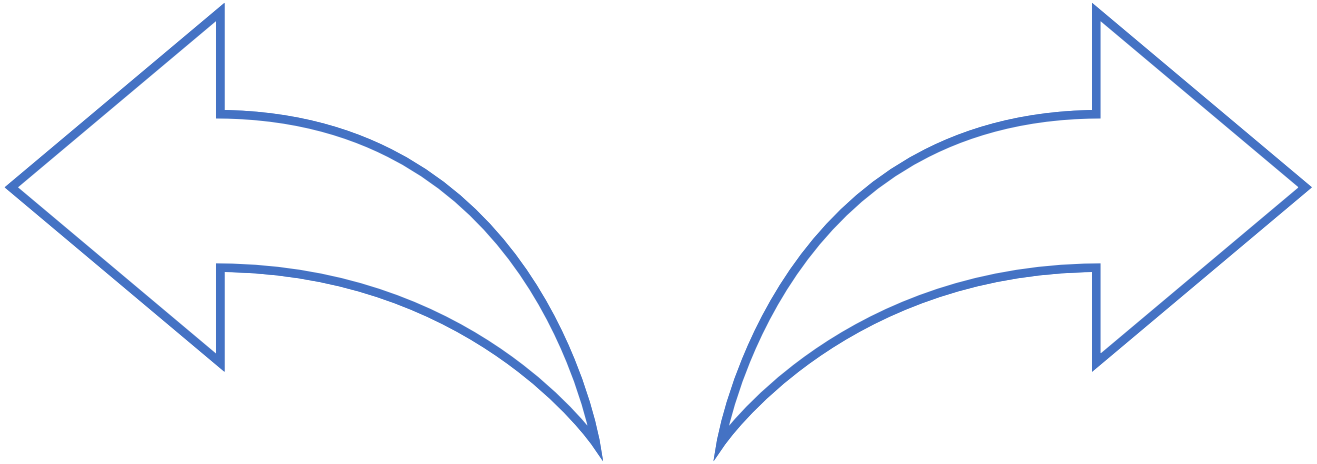


2. Model Design & Training

Model Architecture

- How to find suitable Model Architecture?
  - Manual approach
    - Experience required → Use existing architectures as a baseline
    - Choose training hyperparameters → Number of epochs, learning rate, loss function,…
  - Network Architecture Search
    - Find optimal model architecture and hyperparameters automatically
    - Constrained optimization problem: Accuracy, ROM, RAM
    - Needs many computational resources

2. Model Design & Training



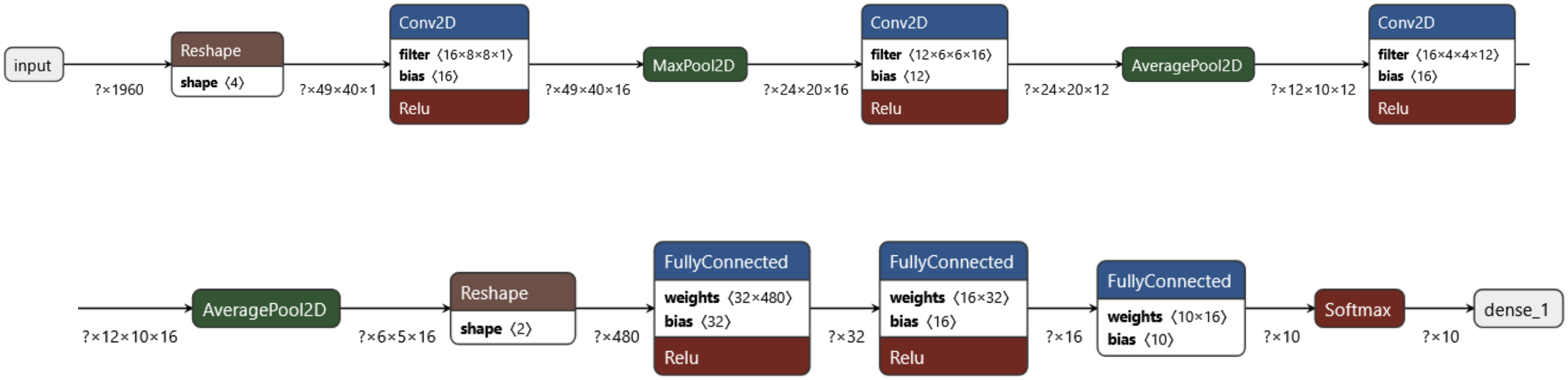
Manual Model Design?

Network Architecture Search?

