

CS249r: Frameworks



Sep. 25, 2023

Goals for today

1. Logistics
2. Lecture
3. Paper discussions
4. Guest speaker



Course Logistics

Class Schedule

	Start	~End
Lecture	12:45 PM	1:40 PM
Break	1:40 PM	1:45 PM
Paper Discussion (combined)	1:45 PM	2:25 PM
Break	2:25 PM	2:30 PM
Guest Lecture	2:30 PM	3:30 PM

My Office Hours

- Every Monday
 - After class @ 3:30pm
 - Meet in class / office
 - My focus
 - Technical material
 - Lecture content
 - Paper readings / discussions etc.

Updated Office Hours



Matthew Stewart

Postdoctoral Researcher,
Harvard University

Office Hours: 5-6 PM Thursday
(SEC 1.412)



Ikechukwu Uchendu

Computer Science, PhD
Student, Harvard University

Office Hours: 9-10 AM
Wednesday (TBD)



Jason Jabbour

Computer Science, PhD
Student, Harvard University

Office Hours: 11-12 PM Friday
(TBD)



Jessica Quayle

Computer Science, PhD
Student, Harvard University

Office Hours: 1-2 PM Tuesday
(SEC 5.403)

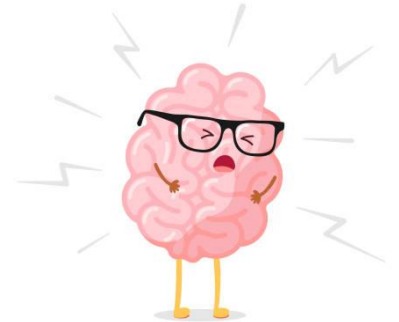
Paper Discussion Sign-up Sheet

1. Please take a few minutes to sign up for next week's paper discussion:
2. Submit to Canvas:
 - a. Paper Reading Group - 1 for your first time
 - b. Paper Reading Group - 2 for your second time
3. **Extra Credit for signing up for 3rd spot**
 - a. **2.5% credit!**



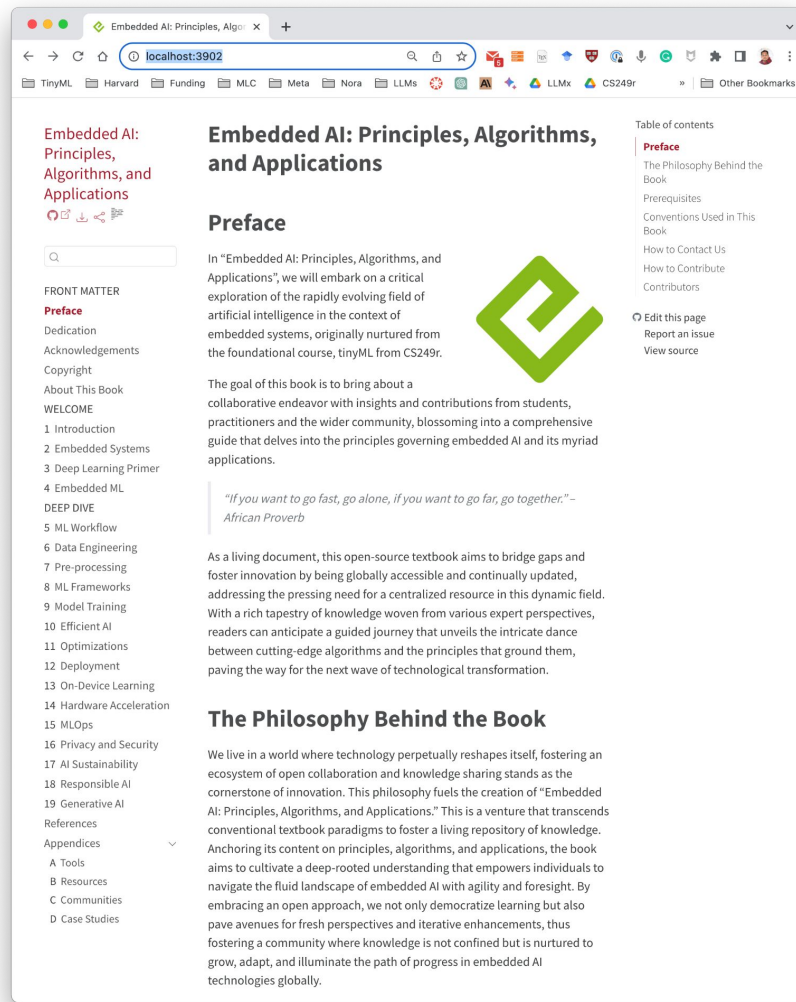
Assignment #1

- Any issues with the hardware?
 - To be expected when working with embedded systems ツ
- New Due Dates:
 - **Part 1:** Oct 2nd
 - **Part 2:** Oct 10th (due to Indigenous Peoples Day)
- Questions?



Book

- Updates
 - Week of topic
 - i. Meet with Matthew & VJ
 - ii. Create a google doc
 - iii. Share with staff
 - iv. Iterate on rough outline →
 - v. Start drafting collaboratively
 - vi. Push to GitHub
 - vii. Peer review



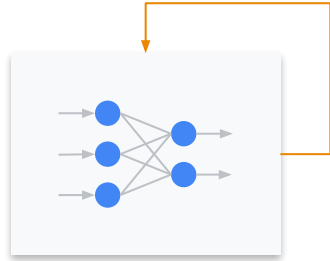
Project Sign-Ups

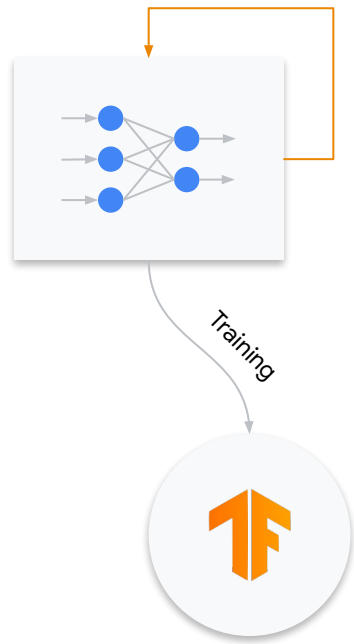
- Join the #projects channel and post your interests
- Once you find a group, sign up on this spreadsheet:
 - (same sheet used for scribing sign-up)

