Aneel Damaraju

Abhishek Sharma

Jason Jabbour

Tensorflow Lite Micro

CS 249 TinyML

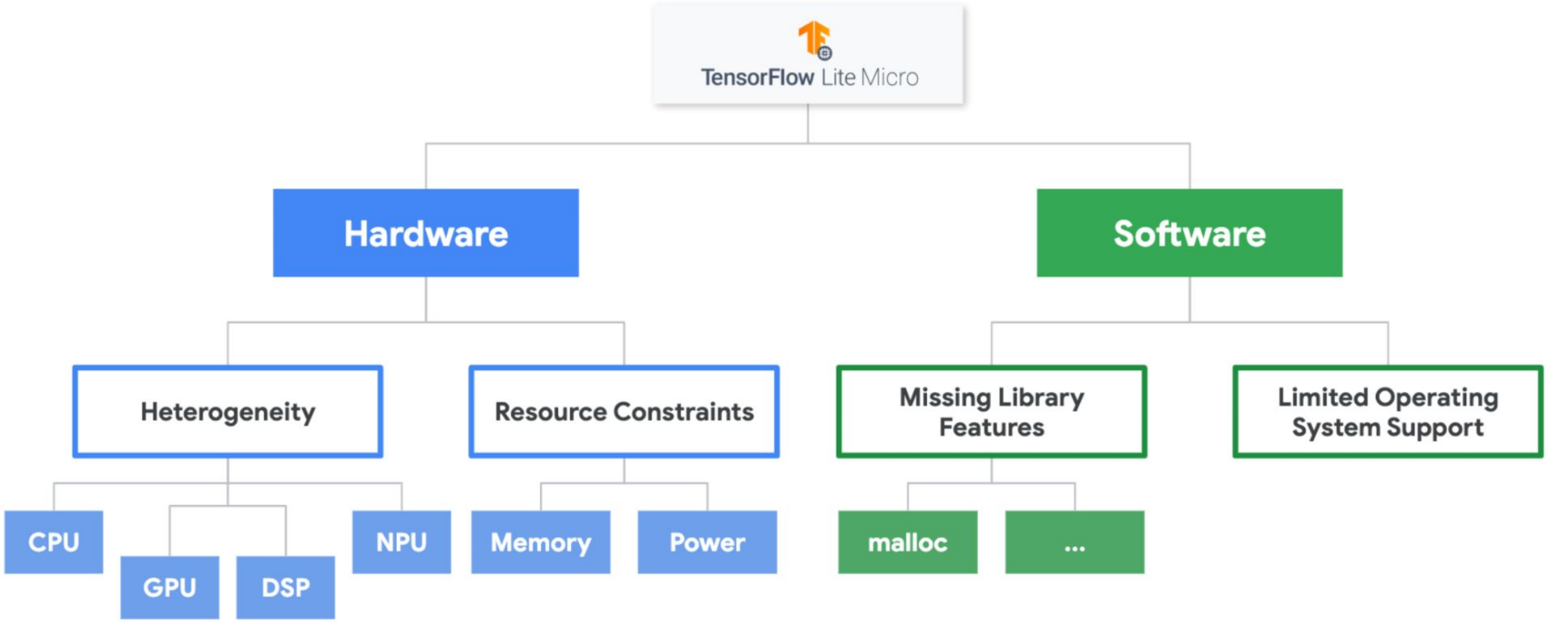
Roadmap

- High-Level Tensorflow Lite Micro Sales Pitch
- Review of Important Concepts
- Under the hood of Tensorflow Lite Micro
- Tensorflow Lite Micro Demo

Roadmap

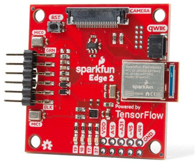
- High-Level Tensorflow Lite Micro Sales Pitch
- Review of Important Concepts
- Under the hood of Tensorflow Lite Micro
- Tensorflow Lite Micro Demo

What is TFLite Micro?

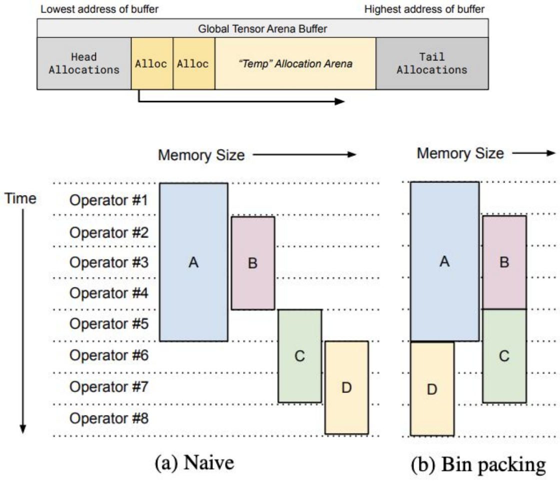


What is TFLite Micro?

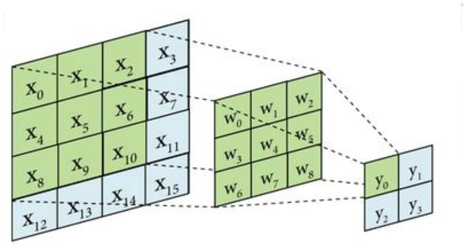
Hardware agnostic, with minimal external dependencies

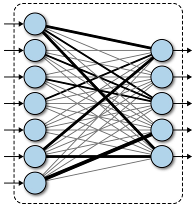


Minimal memory overhead for trained models



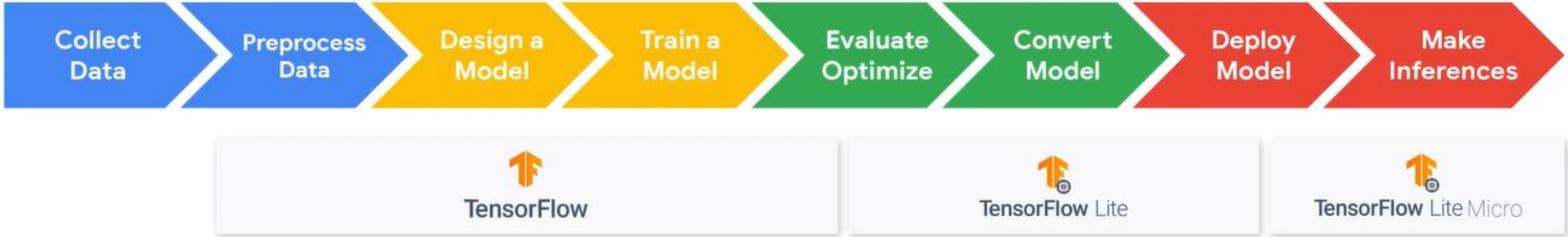
Uses TFLite to cover a **wide variety of model architectures**