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

Demo: Tensorflow & Keras

MicroKWS Architecture



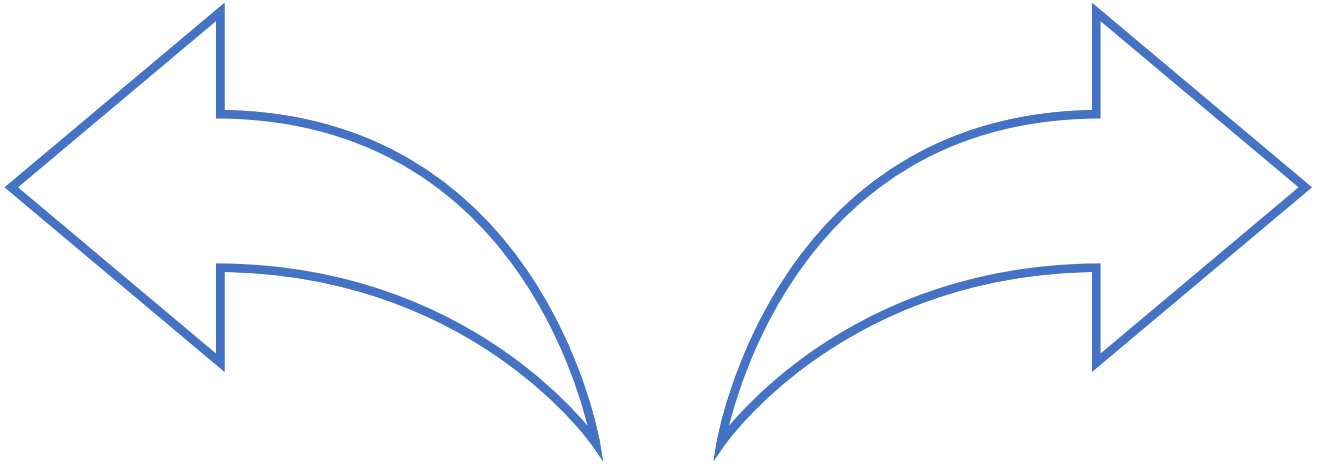
3. Quantization

Quantization

- Compression Techniques
  - Integer Quantization: Add data types
  - Pruning: Introduce sparsity (add zeros)
  - Weight clustering: Create codebook and encode weights
  - Other compression techniques (Huffman, Golomb Rice,…)

- Quantization Modes
  - Quantization-aware Training (QAT): Using Fake-Quantization Layers
  - Post Training Quantization (PTQ): Requires Re-Training to fix weights iteratively

3. Quantization



Quantization-aware Training?

Post-Training Quantization?

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

Demo: TFLite

4. Model Optimization and Code Generation

Open-source Frameworks for Embedded ML

- **TFLite for Microcontrollers (TFLM)** TensorFlow Lite
  - Part of Google's Tensorflow Lite (LiteRT)
  - Hardcoded NN kernel implementations
  - Interpretation of model at runtime
  - Vendor kernel libraries can be used easily

- **Executorch**
  - Part of the Pytorch project
  - Driven by Meta as alternative for TFLM
  - Targeting any embedded devices (Minimal runtime)
  - Supports State of the Art model types (LLMs,…)

- **(Micro)TVM**
  - ML Compiler-driven approach
  - Various intermediate representations (Irs) for optimization
  - Supports many different frontends and backends
  - Allows auto-tuning kernels on target device
  - Accelerator & kernel library integration possible
  - Less overheads due to Ahead-of-Time compilation