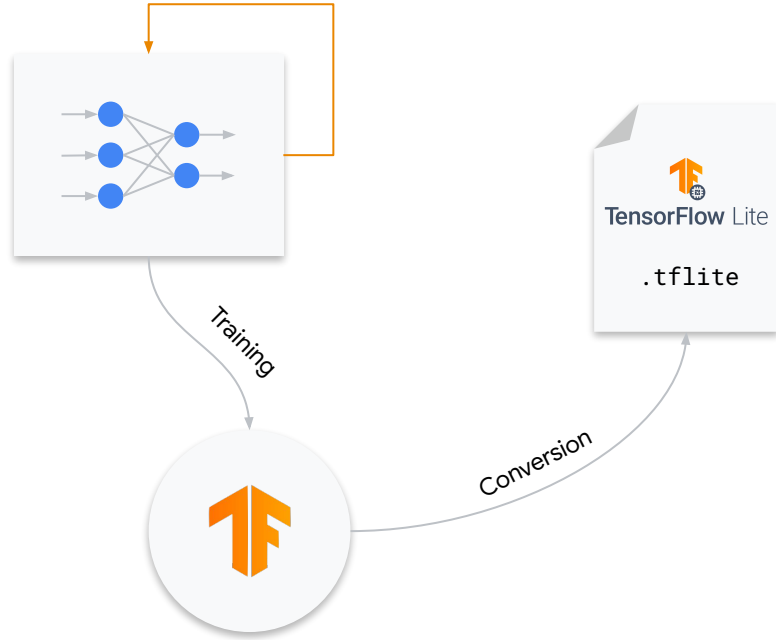


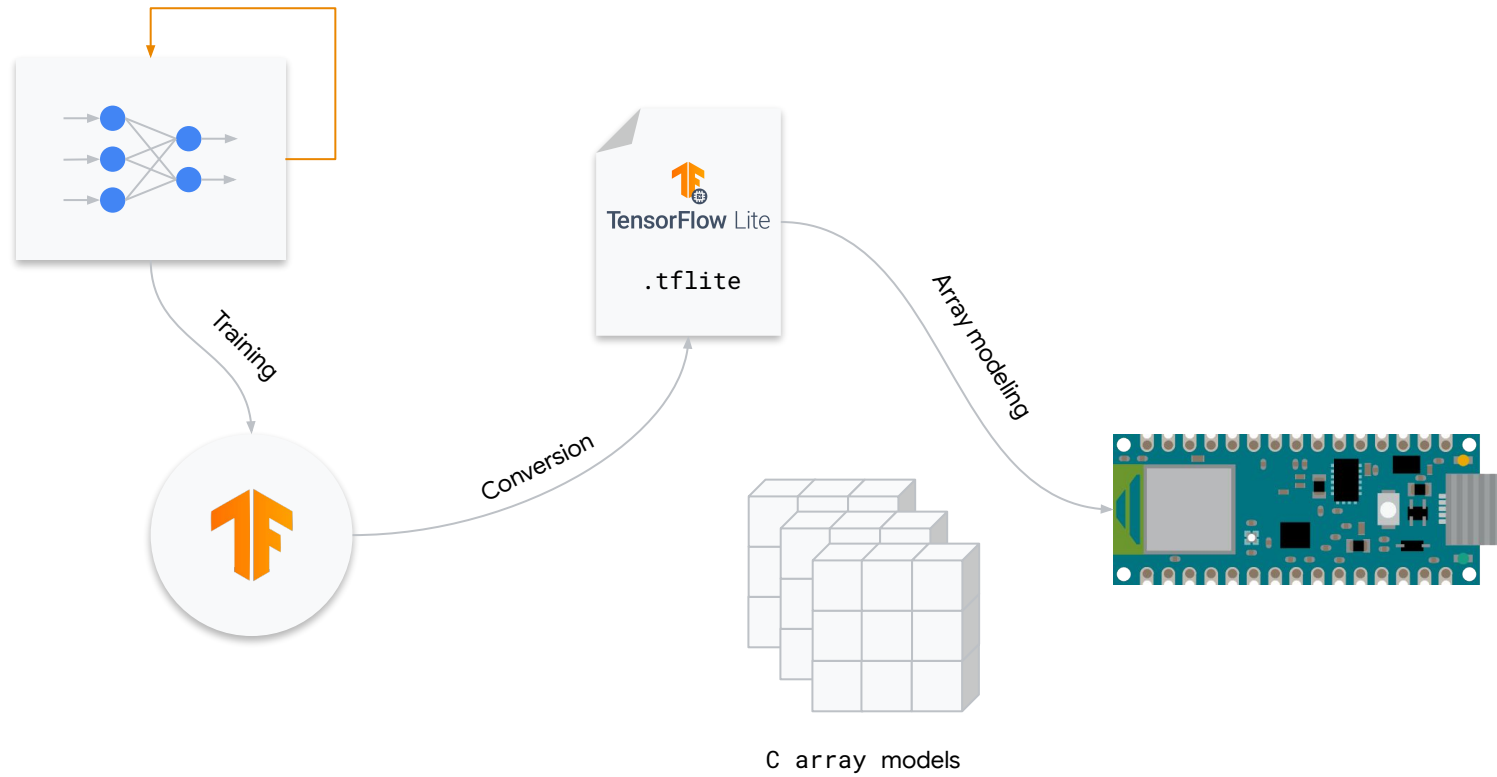


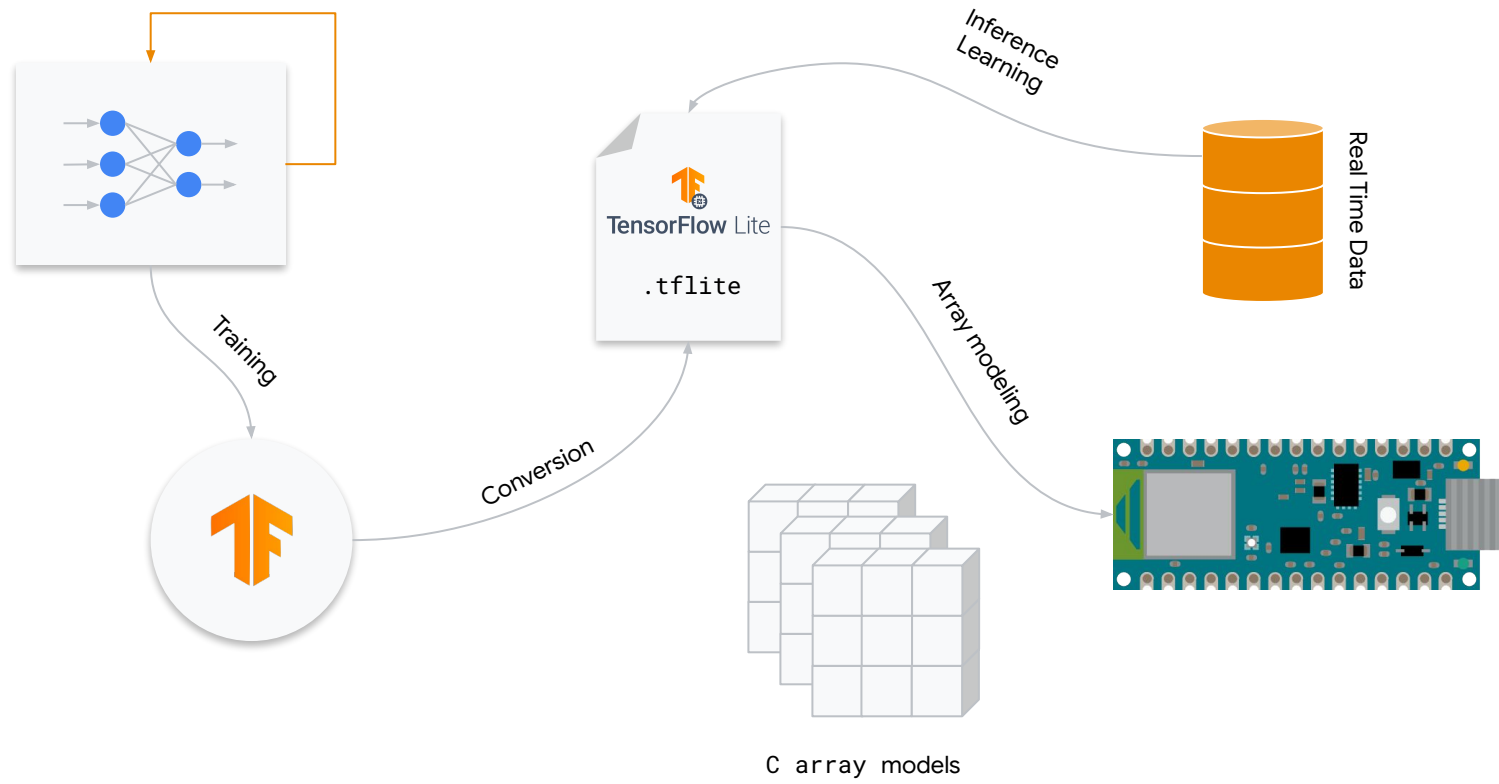
Lecture







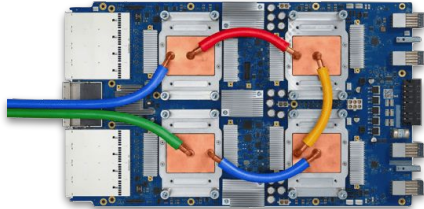




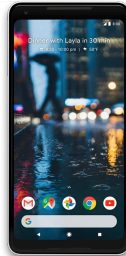
Framework Differences



TensorFlow

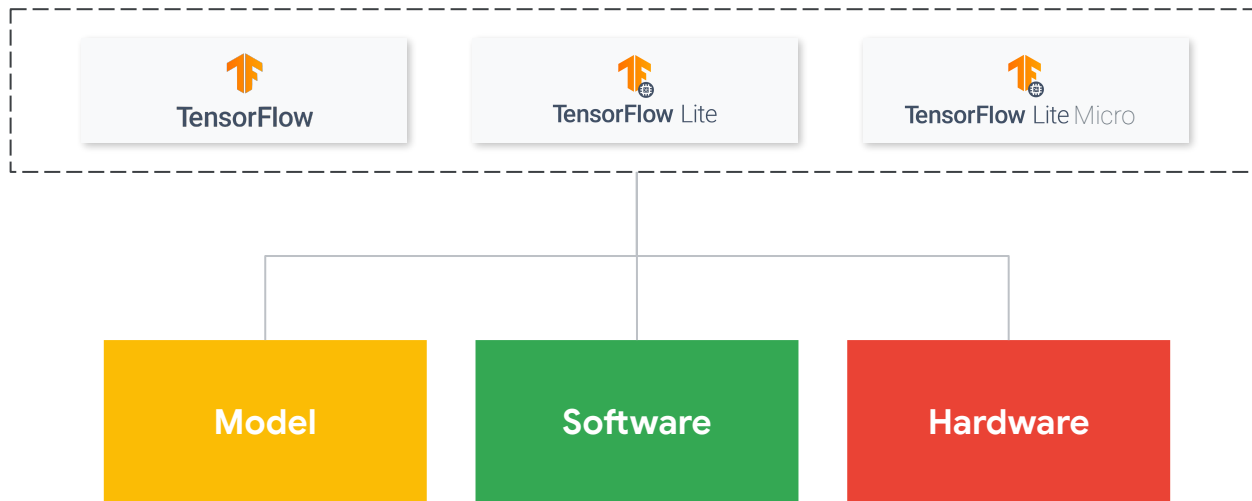


TensorFlow Lite









TensorFlow Lite Micro





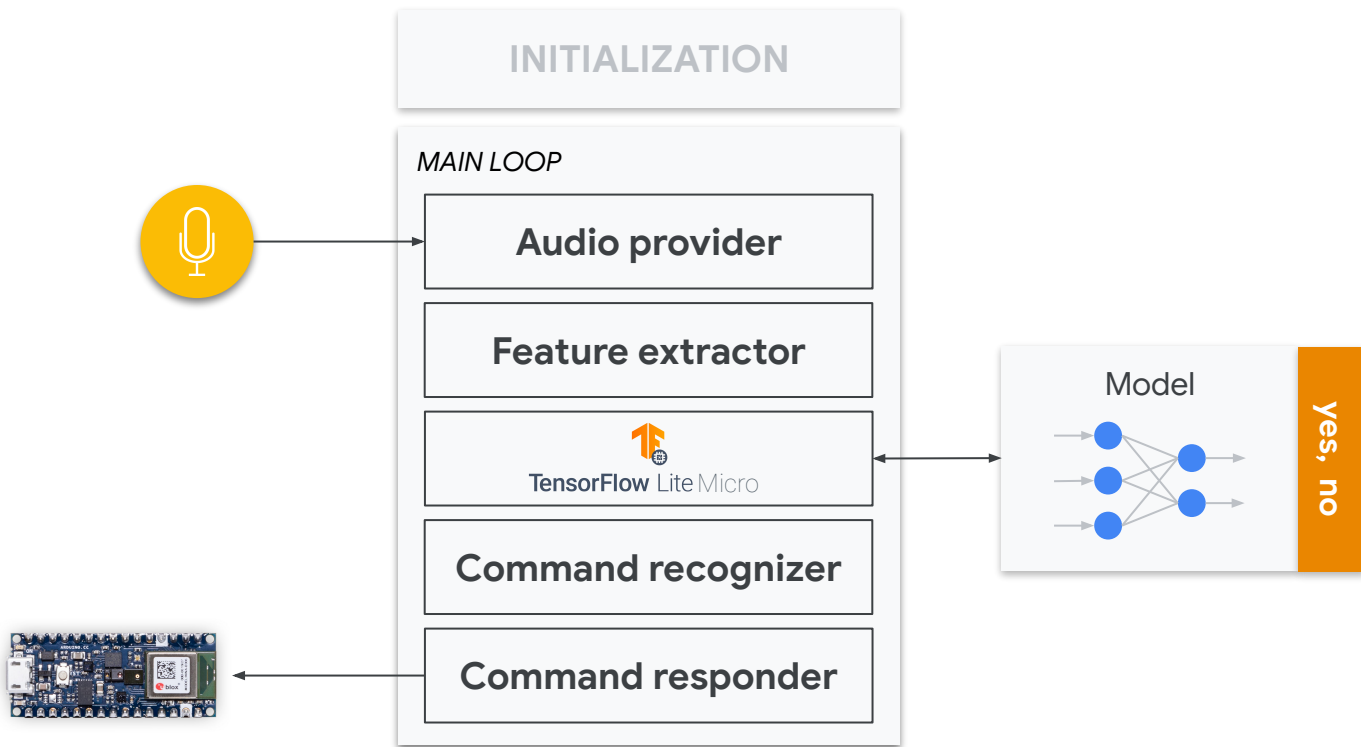
Model	 TensorFlow	 TensorFlow Lite	 TensorFlow Lite Micro
Training	Yes	No	No
Inference	Yes <i>(but inefficient on edge)</i>	Yes <i>(and efficient)</i>	Yes <i>(and even more efficient)</i>
How Many Ops	~1400	~130	~50
Native Quantization Tooling + Support	No	Yes	Yes

Software	 TensorFlow	 TensorFlow Lite	 TensorFlow Lite Micro
Needs an OS	Yes	Yes	No
Memory Mapping of Models	No	Yes	Yes
Delegation to accelerators	Yes	Yes	No

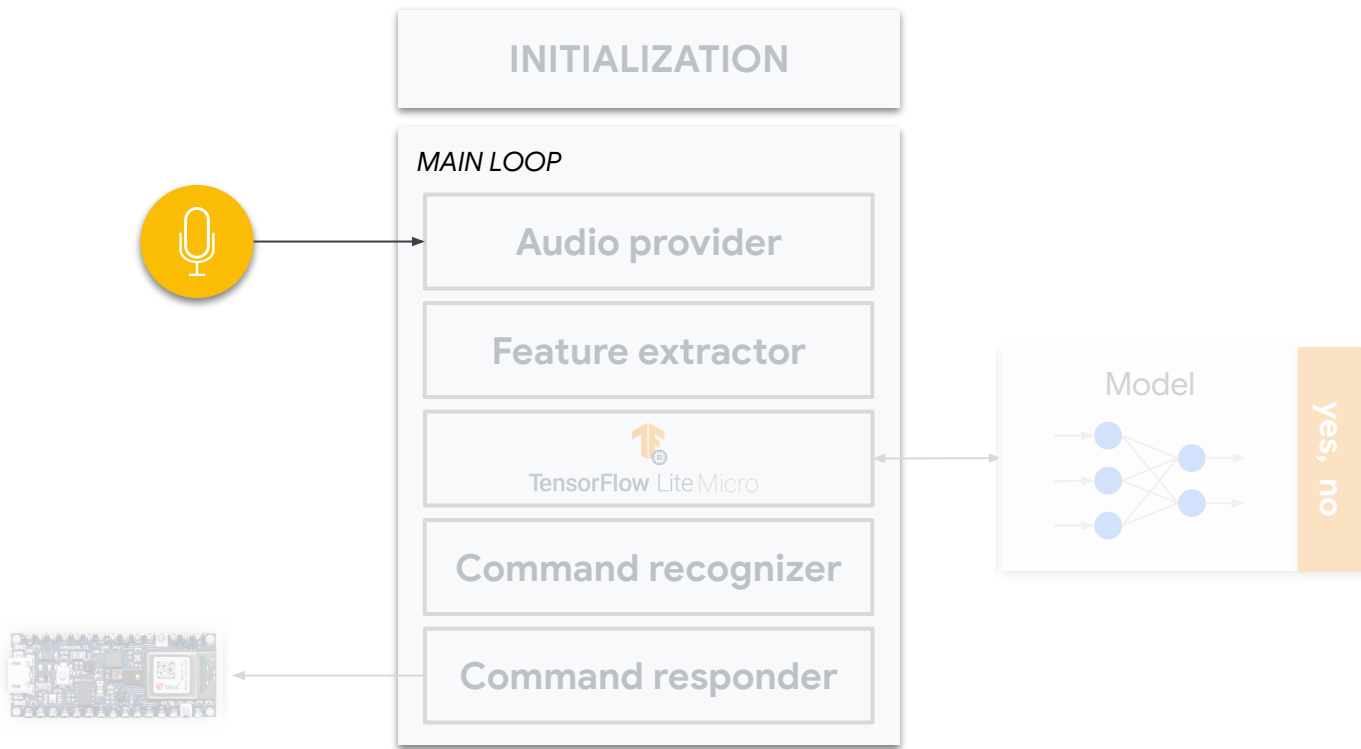
Hardware	 TensorFlow	 TensorFlow Lite	 TensorFlow Lite Micro
Base Binary Size	3MB+	100KB	~10 KB
Base Memory Footprint	~5MB	300KB	20KB
Optimized Architectures	X86, TPUs, GPUs	Arm Cortex A, x86	Arm Cortex M, DSPs, MCUs