...and **many more!**

TFLM in a Machine Learning Pipeline



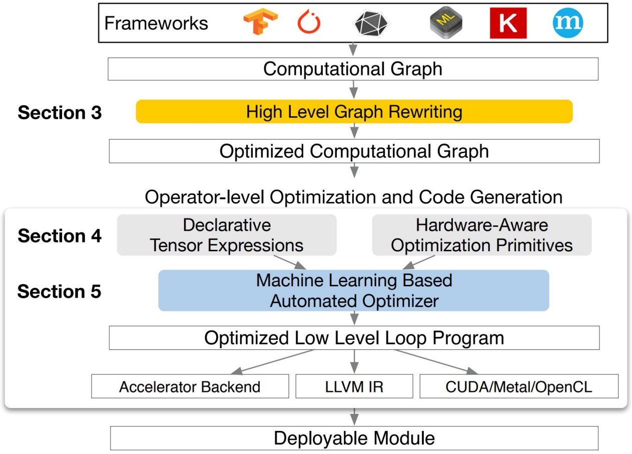
Ecosystem of TinyML Frameworks

tvm

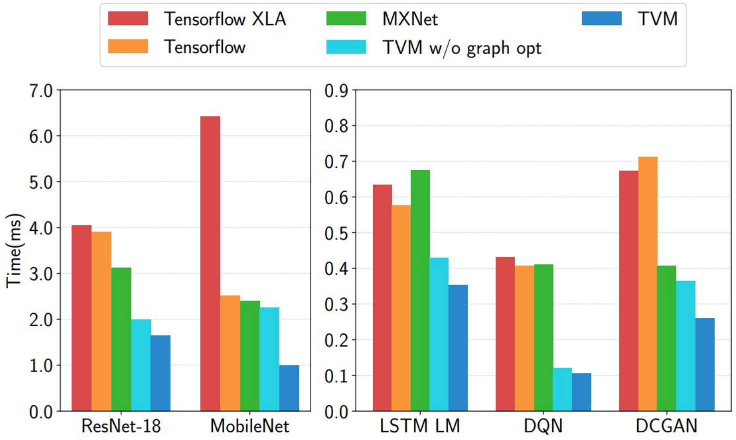
uTensor

STM32 Cube.AI

Ecosystem of TinyML Frameworks



SVM System Overview

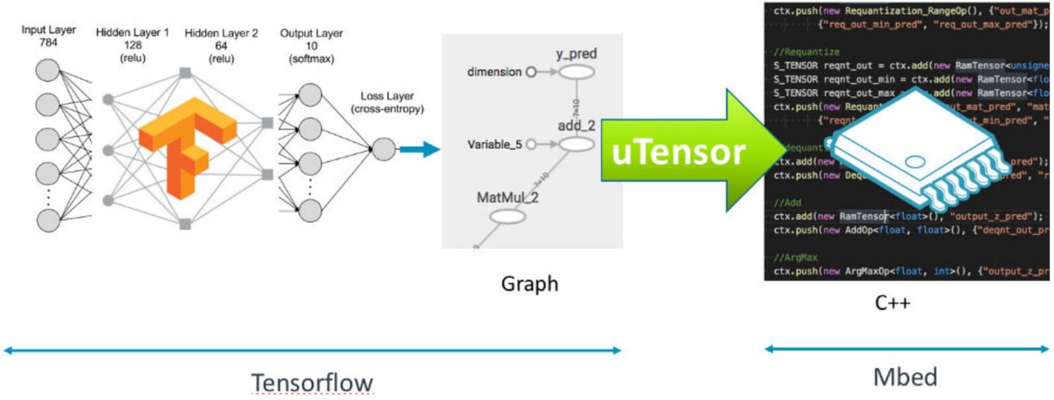


TVM Time Comparison

Compiler that exposes **graph-level** and **operator-level optimizations** to provide performance portability to deep learning workloads across diverse hardware back-ends

Ecosystem of TinyML Frameworks

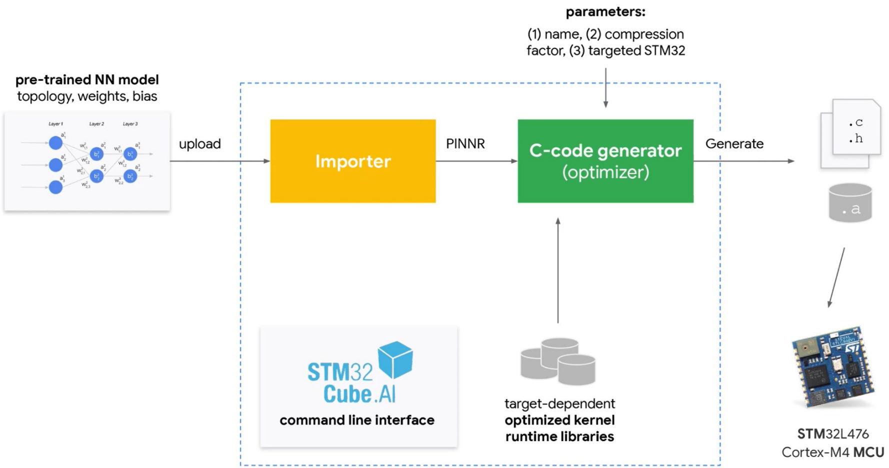
uTensor



TVM Time Comparison

Framework that **converts ML models** to **C++ source files** to increase human-readability and ease of editing with a core runtime **size of 2KB**

Ecosystem of TinyML Frameworks



STM32Cube.AI Workflow

Generates an internal representation of a network, **generates optimized code** using a **compiler** and **optimized kernel libraries**, generates executable code runnable on a microcontroller

Roadmap

- High-Level Tensorflow Lite Micro Sales Pitch
- Review of Important Concepts
- Under the hood of Tensorflow Lite Micro
- Tensorflow Lite Micro Demo

Embedded System

Computer hardware system with software that:

- is managed by **microcontrollers**, digital signal processors (DSP), application specific integrated circuit (ASIC), or field-programmable gate arrays (FPGA)
- is designed to perform a dedicated function under **tight constraints**, either as an independent system or as a part of a large system
- stores programming instructions in **read-only memory** or flash memory chips

Embedded Systems: Hardware

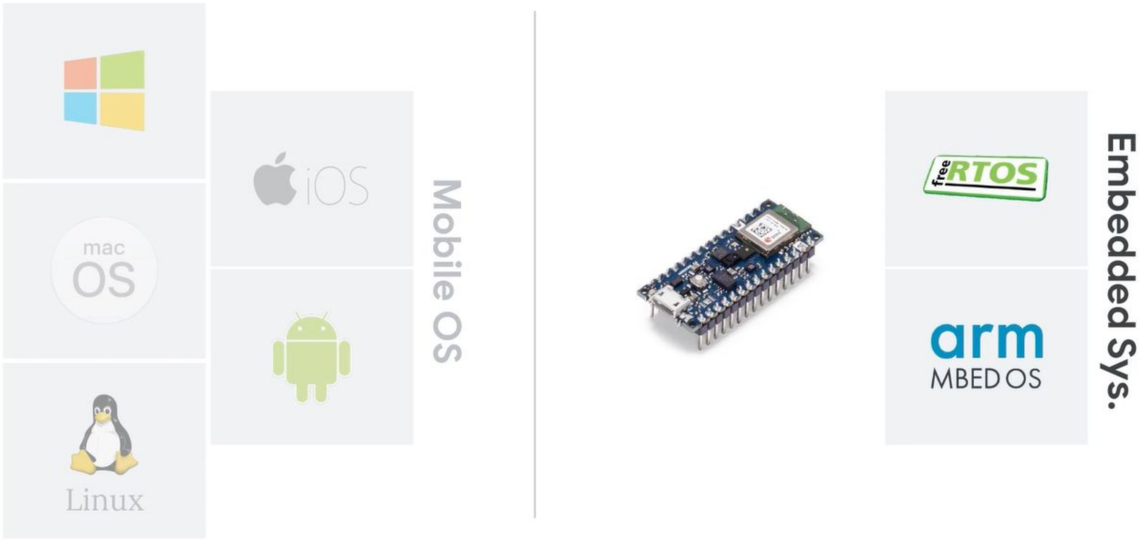
	Board	MCU / ASIC	Clock	Memory	Sensors	Radio
	Himax WE-I Plus EVB	HX6537-A 32-bit EM9D DSP	400 MHz	2MB flash 2MB RAM	Accelerometer, Mic, Camera	None
	Arduino Nano 33 BLE Sense	32-bit nRF52840	64 MHz	1MB flash 256kB RAM	Mic, IMU, Temp, Humidity, Gesture, Pressure, Proximity, Brightness, Color	BLE
	SparkFun Edge 2	32-bit ArtemisV1	48 MHz	1MB flash 384kB RAM	Accelerometer, Mic, Camera	BLE
	Espressif EYE	32-bit ESP32-D0WD	240 MHz	4MB flash 520kB RAM	Mic, Camera	WiFi, BLE