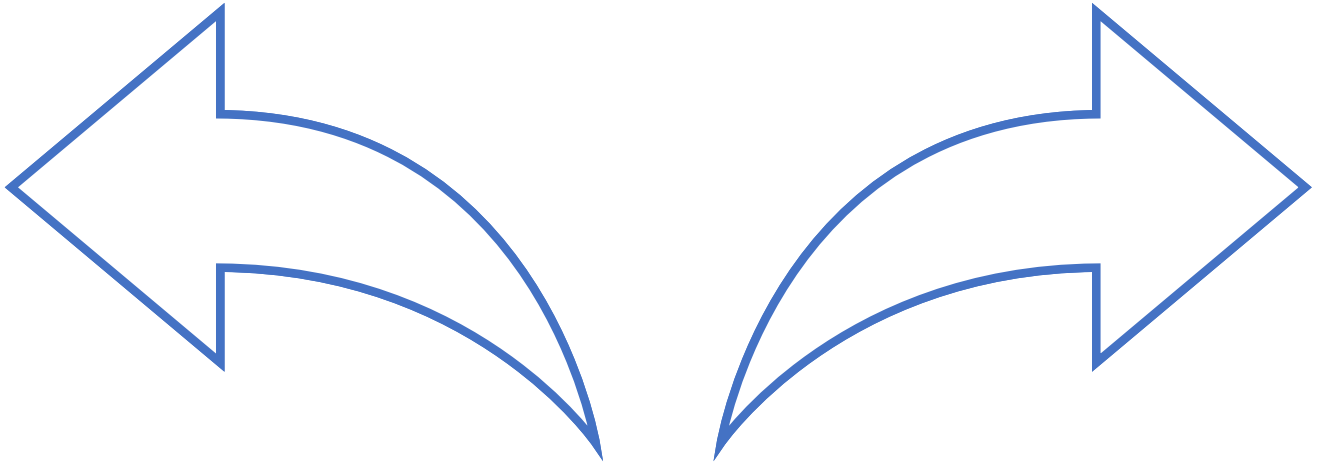
- **(tiny)IREE**
  - Highly integrated with LLVM Compiler
  - Based on MLIR (Multi-level IR) project
  - Very flexible, lots of contributors
  - Supports State of the Art model types (LLMs,…)
  - Limited MCU/Baremetal support

ML Compilers

Traditional Approaches

Modern Approaches

4. Model Optimization & Code Generation (I)



TFLM, Executorch, tinyIREE,…?

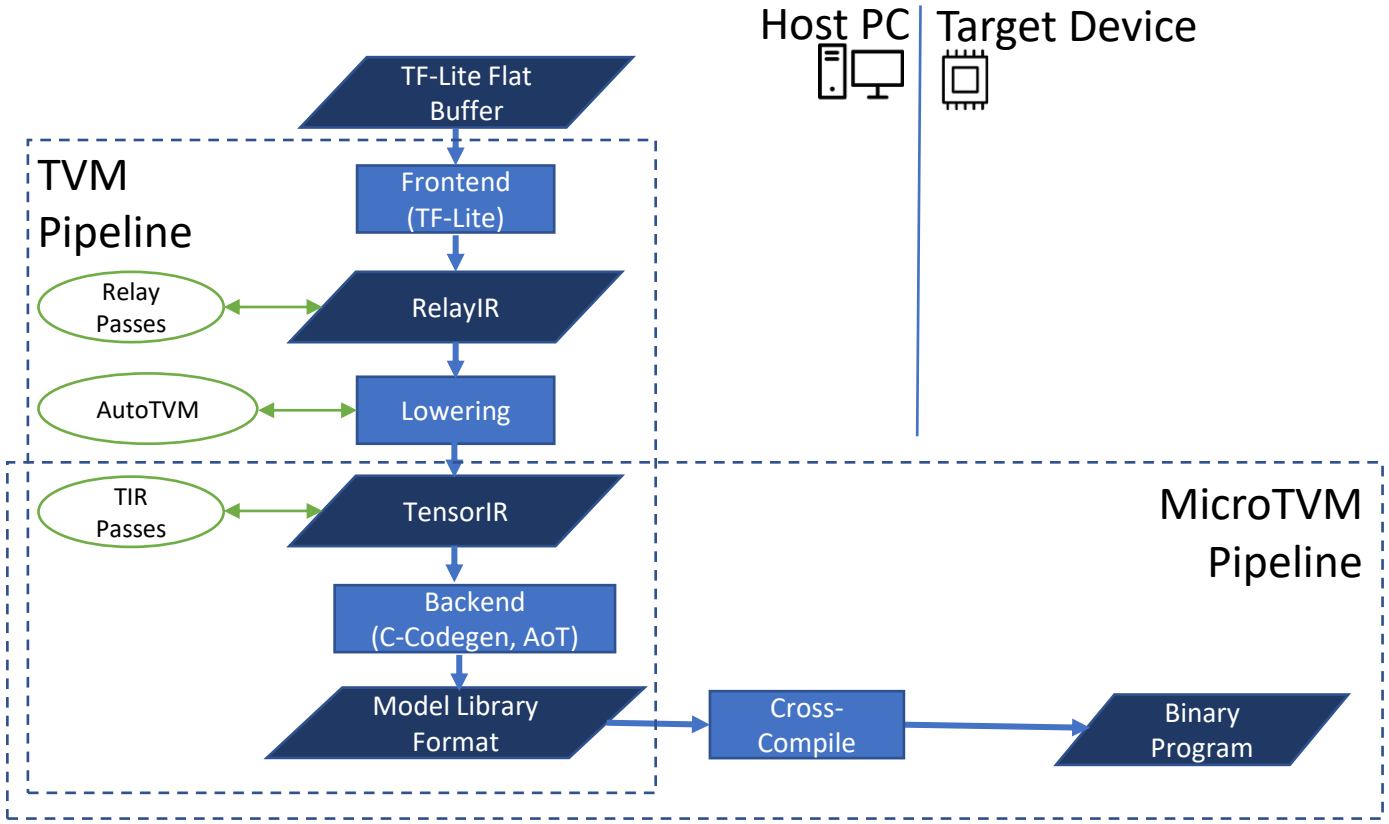
TVM?