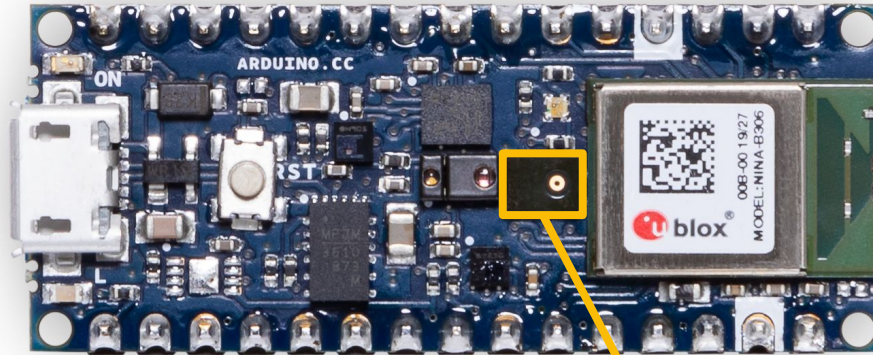
Keyword Spotting Example

Keyword Spotting Components



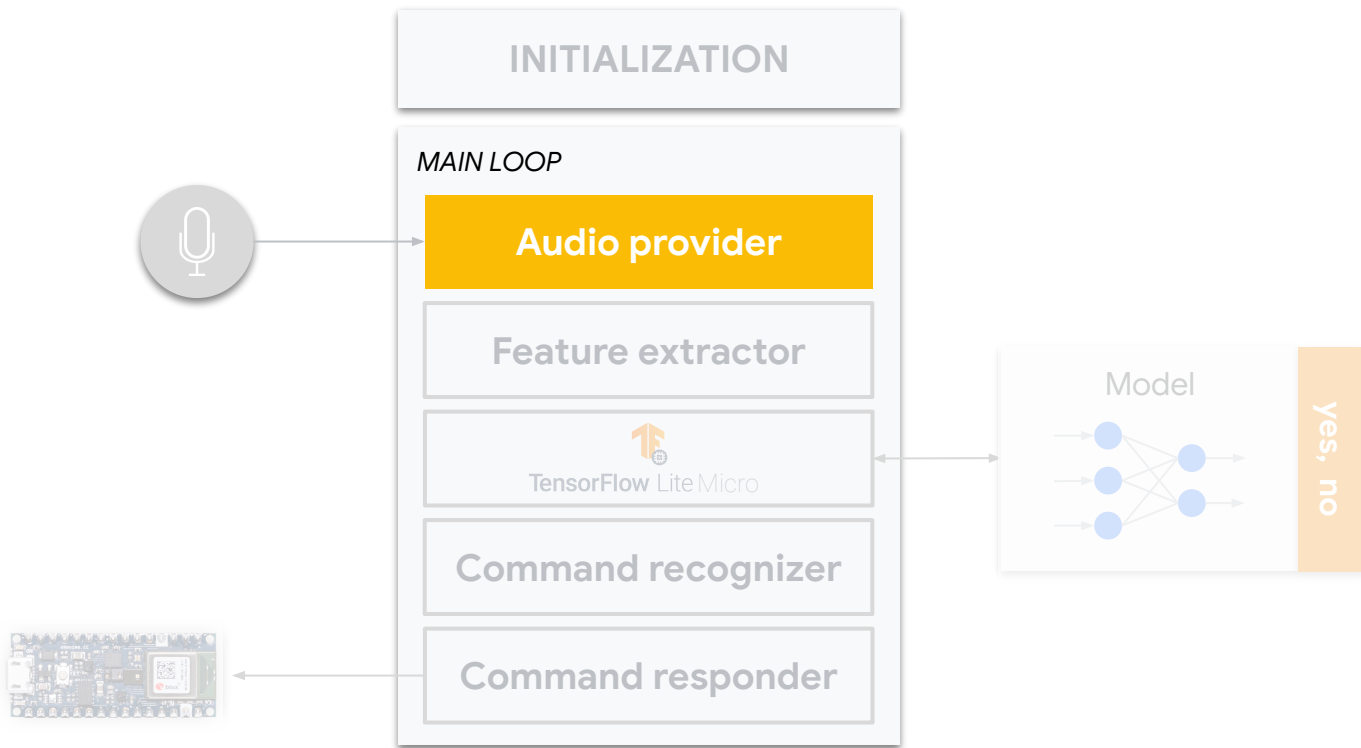
Device Microphone



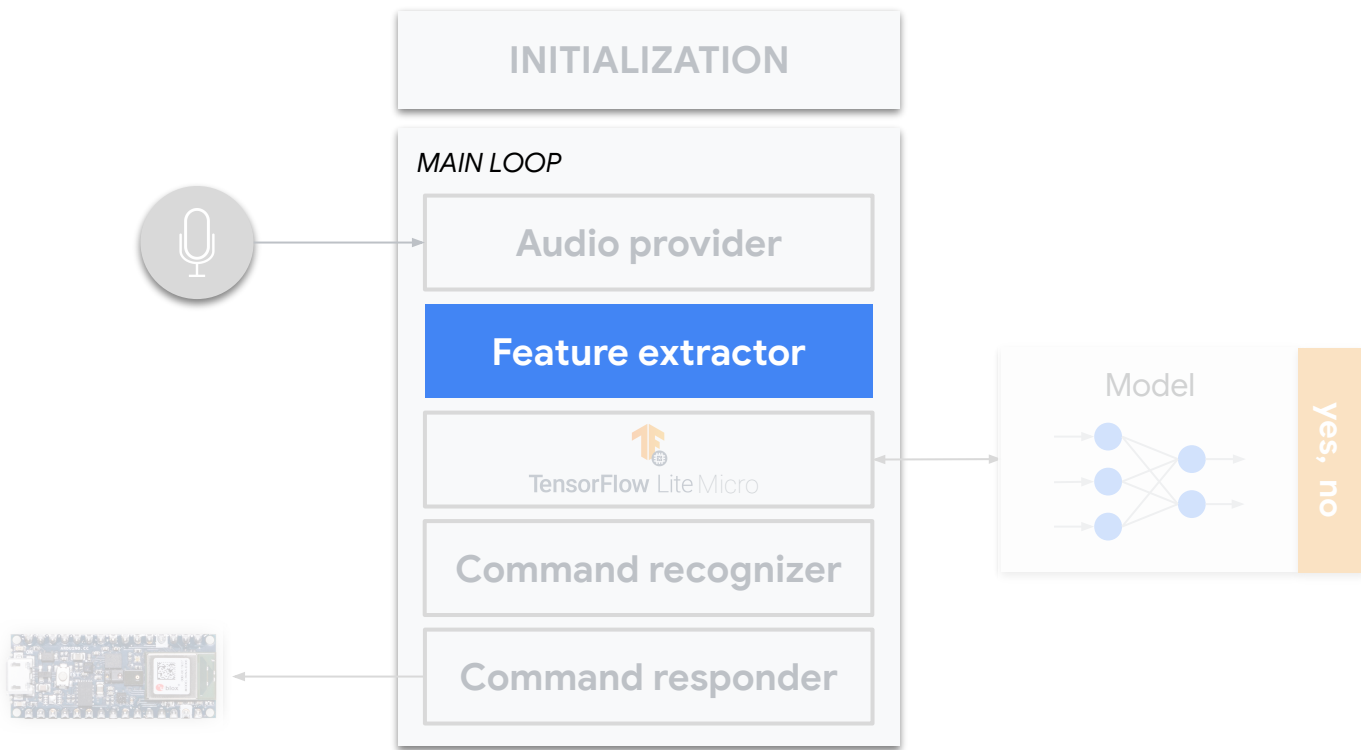


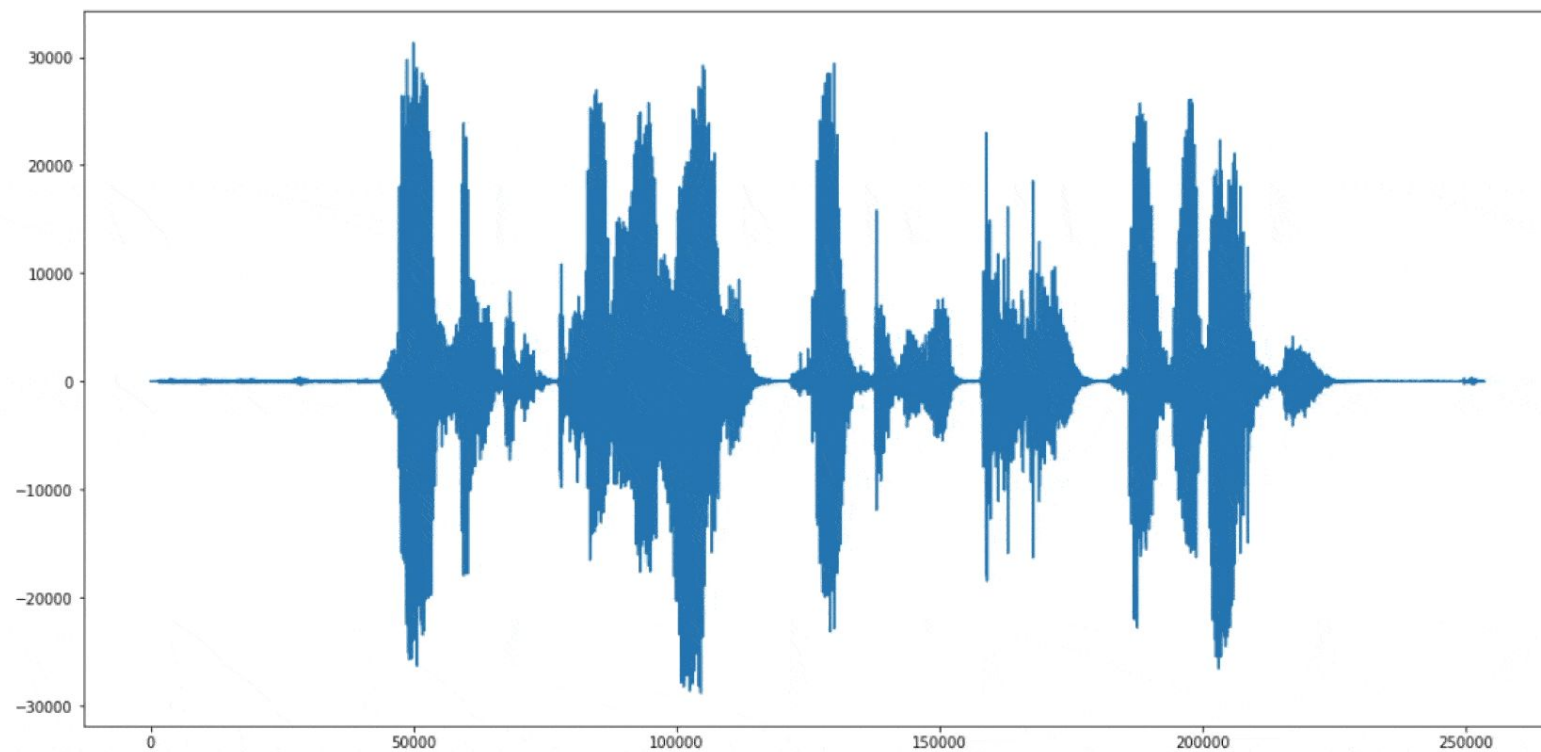
Microphone

Audio Provider

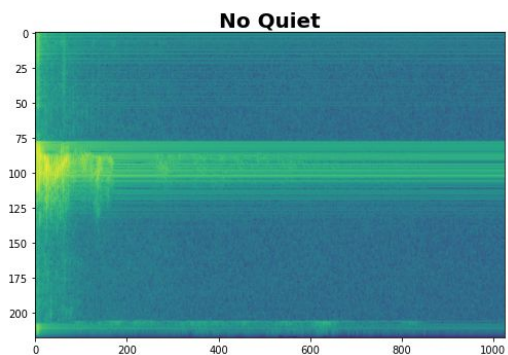
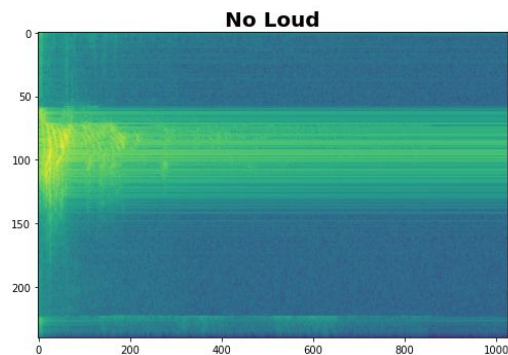
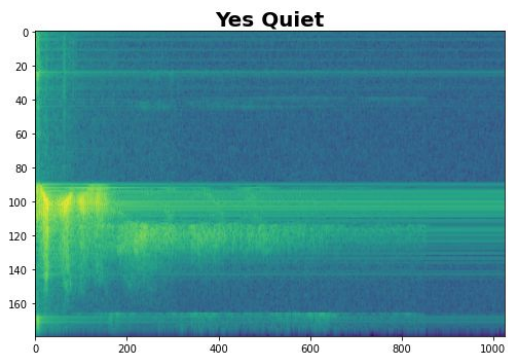
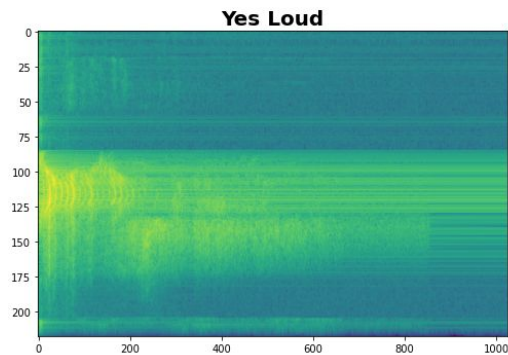


Feature Extractor

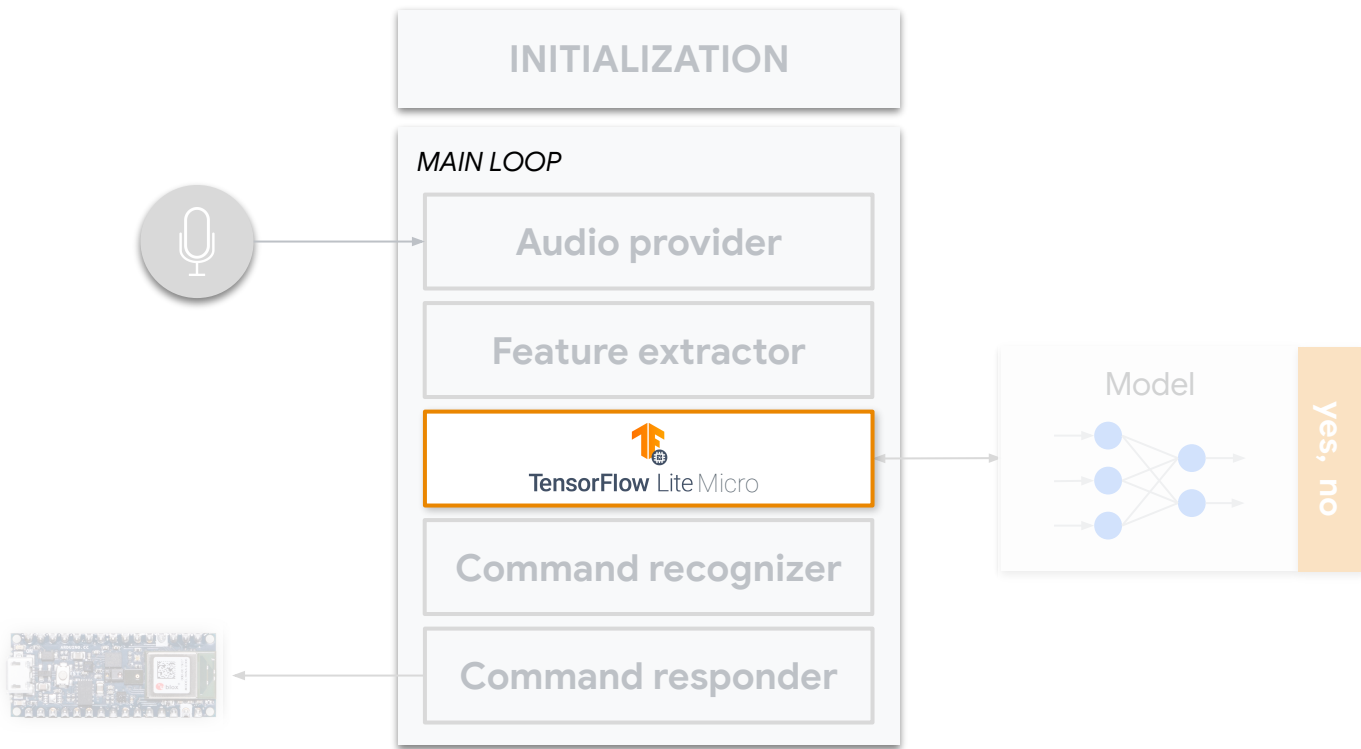




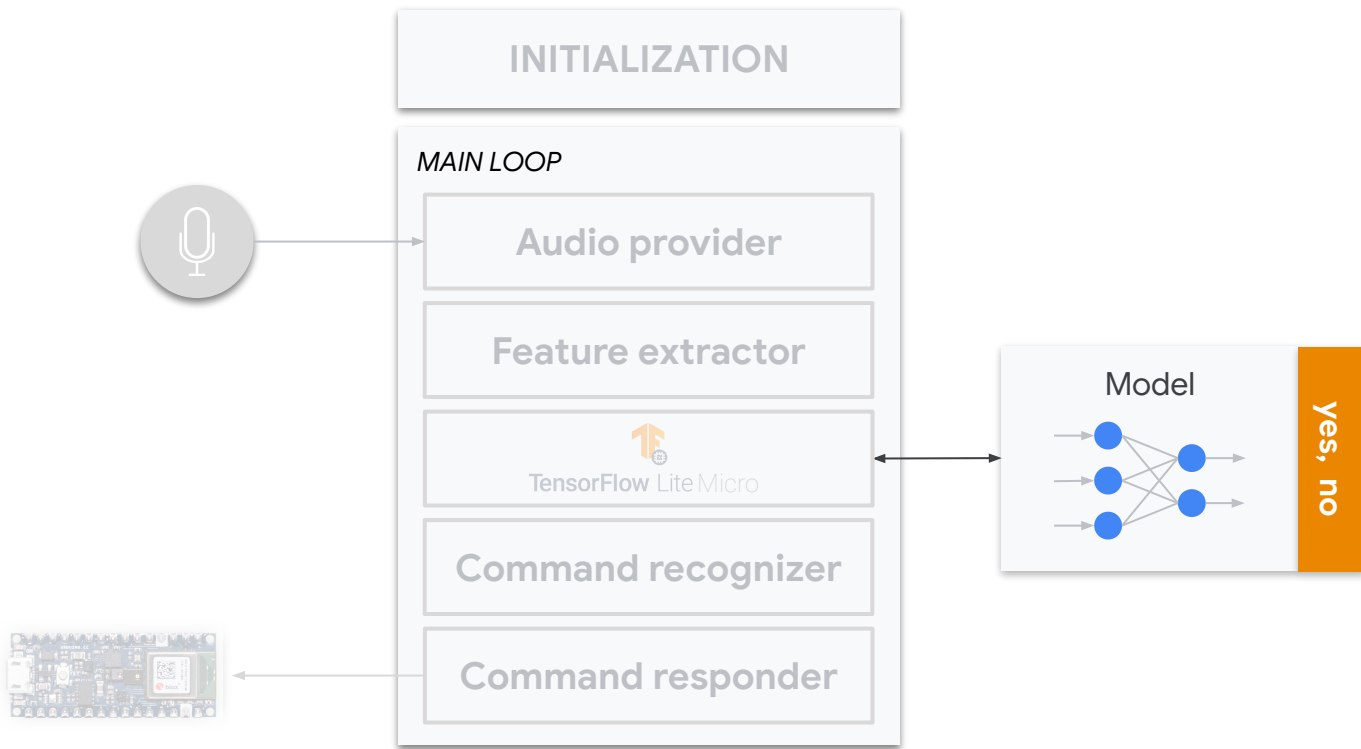
Data Preprocessing: Spectrograms



TFLite Micro Interpreter



Model

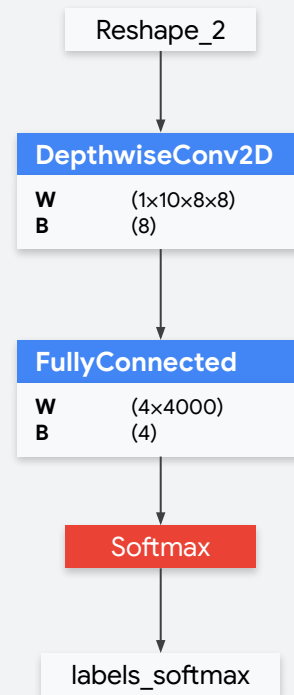


Model Operators

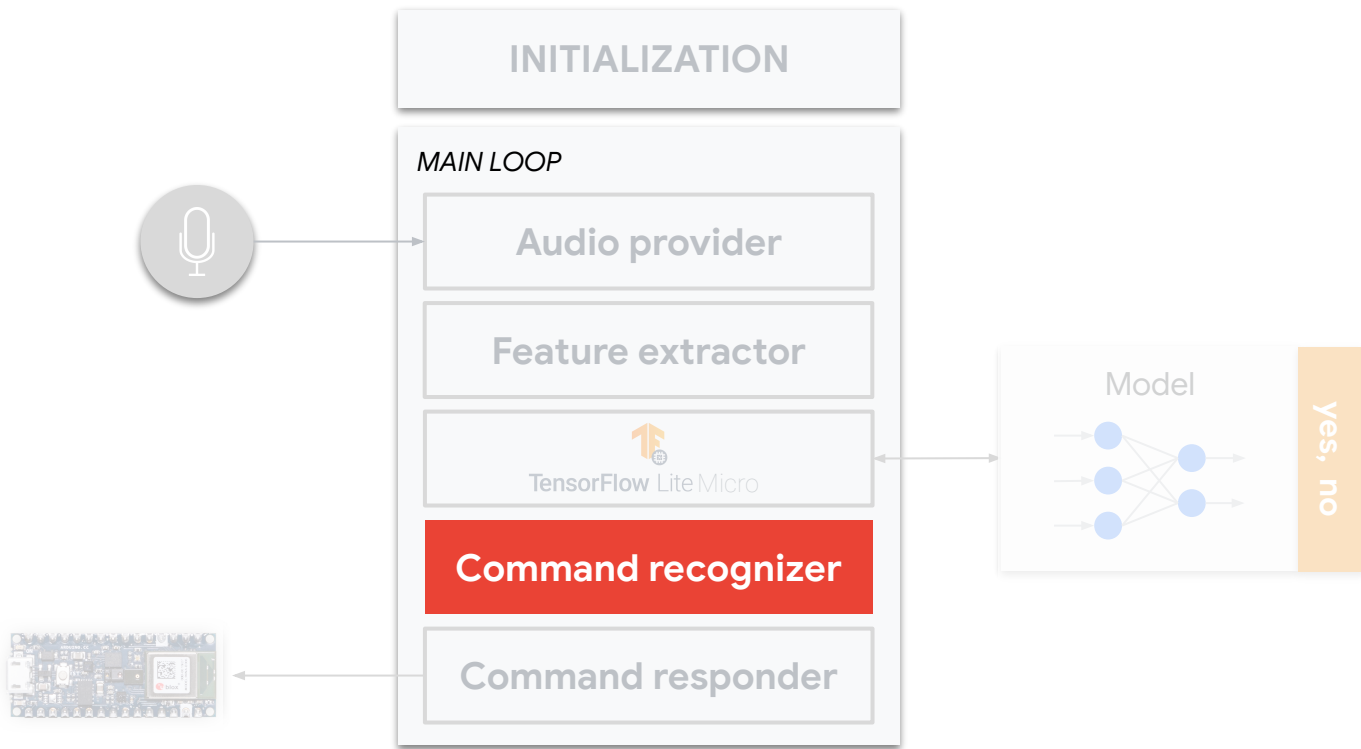
NETRON

<https://netron.app>

TinyConv Keyword Spotting Model



Command Recognizer





“Yes”



“No”

