Machine Learning

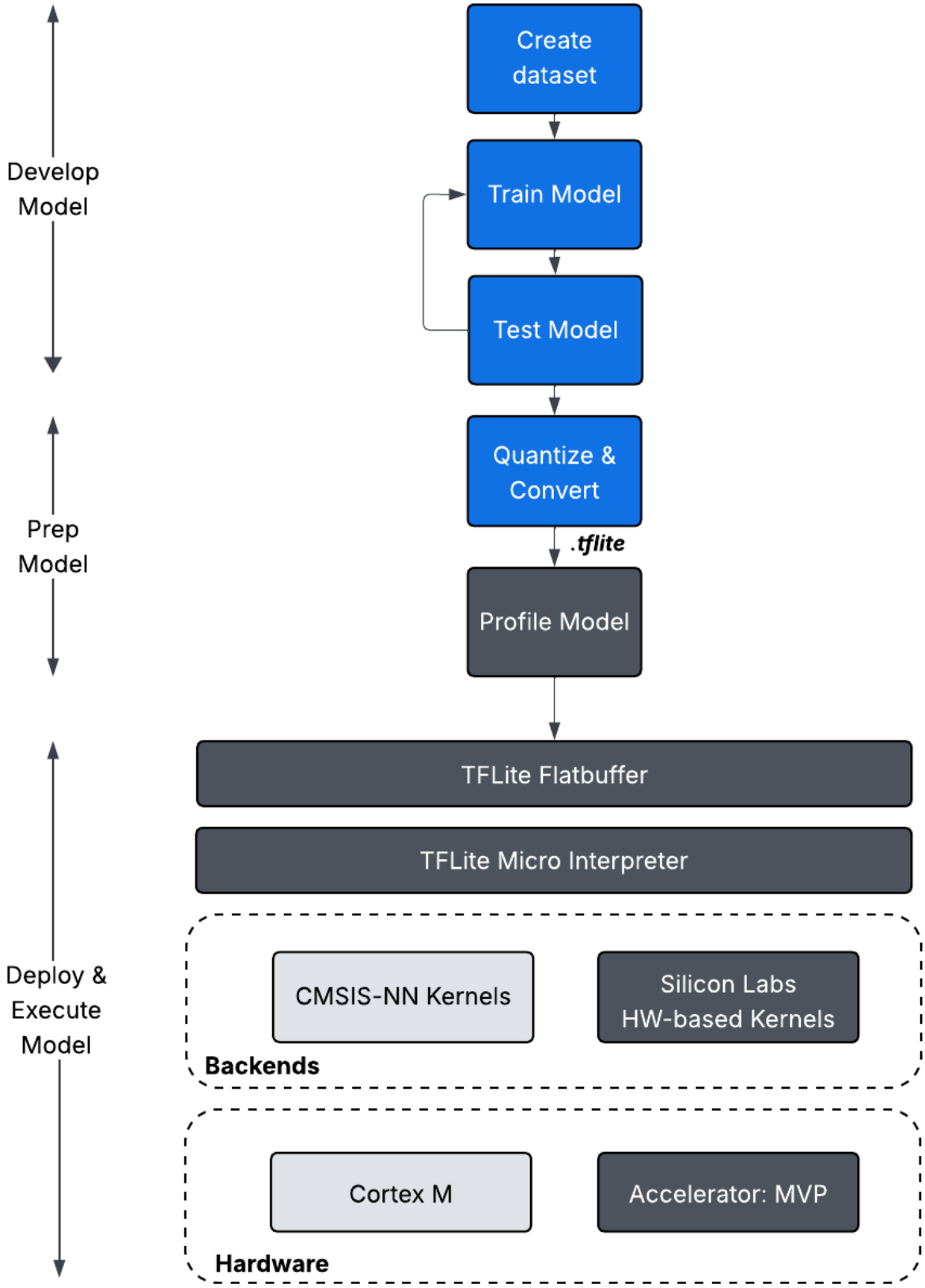
- Developing with Machine Learning
- Release Notes
- Getting Started
  - Getting Started With Machine Learning
  - Tensorflow Lite Micro from Scratch
  - Voice Control Light from Scratch
  - Voice Control Light Demo
- Fundamentals
  - Machine Learning Fundamentals
  - MVP Accelerator
- Developers Guide
  - Machine Learning Developers Guide
  - Add Machine Learning to a New or Existing Project
  - Update or Replace a .tflite File
  - Developing a Model
  - AI/ML Extension Setup
  - Flatbuffer Converter Tool
  - I2S Configuration for SiWx917
  - ML Profiler
- Machine Learning API Reference
  - Microphone I2S Driver for Machine Learning
  - Model Specific Functions
  - Model Specific Variables and Constants
  - IMU - Inertial Measurement Unit
    - IMU Fusion
      - sl_imu_sensor_fusion
    - Vector and Matrix Math
    - Direction Cosine Matrix
  - Audio Feature Generator
  - Image Feature Generator
  - Microphone
  - TensorFlow Lite Micro Debug
  - TensorFlow Lite Micro Init
  - Machine Learning API Reference

- Math Library API Reference
  - Vector functions
  - Matrix functions
  - Utility functions
  - MVP Math Library API
- Additional Topics
  - Sample Applications
    - Overview
    - Image Classifier
  - Third Party Tooling and Solutions

Developing with Machine Learning

Developing with Silicon Labs Machine Learning (AI/ML)

Machine learning is a subset of artificial intelligence (AI) that enables systems to learn from data and improve their performance without being explicitly programmed. It involves algorithms that identify patterns and make decisions based on input data. Deep learning, a specialized branch of machine learning, uses neural networks with many layers to model complex patterns and achieve high accuracy in tasks like image and speech recognition. Together, these technologies power many modern applications, from recommendation systems to autonomous vehicles. As part of the broader AI field, machine learning and deep learning are key to creating systems that can adapt, reason, and act intelligently.



Silicon Labs currently supports TensorFlow Lite for Microcontrollers (TFLM) and associated software as an extension to SiSDK.

The content on these pages is intended for those who want to experiment with or are already developing a Machine Learning application using Silicon Labs technology.

For Silicon Labs Machine Learning product information: See the product pages on silabs.com.

For background on Machine Learning: The Fundamentals section is a good place to start.

To get started with development: See the Getting Started section to get started working with example applications.

If you are already in development: See the Developer's Guide.

If you are using the Machine Learning SDK version 1.3.x or earlier: See the AI/ML Extension Setup guide to transition from v1.x to v2.x of the AI/ML SDK. The earlier versions were supported as a software component of SiSDK, the stack has been moved into an extension now.

For Silicon Labs AI/ML Extension Source Code: See the AI/ML Extension link to GitHub.

Machine Learning

Silicon Labs ML SDK Version 2.2.1 - Release Notes (Mar 25, 2026)

The Silicon Labs ML SDK is provided as an extension to the Simplicity SDK. It enables AI/ML development on Series 2 (EFR and SiWG917) devices using the Tensorflow Lite for Microcontrollers (TFLM) framework.

Click here for earlier releases.

Release Summary

Key Features | API Changes | Bug Fixes | Chip Enablement

Key Features

Added in 2.2.1

Improved calculation of CPU clocks utilization and inference time in ML Model Profiler.

Added in 2.2.0

- New ML Model Profiler helps developers understand the execution performance of their ML models on the target device and within existing applications. It is available in 2 formats:
  - ML Model Profiler Tool: run `.tflite` model on Series 2 devices to analyze performance before integration.
  - ML Model Profiler Component: integrate profiling directly within an existing application to capture model performance during normal execution.
- New sample applications are available for Profiling ML Models, Data Capture from IMU, and Device Tampering Detection.
- Changed sample application names.
- Demos for more EFR and SiWG917 boards.

API Changes

None.

Bug Fixes

Fixed in 2.2.1

Fixed firmware built with debug symbols instead of a release build.

Fixed in 2.2.0

- Blink app behaves differently when running on Series 2, compared to SiWG917.
- Blink and Voice Control applications now support BRD4338a and similar SiWG917 radio boards.
- Flatbuffer converter tool documentation updated to reflect latest changes.
- Misleading error on macOS for applications built using software optimizations for ML models.
- Removed invalid link to I2S pin configuration for SiWG917 voice-based applications.

Chip Enablement

Added in 2.2.1

None.

Added in 2.2.0

- EFR32MG24B310F1536IM48
- EFR32MG24B210F1536IM48
- EFR32MG24B220F1536IM48
- EFR32MG26B510F3200IM68
- EFR32MG26B410F3200IM48
- EFR32MG26B420F3200IM48
- EFR32MG26B510F3200IL136
- EFR32MG26B410F3200IM68
- EFR32MG26B420F3200IM68
- EFR32MG26B510F3200IM48
- EFR32ZG28B312F1024IM48
- EFR32ZG28B312F1024IM68
- EFR32ZG28B322F1024IM68
- EFR32ZG28B312F1024IM68
- SiWG917M111MGTBA
- SiWG917M141XGTBA
- SiWG917Y111MGNBA
- SiWG917Y111MGAB4
- SiWG917Y121MGNB4

Key Features

New Features | Enhancements | Removed Features | Deprecated Features

> Note: See Feature Matrix for a list of any applicable APIs, examples, software variants, modes, hardware, and host interfaces for each feature.

New Features

Added in 2.2.1

Improved calculation of CPU clocks utilization and inference time in ML Model Profiler.

Added in 2.2.0

- New tool, ML Model Profiler, to profile `.tflite` models. For details, see documentation here. This tool currently only supports EFR devices from the Series 2 family. SiWG917 support is forthcoming.
- New component, ML Model Profiler, to enable profiling of `.tflite` models in existing customer applications. Adding the "ML Model Profiler" component to an application or the `ml-profiler` component to the `.slcp` file of an application will enable this feature. This is not be confused with the ML Model Profiler Tool. This component will eventually be a part of the tool as well. The purpose of keeping this component separate is to allow flexibility to profile ML models in user's existing applications alongside other stacks.
- New sample applications:
  - AI/ML - ML Model Profiler Firmware: Profiling ML Models built for Tensorflow Lite for Microcontrollers.
  - AI/ML - SoC Anomaly Detection for EFR32: Device Tampering Detection by using IMU data and time-series analysis of that data.

- AI/ML - SoC Data Capture for EFR32 Baremetal: Data Capture from IMU into a CSV file. This app requires JLink on the user's workstation.

Enhancements

Added in 2.2.1

None.

Added in 2.2.0

- Changed sample app names from `ml_<app_name>_<platform>` pattern to `aiml_[sub_technology_]<soc|rcp|ncp|host>_<app_name>_<platform>_<os|baremetal>`. This was done to allow quicker segregation of apps based on their capabilities.
- Pre-built demo binaries are now available for more EFR and SiWG917 boards. Newly added demos are:
  - BRD2601B
    - AI/ML - SoC Anomaly Detection EFR32 Baremetal
    - AI/ML - SoC Data Capture EFR32 Baremetal
    - AI/ML - SoC Profiler Firmware EFR32 Baremetal
  - BRD2608A
    - AI/ML - SoC Anomaly Detection EFR32 Baremetal
    - AI/ML - SoC Data Capture EFR32 Baremetal
    - AI/ML - SoC Profiler Firmware EFR32 Baremetal
  - BRD4186C
    - AI/ML - SoC Blink EFR32 Baremetal
    - AI/ML - SoC Model Profiler EFR32 Baremetal
  - BRD4187C
    - AI/ML - SoC Blink EFR32 Baremetal
    - AI/ML - SoC Model Profiler EFR32 Baremetal
  - BRD4338A
    - AI/ML - SoC Audio Classifier SiWG917 Baremetal
    - AI/ML - SoC Blink SiWG917 Baremetal
    - AI/ML - SoC Model Profiler SiWG917 Baremetal
    - AI/ML - SoC Voice Control Light SiWG917 Baremetal
  - BRD4339A
    - AI/ML - SoC Model Profiler SiWG917 Baremetal
  - BRD4340B
    - AI/ML - SoC Model Profiler SiWG917 Baremetal
  - BRD4342A
    - AI/ML - SoC Model Profiler SiWG917 Baremetal
  - BRD4343A
    - AI/ML - SoC Model Profiler SiWG917 Baremetal
  - BRD4343B
    - AI/ML - SoC Model Profiler SiWG917 Baremetal
  - BRD4343Q
    - AI/ML - SoC Model Profiler SiWG917 Baremetal

Removed Features

None.

Deprecated Features

None.

API Changes

New APIs | Modified APIs | Removed APIs | Deprecated APIs

New APIs

None.

Modified APIs

None.

Removed APIs

None.

Deprecated APIs

None.

Bug Fixes

Fixed in 2.2.1

None.

Fixed in 2.2.0

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1567613 1463269	Blink app behaves differently when running on Series 2, compared to SiWG917. The LED has a baseline brightness on SiWG917 while blinking but it turns off completely on Series 2. This is due to the different LED driver implementations on the two chip families.	None.	• SiWG917M111MGTBA • BRD4338A • SoC
1479985	Blink and Voice Control applications now support BRD4338a and similar SiWG917 radio boards.	None.	• SiWG917M111MGTBA • BRD4338A • SoC
1438989	Flatbuffer converter tool documentation updated to reflect latest changes.	None.	• Documentation
1479970	Misleading error on macOS for applications built using software optimizations for ML models.	None.	• Silicon Labs ML SDK
1497650	Removed invalid link to I2S pin configuration for SiWG917 voice-based applications.	None.	• SiWG917M111MGTBA • SoC

Chip Enablement

Added in 2.2.1

None.