TVM: An Optimizing ML Compiler Framework

- Offers compiler-infrastructure for optimizing ML models
- Intermediate representations: RelayIR (high-level), TensorIR (low-level)
- Example Flow:
  1. *Frontend:* Load a model of a given type (TFLite, ONNX,...) and convert to relay
  2. Perform graph-level optimizations (Relay Passes)
  3. *Lowering:* Convert Relay interpretation to TIR level
  4. Apply low-level operator optimizations (TIR Passes)
  5. *Backend:* Generate code for the selected Hardware Target (C, LLVM,…) → Artifacts

→ Gaining popularity ⇒ Highly relevant research field

TVM: Kernel Generation Flow



Model Optimizations

- Example Optimizations which can be applied by TVM:
  - Constant Folding
    - Pre-compute graph parts that can be determined statically
  - Loop tiling/reordering/unrolling
    - Exploits spatial and temporal locality of data accesses in loop nests
  - Data layout transformations
    - Adapt data and kernel layout to the constraints given by the targets memory hierarchy
    - i.e. NHWC → NCHWc

Example: Constant Folding

```
X = (Y * 4 * 5) →
X = (Y * 20)
```

Example: Loop Tiling

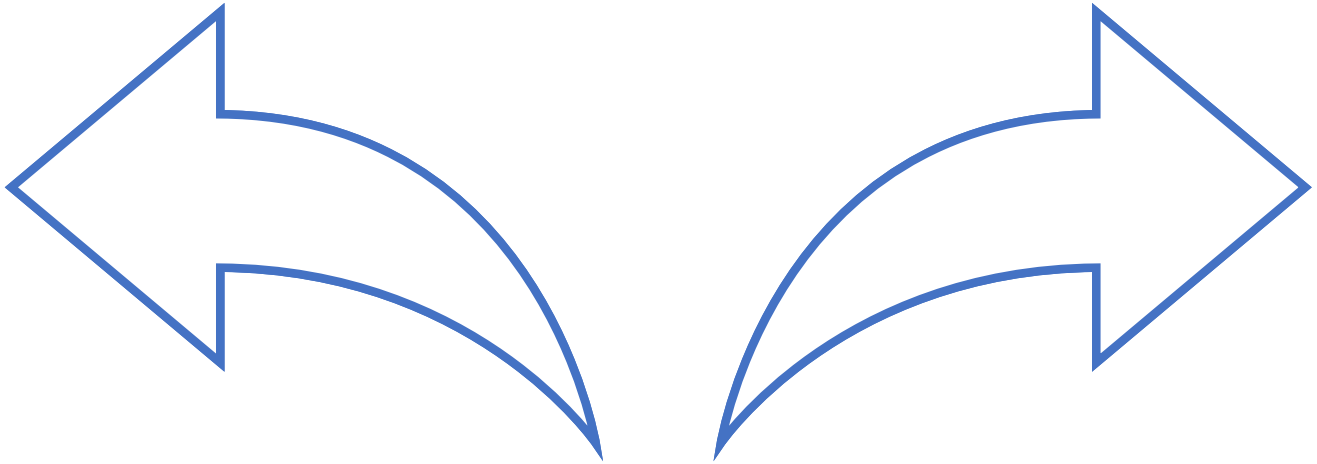
```
for x in range(32) →
for x_outer, x_inner in T.grid(2, 16)
```

Example: Loop Layout Transformations
NHWC (channels-last) → NCHW (channels-first)

Further Optimizations

- How to improve the performance (latency) on a specific target?
  - Use vendor library
    - CMSIS-NN → Popular for ARM MCUs
    - Problem: Often less freedom for optimizations
  - Tuning
    - TVM offers several tuning interfaces: AutoTVM, MetaScheduler,…
    - Benchmark operator implementations on real HW
    - Problem: Very time intensive

4. Model Optimization & Code Generation (II)



AutoTVM?