Embedded Systems: Hardware

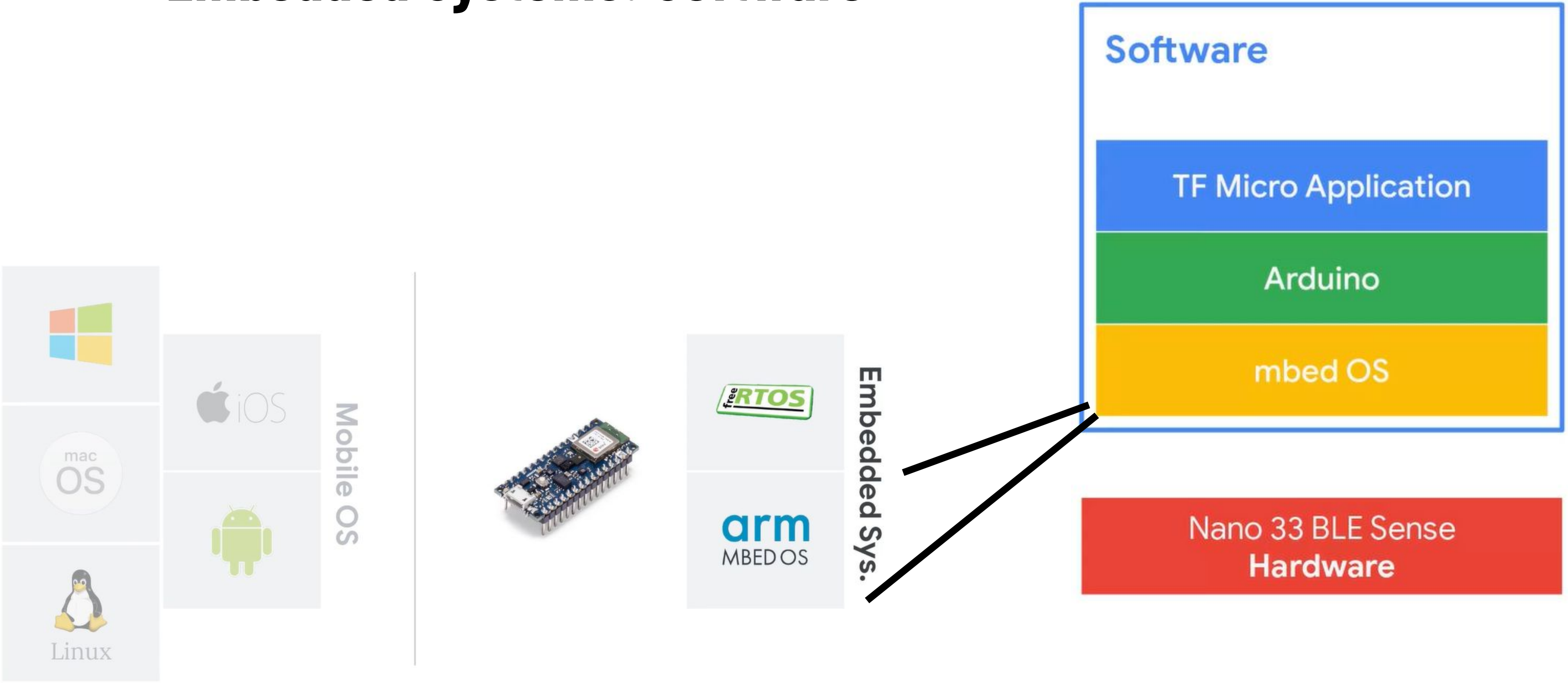
Low Cost

Low Power

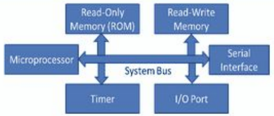
Embedded Systems: Software



Embedded Systems: Software

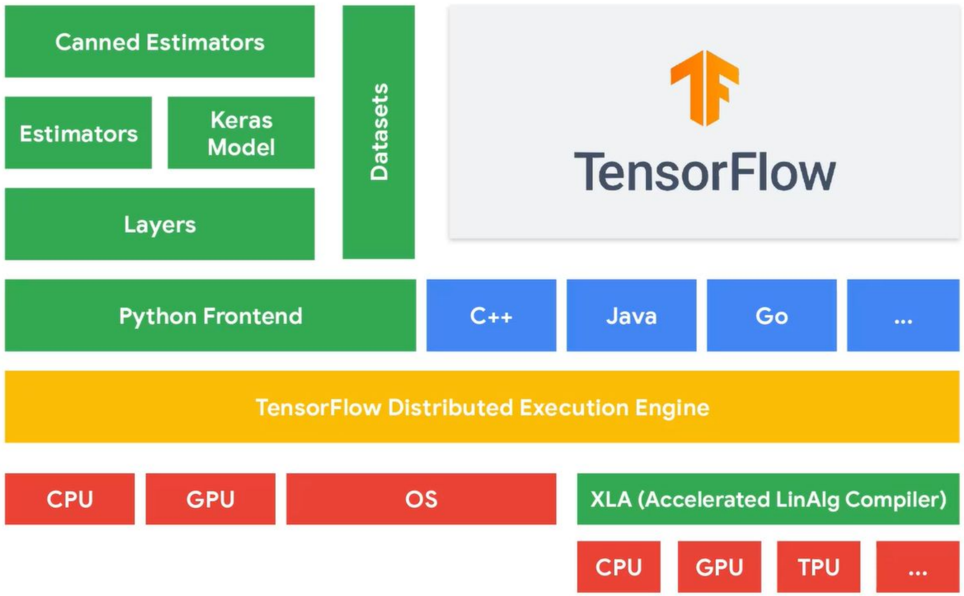


Microprocessor vs Microcontroller

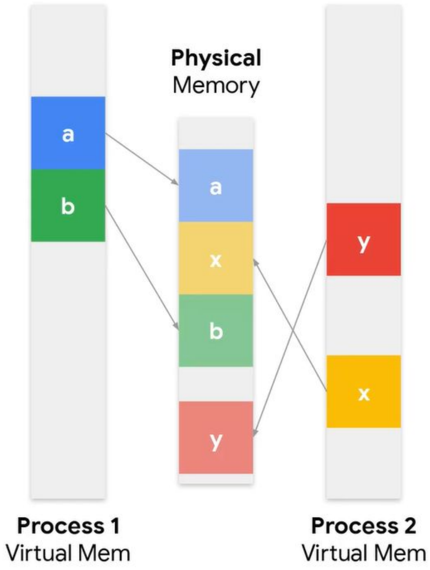
Microprocessor	Micro Controller
	
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

Tensorflow – Tensorflow Lite – Tensorflow Lite Micro

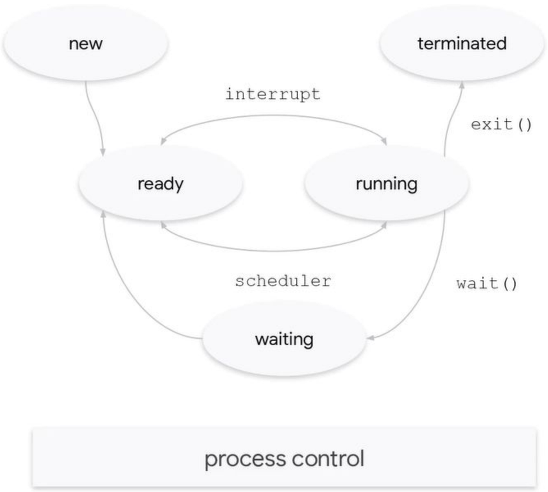
Library = 400MB



Tensorflow – Tensorflow Lite – Tensorflow Lite Micro



Virtual Memory

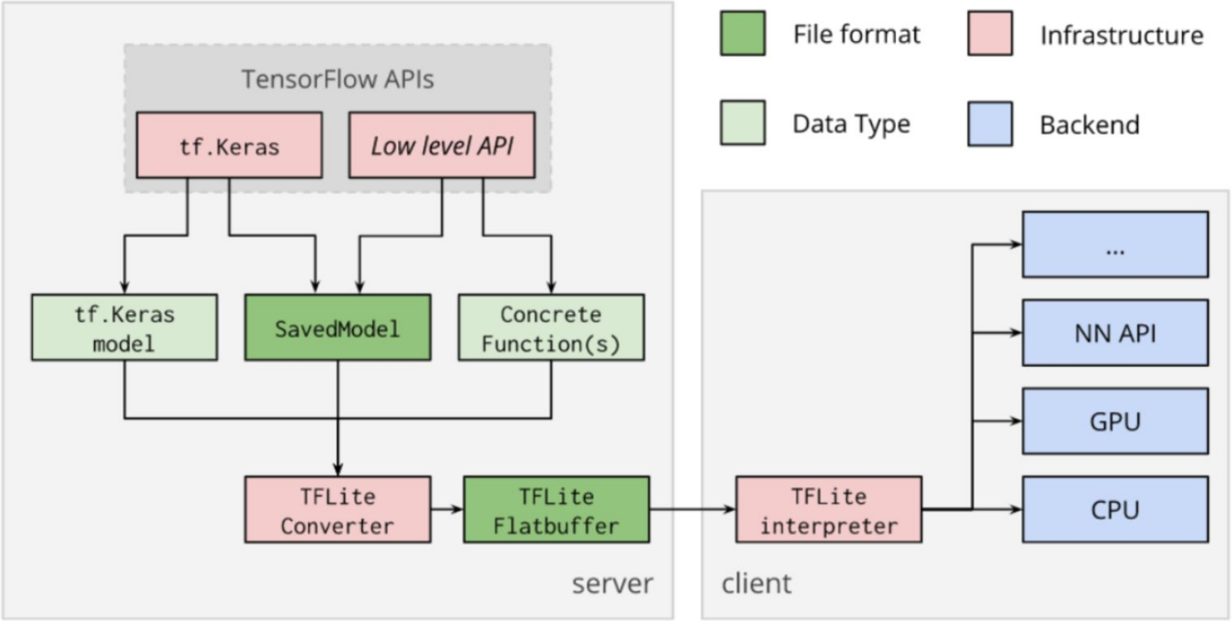


Multi-Threading

Tensorflow – Tensorflow Lite – Tensorflow Lite Micro

	Microprocessor	>	Microcontroller
Platform			
Compute	1GHz–4GHz	~10X	1MHz–400MHz
Memory	512MB–64GB	~10000X	2KB–512KB
Storage	64GB–4TB	~100000X	32KB–2MB
Power	30W–300W	~1000X	150μW–23.5mW