Added in 2.2.0

Chip Family	OPNs / Boards / OPN Combinations	Supported Software Variants (if applicable)	Supported Modes	Supported Host Interfaces
Chip	• EFR32MG24B310F1536IM48 • EFR32MG24B210F1536IM48 • EFR32MG24B220F1536IM48 • EFR32MG26B510F3200IM68 • EFR32MG26B410F3200IM48 • EFR32MG26B420F3200IM48 • EFR32MG26B510F3200IL136 • EFR32MG26B410F3200IM68 • EFR32MG26B420F3200IM68 • EFR32MG26B510F3200IM48 • EFR32ZG28B312F1024IM48 • EFR32ZG28B312F1024IM68 • EFR32ZG28B322F1024IM68 • EFR32ZG28B312F1024IM68 • SiWG917M111MGTBA • SiWG917M141XGTBA • SiWG917Y111MGNBA • SiWG917Y111MGAB4 • SiWG917Y121MGNB4	Standard	SoC	• UART • SPI • I2S • I2C

Application Example Changes

New Examples | Modified Examples | Removed Examples | Deprecated Examples

New Examples

Added in 2.2.1

None.

Added in 2.2.0

Example Name	Description	Supported Software Variants (if applicable)	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
AI/ML - ML Model Profiler Firmware	Profiling ML Models built for Tensorflow Lite for Microcontrollers.	Standard	SoC	• OPN: EFR32xG2x, SiWG917 • Boards: BRD2601B, BRD2608A, BRD4186C, BRD4187C, BRD4338A, BRD4339A, BRD4340B, BRD4342A, BRD4343A, BRD4343B, BRD4343Q • External Hosts: N/A	

Example Name	Description	Supported Software Variants (if applicable)	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
AI/ML - SoC Anomaly Detection for EFR32	Device Tampering Detection by using IMU data and time-series analysis of that data.	Standard	SoC	• OPN: EFR32xG2x • Boards: BRD2601B, BRD2608A • External Hosts: N/A	
AI/ML - SoC Data Capture for EFR32 Baremetal	Data Capture from IMU into a CSV file. This app requires JLink on the user's workstation.	Standard	SoC	• OPN: EFR32xG2x • Boards: BRD2601B, BRD2608A • External Hosts: N/A	

Modified Examples

None.

Removed Examples

None.

Deprecated Examples

None.

Known Issues and Limitations

Refer to the ML Model Profiler Troubleshooting section for known issues and their solutions.

Using This Release

What's in the Release? | Compatible Software | Installation and Use | Help and Feedback

What's in the Release?

Improved calculation of CPU clocks utilization and inference time in ML Model Profiler.

Compatible Software

Software	Compatible Version or Variant
Software Development Kit (SDK)	• Simplicity SDK: 2025.12.2 • WiSeConnect SDK: 4.0.1

Installation and Use

To upgrade your existing software with this release, update Simplicity Studio to the latest, Simplicity SDK to v2025.12.1, WiSeConnect SDK to v4.0.1, and ML SDK to v2.2.1 from Studio installation manager, or download the SDKs from the respective links listed in Compatible Software section above. To update the ML SDK, please refer to AI/ML SDK Setup guide.

To run your first demo, see our Getting Started

To kick start your development, see our Developer's Guide

For information about Secure Vault Integration, see Secure Vault.

To review Security and Software Advisory notifications and manage your notification preferences:

1. Go to https://community.silabs.com/.
2. Log in with your account credentials.
3. Click your profile icon in the upper-right corner of the page.
4. Select Notifications from the dropdown menu.
5. In the Notifications section, go to the My Product Notifications tab to review historical Security and Software Advisory notifications
6. To manage your preferences, use the Manage Notifications tab to customize which product updates and advisories you receive.

To learn more about the software in this release, dive into our online documentation

Help and Feedback

- Contact Silicon Labs Support.
- To use our Ask AI tool to get answers, see the search field at the top of this page.

> Note: Ask AI is experimental.

- Get help from our developer community.

Feature Matrix

Supported Features | Unsupported Features

Supported Features

None.

Unsupported Features

- SiWG917 does not use the built-in TensorFlow component directly; instead, support is provided through automatic code generation handled by advanced configurators.
- Series 2 devices do not support new software optimizations. However, due to their architecture, they are still at least as efficient as SiWG917.
- MVP Compiler is not supported on macOS natively, recommend using a linux container environment.

SDK Release and Maintenance Policy

See our SDK Release and Maintenance Policy.

Getting Started

Getting Started with Machine Learning

Introduction

Silicon Labs TensorFlow Lite for Microcontrollers Integration

Silicon Labs provides robust support for TensorFlow Lite for Microcontrollers (TFLM) as an extension to Simplicity Studio SDK (SiSDK), offering developers flexible options for deploying machine learning models on EFx32 and Si91x microcontrollers using Project Configurator for Simplicity Studio. This guide covers how TensorFlow Lite for Microcontrollers is integrated with the SiSDK using AIML extension for use Silicon Labs' EFx32 and Si91x devices.

TensorFlow Lite for Microcontrollers

TensorFlow is a widely used deep learning framework, with capability for developing and executing neural networks across a variety of platforms. TensorFlow Lite provides an optimized set of tools specifically catered towards machine learning for mobile and embedded devices.

TensorFlow Lite for Microcontrollers (TFLM) specifically provides a C++ library for running machine learning models in embedded environments with tight memory constraints. Silicon Labs provides tools and support for loading and running pre-trained models that are compatible with this library.

AIML Extension

Installing the AI/ML Extension for Silicon Labs Simplicity Studio

For detailed instructions on installing the AI/ML extension, refer to the AI/ML Extension Installation Guide. This extension empowers developers to integrate machine learning capabilities into their Silicon Labs-based projects.

Training and Quantizing a Model

To perform neural network inference on a Silicon Labs device, one first needs a trained model in the TFLite Flatbuffer format. There are two approaches to consider for developers experienced with TensorFlow:

- Following published tutorials for training neural networks using TensorFlow, as outlined in Developing a Model.
- Using Silicon Labs AI/ML partners. See the AI/ML Partners section on Silicon Labs Machine Learning in IoT page for more information.
- Using the Silicon Labs Machine Learning Toolkit, a Python reference package that combines and simplifies all the necessary TensorFlow training steps.

Developing an Inference Application Using Simplicity Studio, SiSDK and AIML extension

After you have a trained and quantized TFLite model, the next step is to set up the TFLM libraries to run inference on a Silicon Labs device.

Project Configurator Setup

The Project Configurator includes TFLM libraries as software components. These software components may be added to any existing project. They are described in the SDK Component Overview. The core components needed for any machine learning project are as follows:

1. TensorFlow Lite Micro. This is the core software component that pulls in all the TFLM dependencies.
2. A supported TFLM kernel implementation. A kernel is a specific hardware/platform implementation of a low level operation used by TensorFlow. Kernel selection can drastically change the performance and computation time of a neural network. By default, the best kernel implementation for the given device is selected automatically.
3. A supported TFLM debug logger. The Project Configurator defaults to using the I/O Stream implementation of the logger. To disable logging entirely, add the Debug Logging Disabled component.

In addition to required TFLM components, software components for obtaining and pre-processing sensor data can be added to the project. As an example for audio applications, Silicon Labs provides an audio feature generator component that includes powerful DSP features to filter and extract features from raw audio data, to be used as a frontend for microphone-based applications. Silicon Labs developed drivers for microphones, accelerometers, and other sensors provide a simple interface for obtaining sensor data to feed to a network.

Tensorflow Lite Micro from Scratch

Machine Learning on Silicon Labs Devices from Scratch

This guide assumes familiarity with the content of the Getting Started Guide. It provides details of working with the TensorFlow API, as an alternative to the automatic initialization provided by Silicon Labs as described in the guide on Adding Machine Learning to a New Project.

Model Inclusion

With the TensorFlow Lite Micro component added in the Project Configurator, the next step is to load the model file into the project. To do this, copy the `.tflite` model file into the `config/tflite` directory of the project. The project configurator provides a tool that will automatically convert `.tflite` files into a `sl_tflite_micro_model` source and header files. The full documentation for this tool is available at Flatbuffer Converter Tool.

To do this step manually, a C array can be created from the `.tflite` using a tool such as `xxd`.

TFLM Initialization and Inference

To instantiate and use the TensorFlow APIs, follow the steps below to add TFLM functionality to the application layer of a project. This guide closely follows the TFLM Getting Started Guide, and is adapted for use with Silicon Labs' projects.

A special note needs to be taken regarding the operations used by TensorFlow. Operations are specific types of computations executed by a layer in the neural network. All operations may be included in a project at once, but doing this may increase the binary size dramatically (>100kB). The more efficient option is to only include the operations necessary to run a specific model. Both options are described in the steps below.

0. Disable the automatic initialization provided by the TensorFlow Lite Micro component

To manually set up TensorFlow, first disable automatic initialization of the model by disabling Automatically initialize model in the configuration header for the TensorFlow Lite Micro component (`sl_tflite_micro_config.h`).

1. Include the library headers

If using a custom, limited set of operations (recommended):

```
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/tflite_bridge/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
```

If using all operations:

```
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
```

2. Include the model header

Because the `autogen/` folder is always included in the project include paths, an imported model may be generically included in any project source file with:

```
#include "sl_tflite_micro_model.h"
```

If not using the Flatbuffer Converter Tool, include the file containing your model definition instead.

3. Define a memory arena

TensorFlow requires a memory arena for runtime storage of input, output, and intermediate arrays. This arena should be statically allocated, and the size of the arena depends on the model used. It is recommended to start with a large arena size during prototyping.

```
constexpr int tensor_arena_size = 10 * 1024;
uint8_t tensor_arena[tensor_arena_size];
```

> Note: After prototyping, it is recommended to manually tune the memory arena size to the model used. After the model is finalized, start with a large arena size and incrementally decrease it until interpreter allocation (described below) fails.

4. Set up logging

This should be performed even if the `Debug Logging Disabled` component is used. It is recommended to instantiate this statically and call the logger init functions during the `app_init()` sequence in `app.cpp`.

```
static tflite::MicroErrorReporter micro_error_reporter;
tflite::ErrorReporter* error_reporter = &amp;micro_error_reporter;
```

5. Load the model

Continuing during the `app_init()` sequence, the next step is to load the model into `tflite`:

```
const tflite::Model* model = ::tflite::GetModel(sl_tflite_model_array);
if (model->version() != TFLITE_SCHEMA_VERSION) {
  TF_LITE_REPORT_ERROR(error_reporter,
      "Model provided is schema version %d not equal "
      "to supported version %d.\n",
      model->version(), TFLITE_SCHEMA_VERSION);
}
```

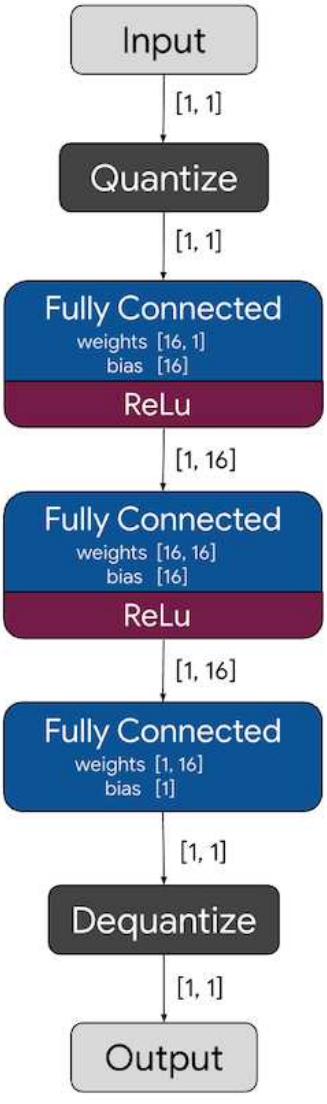
6. Instantiate the operations resolver

If using all operations, this is very straightforward. During `app_init()`, statically instantiate the resolver via:

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

> Note: Loading all operations will result in large increases to the binary size. It is recommended to use a custom set of operations.

If using a custom set of operations, a mutable ops resolver must be configured and initialized. This will vary based on the model and application. To determine the operations utilized in a given `.tflite` file, third party tools such as netron may be used to visualize the network and inspect which operations are in use.



The example below loads the minimal operators required for the TensorFlow hello_world example model. As shown in the Netron visualization, this only requires fully connected layers:

```
#define NUM_OPS 1

static tflite::MicroMutableOpResolver<NUM_OPS> micro_op_resolver;
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
```

If using the Flatbuffer Converter Tool, it generates a C preprocessor macro that automatically sets up the optimal `tflite::MicroMutableOpResolver` for the a flatbuffer:

```
#include "sl_tflite_micro_opcode_resolver.h"

SL_TFLITE_MICRO_OPCODE_RESOLVER(micro_op_resolver, error_reporter);
```

7. Initialize the interpreter

The final step during `app_init()` is to instantiate an interpreter and allocate buffers within the memory arena for the interpreter to use:

```
// static declaration
tflite::MicroInterpreter* interpreter = nullptr;

// initialization in app_init
tflite::MicroInterpreter interpreter(model, micro_op_resolver, tensor_arena,
                                     tensor_arena_size, error_reporter);
interpreter = &interpreter_struct;
TfLiteStatus allocate_status = interpreter.AllocateTensors();
if (allocate_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
  return;
}
```

The allocation will fail if the arena is too small to fit all the operations and buffers required by the model. Adjust the `tensor_arena_size` accordingly to resolve the issue.

8. Run the model

For default behavior in bare metal application, it is recommended to run the model during `app_process_action()` in `app.cpp` in order for periodic inferences to occur during the standard event loop. Running the model involves three stages:

1. Sensor data is pre-processed (if necessary) and then is provided as input to the interpreter.

   ```
   TfLiteTensor* input = interpreter.input(0);
   // stores 0.0 to the input tensor of the model
   input->data.f[0] = 0.;
   ```

   It is important to match the shape of the incoming sensor data to the shape expected by the model. This can optionally be queried by checking properties defined in the `input` struct. An example of this for the hello_world example is shown below:

```
TfLiteTensor* input = interpreter->input(0);
if ((input->dims->size != 1) || (input->type != kTfLiteFloat32)) {
  TF_LITE_REPORT_ERROR(error_reporter,
                       "Bad input tensor parameters in model");
  return;
}
```

2. The interpreter is then invoked to run all layers of the model.

```
TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on x_val: %f\n",
                       static_cast<double>(x_val));
  return;
}
```

3. The output prediction is read from the interpreter.

```
TfLiteTensor* output = interpreter->output(0);
// Obtain the output value from the tensor
float value = output->data.f[0];
```

At this point, application-dependent behavior based on the output prediction should be performed. The application will run inference on each iteration of `app_process_action()`.