CMSIS-NN Kernel Library?

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

Demo: TVM

Example AutoTVM Logs


```
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "conv2d_NCHWc.x86", [["TENSOR", [1, 12, 12, 10], "
    "int16"], [1, 1], [1, 1, 2, 2], [1, 1], "NCHW", "NCHW", "int32"], {}], "config": {"index": 125, "code_hash": null, "entity": [[
    "sp", [-1, 1]], ["tile_ow", "sp", [-1, 8]], ["unroll_kw", "ot", true]]}, "result": [[0.570443], 0, 26.043710947036743, 16540732
    : "0.8.dev0"}
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "dense_nopack.x86", [["TENSOR", [1, 480], "int16"]
    "int32"], {}], "config": {"index": 5, "code_hash": null, "entity": [["tile_y", "sp", [-1, 1]], ["tile_x", "sp", [-1, 32]], ["ti
    .008606], 0, 12.343218803405762, 1654073897.9048553], "version": 0.2, "tvm_version": "0.8.dev0"}
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "conv2d_NCHWc.x86", [["TENSOR", [1, 1, 49, 40], "i
    , [1, 1], [3, 3, 4, 4], [1, 1], "NCHW", "NCHW", "int32"], {}], "config": {"index": 30, "code_hash": null, "entity": [["tile_ic"
    1]], ["tile_ow", "sp", [-1, 40]], ["unroll_kw", "ot", true]]}, "result": [[1.775476], 0, 53.713725328445435, 1654074121.4171865
    .dev0"}
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "dense_pack.x86", [["TENSOR", [1, 480], "int16"],
    "int32"], {}], "config": {"index": 294, "code_hash": null, "entity": [["tile_y", "sp", [-1, 1, 1]], ["tile_x", "sp", [-1, 1, 1]
    ["tile_inner", "sp", [-1, 1]]]}, "result": [[0.025055], 0, 12.116497993469238, 1654074262.356558], "version": 0.2, "tvm_version
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "dense_nopack.x86", [["TENSOR", [1, 32], "int16"],
    "int32"], {}], "config": {"index": 4, "code_hash": null, "entity": [["tile_y", "sp", [-1, 1]], ["tile_x", "sp", [-1, 16]], ["ti
    .000306], 0, 12.548060655593872, 1654074444.2586784], "version": 0.2, "tvm_version": "0.8.dev0"}
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "dense_pack.x86", [["TENSOR", [1, 32], "int16"], [
    "int32"], {}], "config": {"index": 75, "code_hash": null, "entity": [["tile_y", "sp", [-1, 1, 1]], ["tile_x", "sp", [-1, 1, 1]]
    ["tile_inner", "sp", [-1, 1]]]}, "result": [[0.00061], 0, 12.078999519348145, 1654074547.9881606], "version": 0.2, "tvm_version
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "dense_nopack.x86", [["TENSOR", [1, 16], "int16"],
    "int32"], {}], "config": {"index": 3, "code_hash": null, "entity": [["tile_y", "sp", [-1, 1]], ["tile_x", "sp", [-1, 10]], ["ti
    -05], 0, 12.237016439437866, 1654074765.3485973], "version": 0.2, "tvm_version": "0.8.dev0"}
{"input": ["c -keys=cpu -link-params=0 -model=unknown -runtime=c -system-lib=1", "dense_pack.x86", [["TENSOR", [1, 16], "int16"], [
    "int32"], {}], "config": {"index": 36, "code_hash": null, "entity": [["tile_y", "sp", [-1, 1, 1]], ["tile_x", "sp", [-1, 1, 1]]
    ["tile_inner", "sp", [-1, 1]]]}, "result": [[0.000183], 0, 10.38105034828186, 1654074829.1487846], "version": 0.2, "tvm_version
```

Example TVM Kernel

Snippet of C-Code generated by TVM:

