POLITECNICO
MILANO 1863

Hardware Architectures for Embedded and Edge AI

Prof Manuel Roveri – manuel.roveri@polimi.it
Massimo Pavan – massimo.pavan@polimi.it

Exercise session 5 – Keyword spotting training

POLITECNICO
MILANO 1863

Converting the KS model

Converting from tflite to tflite micro

To run the model on device, it needs to be in a c like format so that it can be embedded in the firmware of your application.

```
!apt-get update && apt-get -qq install xxd

MODEL_TFLITE = 'models/TinyConvModel.tflite'
MODEL_TFLITE_MICRO = 'TinyConvModel.cc'
!xxd -i {MODEL_TFLITE} > {MODEL_TFLITE_MICRO}
REPLACE_TEXT = MODEL_TFLITE.replace('/', '_').replace('.', '_')
```

- It should look something like this:

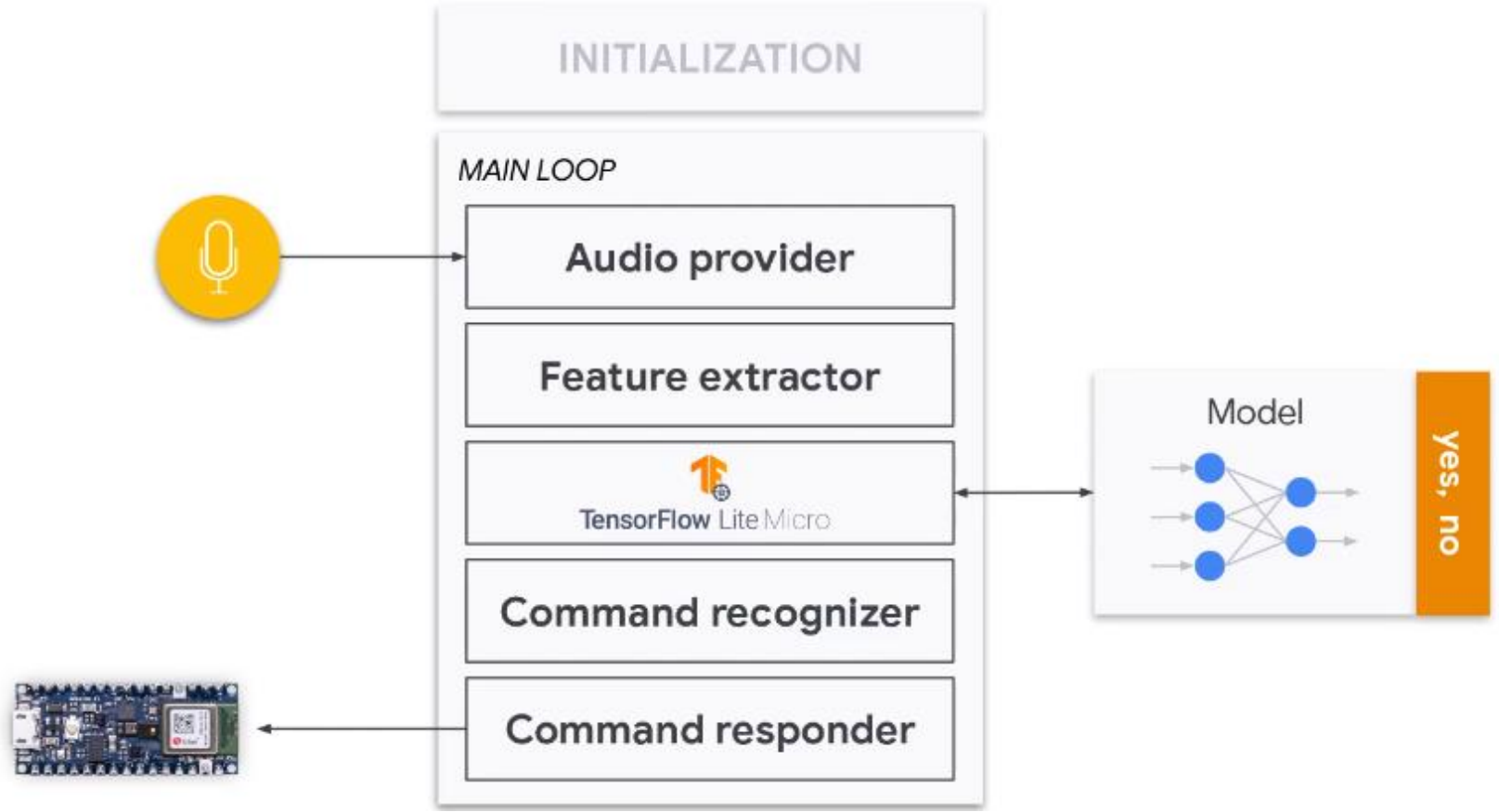
- The memory occupation of this text file is not indicative of the amount of storage memory your model will actually occupy in the firmware.

```
34
35 const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
36     0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
37     0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
38     0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
39     0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
40     0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
41     0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
42     0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
43     0x0b, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f,
44     0x72, 0x75, 0x6e, 0x74, 0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73,
45     0x69, 0x6f, 0x6e, 0x00, 0x0c, 0x00, 0x00, 0x00, 0xd4, 0x41, 0x00, 0x00,
46     0xb4, 0x41, 0x00, 0x00, 0x24, 0x03, 0x00, 0x00, 0xf4, 0x02, 0x00, 0x00,
47     0xec, 0x02, 0x00, 0x00, 0xe4, 0x02, 0x00, 0x00, 0xc4, 0x02, 0x00, 0x00,
48     0xbc, 0x02, 0x00, 0x00, 0x2c, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00,
49     0x1c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x16, 0xbd, 0xff, 0xff,
50     0x04, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e,
51     0x30, 0x00, 0x00, 0x00, 0x94, 0xba, 0xff, 0xff, 0x98, 0xba, 0xff, 0xff,
52     0x32, 0xbd, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x80, 0x02, 0x00, 0x00,
53     0xfa, 0xee, 0x28, 0xc4, 0xee, 0xfe, 0xcf, 0x0f, 0x1e, 0xf7, 0x1f, 0x06,
54     0x0d, 0xed, 0xe9, 0x83, 0x5c, 0xc9, 0x18, 0xe3, 0xf9, 0x14, 0x28, 0x2a,
```

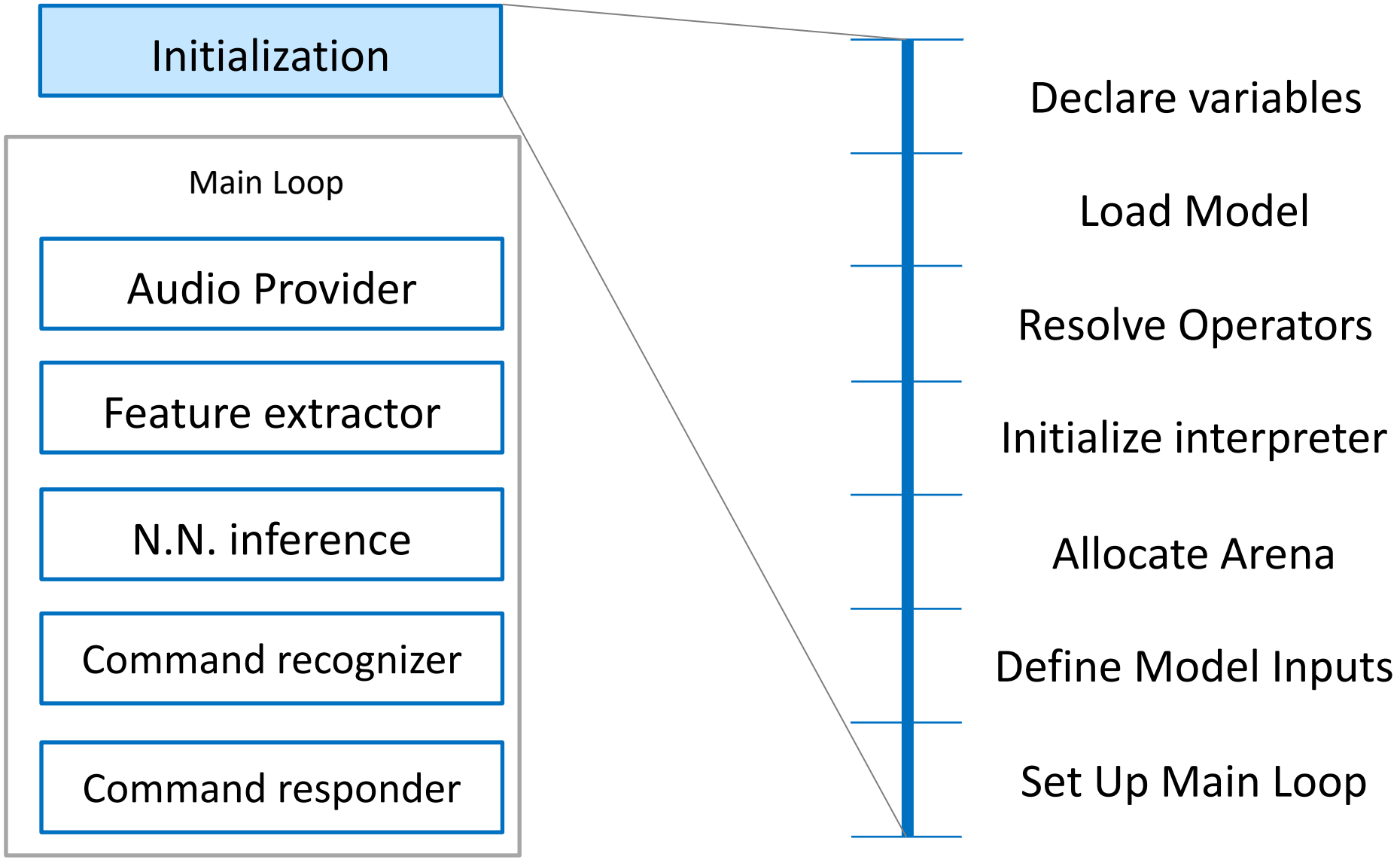
POLITECNICO
MILANO 1863

Deploying keyword spotting

Keyword spotting components



Initialization step by step



- Declare variables
- Load Model
- Resolve Operators
- Initialize interpreter
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

```
// 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;
}
```