Full Code Snippet

After following the steps above and choosing to use the mutable ops resolver, the resulting `app.cpp` now appears as follows:

```
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"

#include "sl_tflite_micro_model.h"

#define NUM_OPS 1

constexpr int tensor_arena_size = 10 * 1024;
uint8_t tensor_arena[tensor_arena_size];

tflite::MicroInterpreter* interpreter = nullptr;

/**************************************************************************//**
 * Initialize application.
 *****************************************************************************/
void app_init(void)
{
  static tflite::MicroErrorReporter micro_error_reporter;
  tflite::ErrorReporter* error_reporter = &micro_error_reporter;

  const tflite::Model* 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.\n",
        model->version(), TFLITE_SCHEMA_VERSION);
  }



  static tflite::MicroMutableOpResolver<NUM_OPS> micro_op_resolver;
  if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
     return;
  }

  static tflite::MicroInterpreter interpreter_struct(model, micro_op_resolver, tensor_arena,
                               tensor_arena_size, error_reporter);
  interpreter = &interpreter_struct;
  TfLiteStatus allocate_status = interpreter.AllocateTensors();
  if (allocate_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
    return;
  }
}

/**************************************************************************//**
 * App ticking function.
 *****************************************************************************/
void app_process_action(void)
{
  // stores 0.0 to the input tensor of the model
  TfLiteTensor* input = interpreter->input(0);
  input->data.f[0] = 0.;

  TfLiteStatus invoke_status = interpreter->Invoke();
  if (invoke_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on x_val: %f\n",
                         static_cast<double>(x_val));
    return;
  }

  TfLiteTensor* output = interpreter->output(0);
  float value = output->data.f[0];
}
```

Examples

As described in the Sample Application Overview, examples developed by the TensorFlow team demonstrating the `hello_world` example described in this guide, as well as a simple speech recognition example `micro_speech`, are included in the Simplicity SDK.

Note that the `micro_speech` example demonstrates use of the `MicroMutableOpResolver` to only load required operations.

Voice Control Light from Scratch

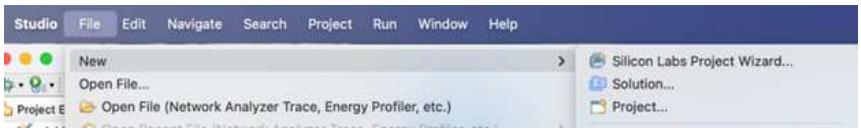
End-to-End Steps to Create a Voice-Controlled Light ML Application from Scratch

This guide details the process of creating a voice-controlled light application using TensorFlow Lite Micro (TFLM) on an EFR32xG24 Development Kit. This example uses the `keyword_spotting_on_off_v3.tflite` model (recommended) for "on" and "off" keyword detection. For more information on model creation, see the MLTK tutorial.

- Hardware: EFR32xG24 Dev Kit Board (BRD2601B Rev A01)
- Software: Simplicity Studio (SiSDK 2024.12 or later)

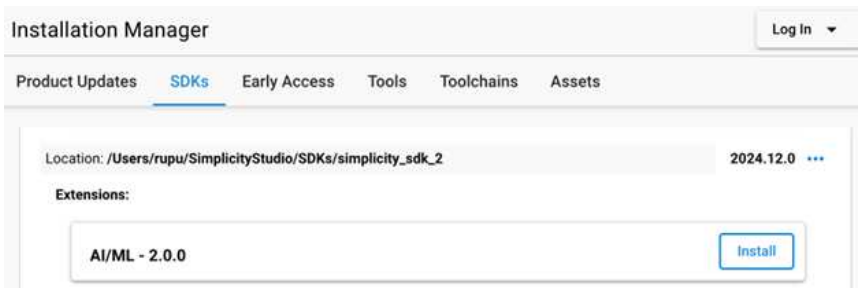
1. Install AI/ML Extension

1. Click Install on the top bar.



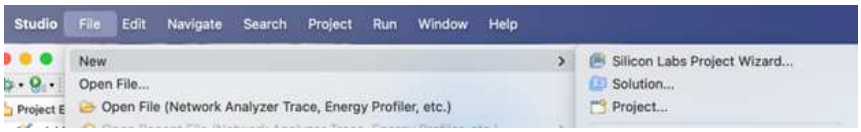
2. Click Manage Installed Packages.

3. Under SDKs, install the latest version of the AI/ML extension (available from SiSDK 2024.12 onwards).

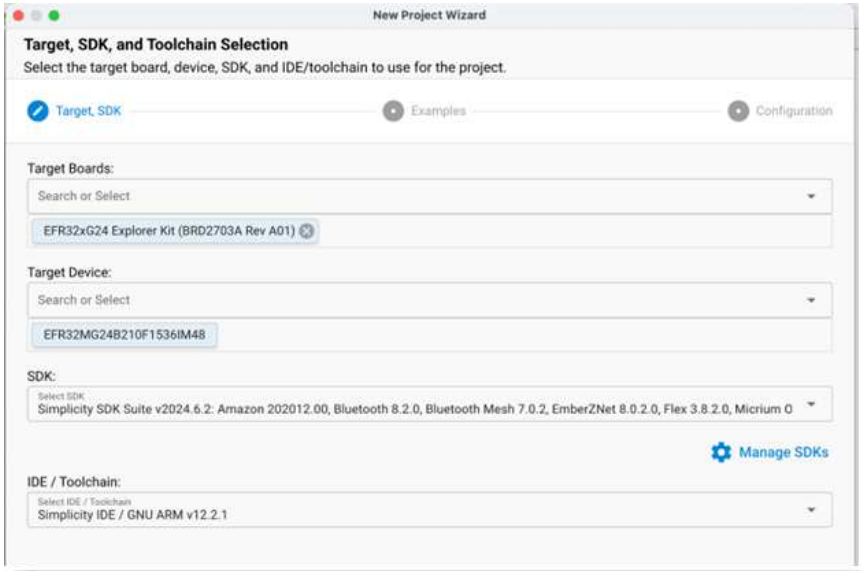


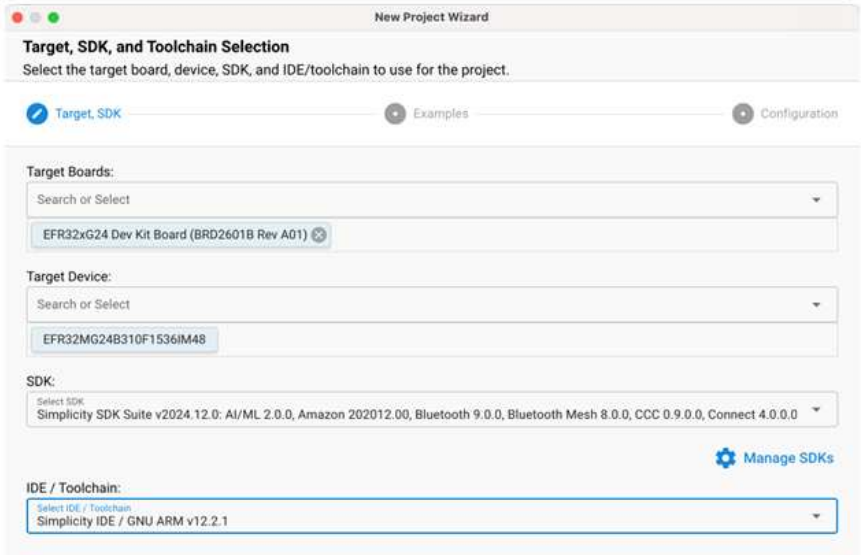
2. Start a New Simplicity Project

1. From the File menu, select New > Silicon Labs Project Wizard.

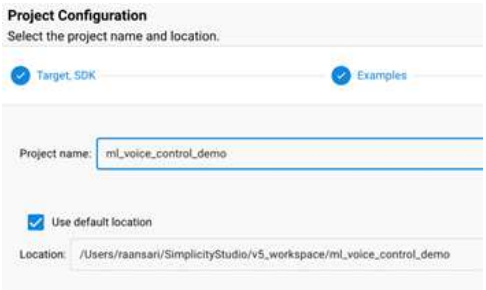


2. Select the target board (EFR32xG24 Development Kit), SDK (Simplicity SDK v2024.12.0 or later), and IDE/Toolchain (e.g., GNU ARM v12.2.1). Click Next.



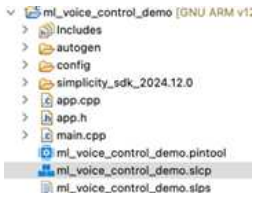


3. Choose Empty C++ Project. Click Next.
4. Give your project a name and click Finish.



3. Add Machine Learning Software Component

1. Open your project file (the one with the `.slcp` extension).



2. Under Software Components, search for "aiml".

3. Enable the AI/ML extension by clicking Enable Extension.

4. Expand AI/ML >> Machine Learning >> TensorFlow. Select TensorFlow Lite Micro and click Install.

5. You will be prompted to select additional components:

- Debug Logging: Choose Debug Logging using IO Stream (if needed) or Debug Logging Disabled. Click Install.
- Kernels: Select MVPv1 Accelerated Kernels. Click Install.

4. Configure the TFLM Component

1. Click Configure in the TensorFlow Lite Micro Software Component.

2. Set the Arena Size. For this example, enter "-1". This tells the system to dynamically determine the optimal arena size at runtime.

5. Include and Convert the Model

1. Create a `tflite` directory inside your project's `config` directory (optional but recommended).
2. Drag and drop the `keyword_spotting_on_off_v2.tflite` file into the `config/tflite` directory (or directly into `config` if you skipped creating the subdirectory).
3. The framework will automatically convert the `.tflite` file into a C array (`sl_tflite_micro_model.c` in the `autogen` directory). The TFLM interpreter is also initialized automatically.

6. Profile the Model (Optional)

Model profiling can be helpful for optimization. For advanced users who wish to analyze model performance, the MLTK Model Profiler Utility can be used. This is not strictly required for this basic example.

7. Run the Model

1. Include TensorFlow Init API: Add the necessary code to initialize the TFLM interpreter.
2. Provide Input Data:
   - Get a pointer to the input tensor: `TfLiteTensor* input = sl_tflite_micro_get_input_tensor();` .
   - Load your input data (microphone audio quantized to int8) into the input tensor: `input->data.int8f[0] = <input array from microphone quantized to int8>;` (See the example code for audio feature generation).
3. Run Inference:
   - Invoke the interpreter: `TfLiteStatus invoke_status = sl_tflite_micro_get_interpreter()->Invoke();`
   - Check for errors: `if (invoke_status != kTfLiteOk) { TF_LITE_REPORT_ERROR(sl_tflite_micro_get_error_reporter(), "bad input tensor parameters in model"); }`
4. Read Output:
   - Get a pointer to the output tensor: `TfLiteTensor* output = sl_tflite_micro_get_output_tensor();`
   - Access the output data: `int8_t value = output->data.int8_tf[0];`

8. Implement Post-Processing

1. Develop an Algorithm: Create an algorithm to interpret the model's output (e.g., the `int8_t value`) and determine whether "on" or "off" was spoken.
2. Trigger Events: Based on the post-processed output, trigger actions like controlling the LED. Refer to the `voice_control_light.cc` , `recognize_commands.cc` , and `recognize_commands.h` files in the aiml-extension examples for guidance on implementing this logic, including LED control and command recognition. You will need to add components for the microphone, audio processing, and LED control to your project.

Voice Control Light Demo

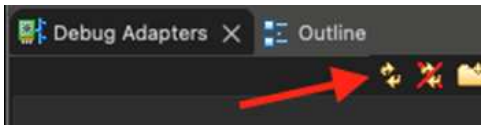
Voice-Control Light Demo - Quick Start Guide

This guide provides instructions for quickly demonstrating the Voice-Control Light application using pre-built binaries. This demo allows you to control an LED on an EFR32xG24 Dev Kit (BRD2601B Rev A01) by speaking "on" or "off" into a microphone.

- Hardware: EFR32xG24 Dev Kit Board (BRD2601B Rev A01)
- Software: Simplicity Studio (SiSDK 2024.12 or later)

Steps

1. Open Simplicity Studio: Launch Simplicity Studio (using the rocket button in the top right corner).
2. Connect your Device: Connect your EFR32xG24 Dev Kit to your computer. Wait 5-10 seconds for the device to be recognized by Simplicity Studio.
    - Troubleshooting: If your device isn't recognized, click the "refresh" button in the Debug Adapters sub-window (usually located at the bottom left).



3. Select your Device: Choose your connected device from the Connected Devices dropdown and click Start.
4. Navigate to the Demo: Go to Example Projects & Demos. In the left-hand context menu, scroll down to Capability and select Machine Learning.
5. Run the Demo: Find the Voice Control Light demo and click Run. This will flash the pre-built binary onto your board.

Fundamentals

Machine Learning Fundamentals

TensorFlow Lite for Microcontrollers is a framework that provides a set of tools for running neural network inference on microcontrollers. It contains a wide selection of kernel operators with good support for 8-bit integer quantized networks. The framework is limited to model inference and does not support training. For information about how to train a neural network, see the Silicon Labs Machine Learning Toolkit (MLTK).

Silicon Labs provides an integration of TensorFlow Lite for Microcontrollers with the Simplicity SDK. See the Getting Started Guides for step-by-step instructions on how to make use of Machine Learning in your project.

SDK Component Overview

The software components required to use TensorFlow Lite for Microcontrollers can be found under Machine Learning > TensorFlow in the software component browser UI in the Simplicity Studio project configurator.