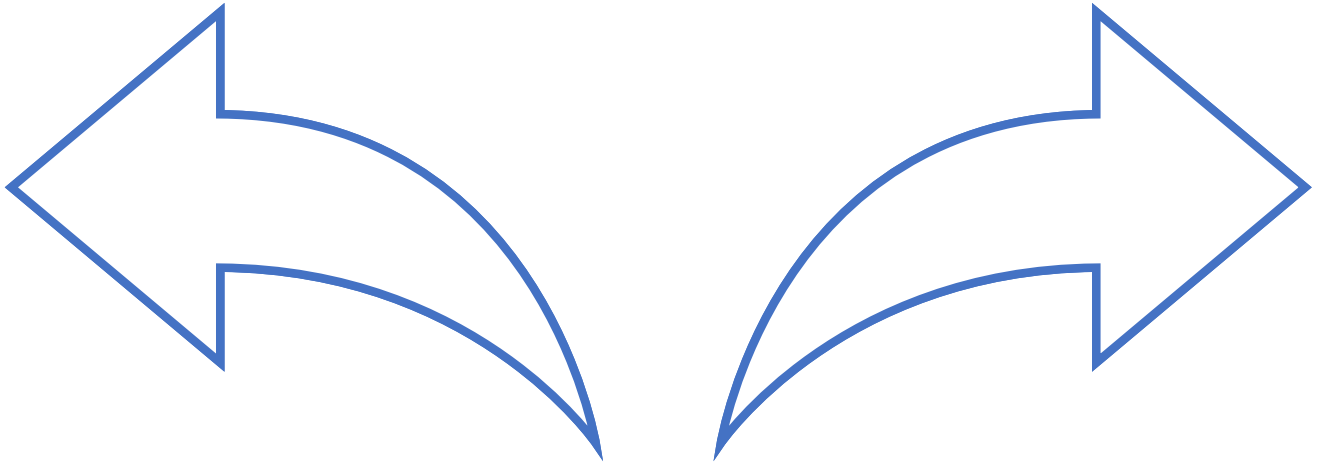
```
TVM_DLL int32_t tvmgen_default_fused_nn_avg_pool2d(float* p0, float* pool_avg, uint8_t* global_const_workspace_4_var, uint8_t* global_workspace_5_v
  void* pool_sum_let = (&(global_workspace_5_var[1262592]));
  for (int32_t ax0_ax1_fused = 0; ax0_ax1_fused < 56; ++ax0_ax1_fused) {
    for (int32_t ax2 = 0; ax2 < 56; ++ax2) {
      for (int32_t ax3_init = 0; ax3_init < 32; ++ax3_init) {
        ((float*)pool_sum_let)[(((ax0_ax1_fused * 1792) + (ax2 * 32)) + ax3_init)] = 0.000000e+00f;
      }
      for (int32_t rv0_rv1_fused = 0; rv0_rv1_fused < 9; ++rv0_rv1_fused) {
        for (int32_t ax3 = 0; ax3 < 32; ++ax3) {
          int32_t cse_var_7 = (rv0_rv1_fused / 3);
          int32_t cse_var_6 = (rv0_rv1_fused % 3);
          int32_t cse_var_5 = (ax0_ax1_fused * 1792);
          int32_t cse_var_4 = (ax2 * 32);
          int32_t cse_var_3 = (cse_var_7 + ax0_ax1_fused);
          int32_t cse_var_2 = (ax2 + cse_var_6);
          int32_t cse_var_1 = ((cse_var_5 + cse_var_4) + ax3);
          ((float*)pool_sum_let)[cse_var_1] = (((float*)pool_sum_let)[cse_var_1] + (((((1 <= cse_var_3) && (cse_var_3 < 57)) && (1 <= cse_var_2)) &
        }
      }
    }
  }
  for (int32_t ax0_ax1_fused_1 = 0; ax0_ax1_fused_1 < 56; ++ax0_ax1_fused_1) {
    for (int32_t ax2_1 = 0; ax2_1 < 56; ++ax2_1) {
      for (int32_t ax3_1 = 0; ax3_1 < 32; ++ax3_1) {
        int32_t cse_var_8 = (((ax0_ax1_fused_1 * 1792) + (ax2_1 * 32)) + ax3_1);
        int32_t v_ = 55 - ax0_ax1_fused_1;
        int32_t v__1 = 1 - ax0_ax1_fused_1;
```

5. Model Deployment

SW Deployment

- How to get our model running on device?
  - Vendor-specific flow
    - Here: ESP-IDF toolchain
    - Provides drivers for microphone,…
  - MicroTVM Deployment
    - Abstraction for actual vendor toolchain → compile & flash easily
    - Allows easy benchmarking and enables tuning

5. Model Deployment



MicroTVM?