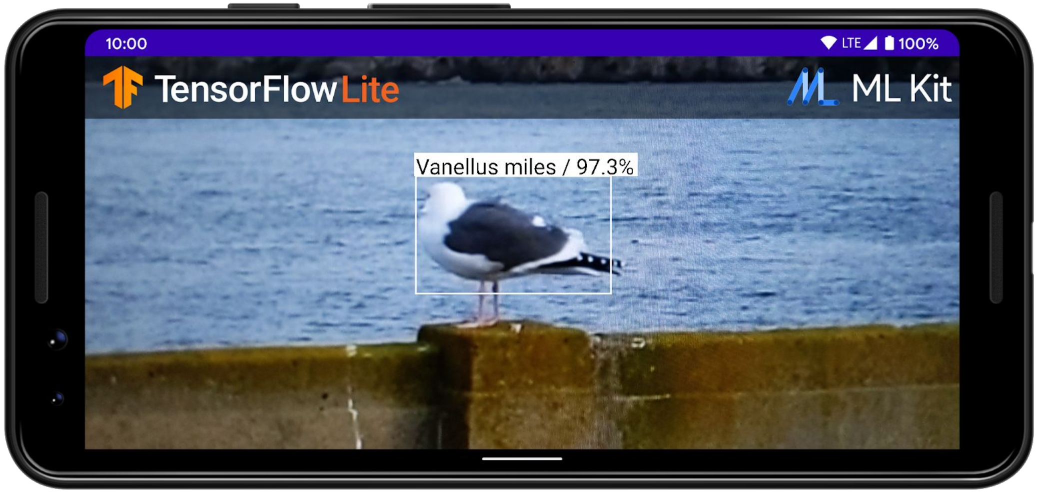
Tensorflow – Tensorflow Lite – Tensorflow Lite Micro



TFLite Conversion Process

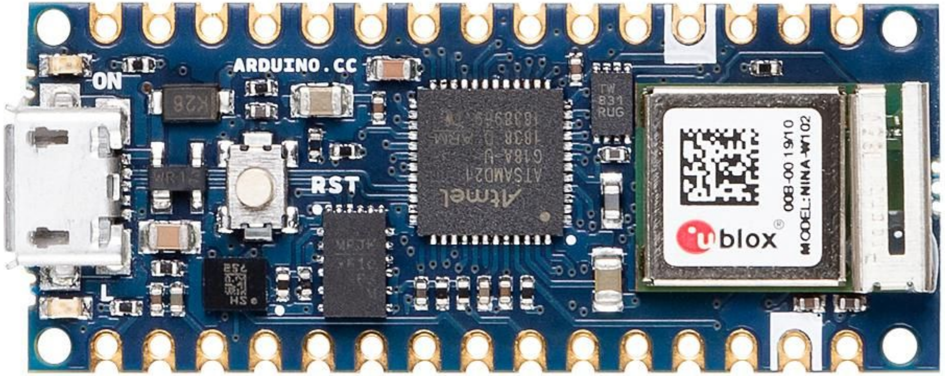
Tensorflow – **Tensorflow Lite** – Tensorflow Lite Micro

Library = 1MB



Tensorflow – Tensorflow Lite – Tensorflow Lite Micro

Library = 16KB



Background: Memory Hierarchy

"SRAM"

"DRAM"

"SSD"

"HDD"

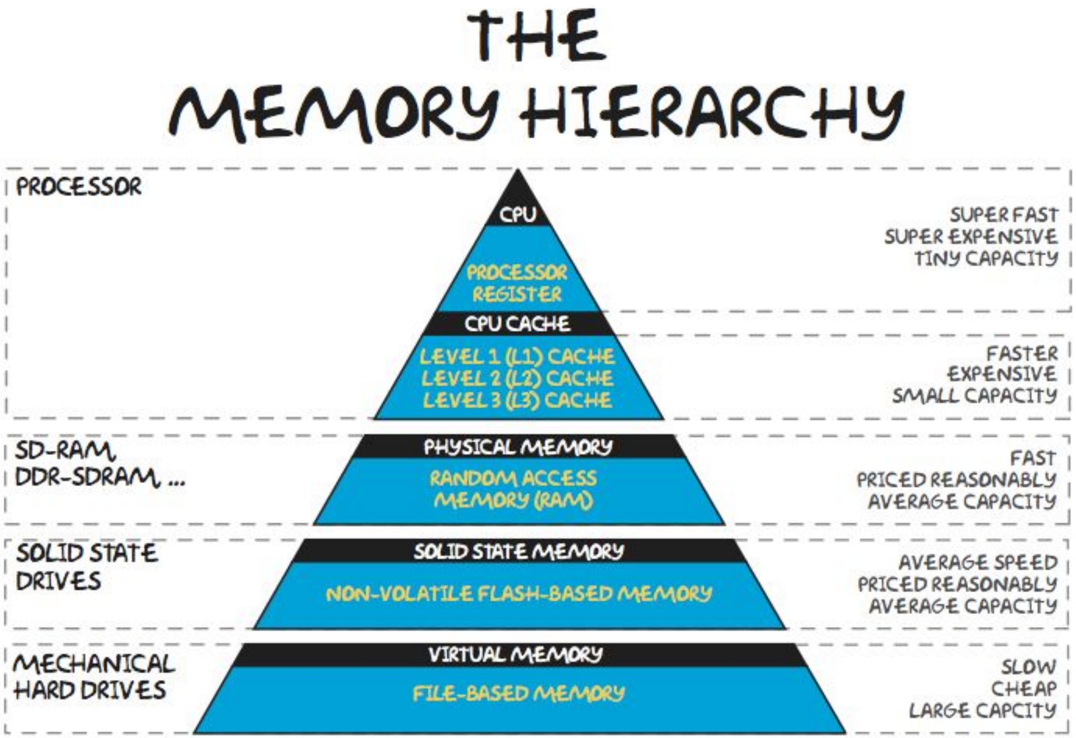
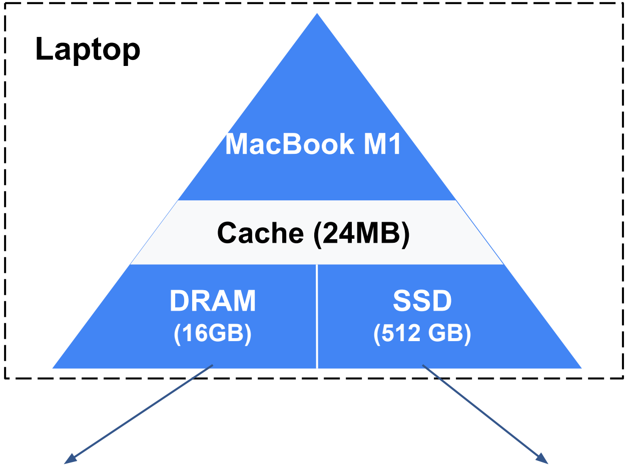


Image via teachbook.com.au

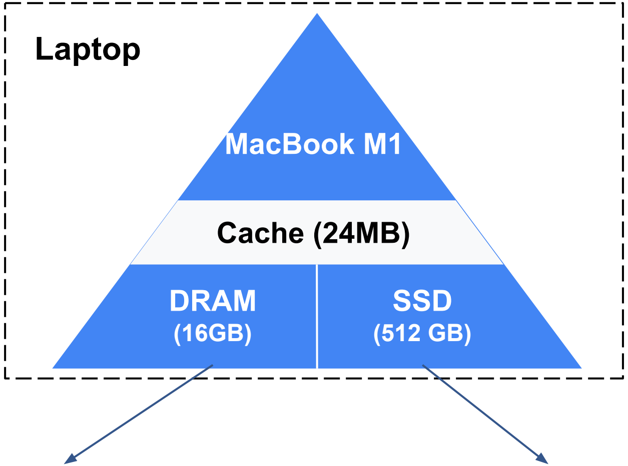
Background: Memory in MCUs

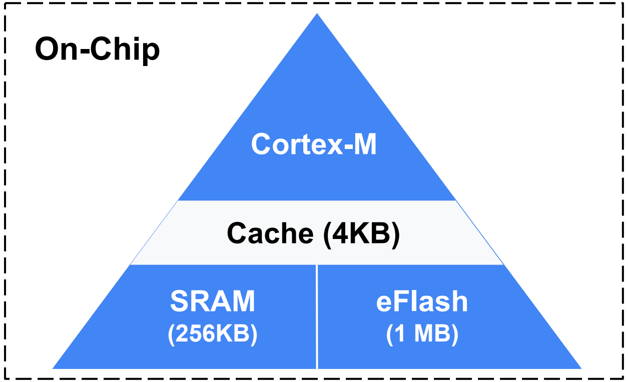


Volatile Memory:
Data is lost when the
computer is turned off

Non-volatile Memory:
Data persists even when the
computer is turned off

Background: Memory in MCUs





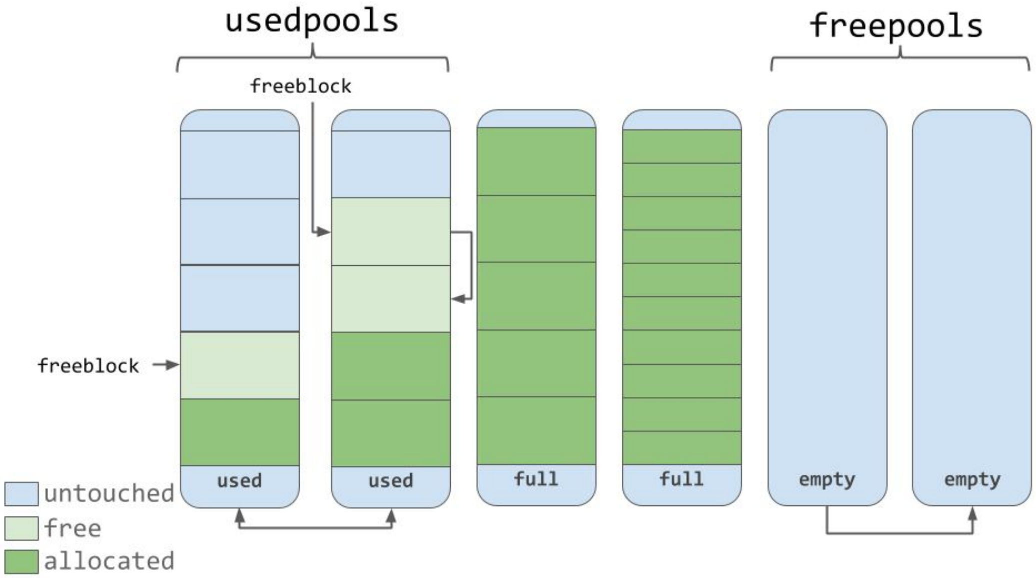
Volatile Memory:
Data is lost when the computer is turned off