Declare variables

Load Model

Resolve Operators

Initialize interpreter

Allocate Arena

Define Model Inputs

Set Up Main Loop

```
// 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;
}
```

Declare variables

Load Model

Resolve Operators

Initialize interpreter

Allocate Arena

Define Model Inputs

Set Up Main Loop

```
34
35  const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
36      0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
37      0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
38      0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
39      0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
40      0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
41      0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
42      0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
43      0x0b, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f,
44      0x72, 0x75, 0x6e, 0x74, 0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73,
45      0x69, 0x6f, 0x6e, 0x00, 0x0c, 0x00, 0x00, 0x00, 0xd4, 0x41, 0x00, 0x00,
46      0xb4, 0x41, 0x00, 0x00, 0x24, 0x03, 0x00, 0x00, 0xf4, 0x02, 0x00, 0x00,
47      0xec, 0x02, 0x00, 0x00, 0xe4, 0x02, 0x00, 0x00, 0xc4, 0x02, 0x00, 0x00,
48      0xbc, 0x02, 0x00, 0x00, 0x2c, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00,
49      0x1c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x16, 0xbd, 0xff, 0xff,
50      0x04, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e,
51      0x30, 0x00, 0x00, 0x00, 0x94, 0xba, 0xff, 0xff, 0x98, 0xba, 0xff, 0xff,
52      0x32, 0xbd, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x80, 0x02, 0x00, 0x00,
53      0xfa, 0xee, 0x28, 0xc4, 0xee, 0xfe, 0xcf, 0x0f, 0x1e, 0xf7, 0x1f, 0x06,
54      0x0d, 0xed, 0xe9, 0x83, 0x5c, 0xc9, 0x18, 0xe3, 0xf9, 0x14, 0x28, 0x2a,
55      0x09, 0xf2, 0x18, 0x34, 0x62, 0xea, 0xef, 0xd6, 0x36, 0xb7, 0x1e, 0xf7,
56      0x3b, 0x22, 0x28, 0x39, 0xc2, 0x9d, 0xf1, 0x07, 0x5e, 0x0b, 0x1e, 0x2c,
57      0x07, 0xdd, 0xfd, 0xc3, 0xd8, 0x4a, 0xf3, 0x28, 0xa7, 0x16, 0xd5, 0xf1,
58      0xc3, 0x05, 0xfd, 0x27, 0xcc, 0xba, 0x1e, 0xcb, 0xd7, 0x3d, 0xd4, 0x29,
59      0x00, 0xfd, 0x28, 0x44, 0xfb, 0xf2, 0xf3, 0xb6, 0x4f, 0xcf, 0x09, 0xf0,
60      0xfa, 0x45, 0x41, 0x49, 0x05, 0xc5, 0x17, 0x5d, 0x64, 0x00, 0xf8, 0xee,
61      0x48, 0x17, 0xf4, 0xe9, 0x2e, 0x4b, 0x2e, 0x3f, 0xdf, 0xee, 0xe4, 0x08,
62      0x38, 0xf1, 0x16, 0x13, 0x2f, 0x2a, 0xed, 0xc2, 0xbf, 0x36, 0xf4, 0x02,
63      0xcf, 0xaa, 0xd2, 0xfa, 0xac, 0x13, 0xf6, 0xe8, 0xb5, 0x68, 0x12, 0xb6,
64      0xce, 0x0e, 0xdf, 0x58, 0xe4, 0x49, 0x14, 0x15, 0x03, 0xed, 0xfa, 0xd4,
65      0x40, 0xa7, 0xf6, 0xca, 0xfb, 0x00, 0x4d, 0x5e, 0xe4, 0x55, 0x1d, 0x30,
66      0x45, 0xe2, 0xfc, 0x01, 0x48, 0x81, 0xe9, 0xf1, 0x1e, 0xfc, 0x21, 0x32,
67      0xed, 0x4b, 0xed, 0xfa, 0x2f, 0xd2, 0xfa, 0xfb, 0x4d, 0xa7, 0xed, 0xc7,
68      0x92, 0xdf, 0xe6, 0xdb, 0xf8, 0x1f, 0xd9, 0xfa, 0x91, 0xf5, 0xe5, 0xc5,
69      0x8c, 0x17, 0x0f, 0xb9, 0xd2, 0xc7, 0xfe, 0x68, 0xd3, 0x51, 0x2e, 0x49,
70      0x1f, 0xbd, 0x01, 0xeb, 0x31, 0x17, 0xf0, 0xef, 0xff, 0xb8, 0x5d, 0x62,
71      0x02, 0x0f, 0x1f, 0x78, 0x6a, 0xb0, 0xf9, 0xfe, 0x4f, 0xcc, 0xd3, 0xff,
72      0x0a, 0x96, 0x1e, 0x2c, 0xed, 0xbc, 0xf4, 0x0b, 0x42, 0xc8, 0xf1, 0xea,
73      0x6e, 0x58, 0xec, 0xc4, 0x99, 0xae, 0xdc, 0xd7, 0x12, 0x87, 0xd8, 0x06,
74      0xa2, 0xc2, 0xe6, 0xa2, 0x81, 0x24, 0xe9, 0xac, 0xce, 0xb6, 0x15, 0x6b,
75      0xba, 0x00, 0x19, 0x58, 0x29, 0xb6, 0xfe, 0x01, 0x25, 0x96, 0xd2, 0xec,
```

Declare variables

Load Model

Resolve Operators

Initialize interpreter

Allocate Arena

Define Model Inputs

Set Up Main Loop

```
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;
}
```

- Declare variables
- Load Model
- Resolve Operators
- Initialize interpreter
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

```
// 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;
```

- Declare variables
- Load Model
- Resolve Operators
- Initialize interpreter
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

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

TfLiteStatus allocate_status =
interpreter->AllocateTensors();

if (allocate_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter,
  "AllocateTensors() failed");
  return;
}
```

- Declare variables
- Load Model
- Resolve Operators
- Initialize interpreter
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

```
// Get information about the memory area to use
// for the model's input.

model_input = interpreter->input(0);

if   ((model_input->dims->size != 2) ||
     (model_input->dims->data[0] != 1) ||
     (model_input->dims->data[1] !=
     (kFeatureSliceCount * kFeatureSliceSize)) ||
     (model_input->type != kTfLiteInt8)) {
          TF_LITE_REPORT_ERROR(error_reporter,
         "Bad input tensor parameters in model");
  return;
}