ESP-IDF?

https://colab.research.google.com/github/tum-ei-eda/micro-kws/blob/workshop/MicroKWS.ipynb

Demo: MicroTVM + ESP-IDF

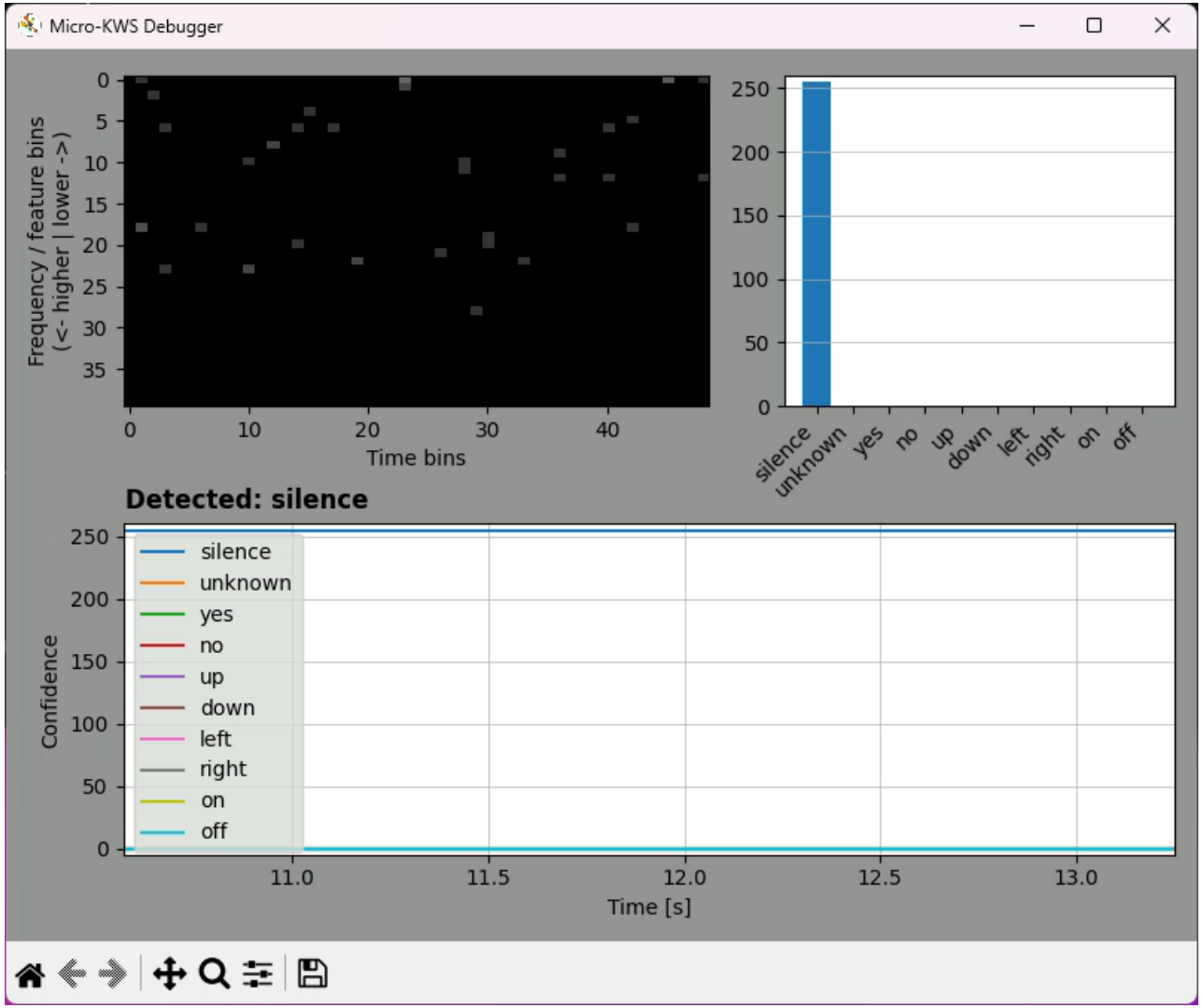
6. Model Evaluation

Real World Performance

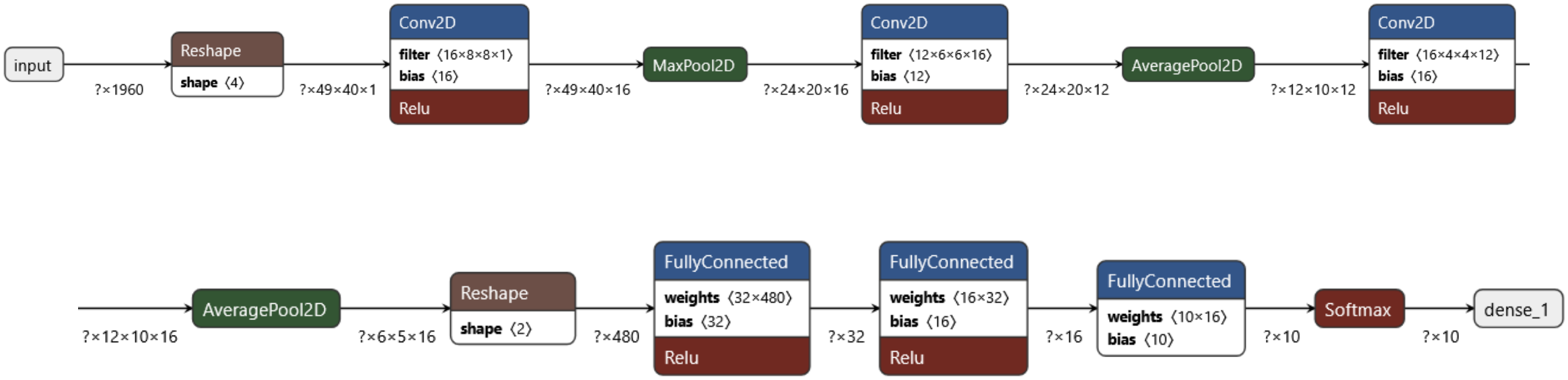
- Relevant metrics
  - Inference time vs. end-to-end latency (including pre-/postprocessing)
  - Training accuracy vs. detection accuracy
  - Model size vs. code size
  - …
- Challenges
  - Stream microphone samples in real-time → Use RTOS?
  - Memory overheads: Drivers, RTOS,…
  - Data not ideal: may contain noise, temporal shifts,… → Posterior Handling
  - …

Demo: Running MicroKWS Example

Optional: MicroKWS Debugger



MicroKWS Architecture



Bonus: MicroKWS Metrics

Layer	Operation	Output Buffer	Weights	Bias	Est. MACs	ROM Footprint [B]	RAM Footprint [B]
IN	-	(1, 49, 40, 1)					1.960
0	CONV_2D	(1, 49, 40, 16)	(16, 8, 8, 1)	(16,)	2.007.040	1.088	31.360
1	MAX_POOL_2D	(1, 24, 40, 16)			0	0	15.360
2	CONV_2D	(1, 24, 20, 12)	(12, 6, 6, 16)	(12,)	3.317.760	6.960	5.760
3	AVG_POOL_2D	(1, 12, 10, 12)			0	0	1.440