Non-volatile Memory:
Data persists even when the computer is turned off

Interpreters vs Compilers

Python: **Interpreted** language

Interpreters vs Compilers

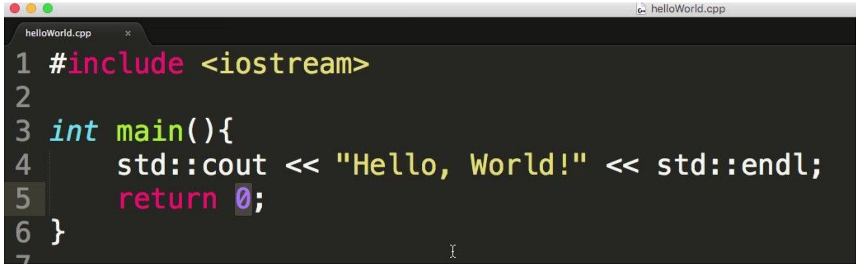


Interpreter Memory Management

Interpreters vs **Compilers**

Python: **Interpreted** language

C++: **Compiled** language



Roadmap

- High-Level Tensorflow Lite Micro Sales Pitch
- Review of Important Concepts
- Under the hood of Tensorflow Lite Micro
- Tensorflow Lite Micro Demo

Memory Allocation

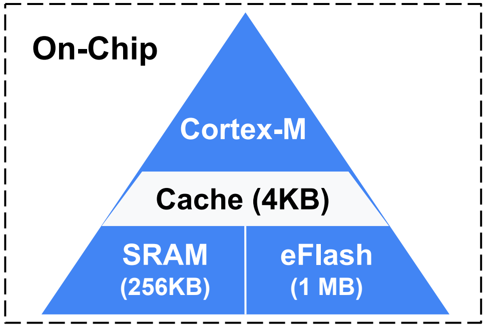
Memory Allocation

Why care about Memory?

Memory Allocation

Why care about Memory?

Memory Limits

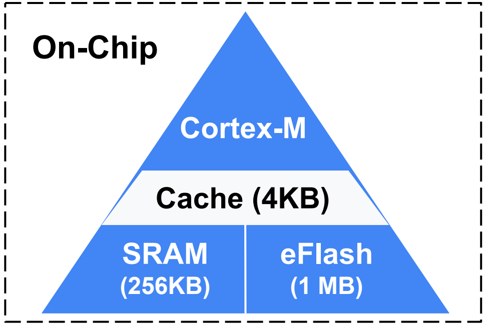


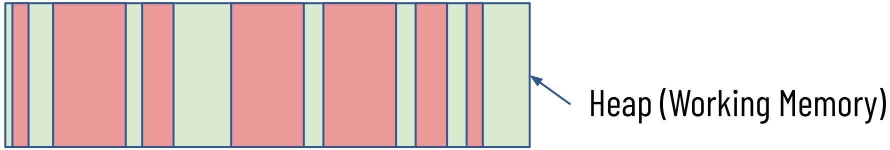
Memory Allocation

Why care about Memory?

Memory Limits

Long-Running Applications





Fragmentation

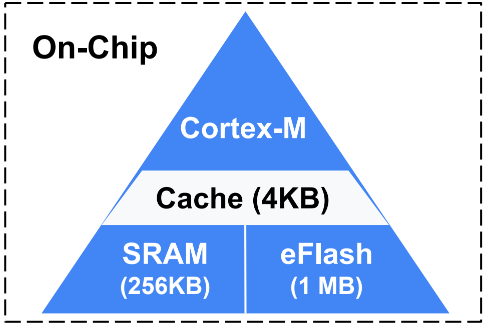
Memory Allocation

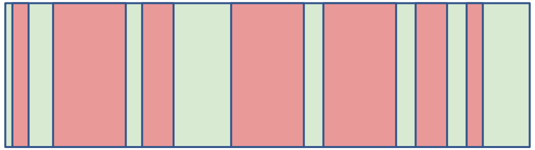
Why care about Memory?

Memory Limits

Long-Running Applications

Lack of OS Support





Fragmentation

~~malloc()~~
~~Virtual Memory~~

How **TFL Micro** solves these challenges

1. Ask developers to **supply a contiguous area of memory** to the interpreter, and in return the framework avoids any other memory allocations

```
constexpr int kTensorArenaSize = 2000;
uint8_t tensor_arena[kTensorArenaSize];

...

static tflite::MicroInterpreter static_interpreter(model, resolver,
  tensor_arena, kTensorArenaSize, error_reporting);
```

How **TFL Micro** solves these challenges

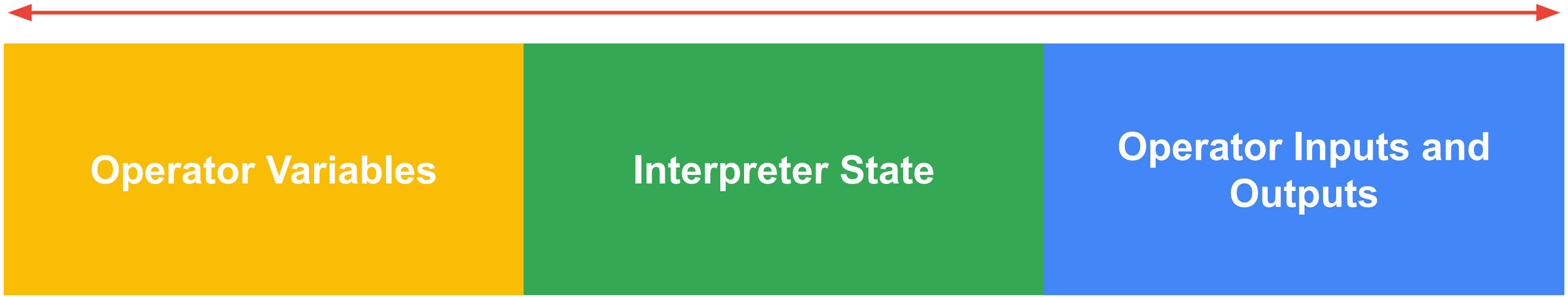
1. Ask developers to **supply a contiguous area of memory** to the interpreter, and in return the framework avoids any other memory allocations

2. Framework **guarantees that it won't allocate from this "arena" after initialization**, so long-running applications won't fail due to fragmentation

How **TFL Micro** solves these challenges

1. Ask developers to **supply a contiguous area of memory** to the interpreter, and in return the framework avoids any other memory allocations
2. Framework **guarantees that it won't allocate from this "arena" after initialization**, so long-running applications won't fail due to fragmentation
3. Ensures clear budget for the memory used by ML, and that the **framework has no dependency on OS facilities needed by malloc or new**

`uint8_t tensor_arena[kTensorArenaSize]`



Loading the Model

```
model = tflite::GetModel(g_model);
```

g_model variable is an array of bytes that we exported from TensorFlow

Holds all of the information about the model, its operators, their connections, and the trained weights

`g_model` FlatBuffer **Format**

Metadata (version, quantization ranges, etc)

Name	*Args*	*Input*	*Output*	*Weights*
Conv2D	3x3	0	1	2
FC	-	1	3	4
Softmax	-	3	5	-

Weight Buffers

Index	*Type*	*Values*
2	Float	0.01, 7.45, 9.23, ...
4	Int8	34, 19, 243, ...
...	...	...

Loading the Model

```
model = tflite::GetModel(g_model);
```

GetModel call lets us access the model's data elements

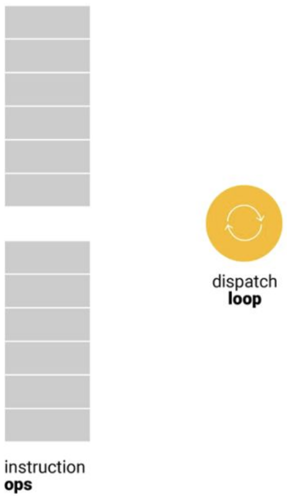
This call doesn't do any data movement or copying, so it's very memory efficient, it just creates a thin wrapper to read the information in-place.