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

Declare variables

Load Model

Resolve Operators

Initialize interpreter

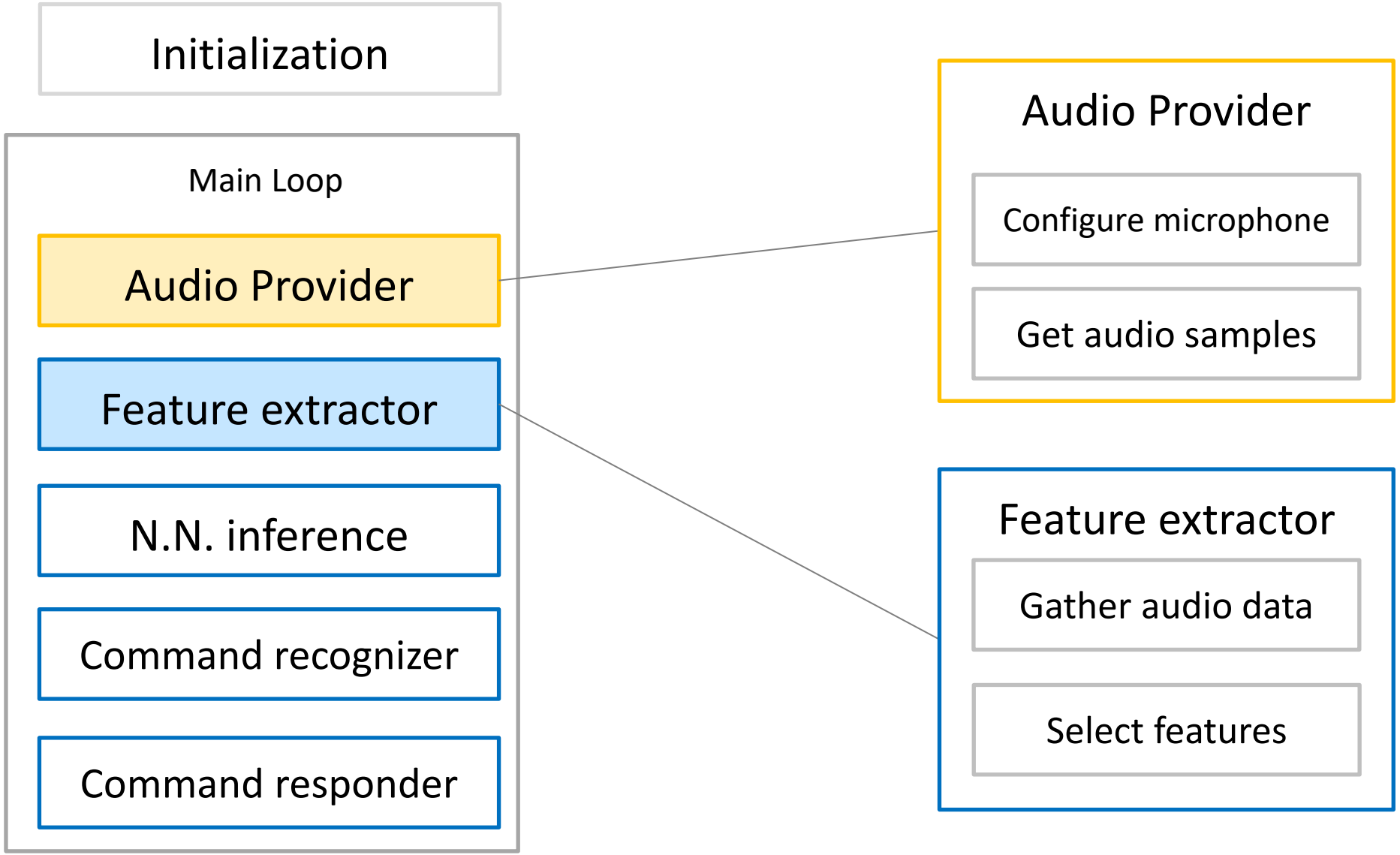
Allocate Arena

Define Model Inputs

Set Up Main Loop

```
// Prepare to access the audio spectrograms
// from a microphone or other source that will
// provide the inputs to the neural network.

static FeatureProvider
static_feature_provider(kFeatureElementCount,
                                    feature_buffer);

feature_provider = &static_feature_provider;

static RecognizeCommands
static_recognizer(error_reporter);
recognizer = &static_recognizer;
```

Main Loop step by step



Audio Provider

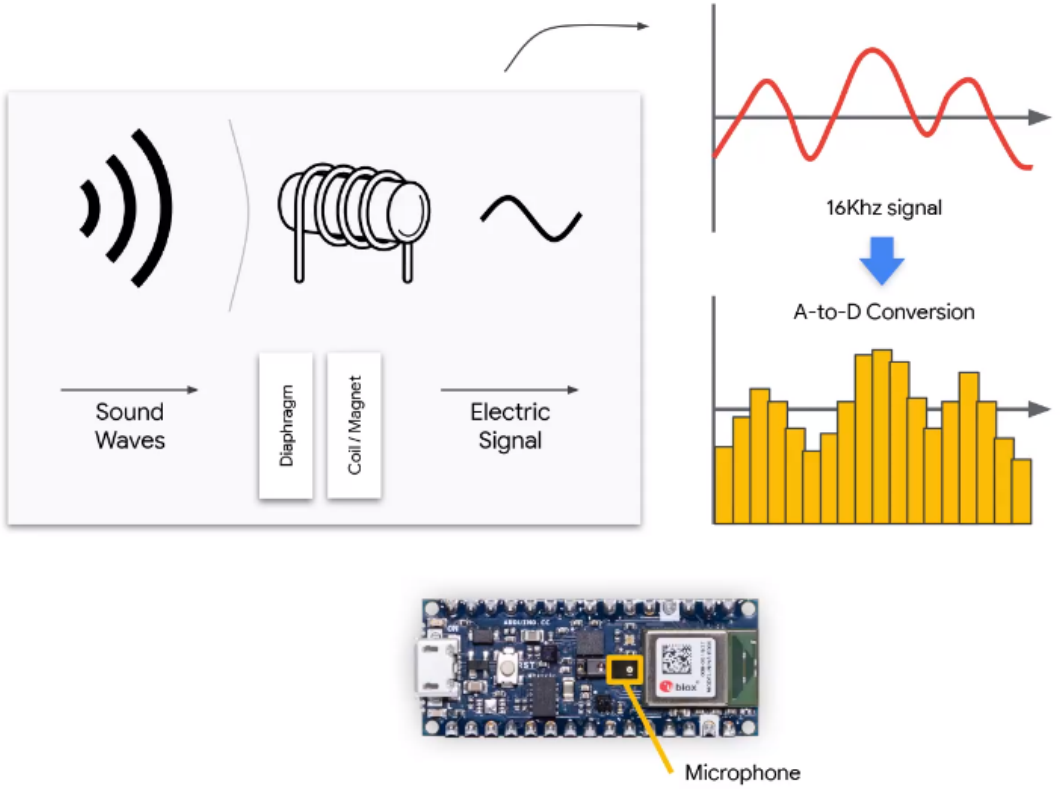
Configure microphone

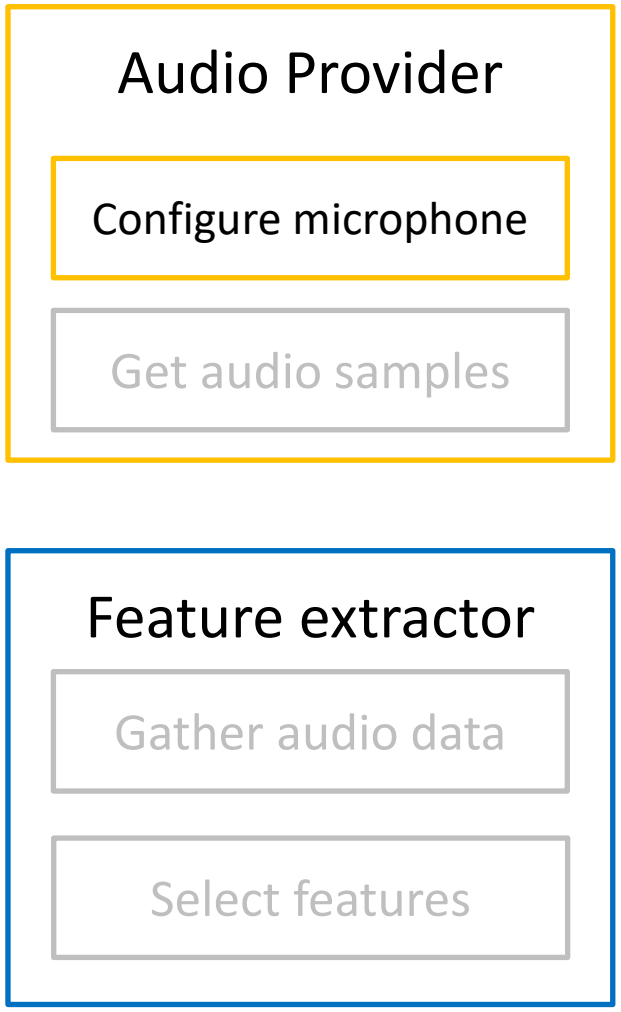
Get audio samples

Feature extractor

Gather audio data

Select features






```
TfLiteStatus InitAudioRecording(tflite::ErrorReporter* error_reporter) {
  // Hook up the callback that will be called with each sample
  PDM.onReceive(CaptureSamples);

  // Start listening for audio: MONO @ 16KHz with gain at 20
  PDM.begin(1, kAudioSampleFrequency);
  PDM.setGain(20);

  // Block until we have our first audio sample
  while (!g_latest_audio_timestamp) {
  }

  return kTfLiteOk;
}
```

Audio Provider

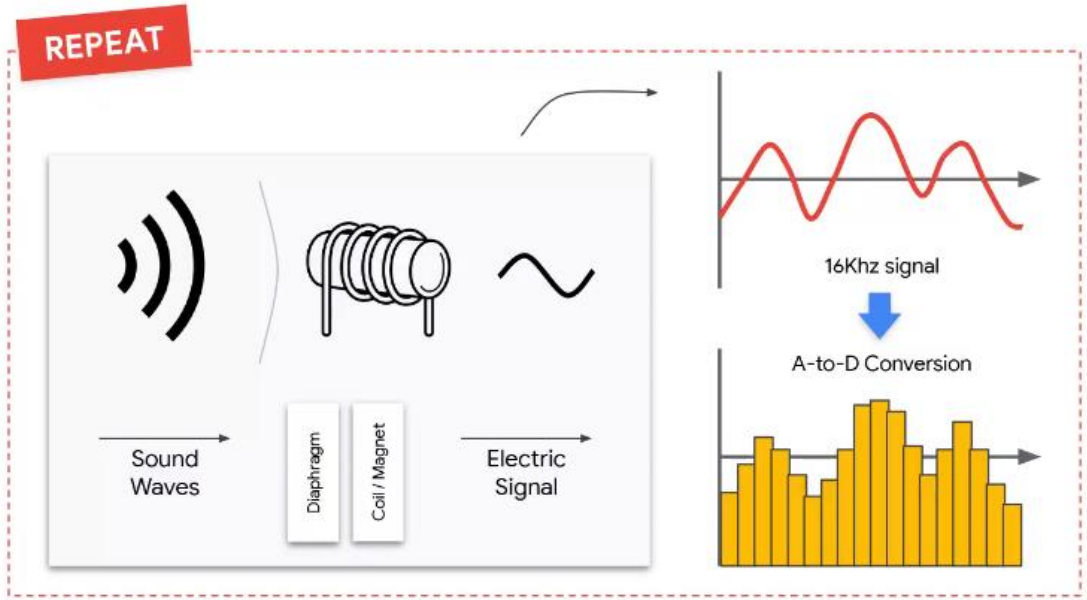
Configure microphone

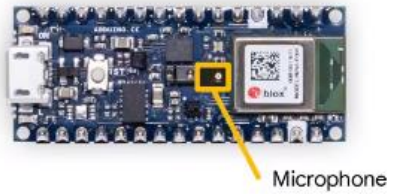
Get audio samples

Feature extractor

Gather audio data

Select features






```
TfLiteStatus InitAudioRecording(tflite::ErrorReporter* error_reporter)
```

