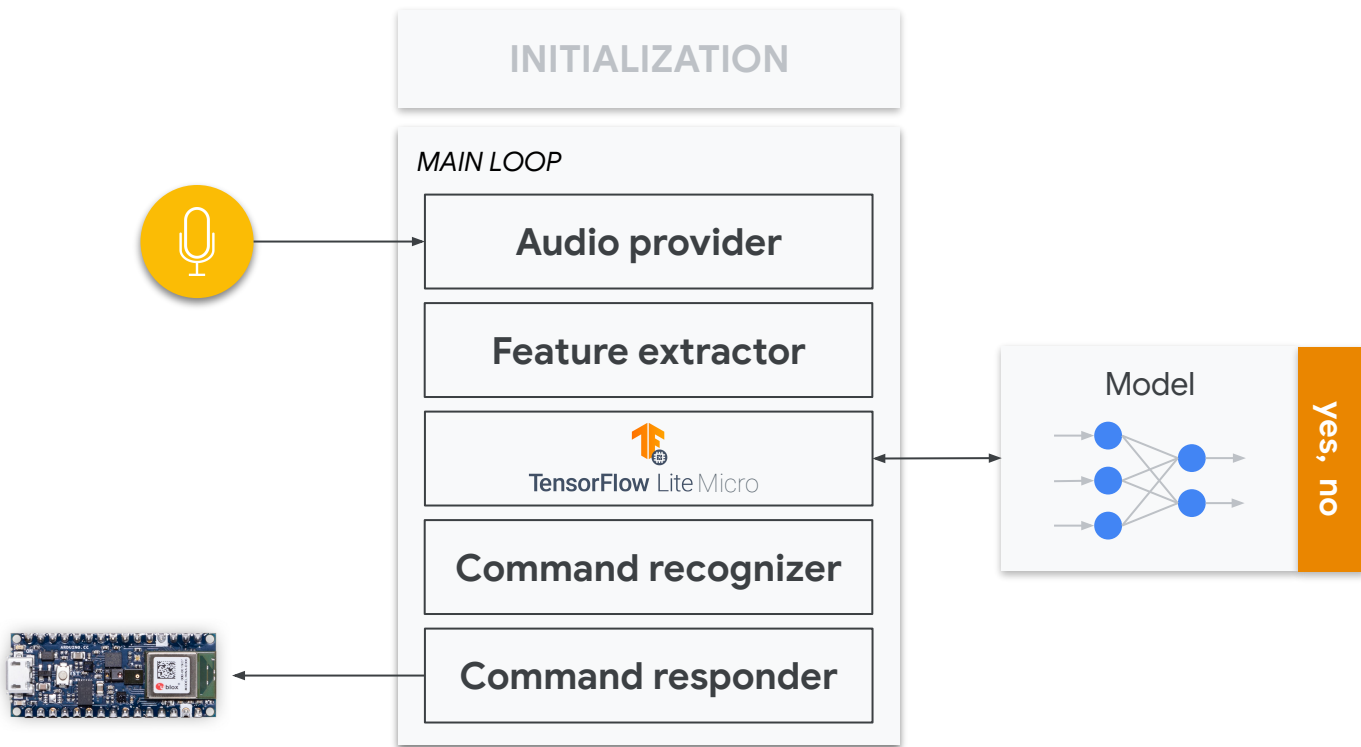


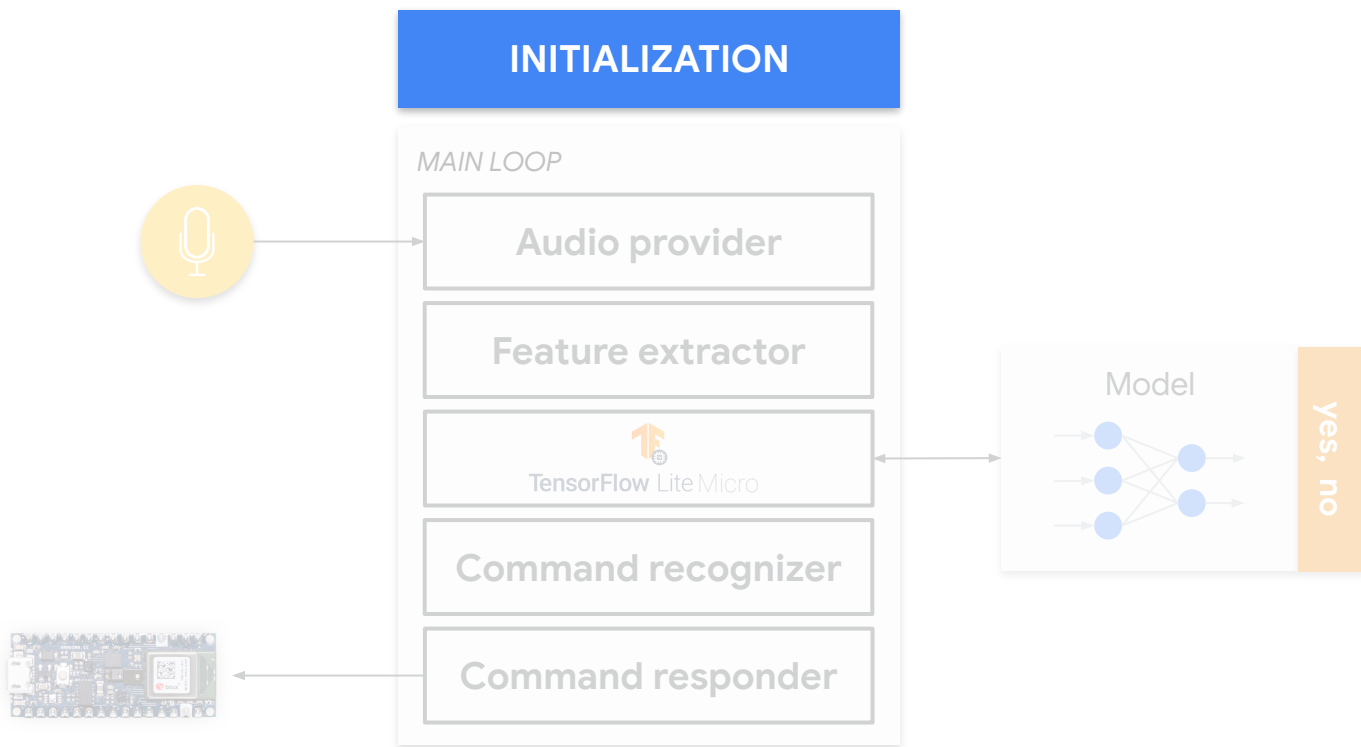
KWS Initialization

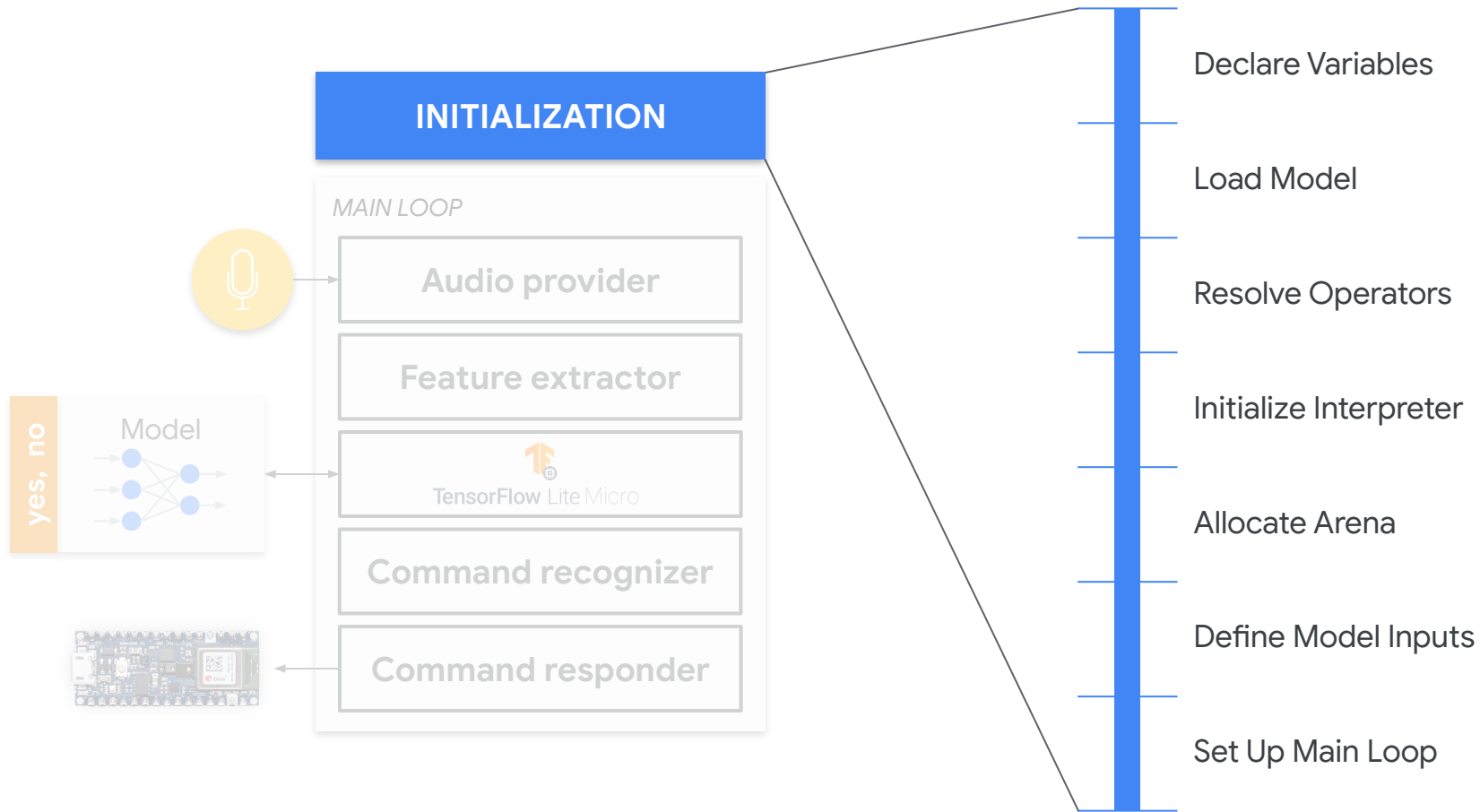


Keyword Spotting Components



Keyword Spotting Components







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

```
// 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() != TFLITE_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;  