TensorFlow Lite Micro

This component contains the full TensorFlow Lite for Microcontrollers framework, and automatically pulls in the most optimal implementation of kernels for the device selected for the project by default. To use TensorFlow Lite Micro, this component is the only one that needs to be explicitly installed. It is however possible to manually install different kernel implementations if so desired, for instance to compare inference performance or code size, and to manually install a different debug logging implementation. By default, the TensorFlow Lite Micro component makes use of the Flatbuffer Converter Tool to convert a `.tflite` file into a C array and to initialize this neural network model automatically. See the section on automatic initialization for more details.

Kernel Implementations

Reference Kernels

This component provides unoptimized software implementations of all kernels. This is a default implementation that is designed to be easy to read and can run on any platform. As a result, these kernels may run more slowly than an optimal implementation.

CMSIS-NN Optimized Kernels

Some kernels have implementations that have been optimized for certain CPU architectures using the CMSIS-NN library. Using these kernels when available can improve inference performance significantly. By enabling this component, the available optimized kernel implementations are added to the project, replacing the corresponding reference kernel implementations. The remaining kernels fall back to using the reference implementations by depending on the reference kernel component.

MVP Accelerated Kernels

Some kernels have implementations optimized for the MVP accelerator available on select Silicon Labs parts. Using these kernels will improve inference performance. By enabling this component, the available accelerated kernel implementations are added to the project, replacing the corresponding optimized or reference kernel implementations. The remaining kernels fall back to use the optimized or reference implementations by depending on the corresponding components. See more details about the accelerator to learn what kernels are supported, and what constraints apply.

Debug Logging using I/O Stream / Disabled

Debug logging is used in TensorFlow to display debug and error information. Additionally, it can be used to display inference results. Debug logging is enabled by default, with an implementation that uses I/O Stream to print over UART to the virtual COM port on development kits (VCOM). Logging can be disabled by ensuring that the component "Debug Logging Disabled" is included in the project.

TensorFlow Third Party Dependencies

A specific version of the CMSIS-NN library is used as with TensorFlow Lite for Microcontrollers to optimize certain kernels. This library is included in the project together with TensorFlow Lite for Microcontrollers. TensorFlow depends on a bleeding-edge version of CMSIS-NN, while the rest of the Simplicity SDK uses a stable CMSIS release. It is strongly recommended to avoid using functions from the Simplicity SDK version of CMSIS-DSP and CMSIS-NN elsewhere in the project and instead use the version bundled with TensorFlow Lite for Microcontrollers to avoid versioning conflicts between the two.

Audio Feature Generator

The audio feature generator can be used to extract time-frequency features from an audio signal for use with machine learning (ML) audio classification applications. The generated feature array is a mel-scaled spectrogram, representing the frequency information of the signal of a given sample length of audio.

When used together with the Flatbuffer Converter Tool, the audio feature generator by default consumes its configuration settings from the model parameters of the `.tflite` flatbuffer. Such metadata can be added to the flatbuffer by using the Silicon Labs Machine Learning Toolkit. This ensures that the settings used during inference on the embedded device match the settings used during training. If models without such metadata are used, the configuration option "Enable Manual Frontend Configurations" can be enabled, and configuration values set in the configuration header `sl_ml_audio_feature_generation_config.h`.

Automatic Initialization of Default Model

When the TensorFlow Lite Micro component is added to the project, it will by default attempt to automatically initialize a default model using the TFLite Micro Init API. It performs initialization of TensorFlow Lite Micro by creating an opcode resolver and interpreter for the given flatbuffer. In addition, it creates the tensor arena buffer.

The model used by the automatic initialization code comes from the Flatbuffer Converter Tool. If the flatbuffer was produced using the MLTK, it may contain metadata about the necessary tensor arena size. If such information is present, it will be automatically initialized to the correct size. If a non-MLTK flatbuffer is used, the tensor arena size must be configured manually using the configuration file for the TensorFlow Lite Micro component.

If automatic initialization at startup is not desired, this can be turned off using the Automatically initialize model (`SL_TFLITE_MICRO_INTERPRETER_INIT_ENABLE`) configuration option.

Version

The Silicon Labs AI/ML extension incorporates TensorFlow Lite for Microcontrollers version #02414075e7f718a2d0412775fcadbf28fb4cc5aa in `third_party/tflite-micro/`. The core TensorFlow Lite for Microcontrollers offering is unpatched, all additional content for Silicon Labs devices is delivered in the extension's root directory.

Third-party Tools and Partners

Tools

- Netron is a visualization tool for neural networks, compatible with `.tflite` model files. This is useful for viewing the operations used in a model, the sizes of tensors and kernels, etc.

AI/ML Partners

Silicon Labs AI/ML partners provide expertise and platforms for data collection, model development, and training. See the technology partner pages to learn more.

MVP Accelerator

MVP Accelerator

The MVP accelerator is a co-processor designed to perform matrix and vector operations. Using hardware accelerated kernel implementations will reduce neural network inference time, as well as off-load the main processor to allow it to perform other tasks or go to sleep.

Silicon Labs has implemented common neural network operators as programs to be executed on the MVP and integrated these with TensorFlow Lite for Microcontrollers. The MVP has 5 array controllers, each of which can support iterating in 3 independent dimensions. Each dimension is limited to 1024 elements, with a stride between each element of 2047. The limiting factor for most neural network operations is therefore the product of the width and depth dimensions, since this becomes the stride in the height dimension.

All MVP-accelerated operations take signed 8-bit integers as input and output. If the inner dimension of the tensor has even size, each element can contain two int8 values, interpreted as a single complex int8 value by the accelerator. The accelerator can then effectively support 2048 int8 values. If the inner dimension is odd, the accelerator must perform one computation at a time, which reduces performance and limits the dimension size to 1024 int8 values.

The operators listed below will be accelerated using the MVP if tensor sizes allow. If a specific tensor cannot be accelerated, the implementation will automatically fall back to using optimized (CMSIS-NN) or reference kernel implementations at runtime. To maximize the likelihood that an operator is supported by the accelerator, use even-valued numbers of channels when designing the model.

Internally, the MVP accelerator uses 16-bit floating point math, even when taking 8-bit integers as input. This means that there is a slight reduction in accuracy of computations, which may be especially noticeable when performing operations that accumulate many elements.

For more information about the MVP hardware accelerator, see the reference manual for EFR32xG24.

Accelerated TensorFlow operators

Add

TensorFlow operator name: `ADD`

Any tensor size is supported.

FullyConnected (Dense)

TensorFlow operator name: `FULLY_CONNECTED` , `FULLY_CONNECTED_INT8`

Supports tensors where all dimensions are within the 1024 element limit. Also supports larger tensors where the size of the last dimension is decomposable into two factors that are both within the 1024 element limit.

AveragePool2D

TensorFlow operator name: `AVERAGE_POOL_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit.

MaxPool2D

TensorFlow operator name: `MAX_POOL_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit.

Conv2D

TensorFlow operator name: `CONV_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit.

DepthwiseConv2D

TensorFlow operator name: `DEPTHWISE_CONV_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit. Dilation is not supported.

TransposeConv2D

TensorFlow operator name: `TRANSPOSE_CONV_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit. Dilation is not supported.

Suspending Execution While Waiting for Accelerator

The software API for the MVP accelerator is blocking, meaning that any call to the MVP driver will wait for completion before returning from the function call. To save energy, the driver can optionally suspend execution of the main processor while waiting for the accelerator to complete an operation. By default, the main processor busy-waits for the accelerator.

No sleep (0)

When the "No sleep" option is used, the MCU core will busy-wait for the MVP to finish. This is the option which provides the fastest MVP execution time. The "No sleep" option can be used in a bare metal application or an application using a real-time operating system (RTOS).

Enter EM1 (1)

When the "Enter EM1" option is used, the MCU will be put into Energy Mode 1 whenever the driver waits for an MVP program to complete. The "Enter EM1" option is not safe to use in an application using RTOS because it will prevent proper RTOS scheduling.

Yield RTOS thread (2)

When the "Yield RTOS thread" option is used, the task waiting for the MVP program to complete will yield, allowing other tasks in the system to run or potentially let the scheduler put the system into a sleep mode. The "Yield RTOS thread" requires that the application is using an RTOS.

The power mode of the MVP driver can be configured by setting the `SL_MVP_POWER_MODE` configuration option in the `sl_mvp_config.h` configuration header.

Developers Guide

Machine Learning Developers Guide

The Developer's Guide content is organized as follows:

- Add Machine Learning to a New or Existing Project: Describes the process of adding a machine learning model to a new or an existing project.
- Update or Replace a .tflite File: Describes steps to update and replace a machine learning model in your project.
- Developing a Model: Explains the procedure to develop a machine learning model.
- AI/ML Extension Setup: In-depth explanantion of adding AI/ML extension to the setup.
- Flatbuffer Converter Tool: Shows how a flatbuffer is converted to bytes for including into header file.
- I2S Configuration for SiWx917: Provides details on configuring I2S for AI/ML audio apps on SiWx917.

Add Machine Learning to a New or Existing Project

Add Machine Learning to a New or Existing Project

This guide provides details of adding Machine Learning to a new or existing project, making use of the wrapper APIs for TensorFlow Lite for Microcontrollers provided by Silicon Labs for automatic initialization of the TFLM framework.

The guide assumes that a project already exists in the Simplicity Studio workspace and you have installed the AI/ML extension. If you're starting from scratch, you may start with any sample application or the Empty C++ application. Please refer to AI/ML Extension Setup for instructions to install the AI/ML extension. TFLM has a C++ API, so the application code interfacing with it will also need to be written in C++. If you're starting with an application that is predominantly C code, see the section on interfacing with C code for tips on how to structure your project by adding a separate C++ file for the TFLM interface.

Install the TensorFlow Lite Micro Component

1. Open your project file (the one with the `.slcp` extension).

2. Under Software Components, search for "aiml".

3. Enable the AI/ML extension by clicking Enable Extension.

4. Expand: AI/ML > Machine Learning > TensorFlow. Select TensorFlow Lite Micro and click Install.

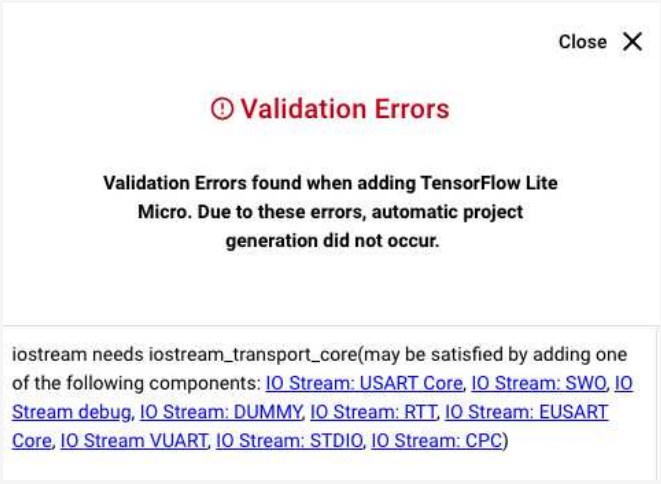
> Note: Skip this step for SiWx917 applications.

5. You will be prompted to select additional components:

> Note: Skip this step for SiWx917 applications.

- Debug Logging: Choose Debug Logging using IO Stream (if needed) or Debug Logging Disabled. Click Install.

> NOTE: If your project didn't already contain an I/O Stream implementation, you may get a dependency validation warning. This is not a problem, but simply means that a choice of I/O Stream backend needs to be made. The USART or EUSART backends are the most common, as these can communicate with a connected PC through the development kit virtual COM port (VCOM).



- IO Stream: You may also need to install EUSART or USART component if you don't see output on your serial console.

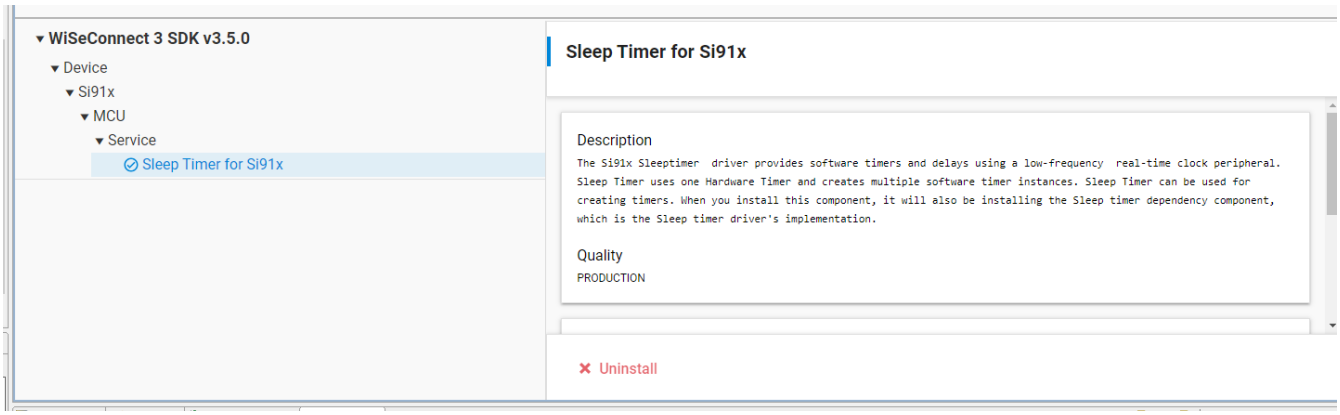
Accept the default suggestion of "vcom" as the instance name, which will automatically configure the pinout to connect to the development board's VCOM lines. If you're using your own hardware, you can set any instance name and configure the pinout manually.