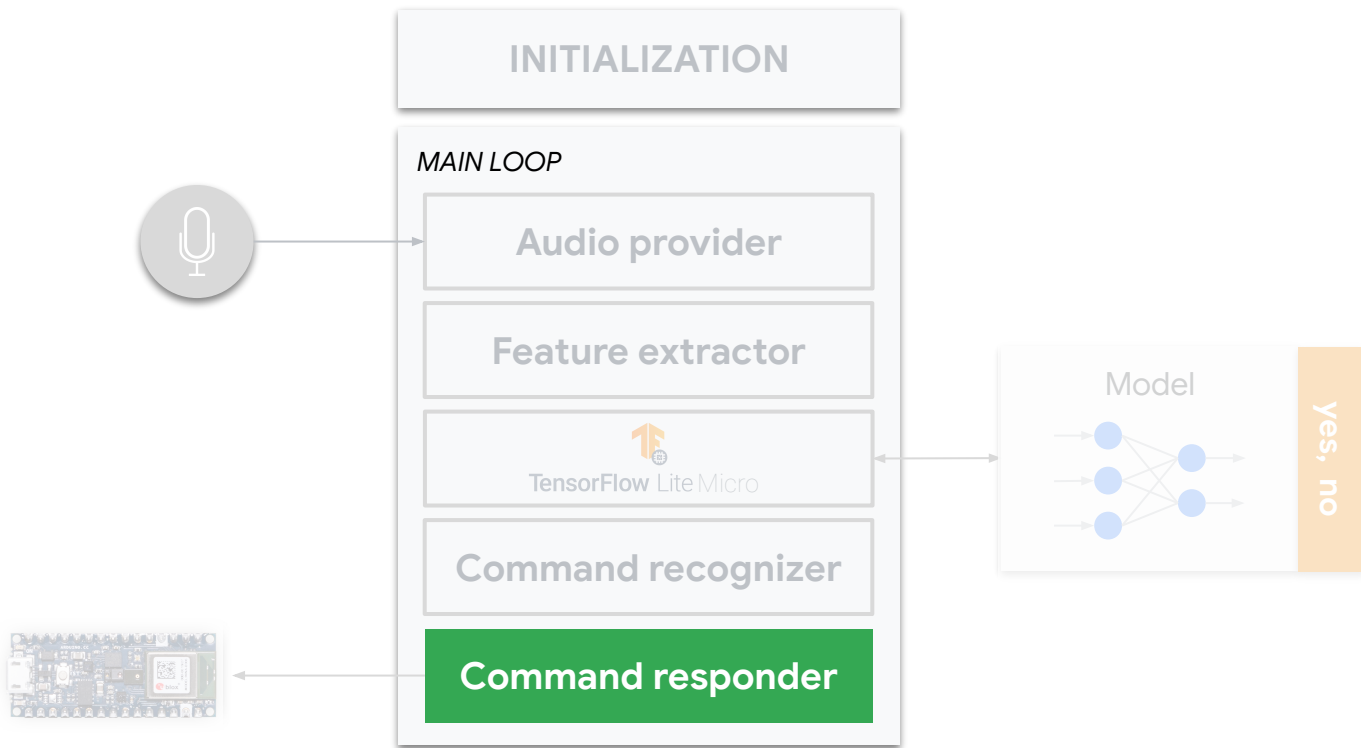


“Up”

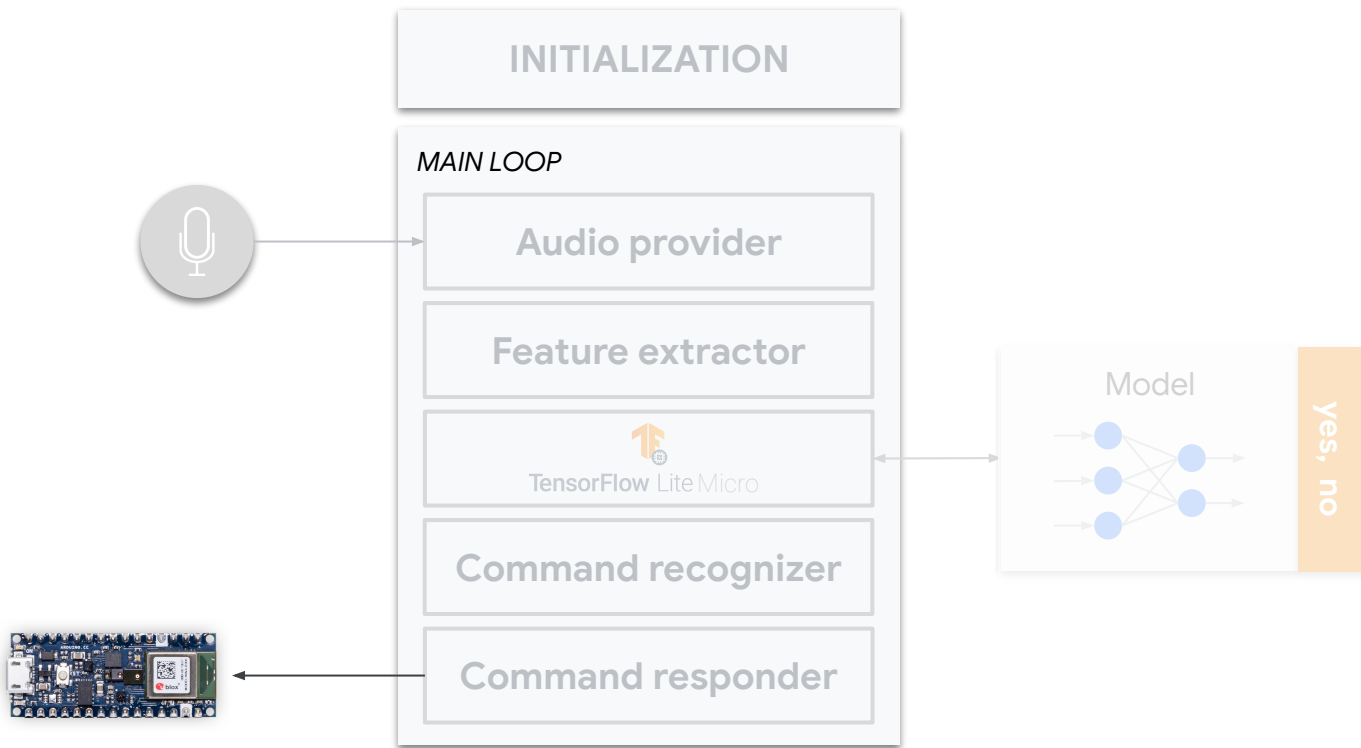


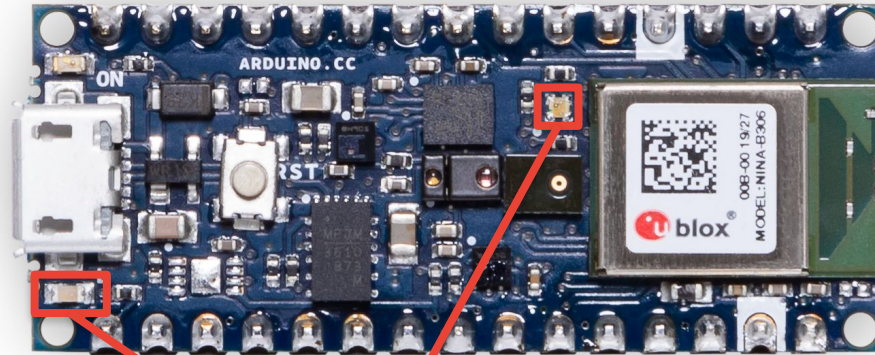
“Down”

Command Responder



Device Response

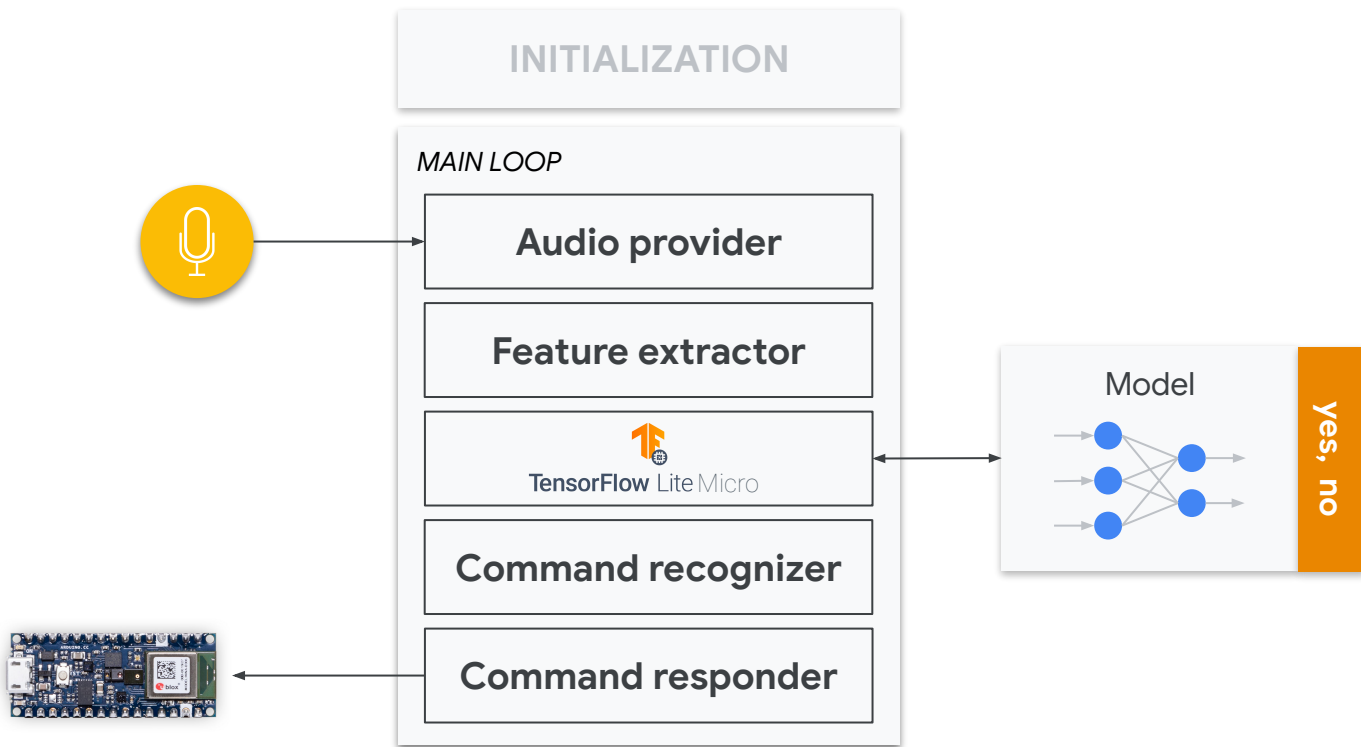




LEDs



Keyword Spotting Components

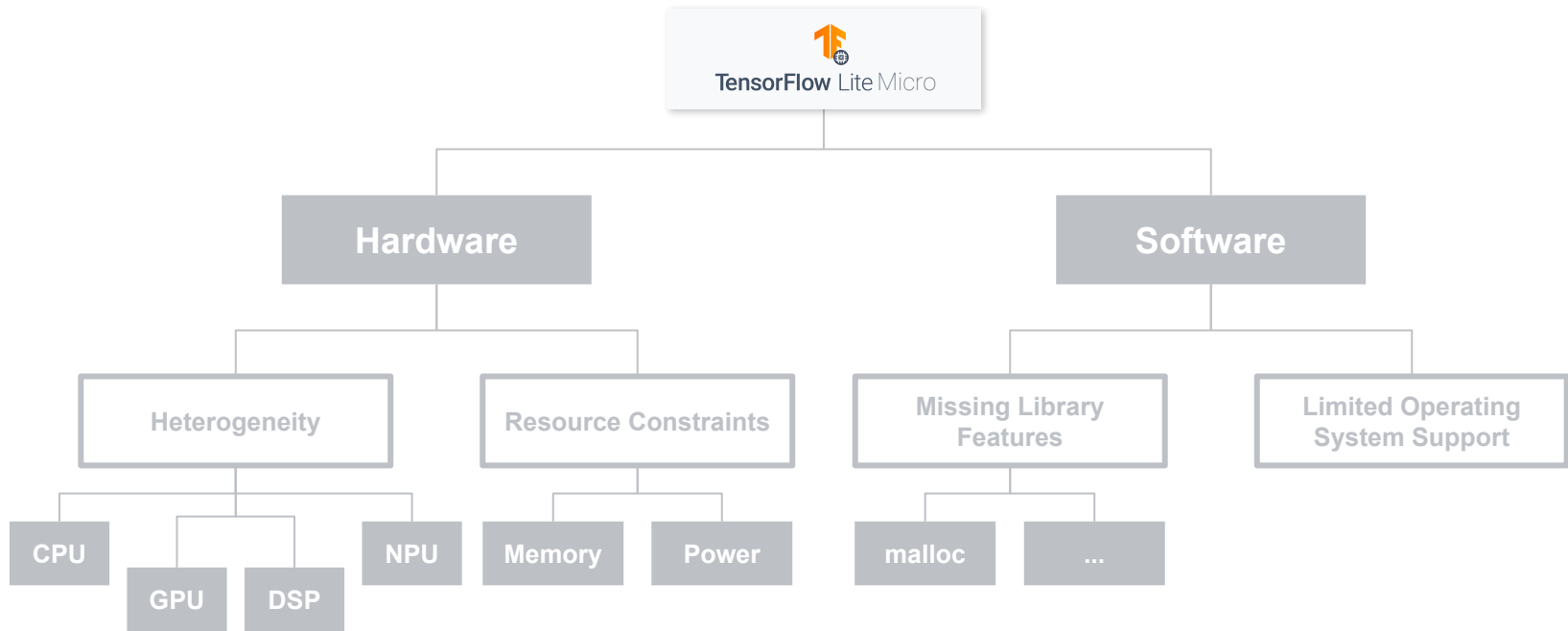


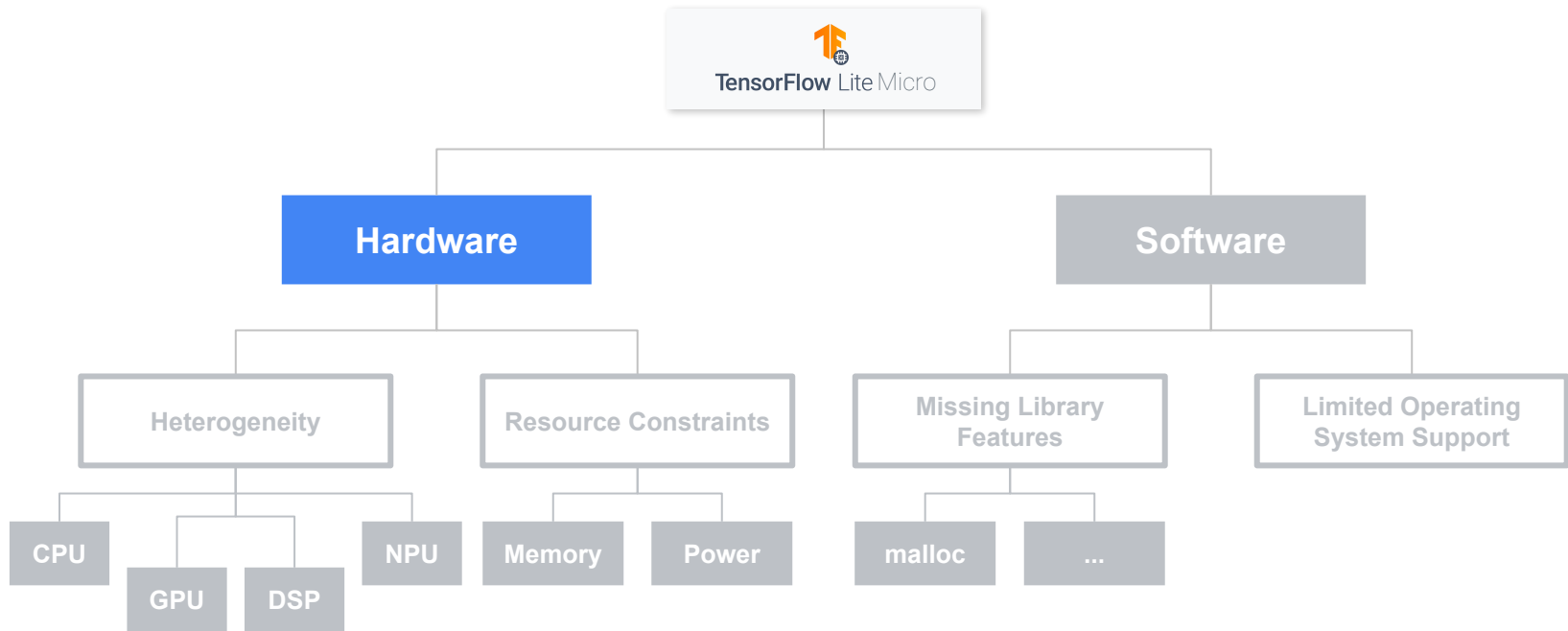
TensorFlow Lite Micro

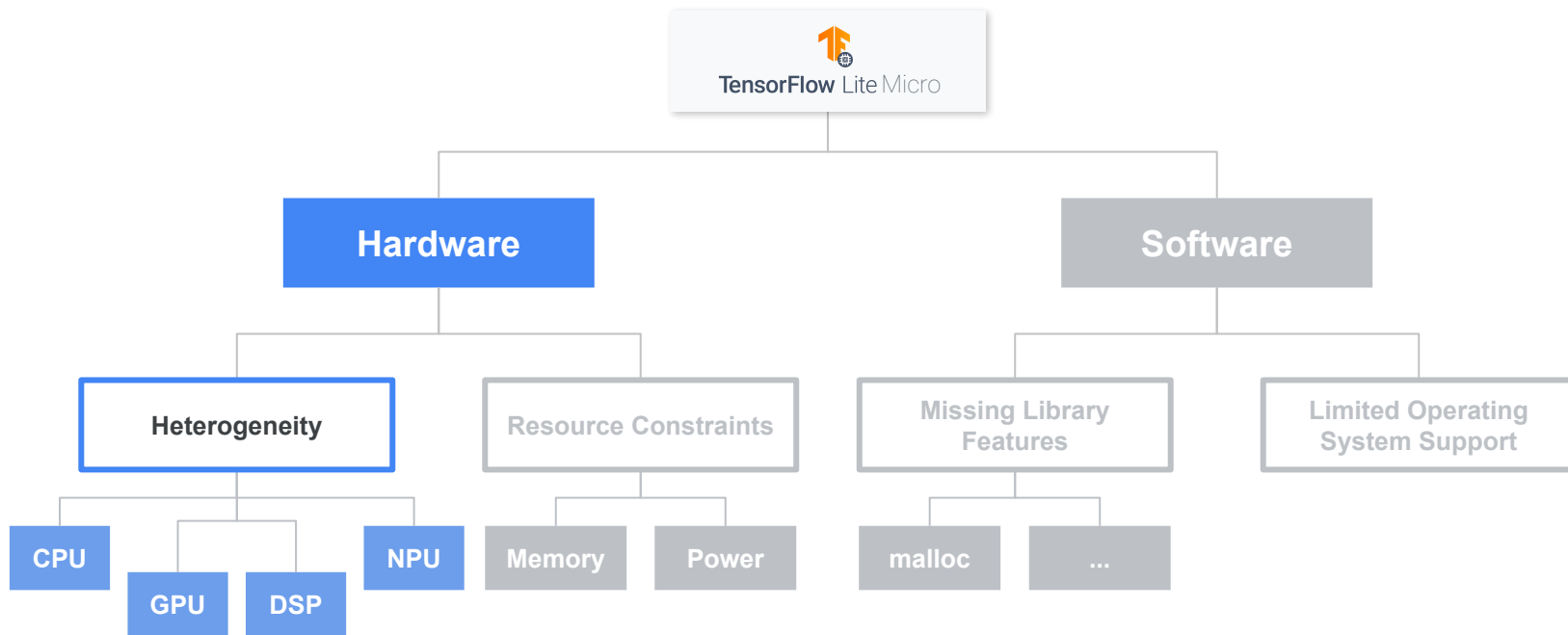
Embedded Machine Learning on TinyML Systems