4	CONV_2D	(1, 12, 10, 16)	(16, 4, 4, 12)	(16,)	368.640	3.136	1.920
5	AVG_POOL_2D	(1, 6, 5, 16)			0	0	480
6	RESHAPE	(1, 480)			0	0	480
7	FULLY_CONNECTED	(1, 32)	(32, 480)	(32,)	15.360	15.488	32
8	FULLY_CONNECTED	(1, 16)	(16, 32)	(16,)	512	576	16
9	FULLY_CONNECTED	(1, 10)	(10, 16)	(10,)	160	200	10
OUT	SOFTMAX	(1, 10)			0	0	10
	TOTAL				5.7 M	28 kB	59 kB

Σ

No Memory Planning

Bonus: MicroKWS Metrics

Layer	Operation	Output Buffer	Weights	Bias	Est. MACs	ROM Footprint [B]	RAM Footprint [B]
IN	-	(1, 49, 40, 1)					1.960
0	CONV_2D	(1, 49, 40, 16)	(16, 8, 8, 1)	(16,)	2.007.040	1.088	31.360
1	MAX_POOL_2D	(1, 24, 40, 16)			0	0	15.360
2	CONV_2D	(1, 24, 20, 12)	(12, 6, 6, 16)	(12,)	3.317.760	6.960	5.760
3	AVG_POOL_2D	(1, 12, 10, 12)			0	0	1.440
4	CONV_2D	(1, 12, 10, 16)	(16, 4, 4, 12)	(16,)	368.640	3.136	1.920
5	AVG_POOL_2D	(1, 6, 5, 16)			0	0	480
6	RESHAPE	(1, 480)			0	0	480
7	FULLY_CONNECTED	(1, 32)	(32, 480)	(32,)	15.360	15.488	32
8	FULLY_CONNECTED	(1, 16)	(16, 32)	(16,)	512	576	16
9	FULLY_CONNECTED	(1, 10)	(10, 16)	(10,)	160	200	10
OUT	SOFTMAX	(1, 10)			0	0	10
	TOTAL				5.7 M	28 kB	47 kB

Σ

(-21%)

With Memory Planning