}
```

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, 0x00, 0x18, 0x00, 0x12, 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     0xa0, 0x96, 0x1e, 0x2c, 0xed, 0xbc, 0xf4, 0x0b, 0xc8, 0x42, 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

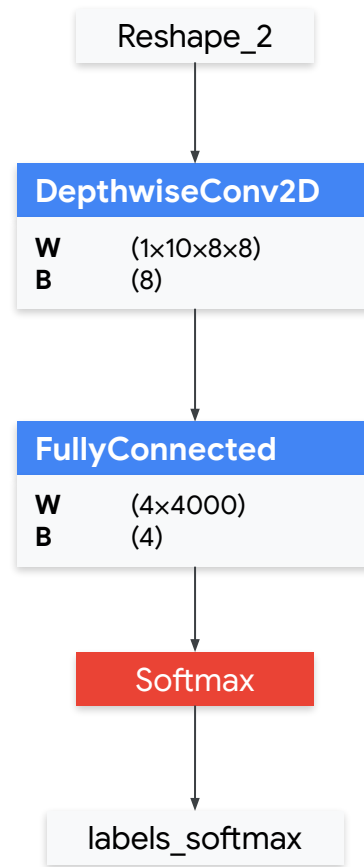
Initialize Interpreter

Allocate Arena

Define Model Inputs

Set Up Main Loop

```
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, 0x00, 0x18, 0x00, 0x12, 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     0xa0, 0x96, 0x1e, 0x2c, 0xed, 0xbc, 0xf4, 0x0b, 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

```
// 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;  
} // namespace
```