0	1	2	3	...			N

Audio Provider

Configure microphone

Get audio samples

Feature extractor

Gather audio data

Select features

```
// This is an abstraction around an audio source like a microphone, and is
// expected to return 16-bit PCM sample data for a given point in time. The
// sample data itself should be used as quickly as possible by the caller, since
// to allow memory optimizations there are no guarantees that the samples won't
// be overwritten by new data in the future. In practice, implementations should
// ensure that there's a reasonable time allowed for clients to access the data
// before any reuse.

// The reference implementation can have no platform-specific dependencies, so
// it just returns an array filled with zeros. For real applications, you should
// ensure there's a specialized implementation that accesses hardware APIs.

TfLiteStatus GetAudioSamples(tflite::ErrorReporter* error_reporter,
                             int start_ms, int duration_ms,
                             int* audio_samples_size, int16_t** audio_samples);
```

Audio Provider

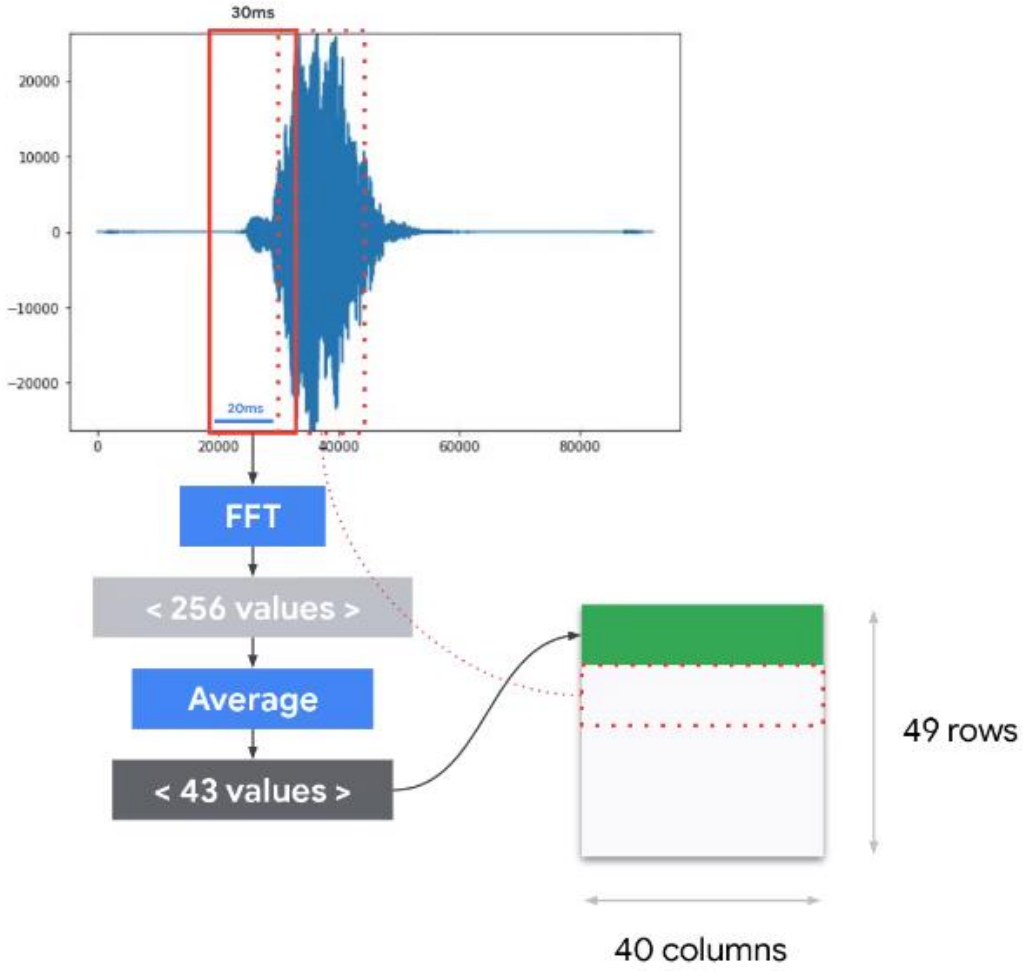
Configure microphone

Get audio samples

Feature extractor

Gather audio data

Select features



Audio Provider

Configure microphone

Get audio samples

Feature extractor

Gather audio data

Select features

30ms

```
void setup() {
// Prepare to access the audio spectrograms from a microphone
// or other source that will provide the inputs to the neural network.

static FeatureProvider static_feature_provider(kFeatureElementCount,
                                               feature_buffer);

...
}

// The name of this function is important for Arduino compatibility.
void loop() {
  // Fetch the spectrogram for the current time.

  const int32_t current_time = LatestAudioTimestamp();
  int how_many_new_slices = 0;
  TfLiteStatus feature_status = feature_provider->PopulateFeatureData(
      error_reporter, previous_time, current_time,&how_many_new_slices);
  if (feature_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "Feature generation failed");
    Return;
}
```