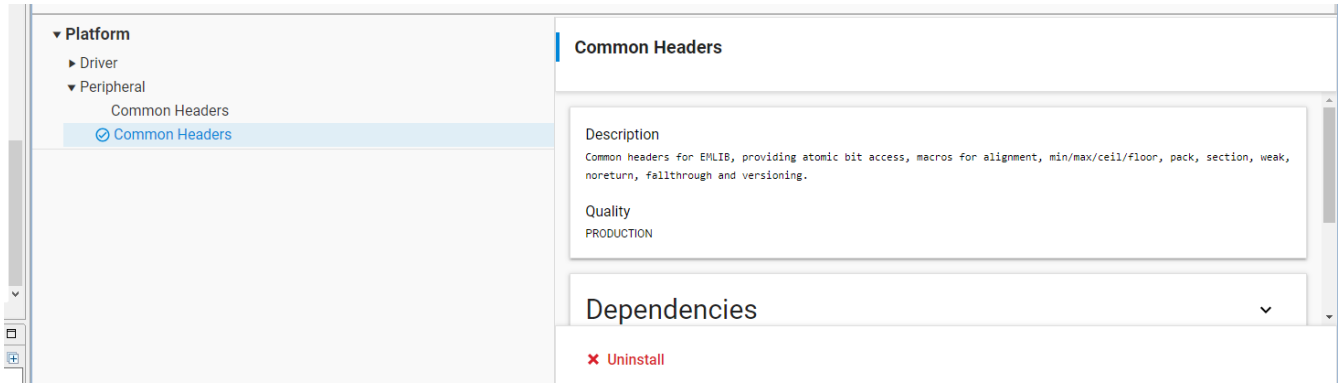
- Kernels: Select MVPv1 Accelerated Kernels. Click Install.

Additional Software Components and C++ Build Settings

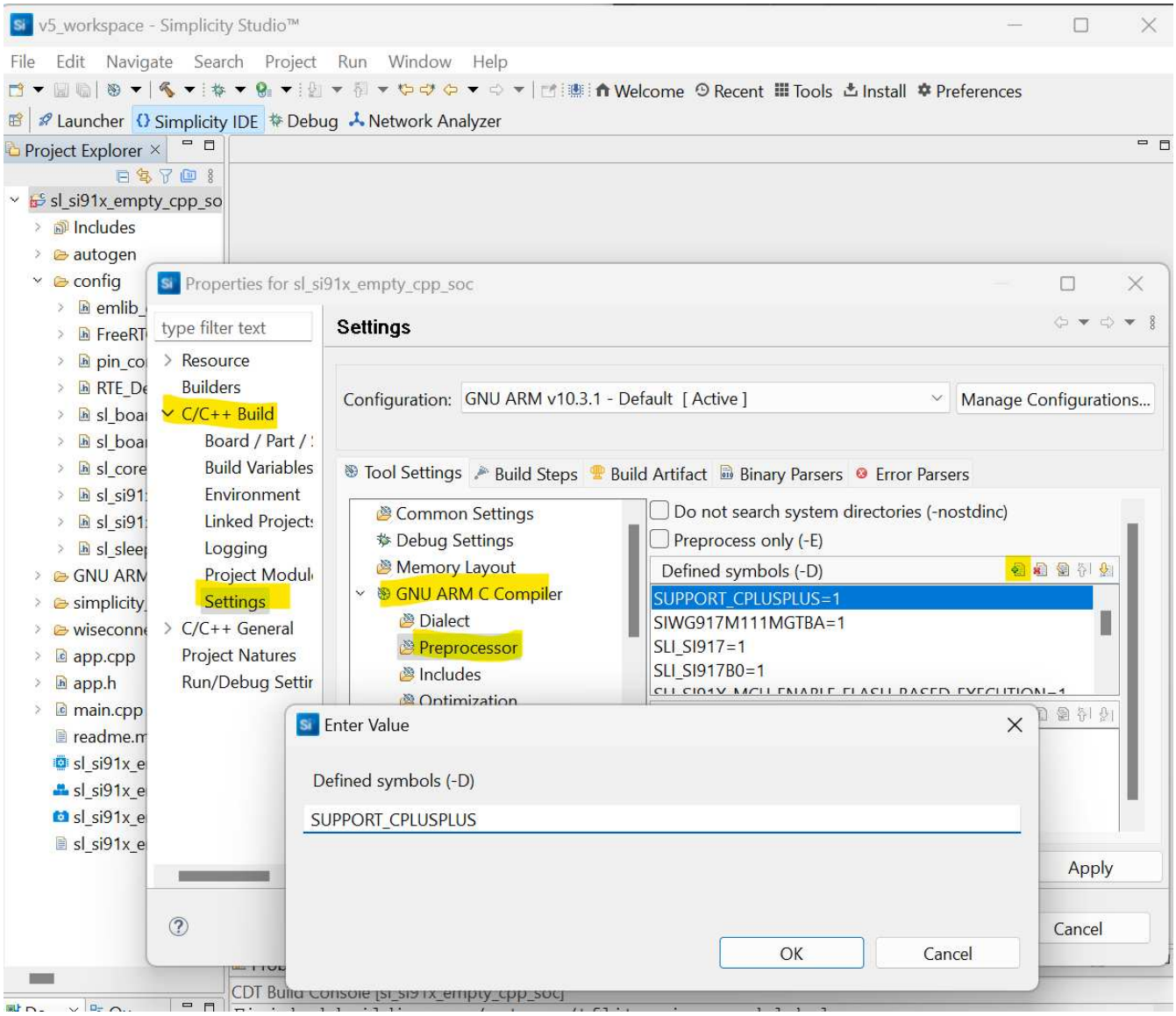
> Note: Skip to Model Inclusion section for Series 2 devices. This section is only applicable for SiWx917 devices.

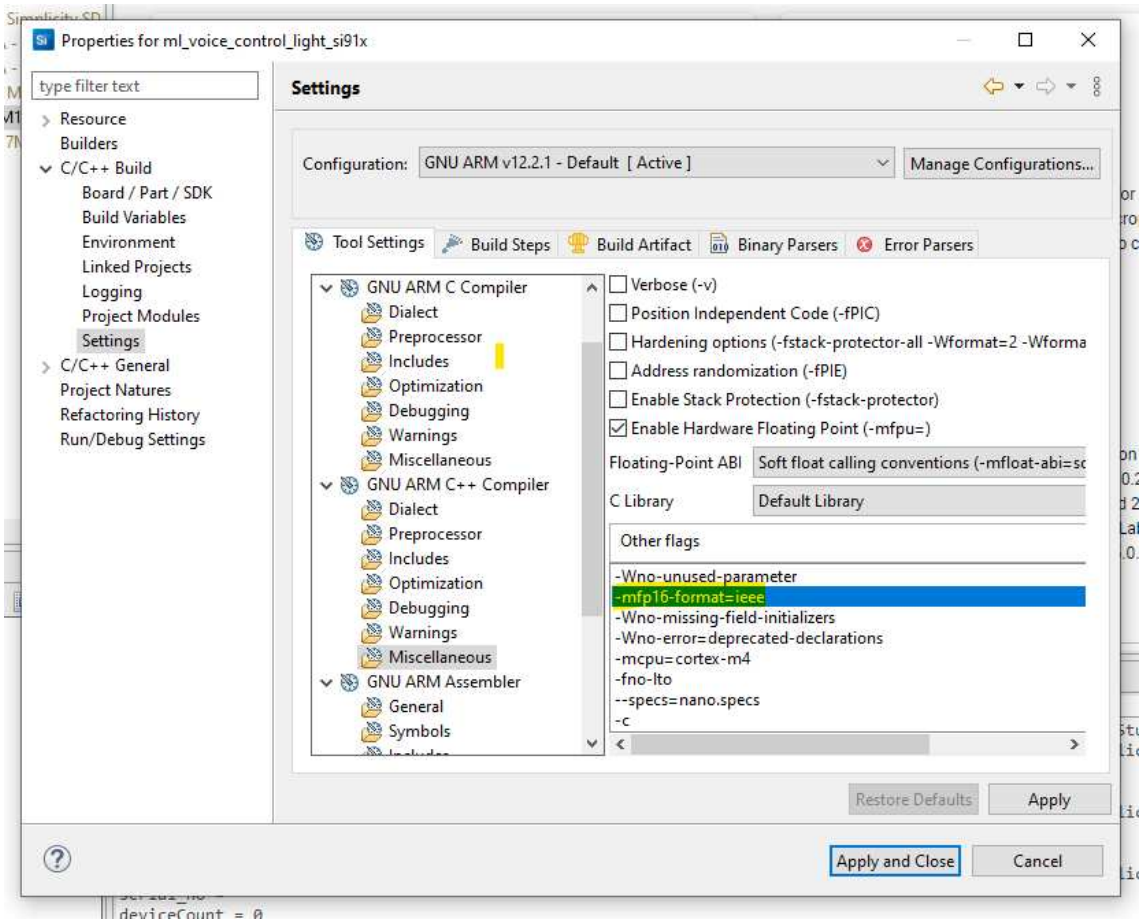
1. Ensure the following components are installed in your project.
   - WiseConnect SDK > Device > Si91X > MCU > Service > Power Manager > Sleep Timer for Si91x
   - Platform > Peripheral > Common Headers





2. Update your C++ build settings as follows:
   - In the C preprocessor defines, add: `SUPPORT_CPLUSPLUS`.
   - In GNU ARM C++ Compiler > Miscellaneous settings, add: `-mfp16-format=ieee`.





Alternate method:
You can also add these settings directly to your project's `.slcp` file:

- Open the `.slcp` file in a text editor.
- Add `SUPPORT_CPLUSPLUS` to the `define` section for C preprocessor defines. For example:

```
define:
  - name: SUPPORT_CPLUSPLUS
    value: 1
```

- Add `-mfp16-format=ieee` to the `toolchain_settings` section for C++ compiler flags, for example:

```
toolchain_settings:
  - option: gcc_compiler_option
    value: -mfp16-format=ieee
```

- Save the file and regenerate your project to apply the changes.

Model Inclusion

With the TensorFlow Lite Micro component added in the Project Configurator, the next step is to load the model file into the project. To do this, create a `tflite` directory inside the `config` directory of the project, and copy the `.tflite` model file into it. The project configurator provides a tool that will automatically convert `.tflite` files into `sl_tflite_micro_model` source and header files. The full documentation for this tool is available at Flatbuffer Converter Tool.

For SiWx917 devices, after copying the `.tflite` model file into the project, a header file named `sl_ml_model_<model_name>.h` is generated. This header provides access to the C array obtained from the converted `.tflite` file, as well as APIs to initialize and run the model.

Automatic Initialization

The TensorFlow framework is automatically initialized using the system initialization framework described in SDK Programming Model. This includes allocating a tensor arena, instantiating an interpreter and loading the model.

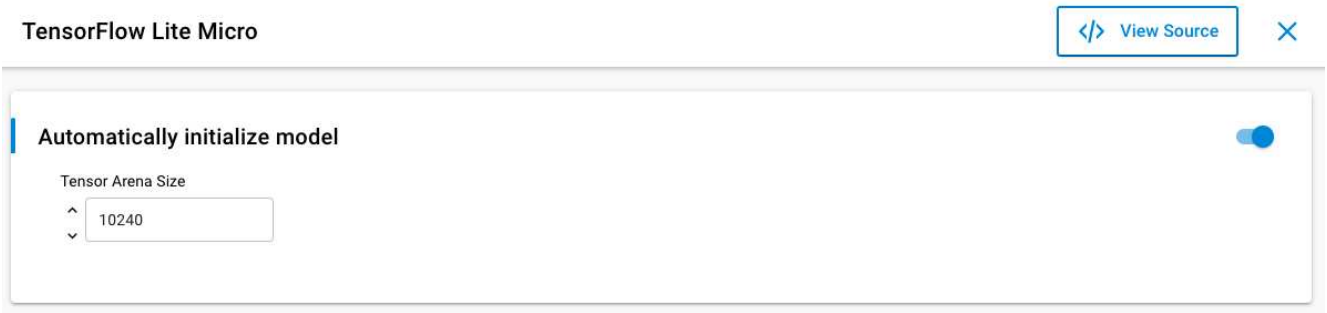
Configuration

> Note: All configuration for SiWx917 is taken care of automatically.

If the model was produced using the Silicon Labs Machine Learning Toolkit (MLTK), it already contains metadata indicating the required size of the Tensor Arena, the memory area used by TensorFlow for runtime storage of input, output, and intermediate arrays. The required size of the arena depends on the model used.

If not using the MLTK, the arena size needs to be configured. This can be done in two ways:

- [Automatic] Set the arena size to -1, and it will attempt to automatically infer the size upon initialization.
- [Manual] Start with a large number during development, and reduce the allocation until initialization fails as part of size optimization.



Run the Model for Series 2 Devices

Include the Silicon Labs TensorFlow Init API

```
#include "sl_tflite_micro_init.h"
```

For default behavior in bare metal application, it is recommended to run the model during `app_process_action()` in `app.cpp` to ensure that periodic inferences occur during the standard event loop. Running the model involves three stages:

Provide Input to the Interpreter

Sensor data is pre-processed (if necessary) and then is provided as input to the interpreter.

```
TfLiteTensor* input = sl_tflite_micro_get_input_tensor();
// stores 0.0 to the input tensor of the model
input->data.f[0] = 0.;
```

Run Inference

The interpreter is then invoked to run all layers of the model.

```
TfLiteStatus invoke_status = sl_tflite_micro_get_interpreter()->Invoke();
if (invoke_status != kTfLiteOk) {
TF_LITE_REPORT_ERROR(sl_tflite_micro_get_error_reporter(),
                     "Bad input tensor parameters in model");
return;
}
```

Read Output for Series 2 devices

The output prediction is read from the interpreter.

```
TfLiteTensor* output = sl_tflite_micro_get_output_tensor();
// Obtain the output value from the tensor
float value = output->data.f[0];
```

At this point, application-dependent behavior based on the output prediction should be performed. The application will run inference on each iteration of `app_process_action()`.