```
// 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;
```

```
// 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

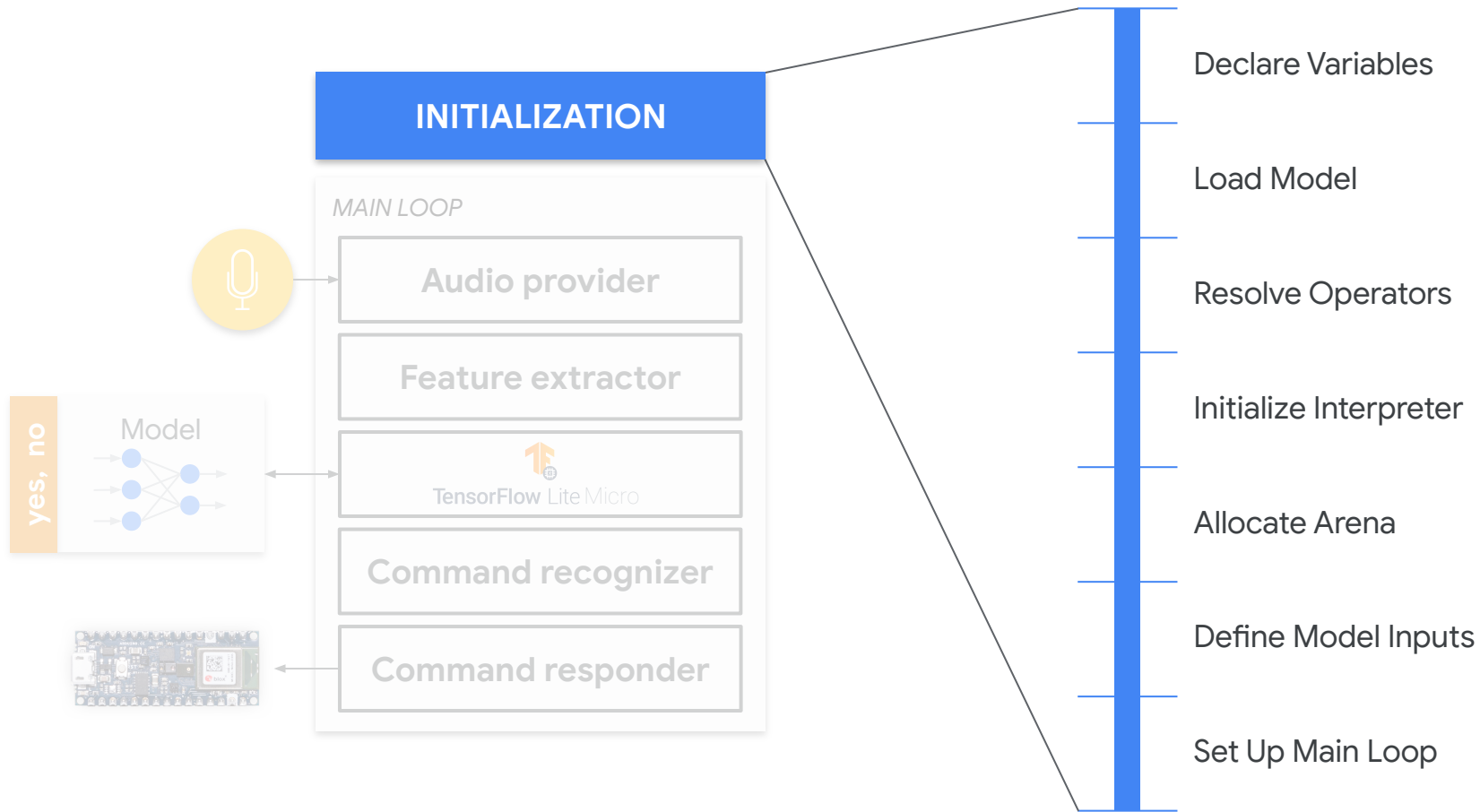
Set Up Main Loop

```
// 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;
```

INITIALIZATION

Initialize Interpreter

Define Model Inputs

Set Up Main Loop

Declare Variables

Load Model

Resolve Operators

Initialize Interpreter

Allocate Arena

Define Model Inputs

Set Up Main Loop