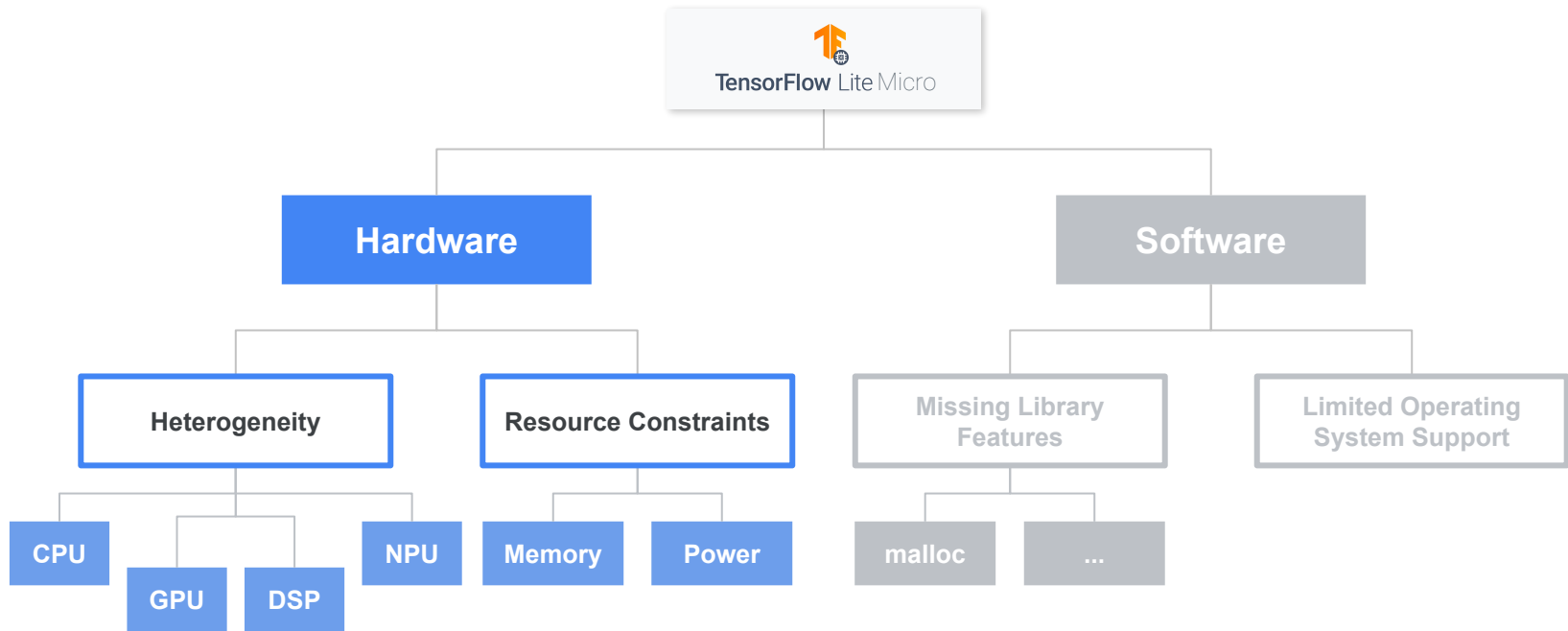










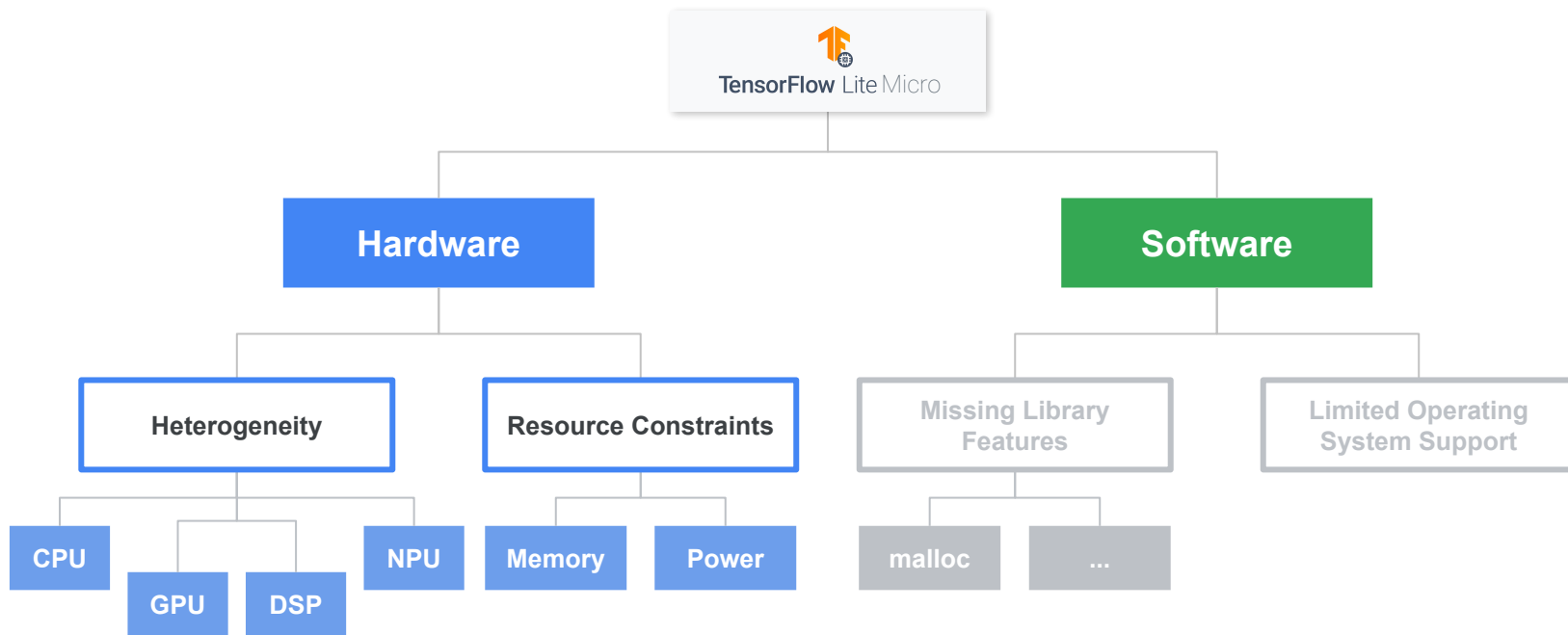


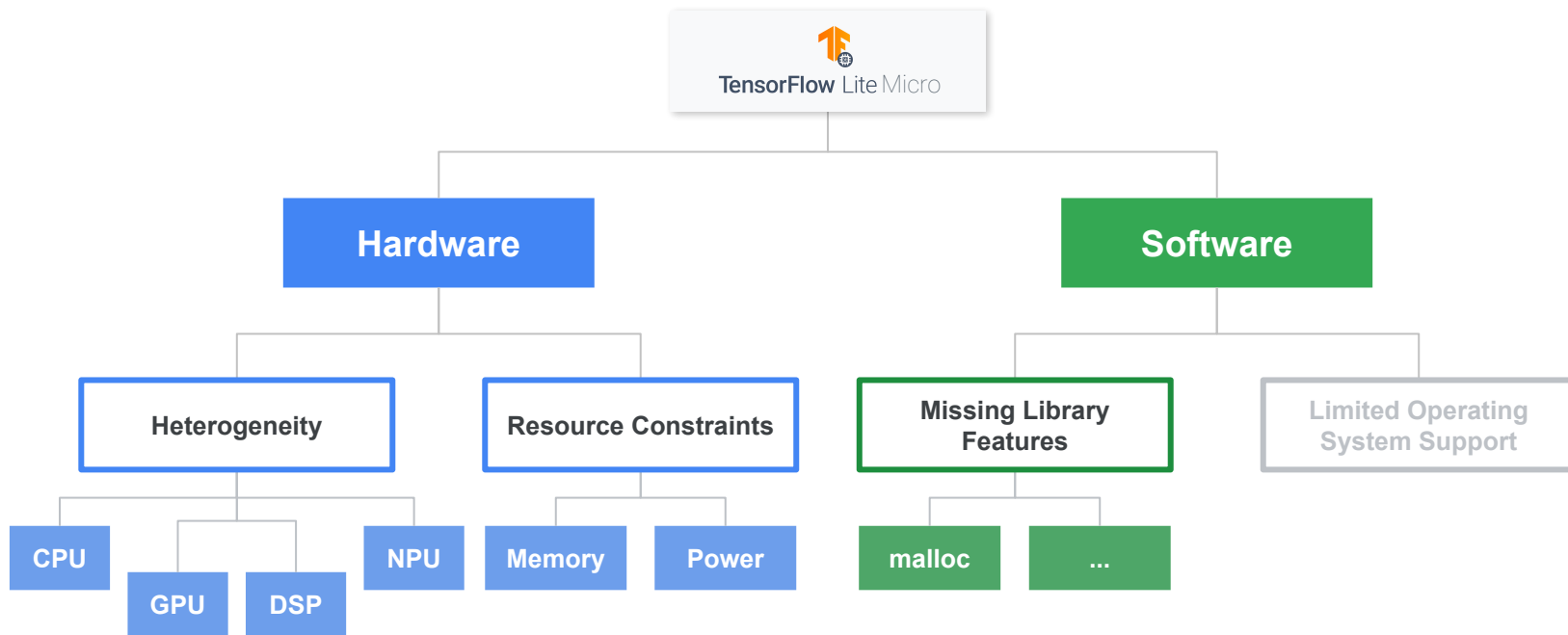
System	Chrom-ART Accelerator™	2-Mbyte dual-bank Flash memory
SMPS, LDO, USB and backup regulators POR/PDR/PVD/BOR	JPEG Codec Acceleration	RAM 1056KB incl. 64KB ITCM
Multi-power domains	Cache I/D 16+16 Kbytes	FMC/SRAM/NOR/NAND/SDRAM
Xtal oscillators 32 kHz + 4 ~48 MHz		Dual Quad-SPI
Internal RC oscillators 32 kHz + 4, 48 & 64 MHz		1024-byte + 4-Kbyte backup SRAM
3x PLL	<div>Arm® Cortex®-M7 480 MHz</div> <div>+</div> <div>Arm® Cortex®-M4 240 MHz</div>	Connectivity
Clock control		TFT LCD controller
RTC/AWU		MPI-DSI
1x SysTick timer		HDMI-CEC
2x watchdogs (independent and window)		6x SPI, 3x I²S, 4x I²C
82/114/140/168 I/Os		Camera interface
Cyclic redundancy check (CRC)		Ethernet MAC 10/100 with IEEE 1588
Unique ID		MDIO slave
		2x FDCAN (Flexible Data rate)
		1x USB 2.0 OTG FS/HS
Control	Floating point unit (DP-FPU)	1x USB 2.0 OTG FS
2x 16-bit motor control PWM synchronized AC timer	Nested vector interrupt controller (NVIC)	2x SDMMC
10x 16-bit timers 2x 32-bit timers	JTAG/SW debug/ETM	4x USART + 4 UART
5x Low-power timer	Memory Protection Unit (MPU)	LIN, smartcard, IrDA, modem control
16-bit High res. timer	ROP, PC-ROP anti-tamper	1x Low-power UART
		4x SAI (Serial audio interface)
		SPDIF input x4
		DFSDM (8 inputs/4 filters)
		SWP (Single Wire Protocol)
		Analog
	AXI and Multi-AHB bus matrix	2x 12-bit, 2-channel DACs
	4x DMA	3 x 16-bit ADC (up to 3.6 Msp/s)
	True random number generator (RNG)	20 channels/up to 2 MSPS
		Temperature sensor
		2x COMP
		2x OpAmp

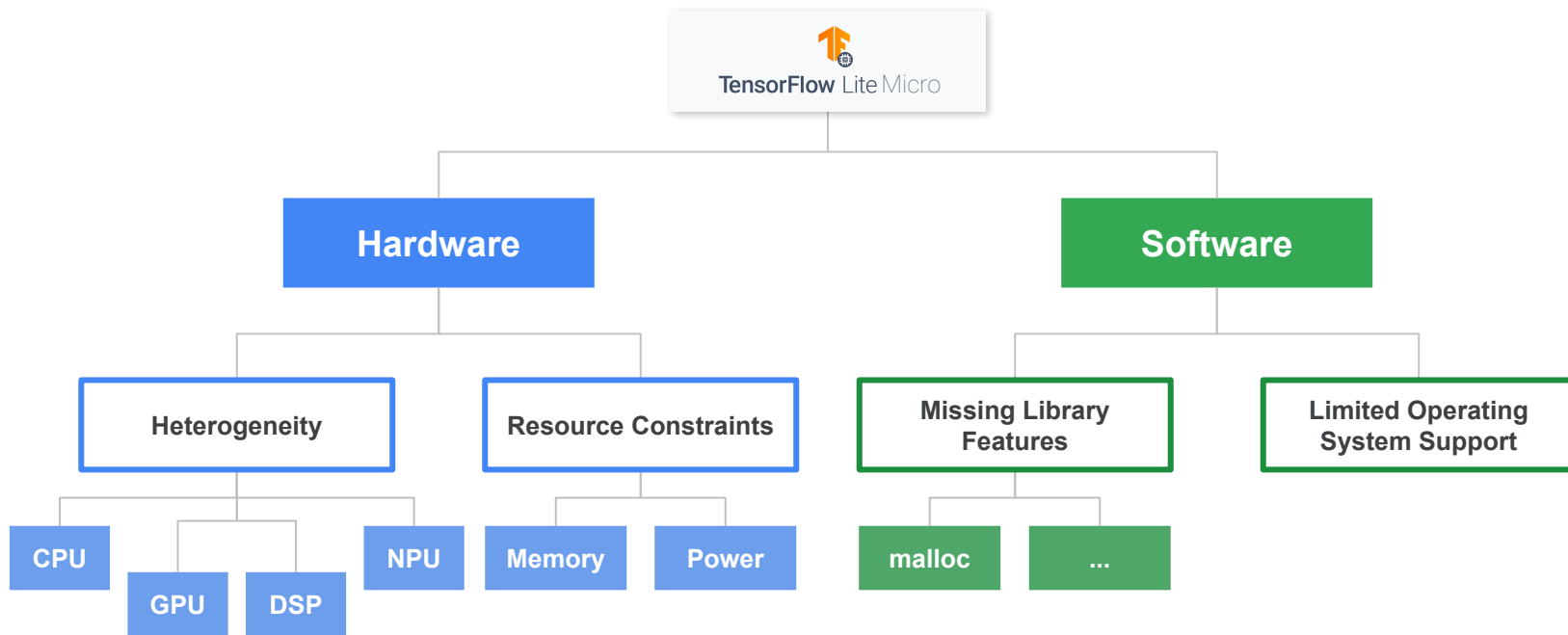
STM32H747xI/G devices are based on the high-performance **Arm® Cortex®-M7** and **Cortex®-M4 32-bit RISC** cores. The Cortex®-M7 core operates at up to 480 MHz and the Cortex®-M4 core at up to 240 MHz. Both cores feature a **floating point unit (FPU)** which supports Arm® single- and double-precision (Cortex®-M7 core) operations and conversions (IEEE 754 compliant), including a full set of **DSP instructions** and a memory protection unit (MPU) to enhance application security.