Full Code Snippet (Series 2)

After following the steps above, the resulting `app.cpp` now appears as follows:

```
#include "sl_tflite_micro_init.h"

/*******************************************************************************//**
 * Initialize application.
 ******************************************************************************/
void app_init(void)
{
  // Init happens automatically
}

/*******************************************************************************//**
 * App ticking function.
 ******************************************************************************/
void app_process_action(void)
{
  TfLiteTensor* input = sl_tflite_micro_get_input_tensor();
  // stores 0.0 to the input tensor of the model
  input->data.f[0] = 0.;

  TfLiteStatus invoke_status = sl_tflite_micro_get_interpreter()->Invoke();
  if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(sl_tflite_micro_get_error_reporter(),
                       "Bad input tensor parameters in model");
  return;
  }

  TfLiteTensor* output = sl_tflite_micro_get_output_tensor();
  // Obtain the output value from the tensor
  float value = output->data.f[0];
}
```

Run the Model for SiWx917 Devices

For SiWx917 devices, the APIs differ from Series 2. Use the generated model C++ APIs to initialize and run your model. Below is a general guide for using these APIs:

1. Include Required Headers

```
//Provides _init() and _run() functions for this model
#include "sl_ml_model_<model_name>.h"
// ...other headers as needed...
```

2. System Initialization

```
sl_system_init();
```

3. Model Initialization

```
//Loads and Initializes the model with name <model_name>
//Initializes its parameters
//Initializes its interpreter
//Initializes its error reporter
//Allocates memory for model's tensors

sl_status_t status = slx_ml_<model_name>_model_init();
if (status != SL_STATUS_OK) {
  printf("Failed to initialize model\n");
  // Handle error
}
```

4. Access and Prepare Input Tensors

```
for (unsigned int i = 0; i < <model_name>_model.n_inputs(); ++i) {
    auto& input_tensor = *<model_name>_model.input(i);
    // Fill input_tensor.data with your input data
}
```

5. Run Inference

```
/// Executes the model with name <model_name> by invoking its interpreter for TFLite Micro and returns execution status.
/// The output is stored in the output tensor of the model itself.
status = slx_ml_model_<model_name>_run();
if (status != SL_STATUS_OK) {
    printf("Error while running inference\n");
    // Handle error
}
```

6. Access Output Tensors

```
for (unsigned int i = 0; i < <model_name>_model.n_outputs(); ++i) {
    auto& output_tensor = *<model_name>_model.output(i);
    // Read results from output_tensor.data
}
```

Full Code Snippet (SiWx917)

Below is a generic code snippet for running a model on SiWx917 devices. Replace `<model_name>` with your actual model name.

`app.h`

```
#ifndef APP_H
#define APP_H

#ifdef __cplusplus
extern "C" {
#endif

void app_init(void);
void app_process_action(void);

#ifdef __cplusplus
}
#endif

#endif  // APP_H
```

`app.cc`

```
#include "app.h"
#include "model_runner.h"

sl_status_t <model_name>_model_status = SL_STATUS_OK;

// Initialization logic
void app_init(void) {
  <model_name>_model_status = model_runner_init();
  if (<model_name>_model_status != SL_STATUS_OK) {
    // Print helpful error message or handle error
    return;
  }
}

void app_process_action(void) {
  if (<model_name>_model_status != SL_STATUS_OK) {
    // Print an error message or indicate error
    return;
  }
  model_runner_loop();
}
```

model_runner.cc

```
#include "sl_ml_model_<model_name>.h"

sl_status_t model_runner_init(void) {
  // Peripheral and other initialization logic

  <model_name>_model_status = slx_ml_<model_name>_model_init();

  if (<model_name>_model_status != SL_STATUS_OK) {
    // Peripheral deinit or cleanup logic
    return SL_STATUS_FAIL;
  }
  return <model_name>_model_status;
}

sl_status_t model_runner_loop(void) {
  // Data capture and pre-processing logic

  <model_name>_model_status = slx_ml_<model_name>_model_run();

  if (<model_name>_model_status != SL_STATUS_OK) {
    // Peripheral deinit or cleanup logic
    return SL_STATUS_FAIL;
  }

  // Post-processing logic

  return <model_name>_model_status;
}
```

Addendum: Interfacing with C Code

If your project is written in C rather than C++, place the code interfacing with TFLM into a separate file that exports a C API through an interface header. For this example, a filename `app_ml.cpp` is assumed that implements the function `ml_process_action()` with the same content as in the example above.

app_ml.h

```
#ifdef __cplusplus
extern "C" {
#endif

void ml_process_action(void);

#ifdef __cplusplus
}
#endif
```

app_ml.cpp

```
#include "app_ml.h"
#include "sl_tflite_micro_init.h"

extern "C" void ml_process_action(void)
{
  // ...
}
```

app.c

```
#include "app_ml.h"
// ...
void app_process_action(void)
{
  ml_process_action();
}
```

Addendum: Series 2 to SiWx917 (and vice versa) App Conversion

If you need to port an application from Series 2 to SiWx917, first remove the TensorFlow Lite Micro component from your project. Then, follow these steps for adding the required software components and C++ build settings. For SiWx917 devices, the APIs and workflow differ from Series 2. Use the generated model C++ APIs as described in the Run the Model for SiWx917 Devices section above. Refer to the provided code snippets and step-by-step instructions on this page for details on initialization, running inference, and accessing input/output tensors. For a complete example, see the Full Code Snippet (SiWx917) section. After updating your project, regenerate it to ensure all dependencies are correctly configured.

Update or Replace a .tflite File

Updating or Replacing the .tflite File in a Project

This guide describes how to swap out the model in an existing project. It assumes that the project uses the Flatbuffer Converter Tool.

Replace the Model

To replace the model in an existing project, drag-and-drop the new model file into the `config/tflite/` directory of the project. If the new model has the same file name as the previous model, accept the prompt to overwrite the file. If the new model has a different name, delete or rename the old `.tflite` file such that it no longer has the `.tflite` extension. The Flatbuffer Converter Tool will automatically execute when the directory watcher notices that a different `.tflite` file is present.

> Note: If you have multiple `.tflite` files in the `config/tflite/` directory, the converter tool will pick the first file in alphabetical order. Hence the recommendation to rename or delete any old models to ensure that the tool uses the correct file.

Inspect the Model

You can take steps to ensure that the automatic regeneration of the C arrays and headers from the `.tflite` file executed as expected. Any of the below steps can be taken if you are ever unsure of what model is part of your application binary.

Check the Model Size

The generated file `autogen/sl_tflite_micro_model.c` defines a size variable `sl_tflite_model_len`. This number can be compared to the file size of the `.tflite` file in bytes, for example, by right-clicking the `.tflite` file and looking at the value of Properties > Resource > Size.

Check the Operators Used by the Model

The generated file `autogen/sl_tflite_micro_opcode_resolver.h` contains multiple calls to `tflite::MicroMutableOpResolver::AddXXX`, where `XXX` is the name of operators used by the model. This can be compared to the operators you know the model *should* use.

Check the Model Parameters

If the `.tflite` file was generated using the Silicon Labs Machine Learning Toolkit, it contains metadata that is generated into `autogen/sl_tflite_micro_model_parameters.h`. These parameters can be compared to the expected parameters.

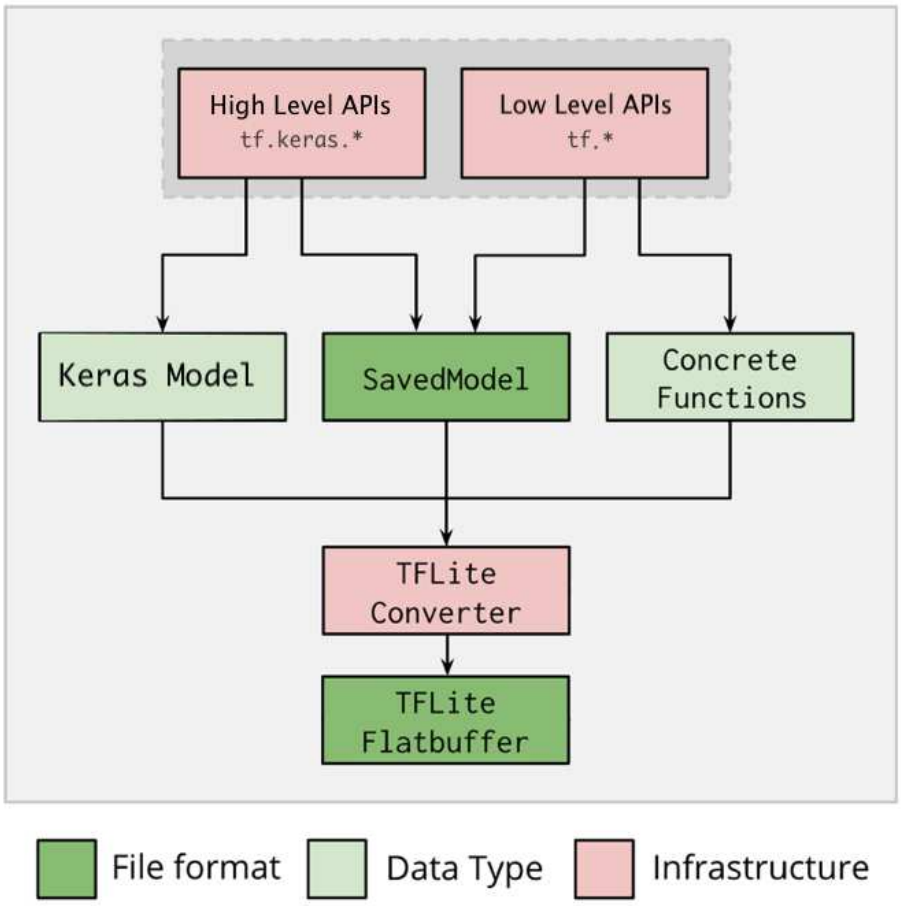
Force Generation of C files

If anything happens that makes the generated `.c` and `.h` get out of sync with the `.tflite` file, the project can be forcefully regenerated by pressing the Force Generation button on the Project Details pane of the Project Configurator.

Developing a Model

Developing a Machine Learning Model

Developing a Model Manually using TensorFlow and Keras



When developing and training neural networks for use in embedded systems, it is important to note the limitations on TFLM that apply to model architecture and training. Embedded platforms also have significant performance constraints that must be considered when designing and evaluating a model. The embedded TLFM documentation links describe these limitations and considerations in detail.

Additionally, the TensorFlow Software Components in Simplicity Studio require a quantized `*.tflite` representation of the trained model. As a result, TensorFlow and Keras are the recommended platforms for model development and training because both platforms are supported by the TensorFlow Lite Converter that generates `.tflite` model representations.

Both TensorFlow and Keras provide guides on model development and training:

TensorFlow Basic Training Loops
- Keras Training and Evaluation

After a model has been created and trained in TensorFlow or Keras, it needs to be converted and serialized into a `*.tflite` file. During model conversion, it is important to optimize the memory usage of the model by quantizing it. It is highly recommended to use integer quantization on Silicon Labs devices.

- TensorFlow Lite Converter
- Quantization Overview

A complete example demonstrating the training, conversion, and quantization of a simple TFLM compatible neural network is available from TensorFlow:

- TensorFlow Hello World Training Example
- Trained Hello World Model (Downloads a .zip file containing the model)

Developing a Model using Silicon Labs AI/ML Partners

See the AI/ML Partners section on the Silicon Labs Machine Learning in IoT page for more information about Silicon Labs AI/ML partners. These partners can help you develop a model that is compatible with Silicon Labs devices.

Developing a Model using the MLTK

> Note: MLTK is experimental and official support is unavailable. It is expected that the MLTK is used by an ML Expert with deep knowledge of TensorFlow and Python, or by a developer willing to learn.

The Silicon Labs Machine Learning Toolkit (MLTK) is a Python package that implements a layer above TensorFlow to help the TensorFlow developer build models that can be successfully deployed on Silicon Labs chips. These scripts are a reference implementation for the audio use case, which includes the use of the Audio Feature Generator on both the training and inference side. This is modified version of the TensorFlow "microfrontend" audio front end.

The MLTK is offered as a self-serve, self-support, fully documented, Python reference package published through GitHub. We are delivering this as an Experimental package, which means it's available "as-is", un-tested, and without support.

See the MLTK documentation for more information.

AI/ML Extension Setup

Installing the AI/ML Extension for Silicon Labs Simplicity Studio

This guide details the installation process for the AI/ML extension within Silicon Labs Simplicity Studio. This extension empowers developers to integrate machine learning capabilities into their Silicon Labs-based projects.

Prerequisites

Before proceeding with the installation, ensure you have the following:

- Silicon Labs Simplicity Studio: The latest version of Simplicity Studio is essential. Download it from the official Silicon Labs website.
- Supported Hardware: Verify that your target Silicon Labs hardware is compatible with the AI/ML extension. Consult the extension's release notes or documentation for a list of supported devices.
- Python Environment (Potentially): Some AI/ML functionalities might rely on a Python environment. It is recommended to have Python 3.7 or higher installed. A virtual environment is strongly advised to manage dependencies effectively.
- Git (Potentially): Git might be required for certain installation methods or for accessing specific resources. Ensure Git is installed on your system.

Installation Methods

The AI/ML extension can be installed via the following methods:

1. Simplicity Installer (Recommended)

This is the most straightforward and recommended approach.

1. Download Simplicity Installer using this link for your OS.
2. Open Simplicity Installer.
3. Click on Installation Wizard.
4. Click Technology Install.
5. Select the AI/ML from the list of available extensions under Optional Packages.
6. Click NEXT. On the next page, accept the Terms of Use and License Agreement, and then click INSTALL to begin the installation.

2. Silicon Labs Tool (command line)

1. Download Silicon Labs Tool (SLT) using this link for your OS.
2. Unzip the downloaded zip file to your preferred location. Add this location to your PATH environment variable so that `slt` command is available globally.
3. Install Simplicity SDK by invoking `slt install simplicity-sdk` on the command line.