Audio Provider

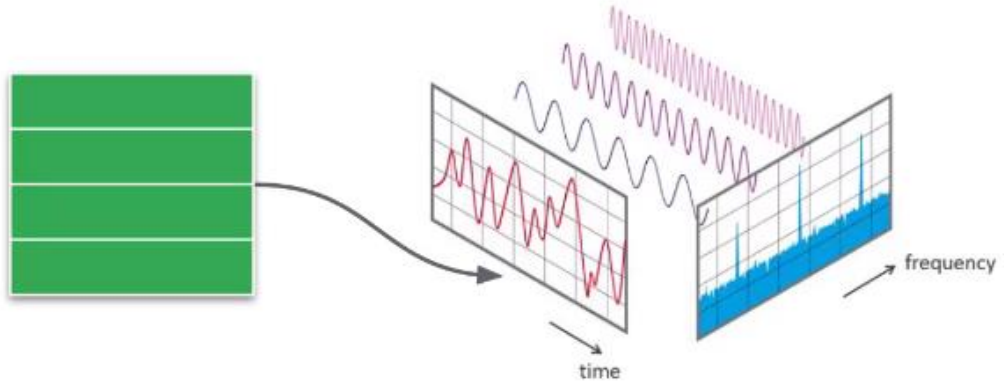
Configure microphone

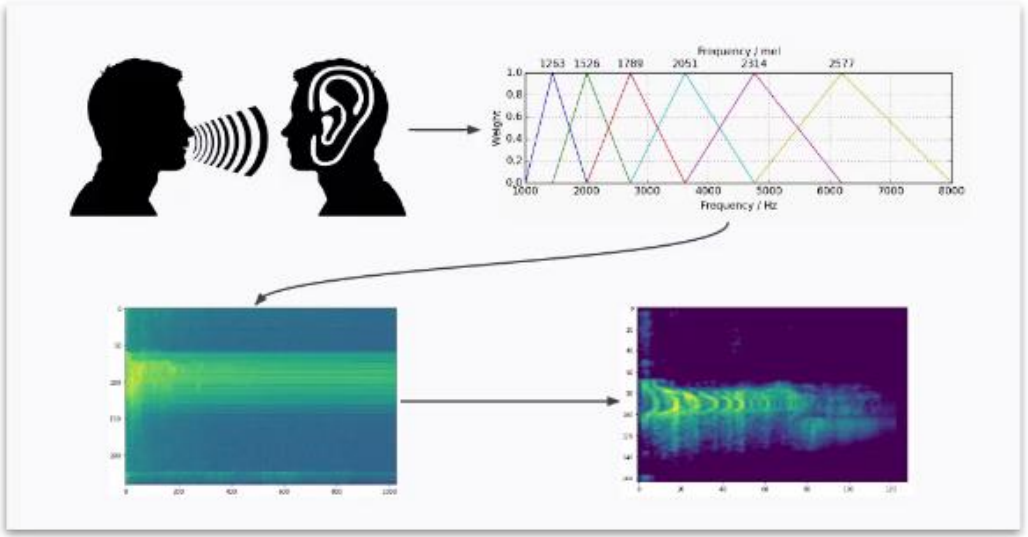
Get audio samples

Feature extractor

Gather audio data

Select features

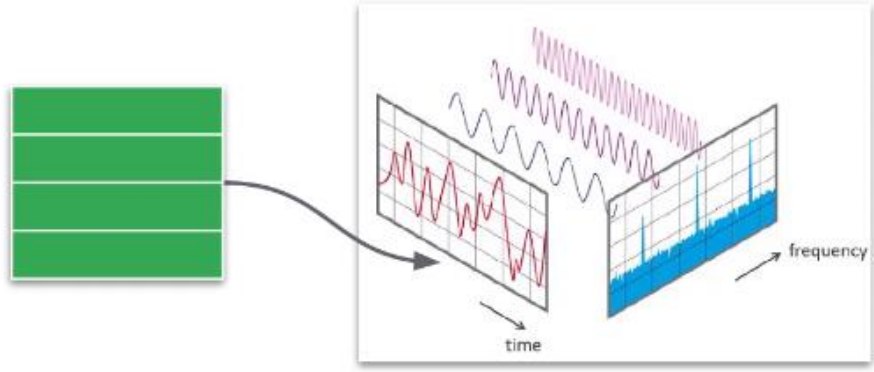




Audio Provider

Configure microphone

Get audio samples

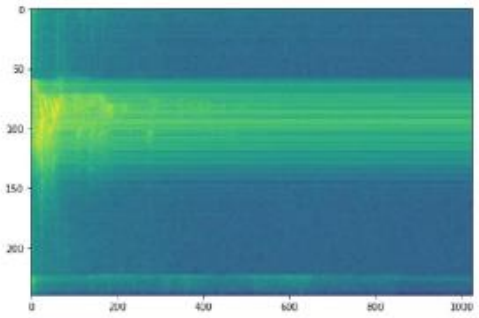


Feature extractor

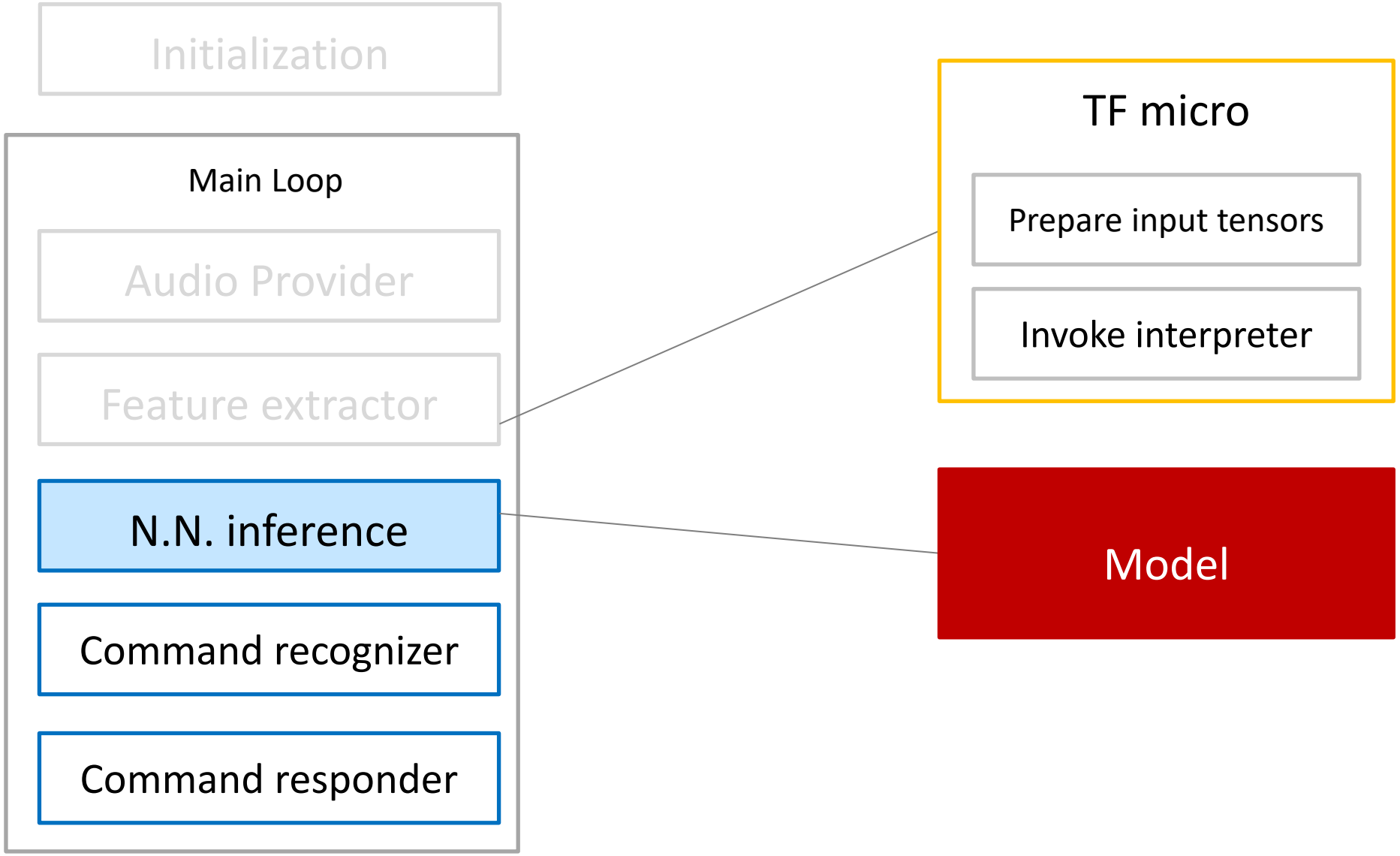
Gather audio data

Select features

```
size_t num_samples_read;
TfLiteStatus generate_status = GenerateMicroFeatures(
    error_reporter, audio_samples, audio_samples_size, kFeatureSliceSize,
    new_slice_data, &num_samples_read);
if (generate_status != kTfLiteOk) {
  return generate_status;
}
```



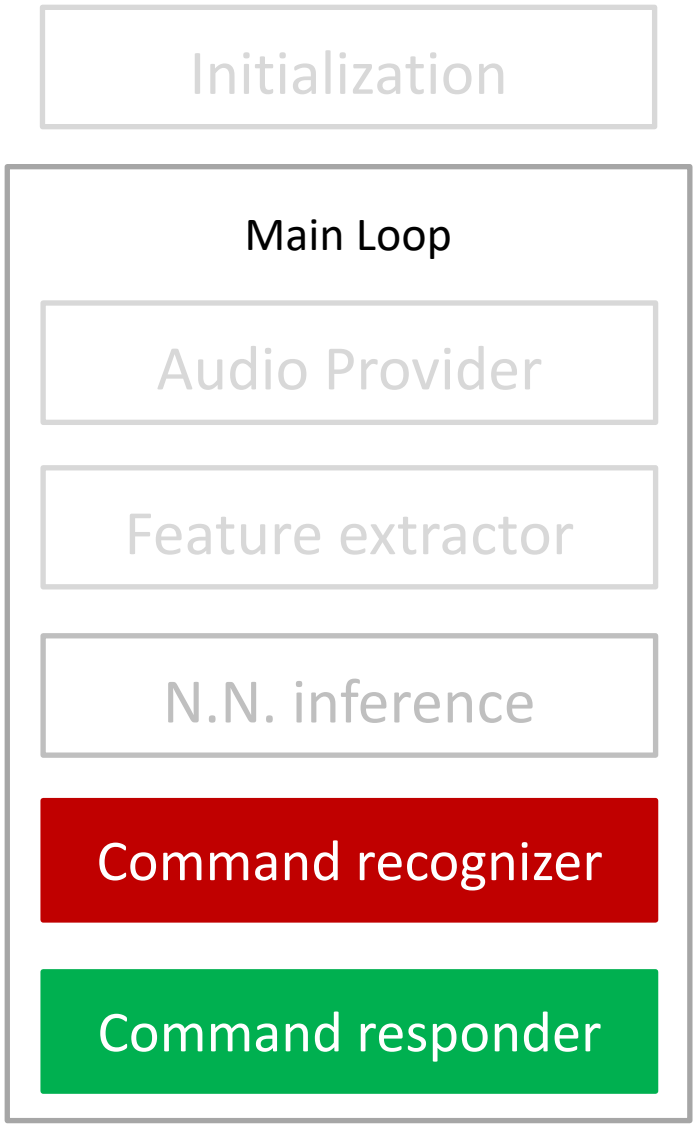
Main Loop step by step




```
// Copy feature buffer to input tensor
for (int i = 0; i < kFeatureElementCount; i++) {
  model_input_buffer[i] = feature_buffer[i];
}

// Run the model on the spectrogram input
// make sure it succeeds.

TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "Invoke
                                        failed");
  return;
}
```


```
// Copy feature buffer to input tensor
for (int i = 0; i < kFeatureElementCount; i++) {
  model_input_buffer[i] = feature_buffer[i];
}

// Run the model on the spectrogram input
// make sure it succeeds.

TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "Invoke
                                        failed");
  return;
}
```