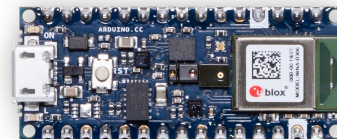
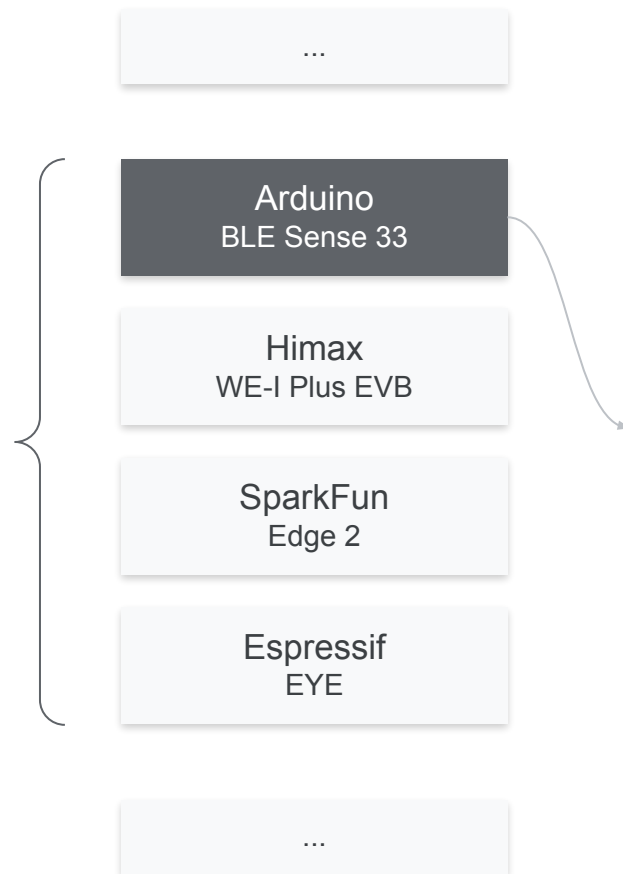
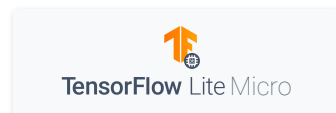
STM32H747xI/G devices incorporate high-speed embedded memories with a dual-bank **flash memory of up to 2 Mbytes**, up to **1 Mbyte of RAM** (including 192 Kbytes of **TCM RAM**, up to 864 Kbytes of user **SRAM** and 4 Kbytes of backup SRAM), as well as an extensive range of enhanced **I/Os and peripherals** connected to APB buses, AHB buses, 2x32-bit multi-AHB bus matrix and a multi layer AXI interconnect supporting internal and external memory access.

All the devices offer three **ADCs**, two **DACs**, two ultra-low power **comparators**, a low-power RTC, a high-resolution timer, 12 general-purpose 16-bit **timers**, two PWM timers for motor control, five low-power timers, a true random number generator (RNG). The devices support four digital filters for external sigma-delta modulators (DFSDM). They also feature standard and advanced communication interfaces.

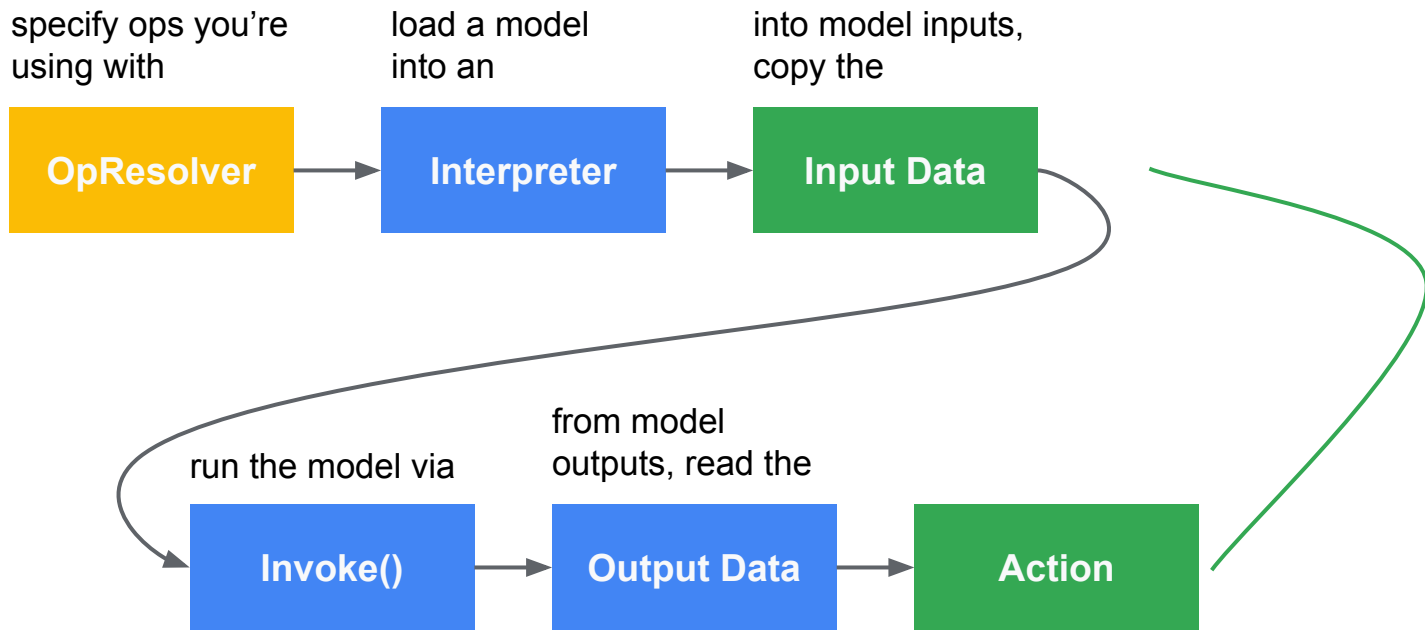








How do you use TFL Micro?

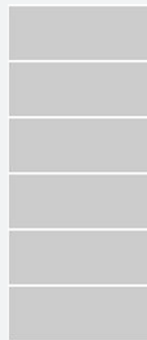
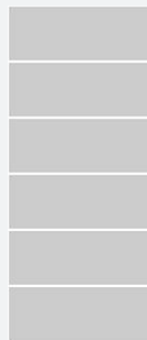


TFLite Micro: Interpreter



TFLite Micro Design

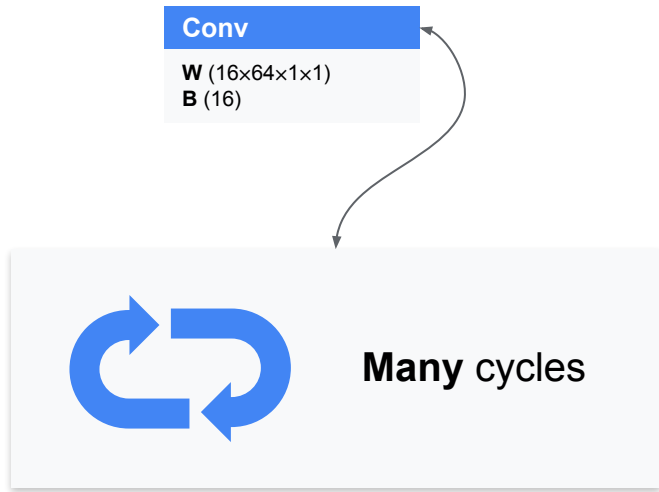
- TFLite Micro uses an **interpreter** design
- Store the model as data and loop through its ops at **runtime**



instruction
ops

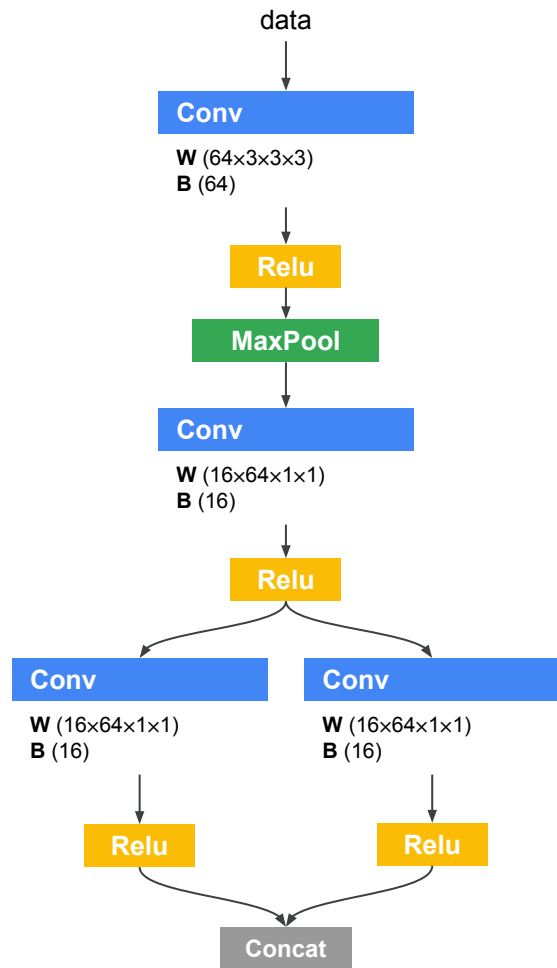


dispatch
loop



ML is Different

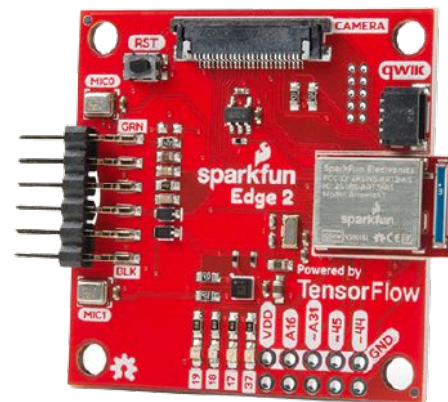
- Each layer like a **Conv** or **softmax** can take tens of thousands or even millions of cycles to complete execution



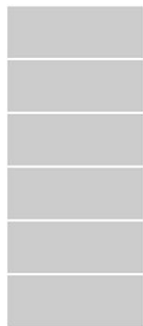
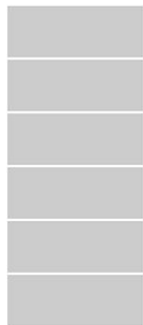
ML is Different

- Parsing overhead is **relatively small** for the TFMicro interpreter when we consider the **overall network graph**

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%



Sparkfun Edge 2
(Apollo 3 **Cortex-M4**)



instruction
ops



dispatch
loop

Interpreter Advantages

- Change the model
without recompiling
the code



instruction
ops



dispatch
loop

Interpreter Advantages

- Change the model **without recompiling** the code
- **Same operator code** can be used across multiple **different models** in the system

Arduino
BLE Sense 33

Himax
WE-I Plus EVB

Espressif
EYE

SparkFun
Edge 2

Interpreter Advantages

- Same **portable** model serialization format can be used **across a lots of systems.**

TFLite Micro

Interpreter Execution

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


TFLite Micro: Model Format

The FlatBuffer File Format




```
// Map the model into a usable data structure. This doesn't involve any
// copying or parsing, it's a very lightweight operation.

model = tflite::GetModel(g_model);
if (model->version() != TFLIGHT_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
        "Model provided is schema version %d not equal "
        "to supported version %d.",
        model->version(), TFLITE_SCHEMA_VERSION);

    return;
}
```



```
// Map the model into a usable data structure. This doesn't involve any
// copying or parsing, it's a very lightweight operation.

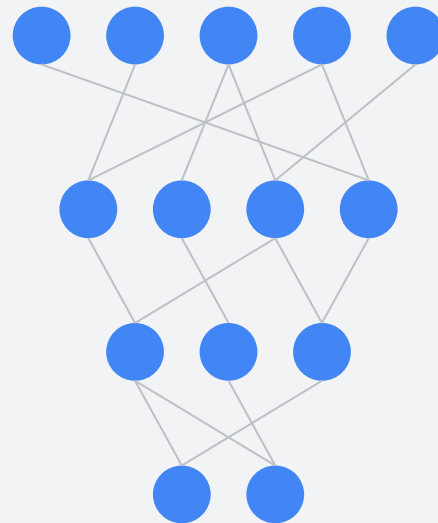
model = tflite::GetModel(g_model);
if (model->version() != TFLIGHT_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
                        "Model provided is schema version %d not equal "
                        "to supported version %d.",
                        model->version(), TFLITE_SCHEMA_VERSION);

    return;
}
```

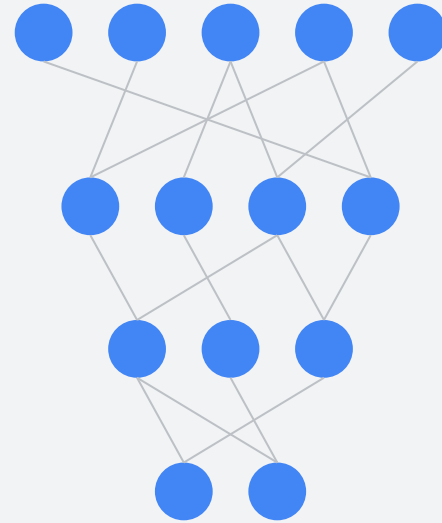


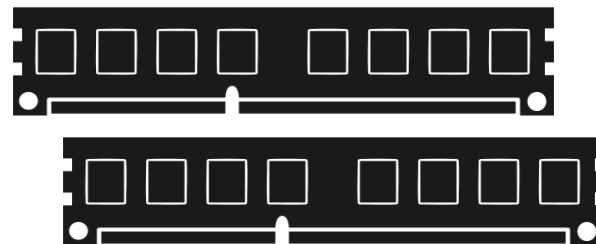
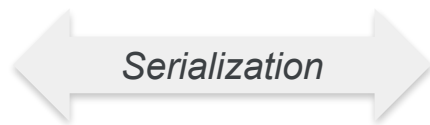
```
// Map the model into a usable data structure. This doesn't involve
any
// copying or parsing, it's a very lightweight operation.

model = tfLite::GetModel(g_model);
if (model->version() != TFLIGHT_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
        "Model provided is schema version %d not equal
"
        "to supported version %d.",
        model->version(), TFLITE_SCHEMA_VERSION);
    return;
}
```



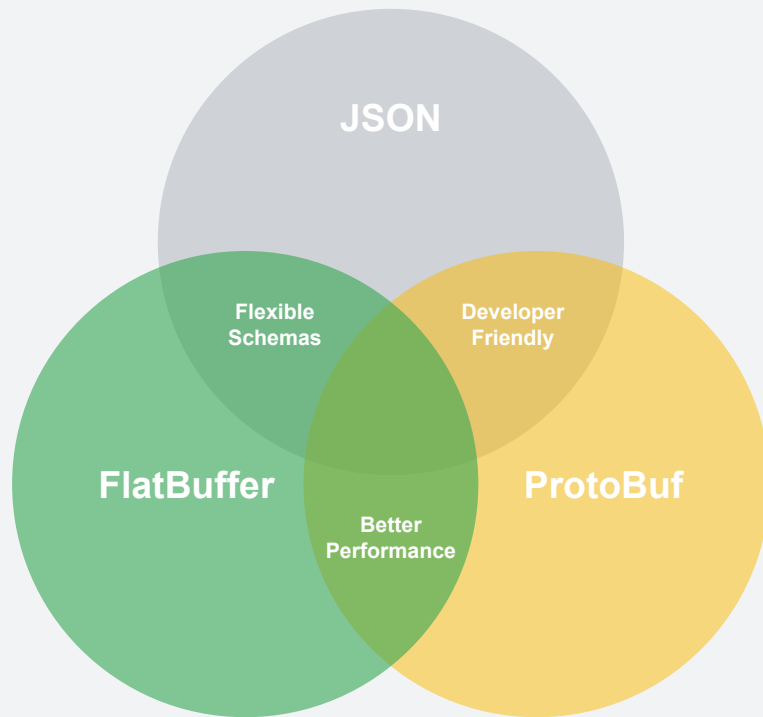
How is `g_model` stored?