Post-Installation Steps

After successful installation:

1. Start Simplicity Studio: Restart Simplicity Studio to ensure the changes take effect.

2. Verify Installation: Check the Simplicity Studio preferences or installed software list to confirm that the AI/ML extension is listed and installed.
3. Explore Documentation and Examples: The AI/ML extension should include documentation and example projects. These resources are crucial for getting started and understanding how to use the extension's features and APIs.

Troubleshooting

If you encounter any issues during installation:

1. Check Simplicity Studio Logs: Examine the Simplicity Studio logs for any error messages. These logs can provide valuable clues for troubleshooting.
2. Review Documentation: Refer to the AI/ML extension's documentation and release notes for troubleshooting tips and known issues.
3. Silicon Labs Community Forum: The Silicon Labs community forum is an excellent resource for finding solutions to common problems or asking for help from other users and Silicon Labs experts.

Updating the Extension

To update the AI/ML extension, follow the same installation steps as described above. Simplicity Studio will typically detect the newer version and guide you through the update process.

Uninstallation

To uninstall the AI/ML extension:

1. Open Simplicity Installer.
2. Click Package Manager.
3. Search for Simplicity SDKs.
4. Click the SDK from which you want to uninstall the AI/ML Extension and click the Trash Bin icon beside it.

Flatbuffer Converter Tool

Flatbuffer Converter Tool

The Flatbuffer Converter Tool helps to convert a `.tflite` file into a C array that can be compiled into a binary for an embedded system. This array can be used with the TensorFlow Lite for Microcontrollers API, which takes a void pointer to a buffer containing the model as an argument to its `tflite::GetModel()` init function.

In addition to converting the flatbuffer into a C array, the tool supports emitting model parameters embedded as metadata in the `.tflite` file as C preprocessor macros.

Input

The tool takes a directory containing one or more `.tflite` files as input. If the directory consists of multiple files, only the first file in alphabetical order is converted.

> Tip: If you have multiple files in the directory, but the one you want to convert isn't the first file in alphabetical order, you can rename the other files to add a `.bak` extension or rename the target file accordingly.

Output

The tool writes its output into multiple files in a single output directory.

Model Array

The tool always emits a pair of files `sl_tflite_micro_model.c` / `.h` , which declares the variables as follows.

- `const uint8_t sl_tflite_model_array[]` containing the full contents of the `.tflite` file
- `const uint32_t sl_tflite_model_len` containing the length of the model array

Opcode Resolver

A header file `sl_tflite_micro_opcode_resolver.h` is also emitted. This file declares a C preprocessor macro `SL_TFLITE_MICRO_OPCODE_RESOLVER(opcode_resolver, error_reporter)` that instantiates a `tflite::MicroMutableOpResolver` object and automatically registers the set of operators required to parse the model.

This macro can be used as part of an initialization sequence to automatically initializing the optimal opcode resolver.

Model Parameters

If the `.tflite` file contains model parameters in its metadata section, a third header file `sl_tflite_micro_model_parameters.h` is emitted.

For every model parameter key-value pair, a C preprocessor macro `SL_TFLITE_MODEL_<key>` is created with the relevant value.

See the MLTK documentation to learn more about embedding model parameters in the `.tflite` file.

SLC Project Configuration Integration

The Flatbuffer Converter Tool integrates with the SLC project configuration tools in Simplicity Studio and on the command line using SLC-CLI. When using these tools, the Flatbuffer Converter is automatically run with the `config/tflite/` directory of the project as input, and the `autogen/` directory as output.

In other words, any file with the `.tflite` extension in the `config/tflite/` directory in the project will be automatically converted when the project is generated. If using Simplicity Studio, a directory watcher ensures that the project is automatically generated if a `.tflite` file is added or removed. This means that it is sufficient to drag-and-drop a `.tflite` file into the project, and it is automatically converted into C code.

Manual Usage

If conversion is desired outside of a full SLC project generation cycle, the flatbuffer converter can be invoked manually. This is generally not necessary, but is an option for advanced users. This can only be done using the SLC-CLI, users of Simplicity Studio are recommended to regenerate the full project.

As Standalone Conversion Command

The Flatbuffer Converter Tool runs as a standalone Python script outside of a project generation flow. Execute the following command using absolute paths for both input and output directories.

- NumPy must be installed. If not install it using:

```
pip install numpy
```

- The output directory must already exist before running the tool. Create one in any location you prefer:

```
mkdir -p /path/to/output/folder
```

- Identify your TFLite model folder where `.tflite` model files are stored. You may also use your own custom `.tflite` model files by supplying their absolute path to `-i`.
- Run `slt where aiml` to locate the AIML package installation path, and then go to the `tool/tflite` directory where `tflite.py` is located.
- Navigate to the folder containing flatbuffer_converter.py:

```
cd "$(slt where aiml)/tool/flatbuffer-converter/flatbuffer_converter.py"
```

- Run the Flatbuffer Converter Tool using:

```
python flatbuffer_converter.py generate </path/to/tflite/model/folder> </path/to/output/folder> <part_number>
```

Tool Help

You can run the help command at any time to view the latest usage instructions for this tool:

```
python flatbuffer_converter.py generate --help
```

```
usage: flatbuffer_converter.py generate [-h] input_dir output_dir part_number

Runs when project is generated using SLC-CLI or Studio.

positional arguments:
  input_dir    Input directory containing .tflite files
  output_dir   Output directory to populate with serialized content.
  part_number  Part Number

options:
  -h, --help   show this help message and exit
```

I2S Configuration for SiWx917

I2S Pin Configuration for ML Audio Applications on SiWx917 platform

The default configuration of I2S pins on SiWx917 does not provide the expected output for classification results on hardware for all AI/ML audio applications. You MUST manually configure the I2S pins using the `.slcp` as described below.

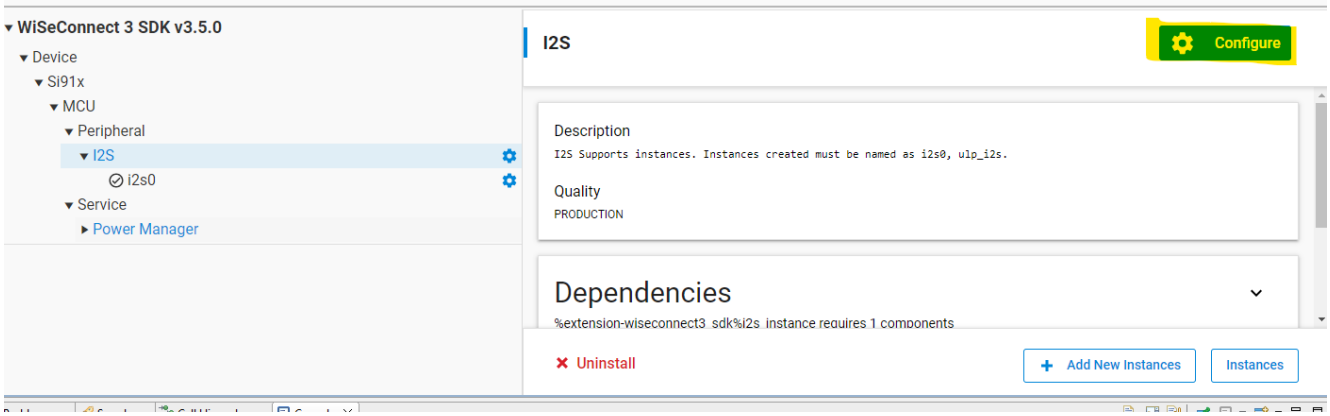
To configure the I2S pins for your board, follow these steps using Simplicity Studio:

1. Enable I2S Peripheral via Software Components

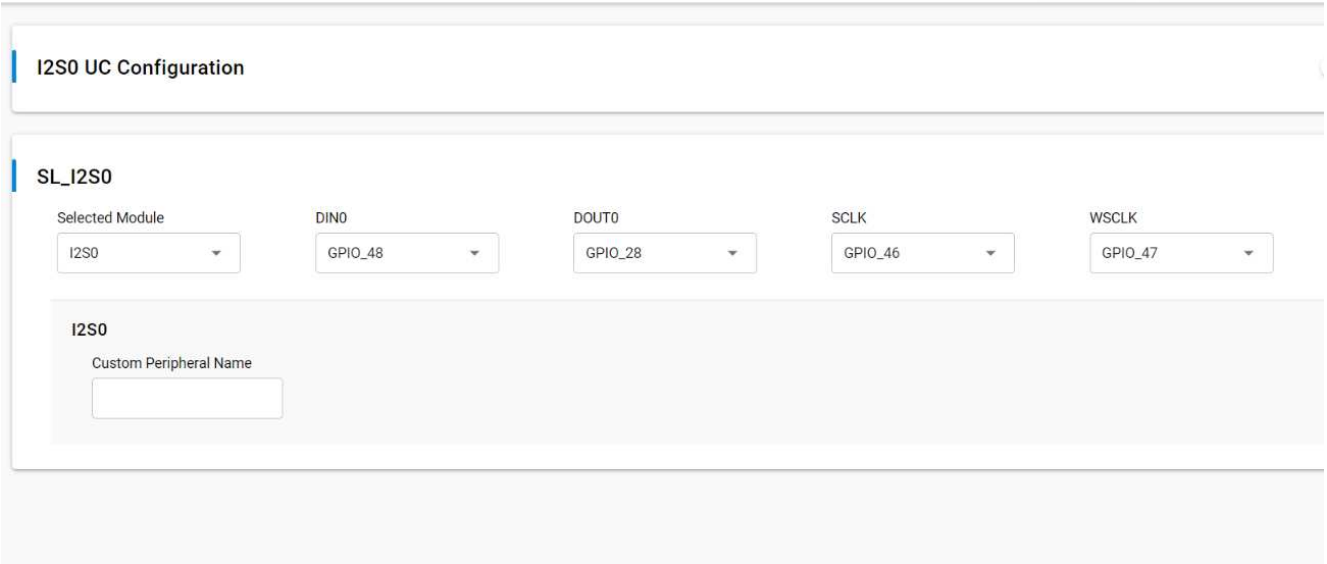
Once the .pintool file has been generated in your project folder, open the .slcp file in Simplicity Studio. After that, navigate to the Software Components tab. Search for and enable the I2S peripheral (e.g., `I2S0`). Navigate to WiSeConnect 3 SDK v3.5.0 > Device > Si91x > MCU > Peripheral > I2S > i2s0 > Configure.

2. Configure I2S Pins

In the Software Components configuration, click the Configure button next to the enabled I2S peripheral.



Assign the required I2S signals (`DIN0` , `SCLK` , and `WSCLK`) to the appropriate GPIO pins as shown in the image below. The configuration tool displays available pins. Ensure you select only those indicated in the image.



3. Save and Generate Code

The tool automatically generates the necessary pin configuration code for your project.

4. Rebuild the Project

Build your project to ensure the new pin configuration is included.

For more details, refer to the Simplicity Studio Software Components documentation.

ML Profiler

Silicon Labs Machine Learning Model Profiler

Overview

The Silicon Labs Machine Learning Model Profiler is a performance analysis tool designed to help engineers understand how TensorFlow Lite for Microcontrollers models (`.tflite`) execute on Silicon Labs embedded devices. The purpose of this tool is also to help developers optimize their models before deploying to production.

The profiler enables users to:

- Correlate layer execution with power usage, clock rate, and memory pressure (Power usage profiling is a planned feature)
- Build intuition for optimizing models to reduce stalls and improve inference latency
- Identify performance bottlenecks during inference
- Understand trade-offs between execution on the ARM Cortex-M MCU and the Matrix Vector Processor (MVP).

The tool is available as:

- a standalone GUI
- a command-line interface (CLI) called `sml`
- a Python API (planned feature)

It can also be launched directly from Simplicity Studio v6.

Who This Tool Is For

This documentation is written primarily for embedded and ML engineers.
Product managers, data scientists, and sales engineers are expected to be sufficiently familiar with machine learning concepts to interpret the results.

The profiler focuses exclusively on execution performance, not model accuracy or output quality.