Loading the Model

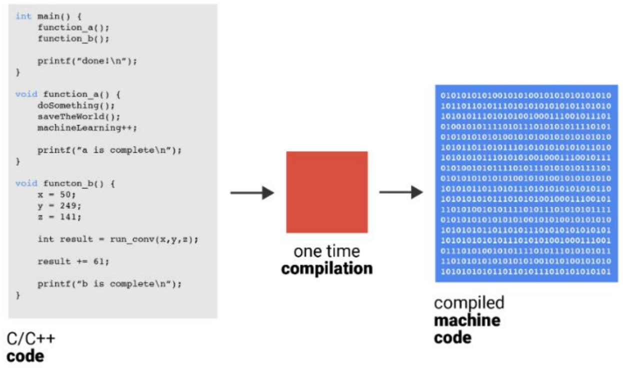
```
model = tflite::GetModel(g_model);
```

The resulting model is a class object that lets us access the model weights and operations.

Interpreter-based design for machine learning



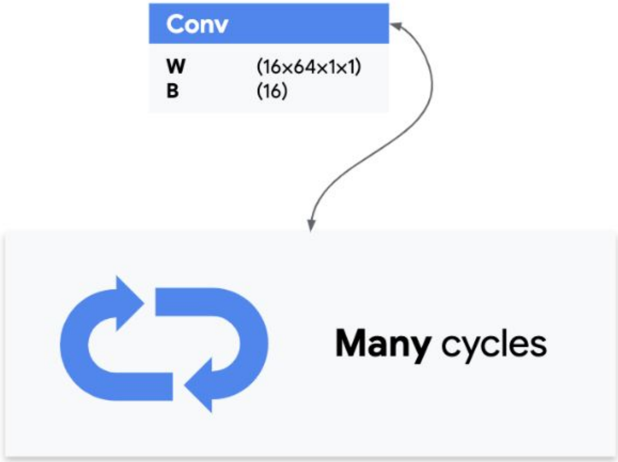
Interpreter
(generally **slower** than compiled code)



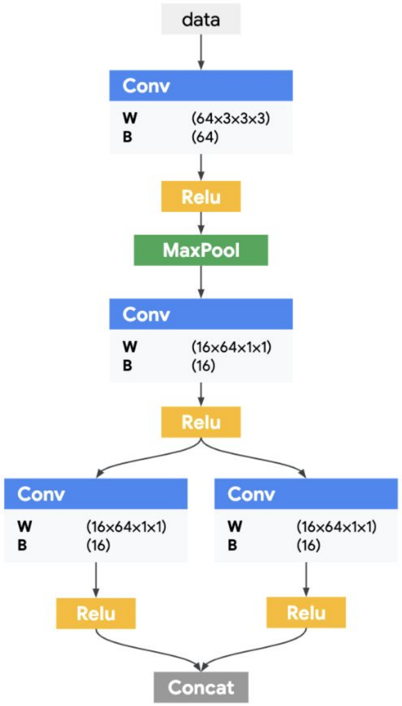
Compiler
(generally **faster** than interpreted code)

Interpreter-based design for machine learning

- A single line of ML code can take up to **millions of operations**
- The bottleneck is **matrix multiplications** and other operations, so an interpreter is used.



ML models have low interpreter overhead



Model	Total Cycles	Calculation Cycles	Interpreter Overhead
Visual Wake Words (Ref)	18,990.8K	18,987.1K	< 0.1%
Google Hotword (Ref)	36.4K	34.9K	4.1%

Example interpreter execution

```
if (op_type == CONV2D) {
  Convolution2d(conv_size, input, output, weights);
} else if (op_type == FULLY_CONNECTED) {
  FullyConnected(input, output, weights)

}
```