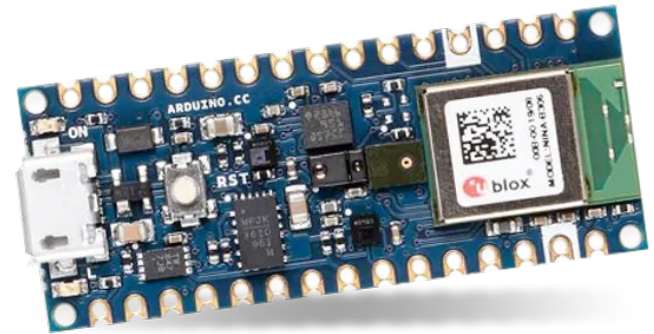
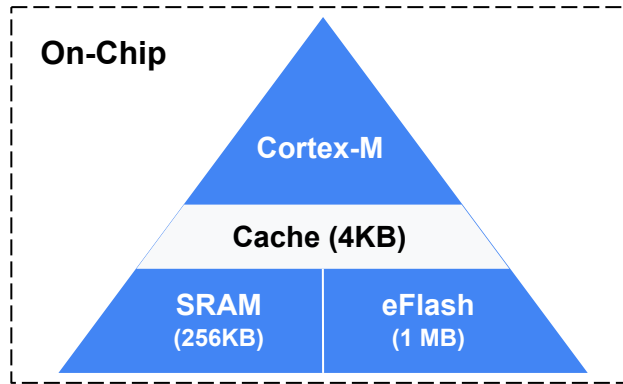


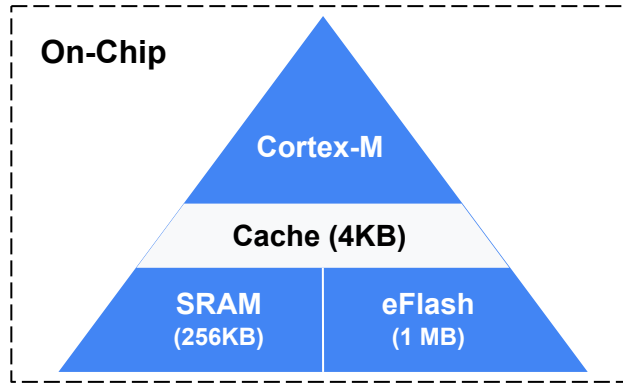


Serialization Libraries

- JSON
- ProtoBuf
- FlatBuffer

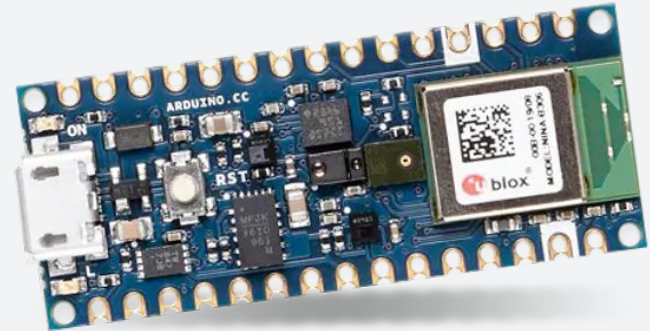






Hardware & Software Limitations

- Limited **OS support**
- Limited **compute**
- Limited **memory**



What is `g_model`?

- **Array of bytes**, and acts as the equivalent of a file on disk
- Holds **all** of the **information about the model**, its **operators**, their **connections**, and the trained **weights**

```
6 // Automatically created from a TensorFlow Lite flatbuffer using the command:
17 // xxd -i model.tflite > model.cc
18
19 // This is a standard TensorFlow Lite model file that has been converted into a
20 // C data array, so it can be easily compiled into a binary for devices that
21 // don't have a file system.
22
23 // See train/README.md for a full description of the creation process.
24
25 #include "model.h"
26
27 // Keep model aligned to 8 bytes to guarantee aligned 64-bit accesses.
28 alignas(8) const unsigned char g_model[] = {
29     0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x12, 0x00,
30     0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14, 0x00,
31     0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
32     0x60, 0x09, 0x00, 0x00, 0xa8, 0x02, 0x00, 0x00, 0x90, 0x02, 0x00, 0x00,
33     0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
34     0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x04, 0x00, 0x08, 0x00,
35     0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x0b, 0x00, 0x00, 0x00,
36     0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f, 0x72, 0x75, 0x6e, 0x74,
37     0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00,
38     0x0c, 0x00, 0x00, 0x00, 0x48, 0x02, 0x00, 0x00, 0x34, 0x02, 0x00, 0x00,
39     0x0c, 0x02, 0x00, 0x00, 0xfc, 0x00, 0x00, 0x00, 0xac, 0x00, 0x00, 0x00,
40     0x8c, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x34, 0x00, 0x00, 0x00,
41     0x2c, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
42     0x04, 0x00, 0x00, 0x00, 0xfe, 0xfd, 0xff, 0x04, 0x00, 0x00, 0x00,
43     0x05, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e, 0x30, 0x00, 0x00, 0x00,
44     0x7c, 0xfd, 0xff, 0xff, 0x80, 0xfd, 0xff, 0xff, 0x84, 0xfd, 0xff, 0xff,
45     ...,
46     0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x06, 0x00, 0x05, 0x00,
47     0x06, 0x00, 0x00, 0x00, 0x00, 0x72, 0x0a, 0x00, 0x0c, 0x00, 0x07, 0x00,
48     0x00, 0x00, 0x08, 0x00, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09,
49     0x04, 0x00, 0x00, 0x00};
50 const int g_model_len = 2512;
```


FlatBuffers

- Does **not require copies** to be made before using the data inside the model



FlatBuffers

- Does **not require copies** to be made before using the data inside the model
- The **format** is formally specified as a **schema file**



FlatBuffers

- Does **not require copies** to be made before using the data inside the model
- The **format** is formally specified as a **schema file**
- Schema file is used to automatically **generate code** to access the information in the **model byte array**



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, ...
...

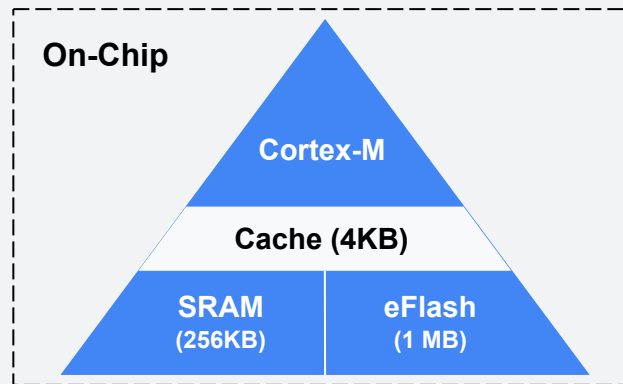
TFLite Micro: Memory Allocation

The Tensor Arena



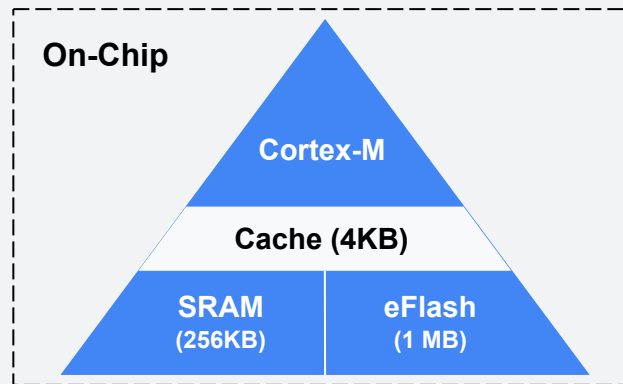
Why Care About **Memory**?

- Embedded systems typically have **only hundreds or tens of kilobytes** of RAM



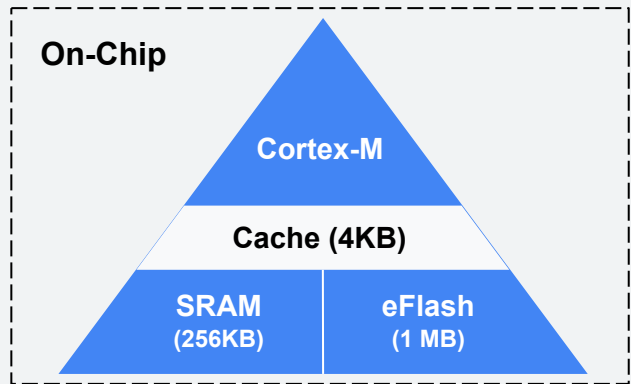
Why Care About **Memory**?

- Embedded systems typically have **only hundreds or tens of kilobytes** of RAM
- **Easy to hit memory limits** when building an end-to-end application



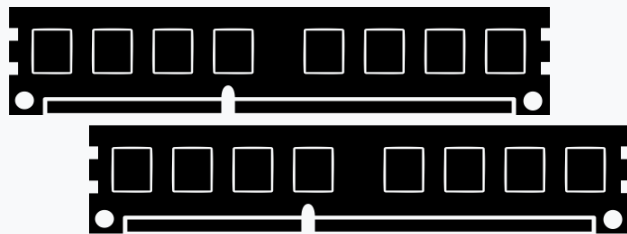
Why Care About **Memory**?

- Embedded systems typically have **only hundreds or tens of kilobytes** of RAM
- **Easy to hit memory limits** when building an end-to-end application
- So any framework that integrates with embedded products **must offer control over how memory usage**



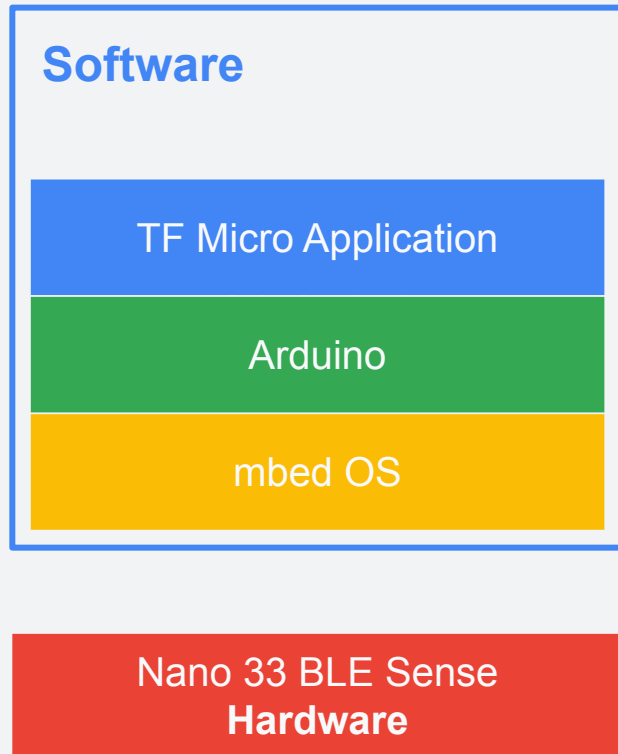
Long-Running Applications

- Products are **expected to run for months** or even years, which poses challenges for memory allocation
- Need to guarantee that memory allocation will not end up **fragmented** → **contiguous memory cannot be allocated** even if there's enough memory overall



Lack of OS Support

- In embedded systems, the standard C and C++ memory APIs (**malloc** and **new**) **rely on operating system support**
- Many devices have **no OS**, or have very **limited functionality**



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);
```

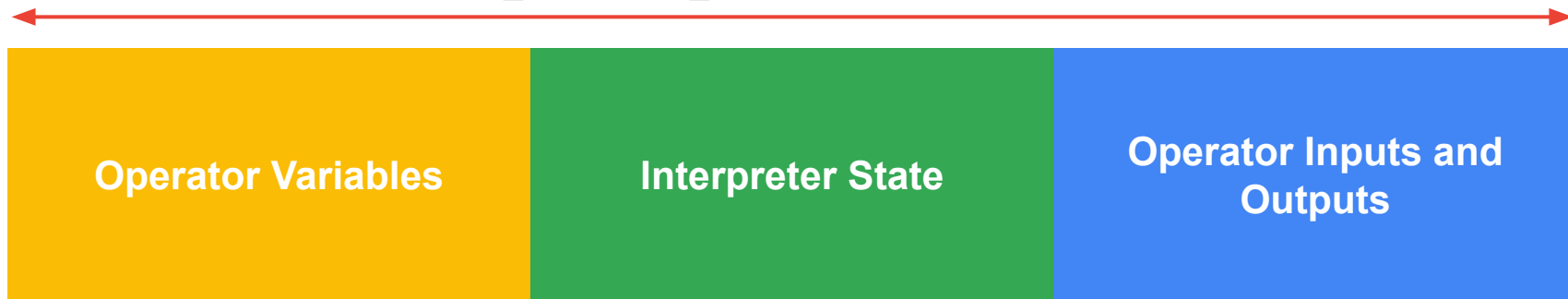

How **TFL Micro** solves these challenges

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]
```



Arena size?

- **Depends on what ops are in the model** (and the parameters of those operations)

```
constexpr int kTensorArenaSize = 2000;  
uint8_t tensor_arena[kTensorArenaSize];  
  
...  
  
static tflite::MicroInterpreter static_interpreter(model,  
    resolver, tensor_arena, kTensorArenaSize, error_reporting);
```


Arena size?

- **Depends on what ops are in the model** (and the parameters of those operations)
- Size of operator inputs and outputs is platform independent, **but different devices can have different operator implementations**

```
constexpr int kTensorArenaSize = 2000;  
uint8_t tensor_arena[kTensorArenaSize];  
  
...  
  
static tflite::MicroInterpreter static_interpreter(model,  
    resolver, tensor_arena, kTensorArenaSize, error_reporting);
```


Arena size?

- **Depends on what ops are in the model** (and the parameters of those operations)
- Size of operator inputs and outputs is platform independent, **but different devices can have different operator implementations**
- → **hard to forecast exact size** of arena needed

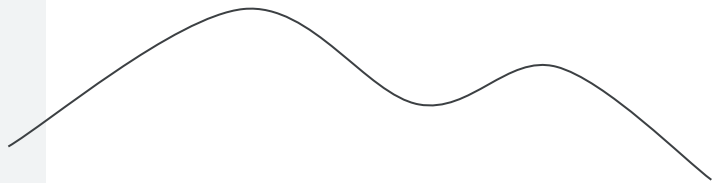
```
constexpr int kTensorArenaSize = 2000;
uint8_t tensor_arena[kTensorArenaSize];

...

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


Solution

- **Create as large an arena as you can** and run your program on-device
- Use the `arena_used_bytes()` function to get the actual size used.
- **Resize the arena to that length** and rebuild
- Best to **do this on your deployment platform**, since different op implementations may need varying scratch buffer sizes



```
constexpr int kTensorArenaSize = 6000;
uint8_t tensor_arena[kTensorArenaSize];

...

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

* Call [`MicroInterpreter::arena_used_bytes\(\)`](#) to get the actual memory size used.

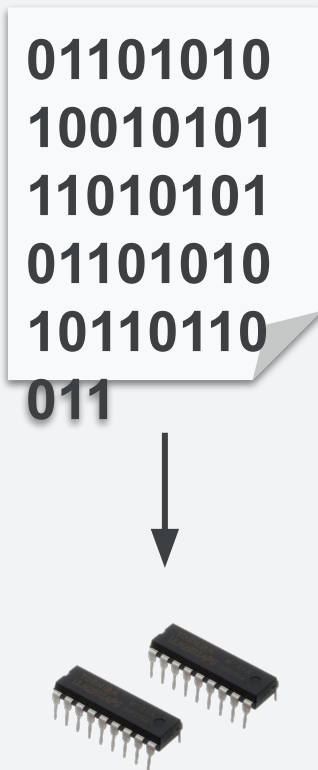
TFLite Micro: NN Operations

The OpsResolver



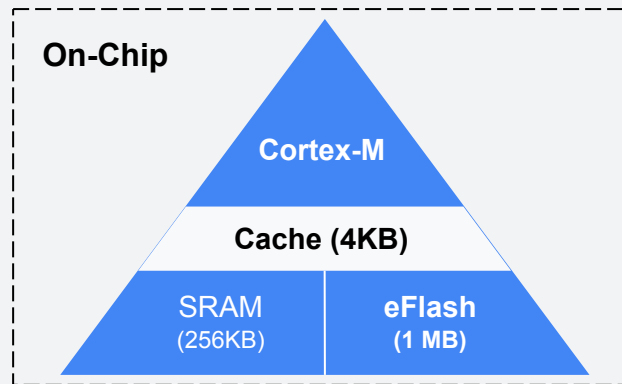
Why Care About Binary Size?

- **Executable code** used by a framework takes up space in Flash



Why Care About Binary Size?