Main Loop step by step

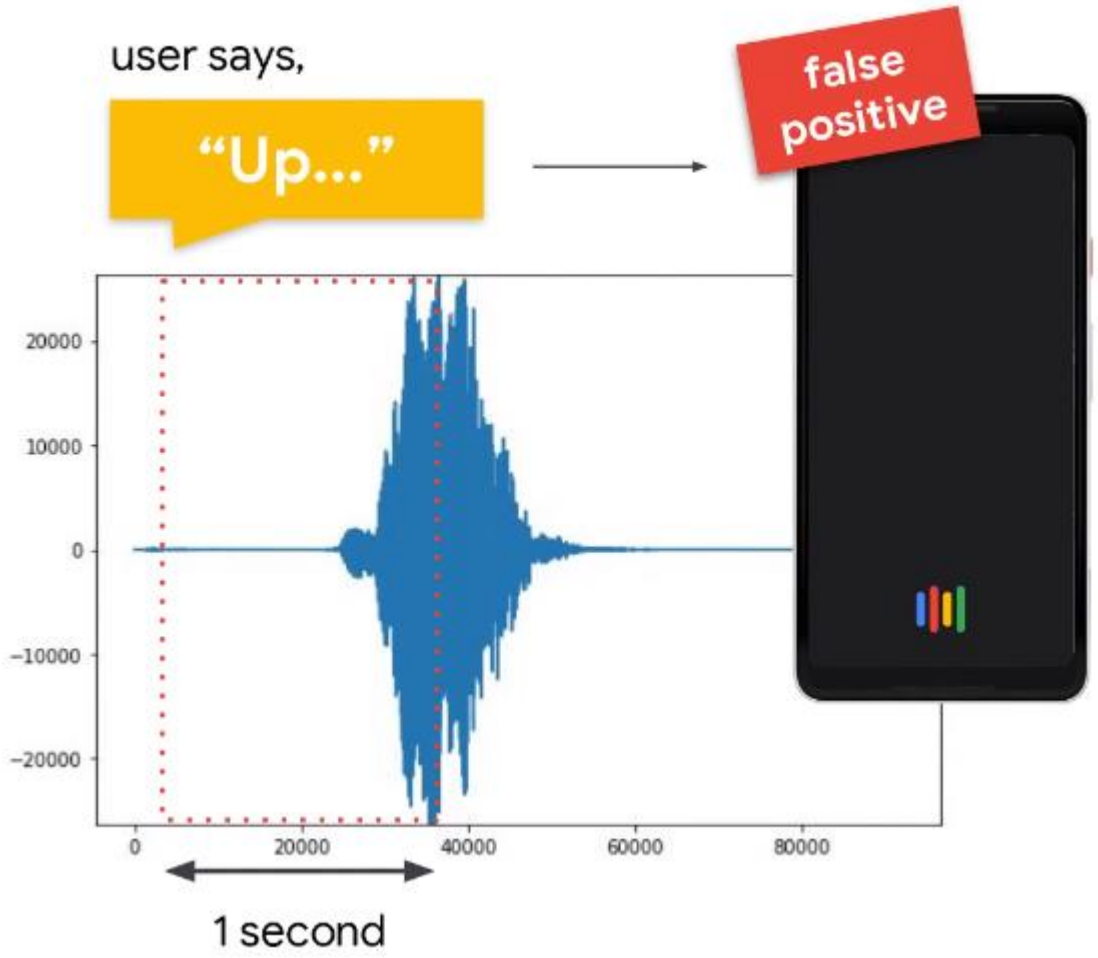


Command recognizer

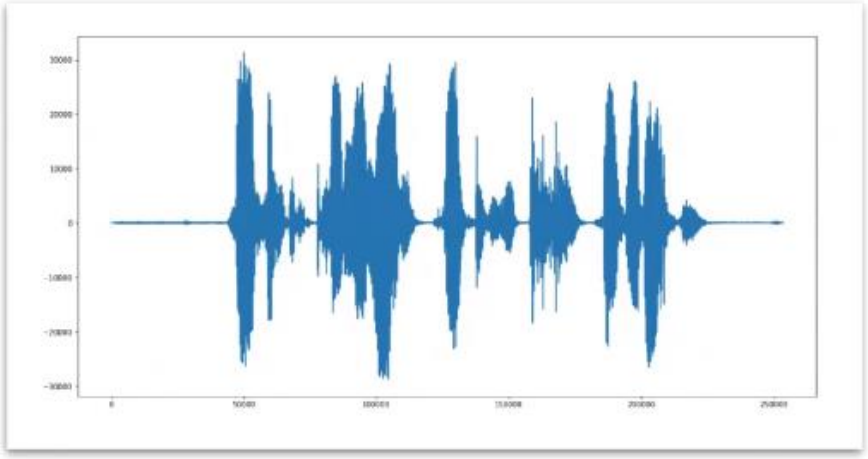
Command responder

Command recognizer

Command responder



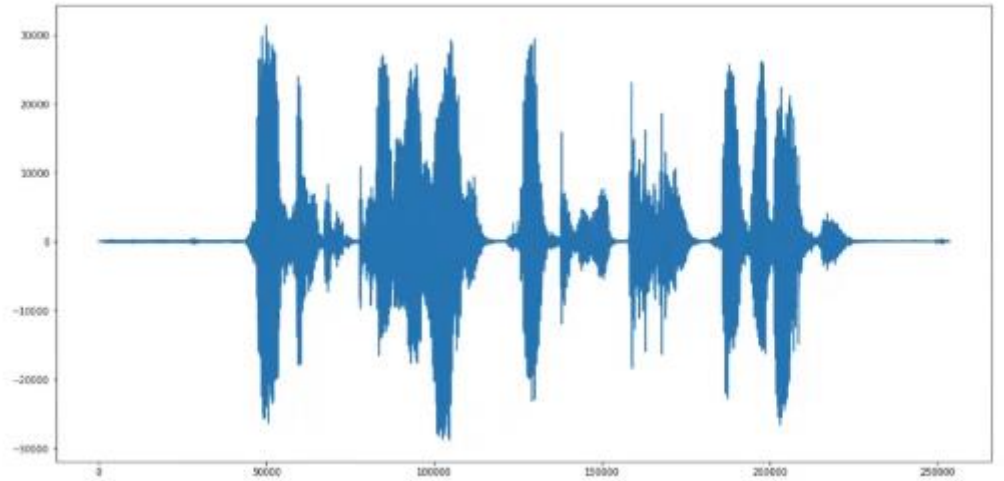
Command recognizer



Command responder

```
// Determine whether a command was recognized based
// on the output of inference

const char* found_command = nullptr;
uint8_t score = 0;
bool is_new_command = false;

TfLiteStatus process_status = recognizer->ProcessLatestResults(
    output, current_time, &found_command, &score, &is_new_command);

if (process_status != kTfLiteOk) {
      TF_LITE_REPORT_ERROR(error_reporter,
      "RecognizeCommands::ProcessLatestResults() failed");
      return;
}
```

Command recognizer

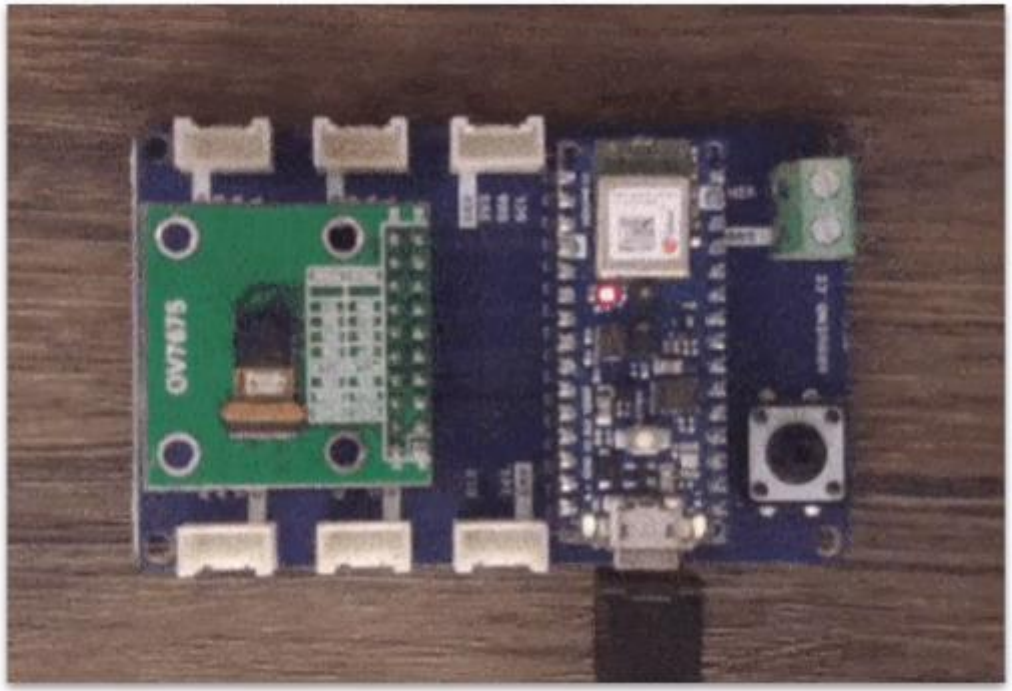


Command responder

Timestep	"Yes"	"No"	"Unknown"
1	55%	35%	20%
2	65%	25%	10%
3	76%	12%	12%
4	88%	7%	5%
5	99%	0.5%	0.5%