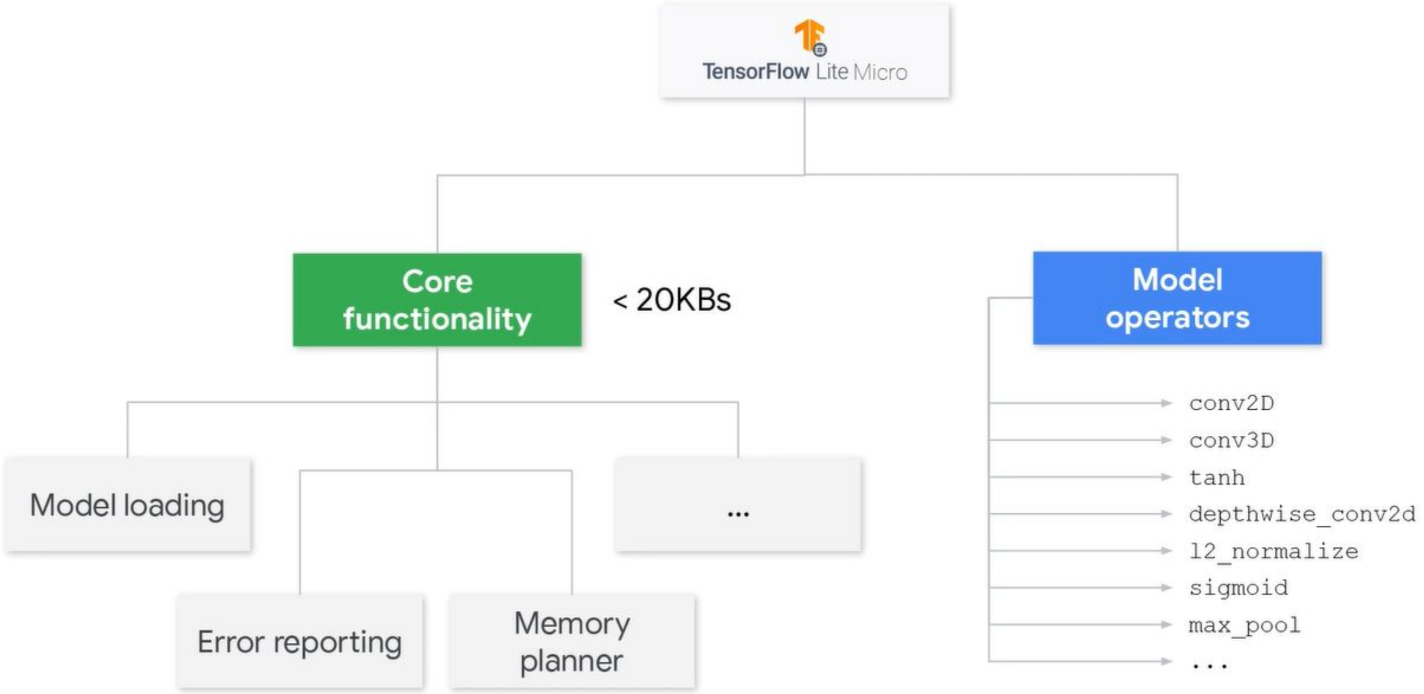
NN Operator Support

TensorFlow

1,400 operators

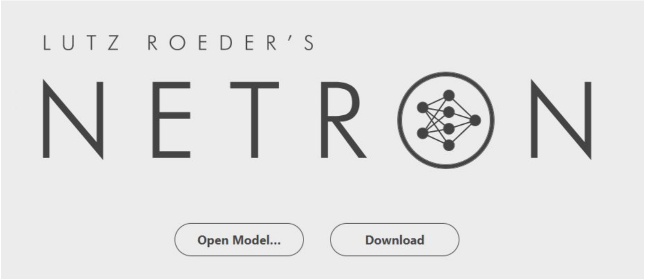
Take up Flash Memory

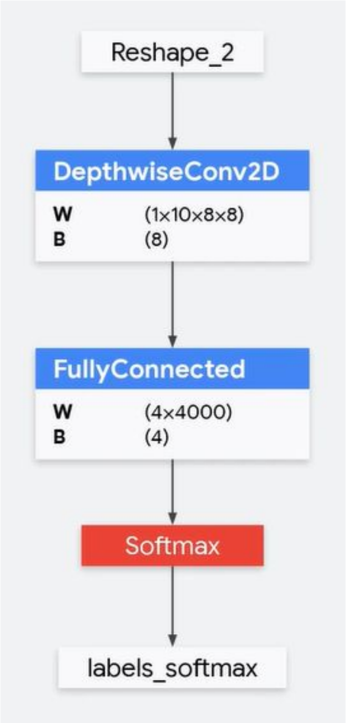
NN Operator Support



Choosing the Necessary Operators

Model

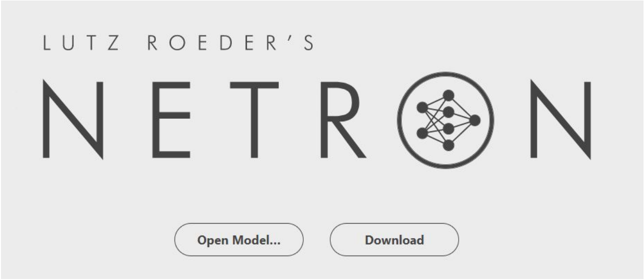


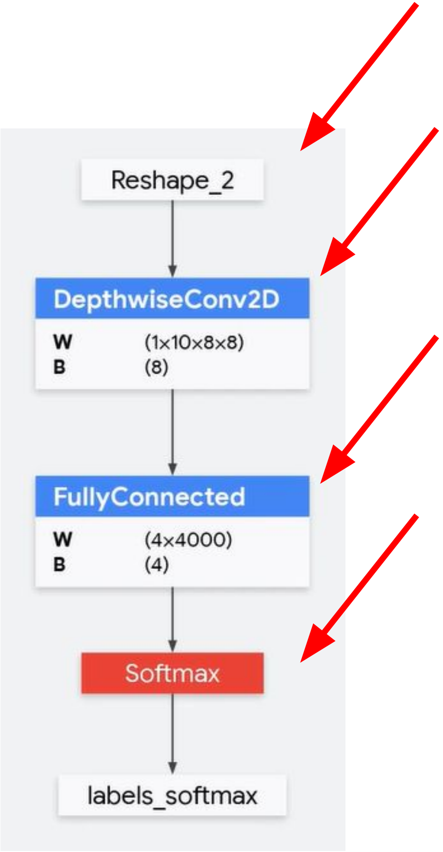


Visible Operators

Choosing the Necessary Operators

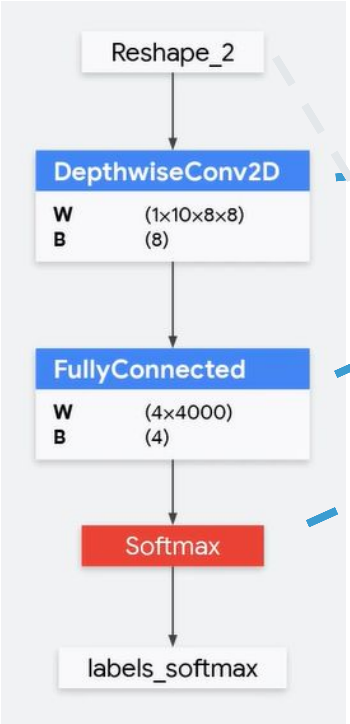
Model





Visible Operators

OpResolver

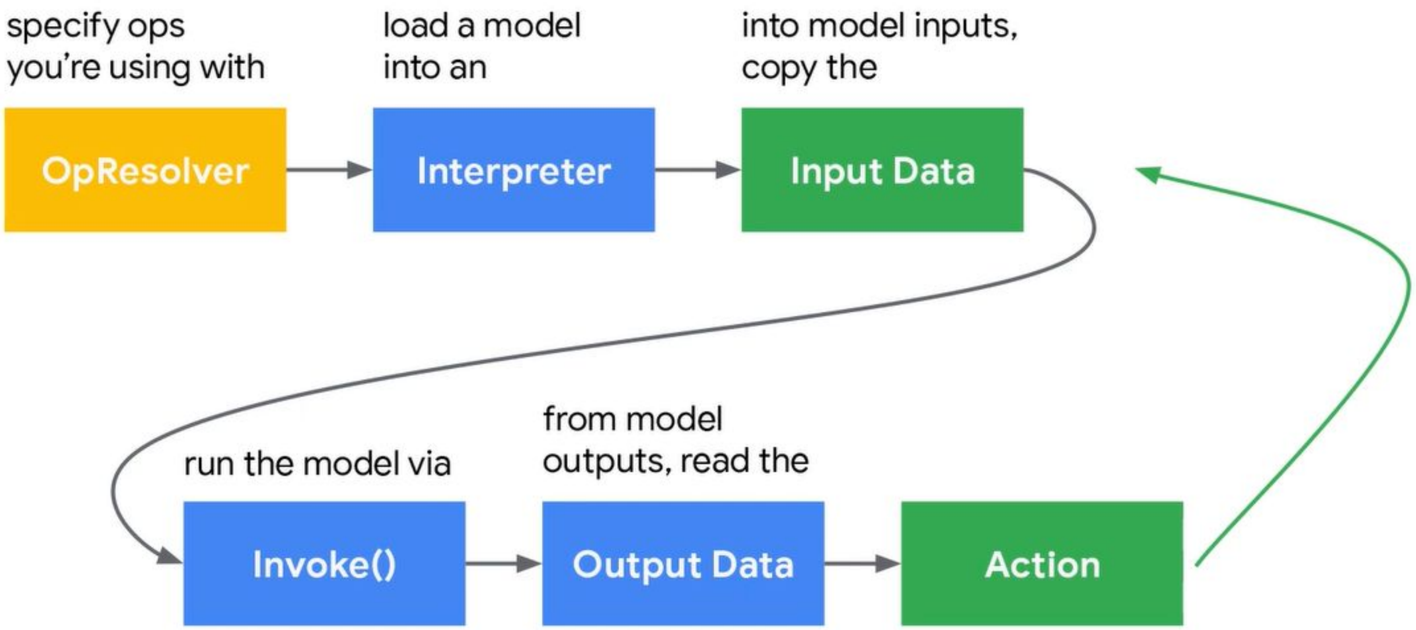



```
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

Roadmap

- High-Level Tensorflow Lite Micro Sales Pitch
- Review of Important Concepts
- Under the hood of Tensorflow Lite Micro
- Tensorflow Lite Micro Demo

TFLM Workflow

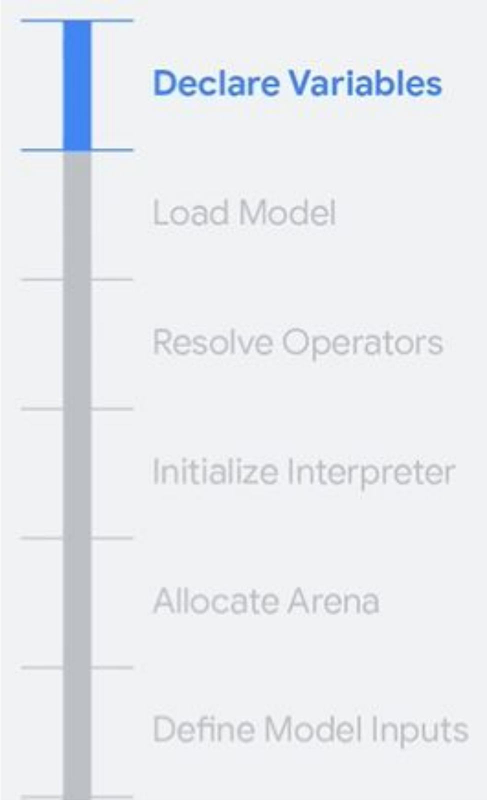


Initialization

- **Declare Variables**
- Load Model
- Resolve Operators
- Initialize Interpreter
- Allocate Arena
- Define Model Inputs

```
// Globals, used for compatibility
// with Arduino-style sketches.

namespace {
    tflite::ErrorReporter* error_reporter = nullptr;
    const tflite::Model* model = nullptr;
    tflite::MicroInterpreter* interpreter = nullptr;
    TfLiteTensor* model_input = nullptr;
    FeatureProvider* feature_provider = nullptr;
    RecognizeCommands* recognizer = nullptr;
    int32_t previous_time = 0;

    // Create an area of memory to use for input,
    // output, and intermediate arrays.
    // The size of this will depend on the model
    // you're using, and may need to be
    // determined by experimentation.

    constexpr int kTensorArenaSize = 10 * 1024;
    uint8_t tensor_arena[kTensorArenaSize];
    int8_t feature_buffer[kFeatureElementCount];
    int8_t* model_input_buffer = nullptr;
}
```

Initialization

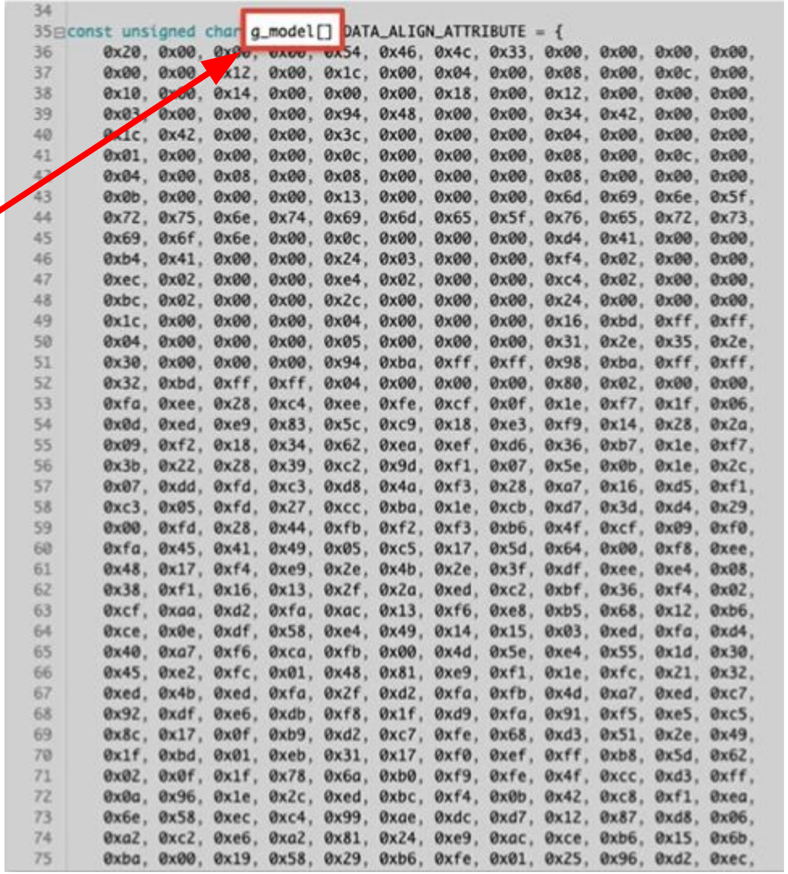