- Executable code used by a framework takes up space in Flash
- **Flash is a limited resource** on embedded devices and often just tens of kilobytes available

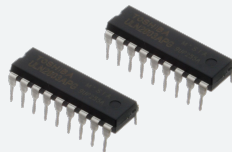


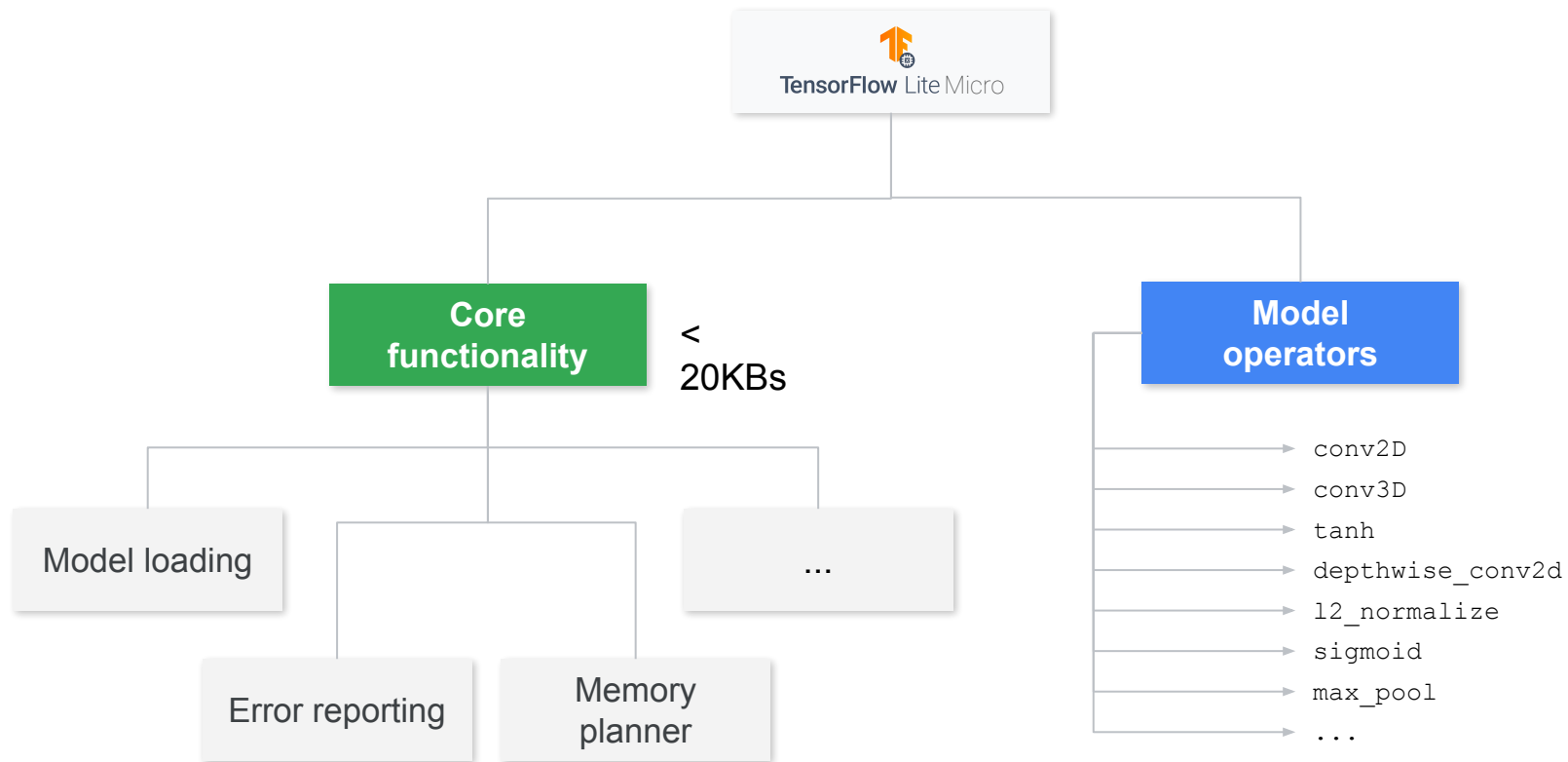
Why Care About Binary Size?

- Executable code used by a framework takes up space in Flash
- Flash is a limited resource on embedded devices and often just tens of kilobytes available
- If compiled **code is too large**, it **won't be usable** by applications.

```
01101010
10010101
11010101
01101010
10110110
```

011





Optimizing Operator Usage in TFL Micro

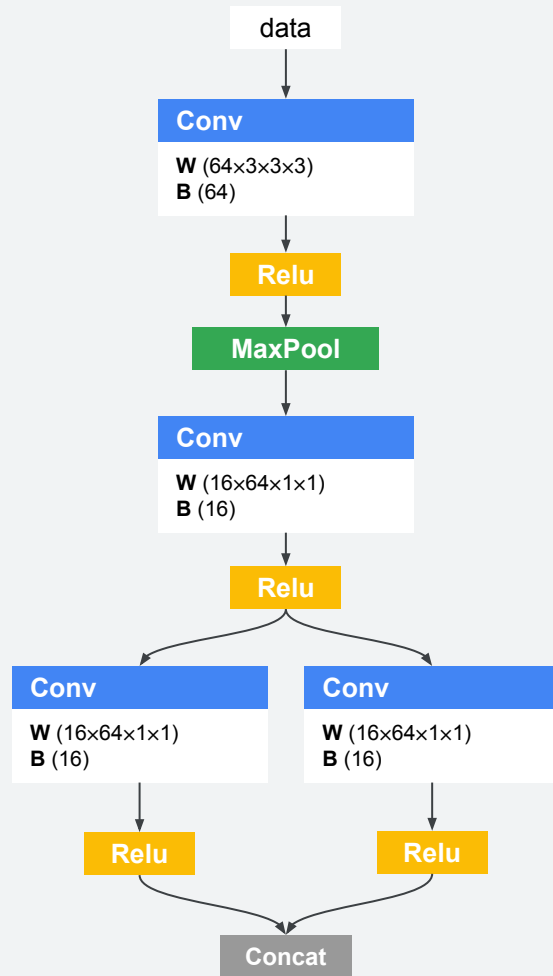
- There are **many operators in TensorFlow** (~1400 and growing)



TensorFlow

Optimizing Operator Usage in TFL Micro

- There are many operators in TensorFlow (~1400 and growing)
- **Not all operators are used** or even needed to perform inference



Optimizing Operator Usage in TFL Micro

- There are **many operators** in **TensorFlow** (~1400 and growing)
- **Not all operators are used** or even needed to perform inference
- Bring in or **load only those that are important** to conserve memory usage

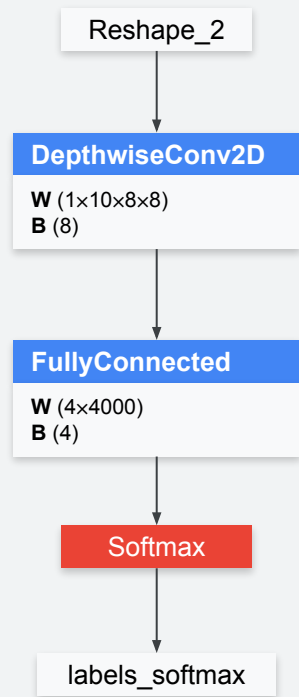


How to Reduce the Size Taken by Ops?

Allow developers to specify which ops they want to be included in the binary

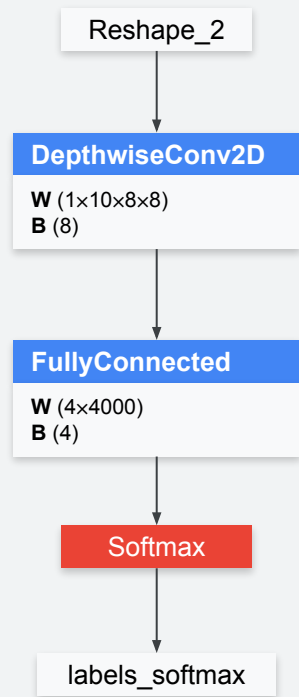
```
tflite::MicroMutableOpResolver<4>  
op_resolver(error_reporter);  
if (op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {  
    return;  
}
```


TinyConv Keyword Spotting Model



Hello!

TinyConv Keyword Spotting Model



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

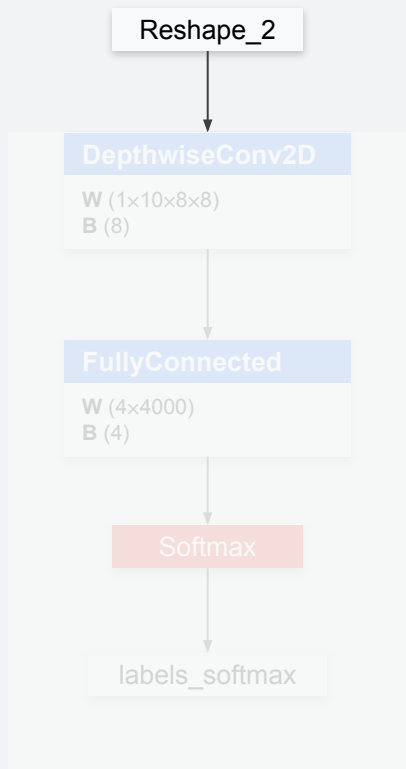
if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
    return;
}

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


TinyConv Keyword Spotting Model



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

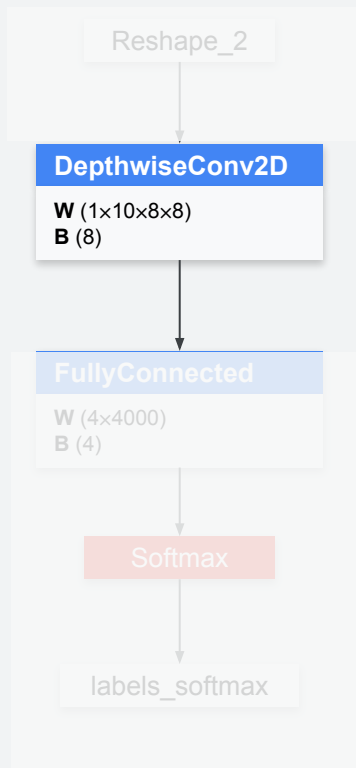
if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
    return;
}

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


TinyConv Keyword Spotting Model



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

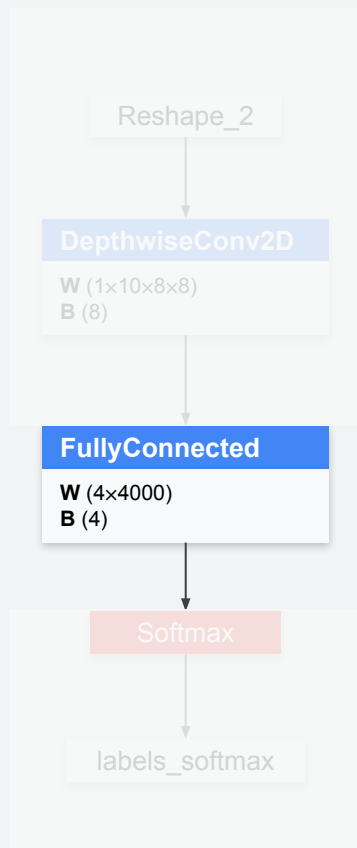
if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
    return;
}

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


TinyConv Keyword Spotting Model



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

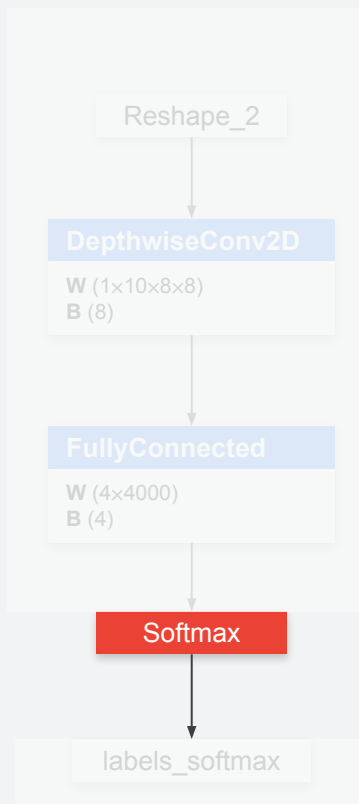
if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
    return;
}

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


TinyConv Keyword Spotting Model



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

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
}

if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
    return;
}

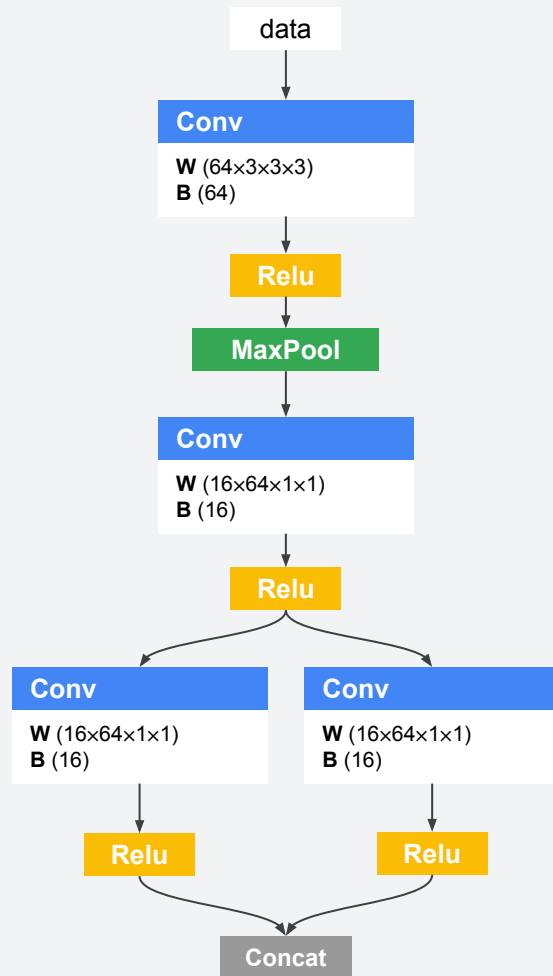
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
    return;
}

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


Which Ops to Include?

NETRON

<https://netron.app>



If memory is not an issue, you can choose to simply include all operators, both used and unused, at the expense of increased memory consumption

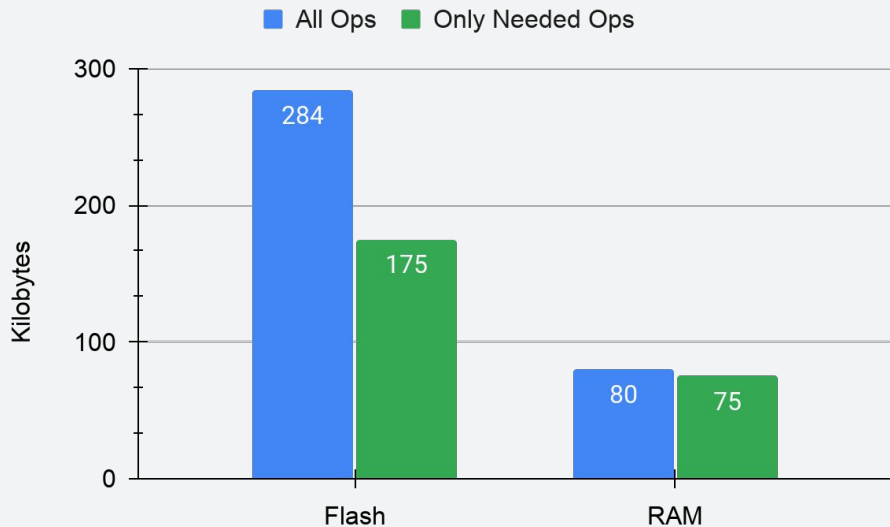
```
static tflite::AllOpsResolver resolver;
```

// Build an interpreter to run the model with.

```
static tflite::MicroInterpreter static_interpreter(  
    model, resolver, tensor_arena, kTensorArenaSize, error_reporter);  
interpreter = &static_interpreter;
```


Memory Improvements

- Selective op registration **reduces memory consumption by 30%**
- **Memory reduction varies by model**, depending on the operators used by the model



In Summary, what is TensorFlow Lite Micro?

Compatible with the TensorFlow training environment.



Built to fit on **embedded systems**:

- Very **small binary footprint**
- **No** dynamic memory allocation
- **No** dependencies on complex parts of the standard C/C++ libraries
- **No** operating system dependencies, **can run on bare metal**
- Designed to be **portable** across a wide variety of systems



Thank You!



Paper Discussions

Paper Discussion #1 - TFLite

[Link](#)

Paper Discussion #2 - MCUNet

[Link](#)



Guest Speaker

Tianqi Chen

Tianqi Chen is a distinguished researcher, primarily recognized as the creator of TVM, an open-source machine learning compiler stack, designed to enable efficient deployment of deep learning models on a variety of hardware platforms. Chen's contributions have been pivotal in progressing machine learning frameworks.



[Personal Website](#)

[Google Scholar](#)