Command recognizer

Command responder

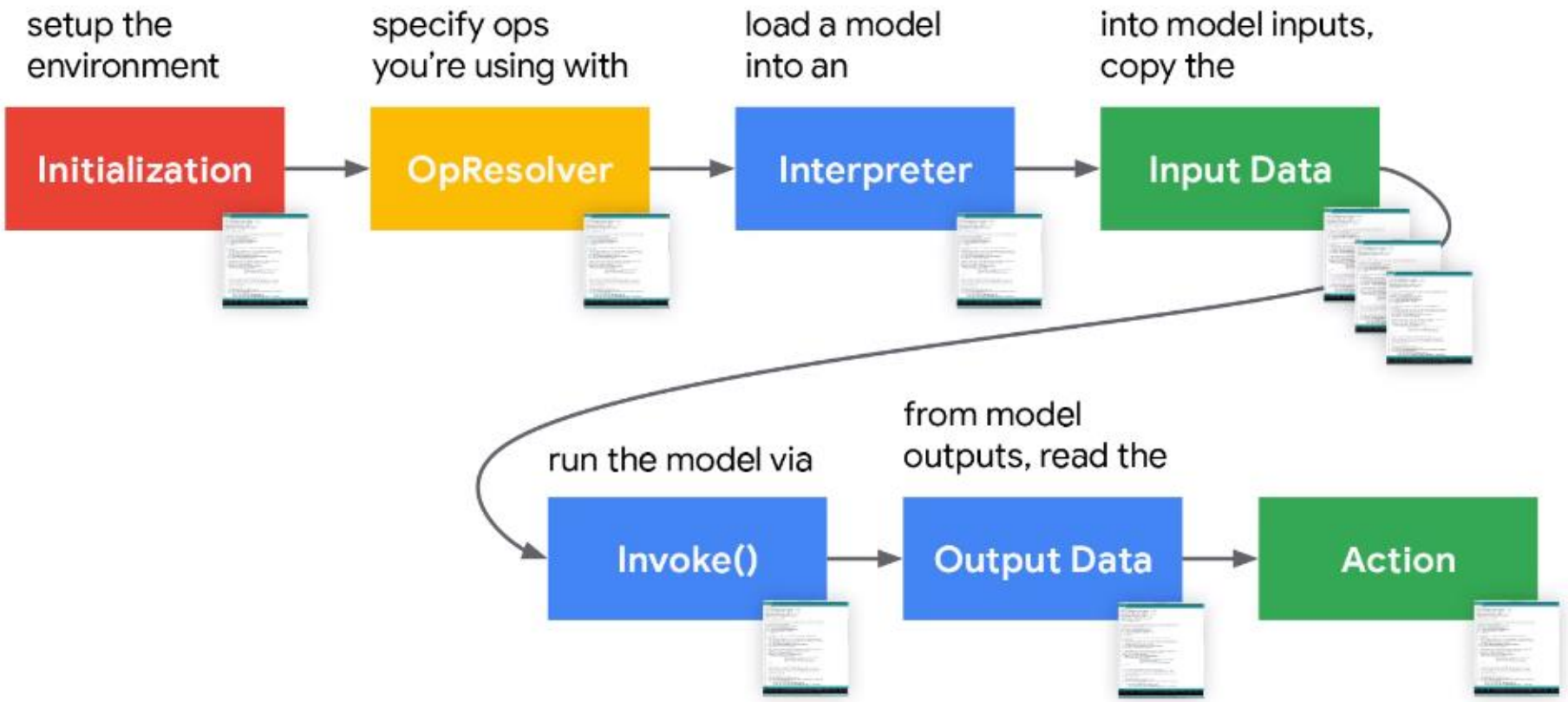


Command recognizer

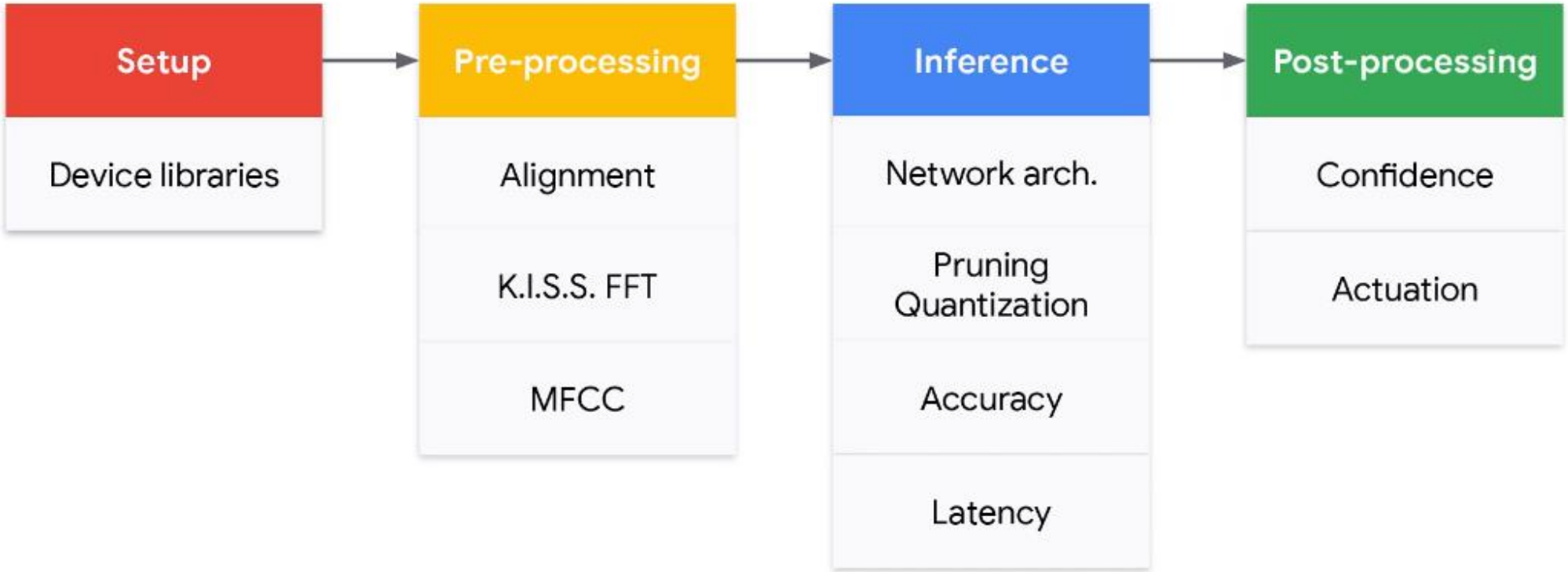
```
// Do something based on the recognized command.
// The default implementation just prints to
// the error console, but you should replace with
// your own function for a real application.

RespondToCommand(error_reporter, current_time,
                 found_command, score, is_new_command);
```

Command responder

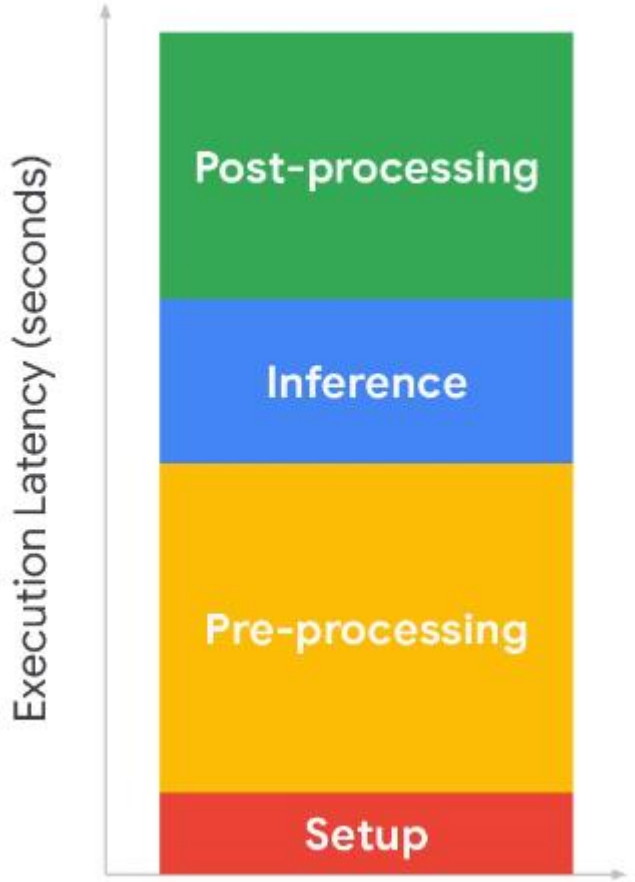
Summing up…

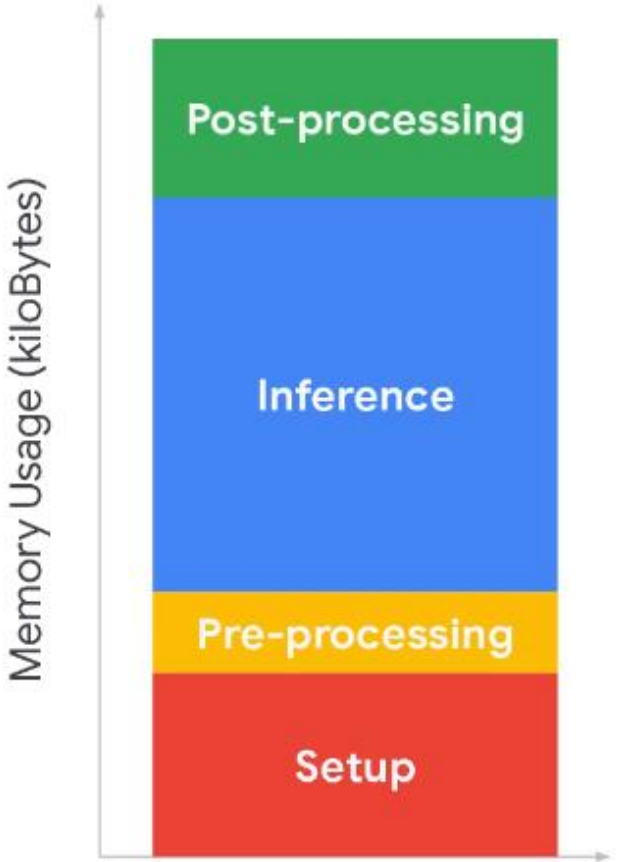

setup the environment
specify ops you're using with
load a model into an
into model inputs, copy the
Initialization
OpResolver
Interpreter
Input Data
run the model via
from model outputs, read the
Invoke()
Output Data
Action

Identify what you need to modify and focus on that part.