```
// Globals, used for compatibility
// with Arduino-style sketches.

namespace {
    tflite::ErrorReporter* error_reporter = nullptr;
    const tflite::Model* model = nullptr;
    tflite::MicroInterpreter* interpreter = nullptr;
    TfLiteTensor* model_input = nullptr;
    FeatureProvider* feature_provider = nullptr;
    RecognizeCommands* recognizer = nullptr;
    int32_t previous_time = 0;

    // Create an area of memory to use for input,
    // output, and intermediate arrays.
    // The size of this will depend on the model
    // you're using, and may need to be
    // determined by experimentation.

    constexpr int kTensorArenaSize = 10 * 1024;
    uint8_t tensor_arena[kTensorArenaSize];
    int8_t feature_buffer[kFeatureElementCount];
    int8_t* model_input_buffer = nullptr;
}
```

Initialization

- Declare Variables
- **Load Model**
- Resolve Operators
- Initialize Interpreter
- Allocate Arena
- Define Model Inputs

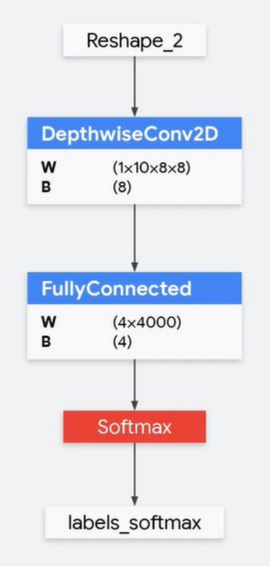
```
model =
tflite::GetModel(g_model);
```



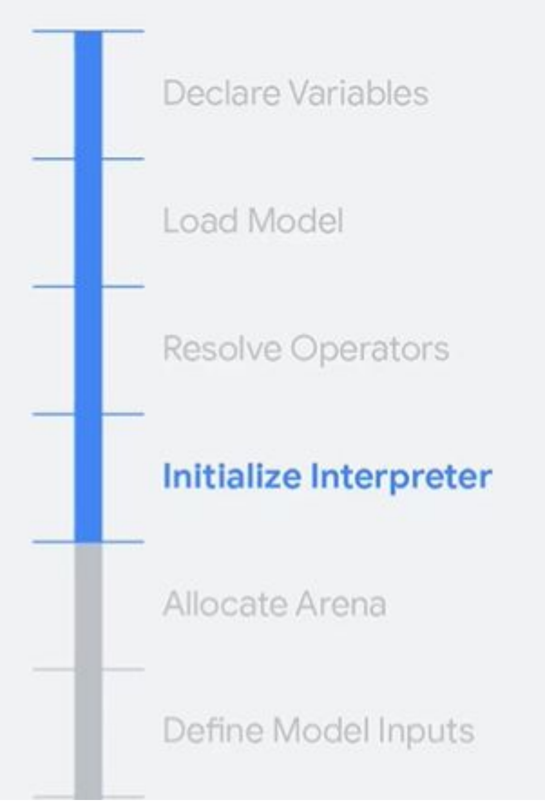
Initialization

- Declare Variables
- Load Model
- **Resolve Operators**
- Initialize Interpreter
- Allocate Arena
- Define Model Inputs

```
static tflite::MicroMutableOpResolver<4>
micro_op_resolver(error_reporter);
if (micro_op_resolver.AddBuiltin(
  tflite::BuiltinOperator_DEPTHWISE_CONV_2D,
  tflite::ops::micro::Register_DEPTHWISE_CONV_2D()) != kTfLiteOk)
{
  return;
}
if (micro_op_resolver.AddBuiltin(
  tflite::BuiltinOperator_FULLY_CONNECTED,
  tflite::ops::micro::Register_FULLY_CONNECTED()) != kTfLiteOk)
{
  return;
}
if (micro_op_resolver.AddBuiltin(
  tflite::BuiltinOperator_SOFTMAX,
  tflite::ops::micro::Register_SOFTMAX()) != kTfLiteOk)
{
  return;
}
if (micro_op_resolver.AddBuiltin(
    tflite::BuiltinOperator_RESHAPE,
    tflite::ops::micro::Register_RESHAPE()) != kTfLiteOk)
{
  return;
}
```

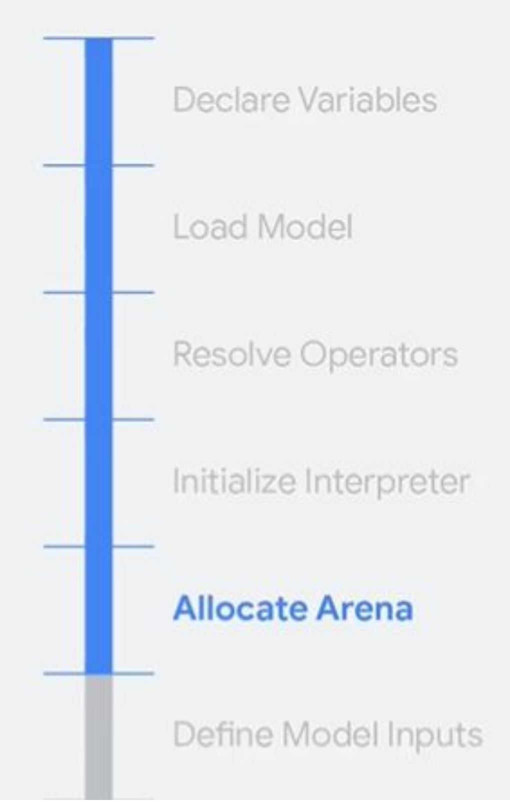


Initialization




```
// Build an interpreter to run the model with.

static tflite::MicroInterpreter static_interpreter(
    model, micro_op_resolver, tensor_arena,
    kTensorArenaSize, error_reporter);

interpreter = &static_interpreter;
```

Initialization




```
// Allocate memory from the tensor_arena for
// the model's tensors.

TfLiteStatus allocate_status =
interpreter->AllocateTensors();
```

Initialization

- Declare Variables
- Load Model
- Resolve Operators
- Initialize Interpreter
- Allocate Arena
- **Define Model Inputs**

```
model_input = interpreter->input(0);

model_input_buffer = model_input->data.int8;
```

TFLM Executive Summary

- TFLM is built to fit with **embedded system constraints**
- Very **small** binary footprint
- **No** dynamic memory allocation
- **No** dependencies on standard C/C++ Libraries
- **No** operating system dependencies

Roadmap

- High-Level Tensorflow Lite Micro Sales Pitch
- Review of Important Concepts
- Under the hood of Tensorflow Lite Micro
- Tensorflow Lite Micro Demo

Extra Slides

References

- David, Robert, et al. "Tensorflow lite micro: Embedded machine learning for tinyml systems." Proceedings of Machine Learning and Systems 3 (2021): 800-811.
- Chen, Tianqi, et al. "{TVM}: An automated {End-to-End} optimizing compiler for deep learning." 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018.
- UTensor (https://github.com/uTensor/uTensor)
- Microprocessor VS Microcontroller (https://www.linquip.com/blog/difference-between-microprocessor-and-microcontroller/)
- Embedded Systems (https://www.heavy.ai/technical-glossary/embedded-systems)
- TFLite Conversion (https://www.tensorflow.org/lite/models/convert)
- Edx TinyML Course (https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning)
- Edx TinyML Course Material (https://github.com/tinyMLx/courseware/tree/master/edX#chapter-44-tensorflow-lite-micro)

Suggested Readings

- Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., ... & Xuesong, X. (2021). Mlperf tiny benchmark. arXiv preprint arXiv:2106.07597. 2106.07597.pdf (arxiv.org)

- David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., ... & Rhodes, R. (2021). Tensorflow lite micro: Embedded machine learning for tinyml systems. Proceedings of Machine Learning and Systems, 3, 800-811. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems (arxiv.org)

- Chen, Tianqi, et al. "{TVM}: An automated {End-to-End} optimizing compiler for deep learning." 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018. https://www.usenix.org/system/files/osdi18-chen.pdf

Additional Links:

- Memory Management in Tensorflow Lite Micro
  - Source: tflite-micro/memory_management.md at main · tensorflow/tflite-micro (github.com)
- MLPerf Tiny Deep Learning Benchmarks for Embedded Devices
  - Source: mlcommons/tiny: MLPerf™ Tiny is an ML benchmark suite for extremely low-power systems such as microcontrollers (github.com)