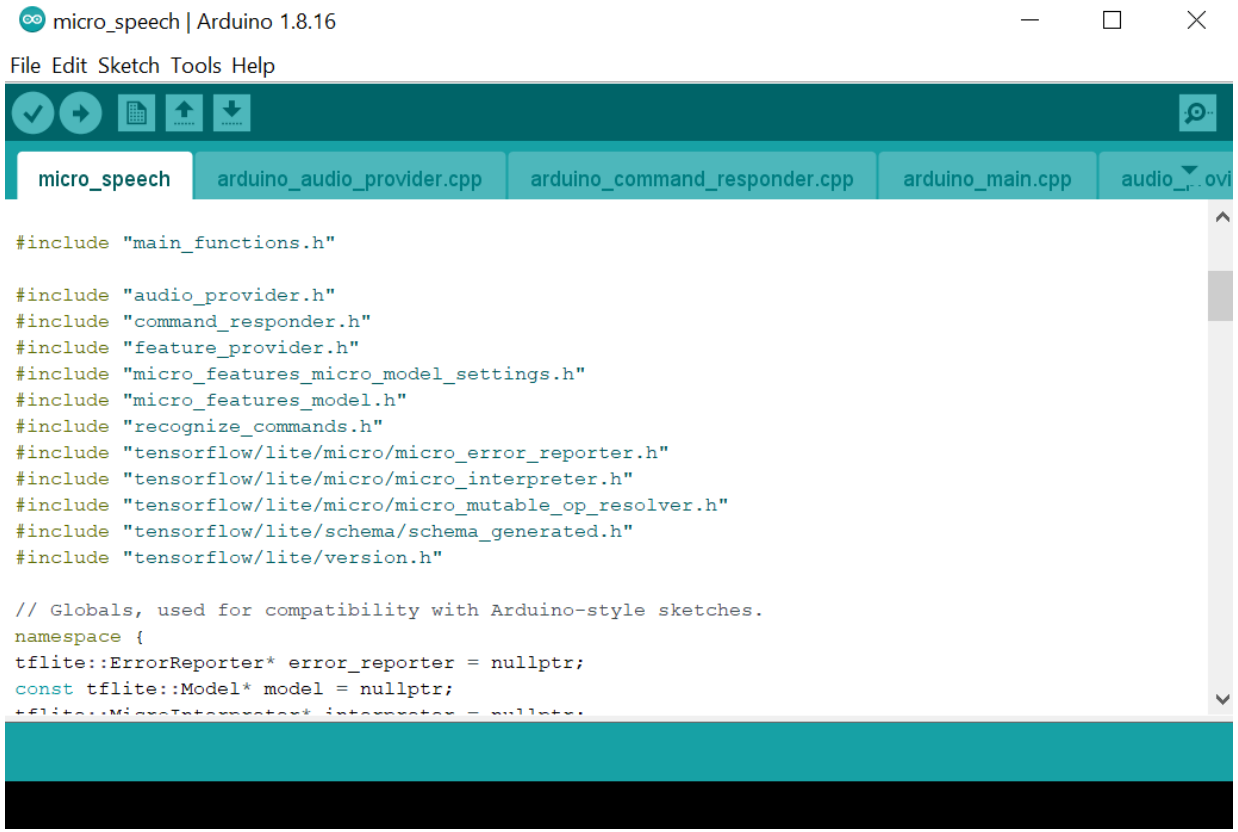


Why caring about all this stuff in ai course? Well…

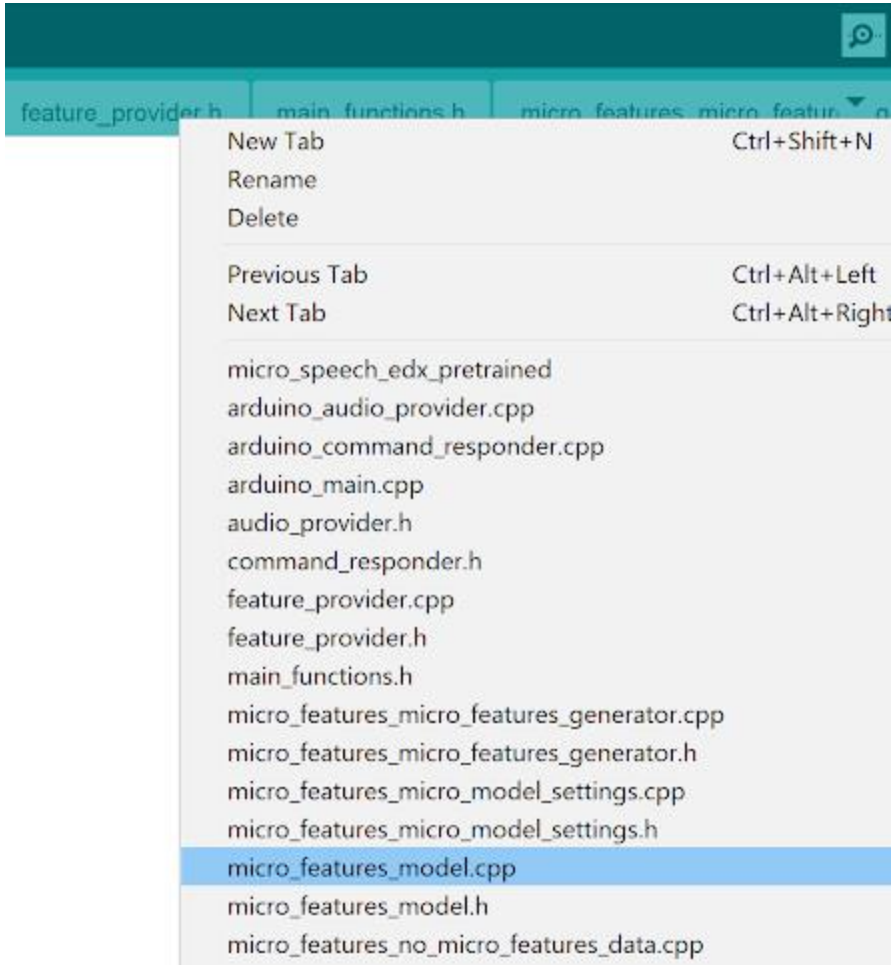


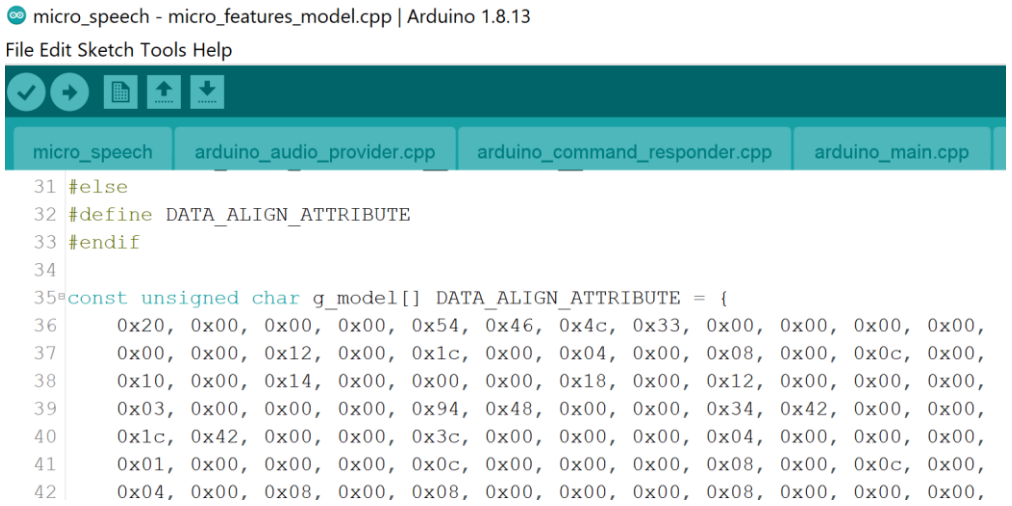


Let's deploy a new model: Arduino IDE + COLAB time



Colab: https://colab.research.google.com/drive/1YH6vXlDzzCRZOT-sLx50TNAE-LhVkcOK?usp=sharing


feature_provider.h
main_functions.h
micro_features_micro_featur
New Tab Ctrl+Shift+N
Rename
Delete
Previous Tab Ctrl+Alt+Left
Next Tab Ctrl+Alt+Right
micro_speech_edx_pretrained
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp
audio_provider.h
command_responder.h
feature_provider.cpp
feature_provider.h
main_functions.h
micro_features_micro_features_generator.cpp
micro_features_micro_features_generator.h
micro_features_micro_model_settings.cpp
micro_features_micro_model_settings.h
micro_features_model.cpp
micro_features_model.h
micro_features_no_micro_features_data.cpp



as the variable type is different in the downloaded or printed model.cc file (and the length!)

```
1590     0x02, 0x00, 0x00, 0x00, 0x00, 0x00
1591     0x06, 0x00, 0x00, 0x00, 0x00, 0x16
1592     0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
1593     0x04, 0x00, 0x00, 0x00, 0x00, 0x00
1594     0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
1595     0x03, 0x00, 0x00, 0x00};
1596 const int g_model_len = 18712;
```

micro_features_micro_model_settings.h

```
constexpr int kCategoryCount
= 5;
```

micro_features_micro_model_settings.cpp

```
const char* kCategoryLabels[kCategoryCount] = {
      "silence",
      "unknown",
      "up",
      "down",
      "go",
};
```

arduino_command_responder.cpp

```
// Red for up -- note here that you do not need to index
//               into the first letter only, just a unique
//               letter combination in the keyword! That
//               said make sure you do not index beyond the
//               end of ANY keyword or you will get an error!,
if(found_command[1] == 'p') {
   last_command_time = current_time;
   digitalWrite(LEDR, LOW);
}
```

```
// Green for down
if(found_command[0] == 'd') {
   last_command_time = current_time;
   digitalWrite(LEDG, LOW);
}
```

```
// Blue for go
if(found_command[0] == 'g') {
   last_command_time = current_time;
   digitalWrite(LEDB, LOW);
}
```

```
// All three for unknown (white)
if(found_command[1] == 'n') {
   last_command_time = current_time;
   digitalWrite(LEDR, LOW);
   digitalWrite(LEDG, LOW);
   digitalWrite(LEDB, LOW);
}
```

POLITECNICO
MILANO 1863

Appendix

Credits and reference

- "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers", Daniel Situnayake, Pete Warden, O'Reilly Media, Inc.
- Online course:
  - https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
- A lot more material on TinyML:
  - http://tinyml.seas.harvard.edu/