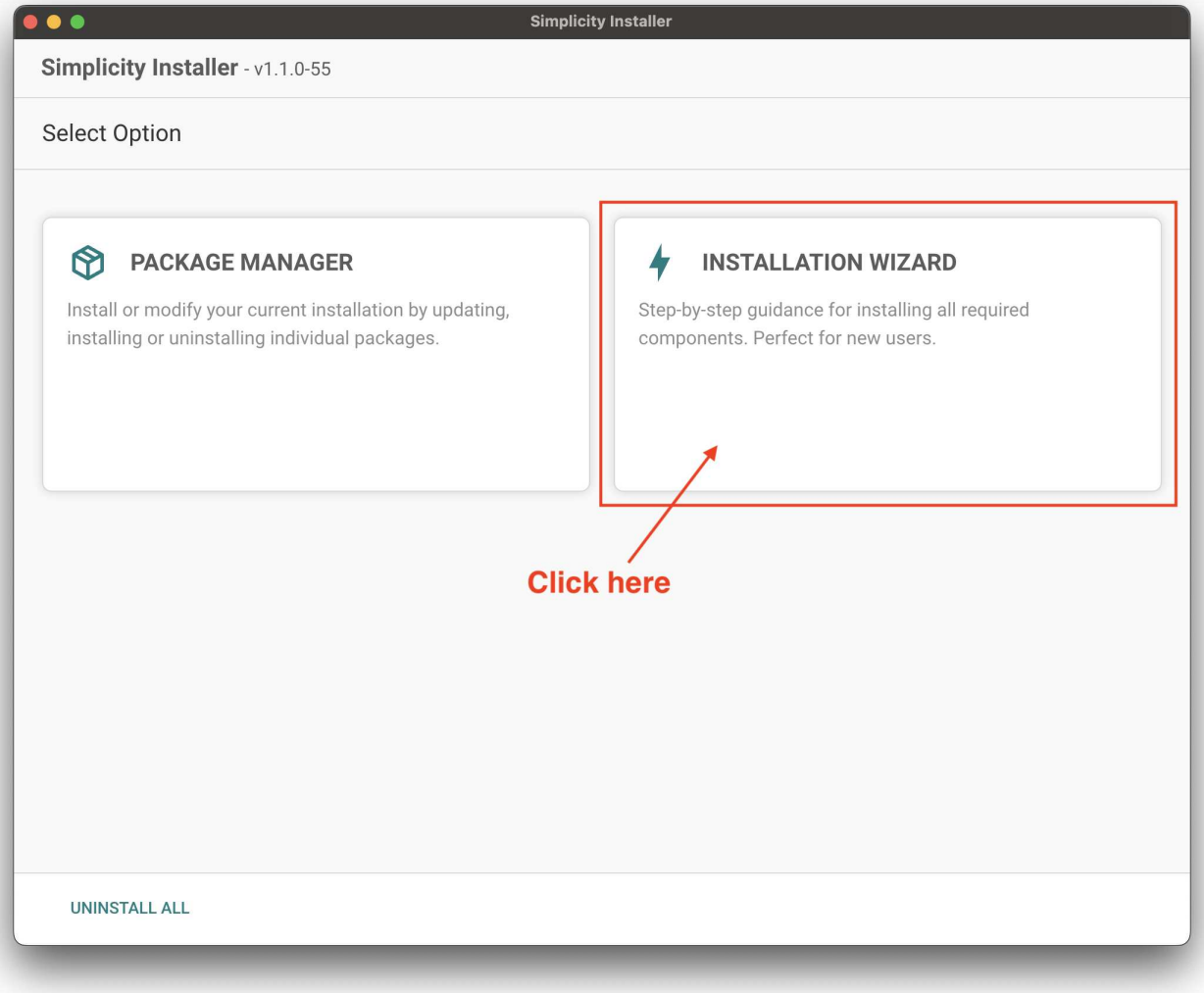
Key Concepts and Terminology

Term	Meaning
Inference	One complete execution of a model
Layer	A neural network layer
Operator	A logical operation within a layer
Kernel	The concrete implementation that executes an operator
MCU	ARM Cortex-M core
MVP	Matrix Vector Processor hardware accelerator
Stall / Wait	Time spent idle due to memory or resource contention
Power	Power consumed during execution
Tensor Arena	Memory allocated for TFLM state
Quantization	Optimization technique to reduce model size
Perfetto	The trace visualization tool used

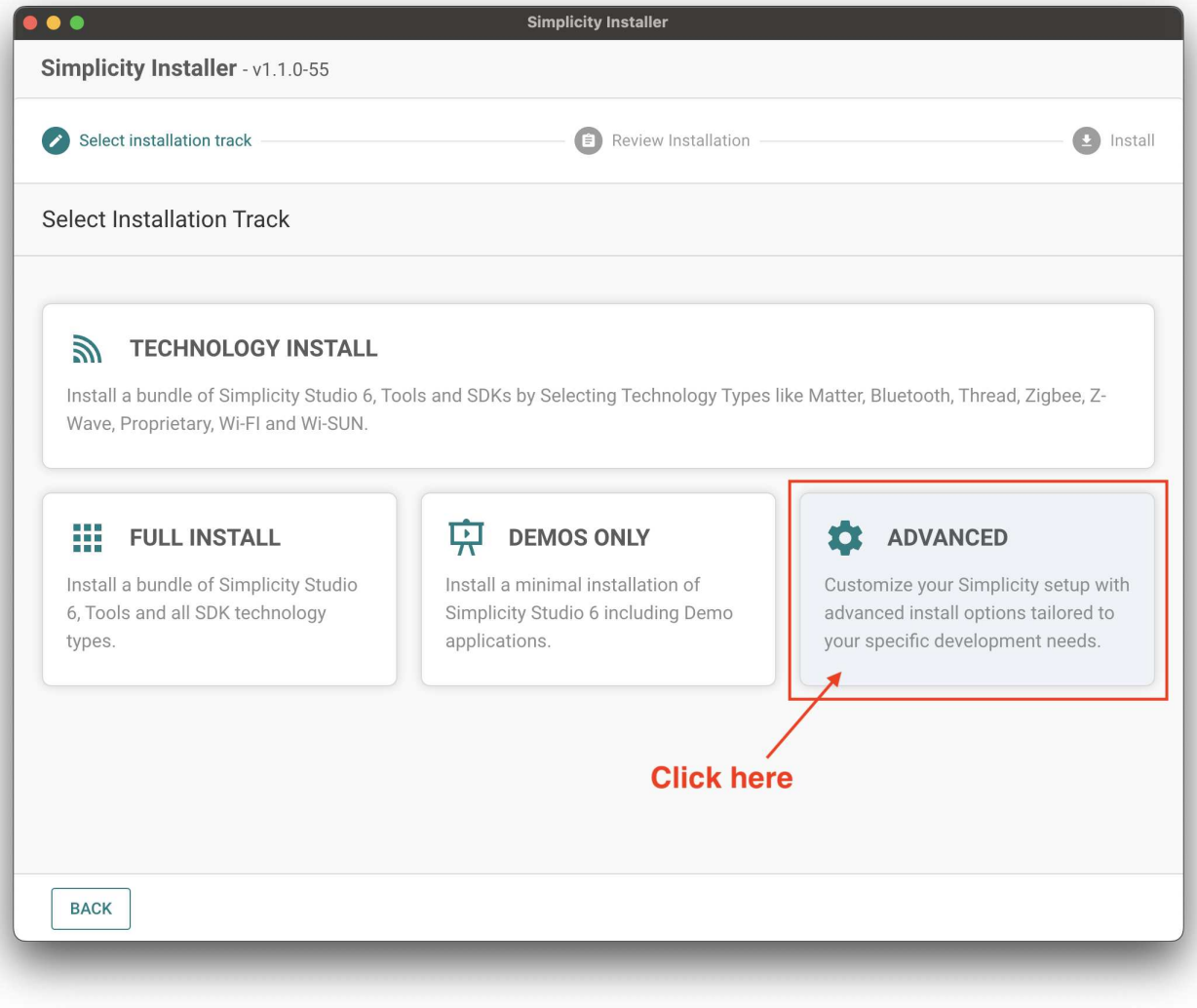
Installation

Using Simplicity Installer

1. Download Simplicity Installer using this link for your OS.
2. Once Installed, open Simplicity Installer and navigate to Installation Wizard.

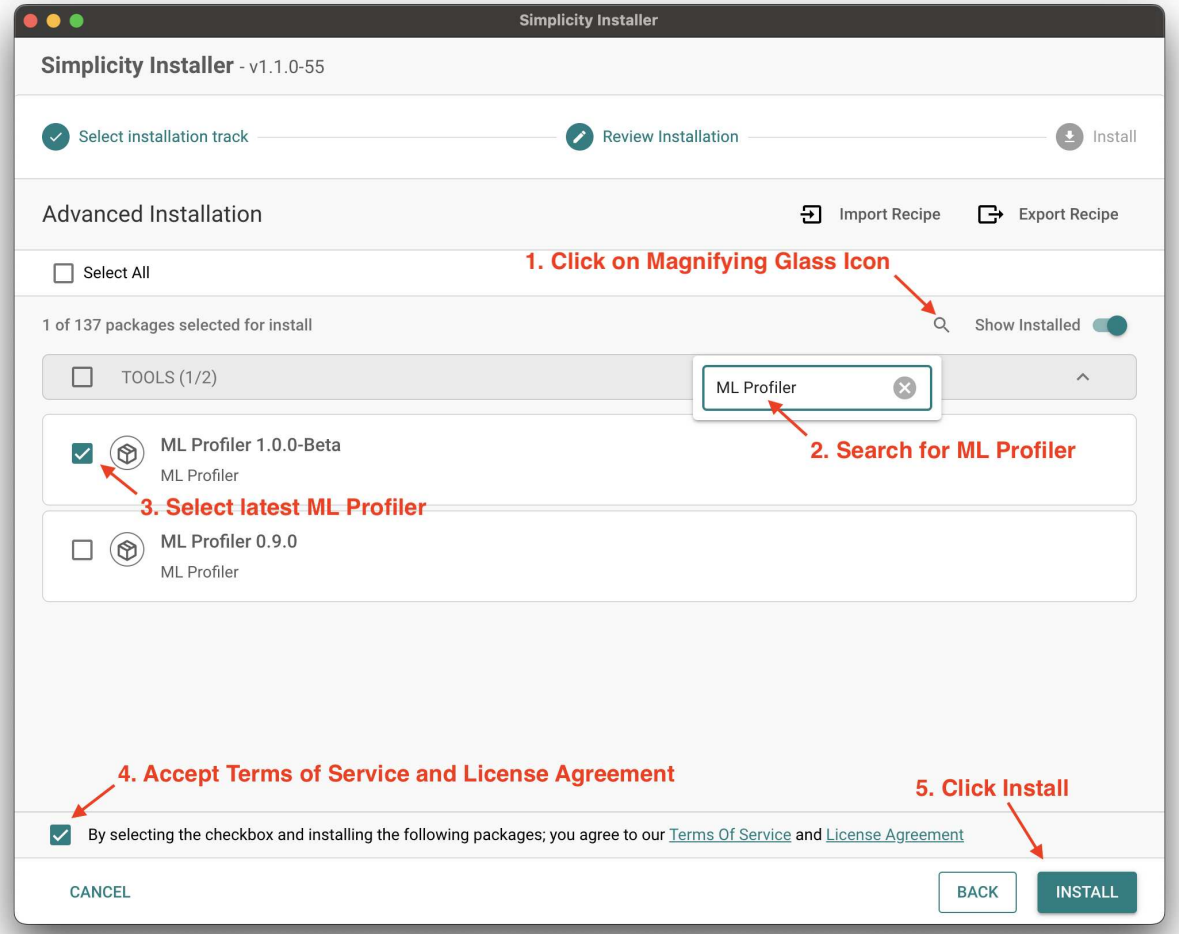


3. Click on Advanced tile.

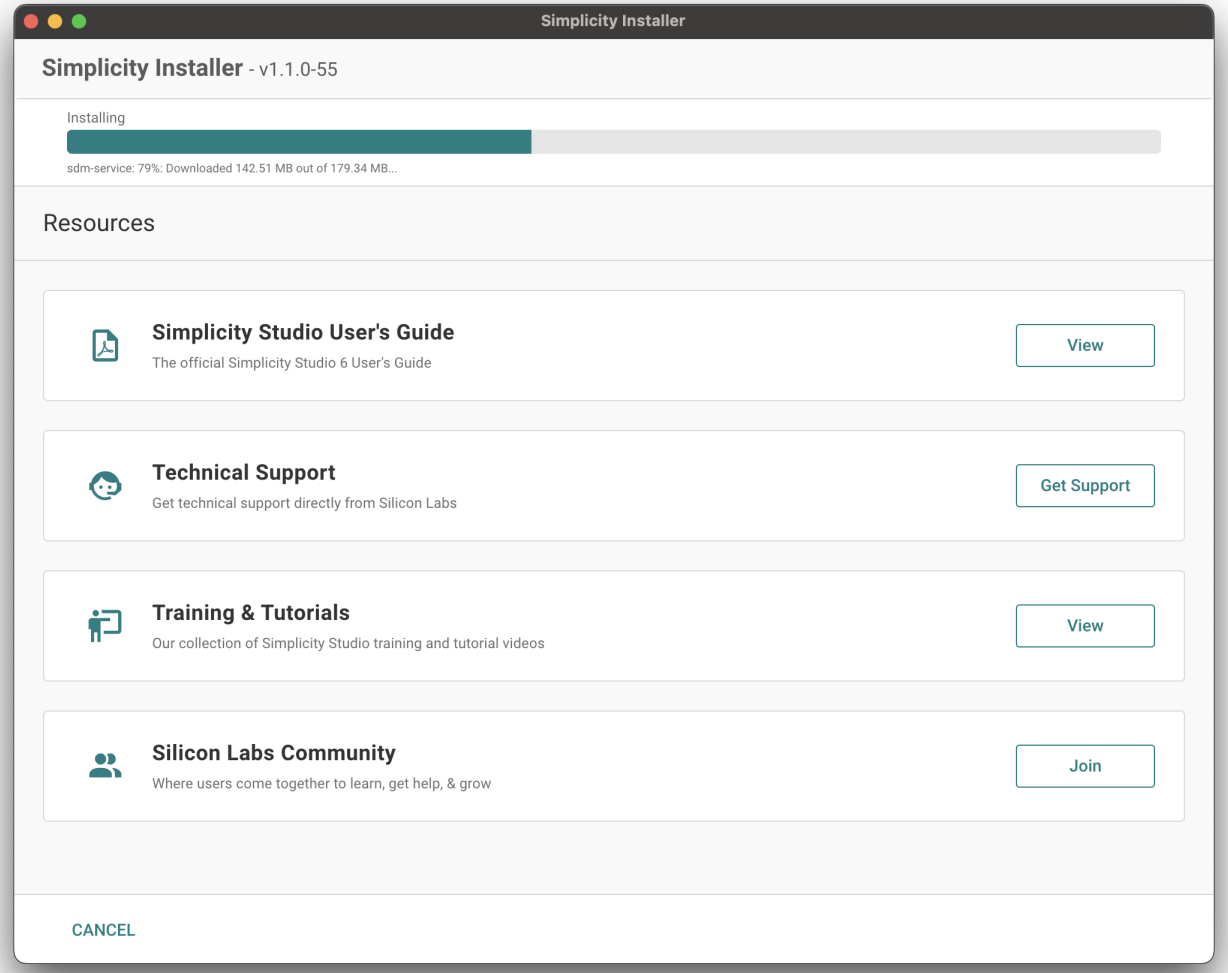


4. Search for ML Profiler:
    1. Click the Magnifying Glass Icon.
    2. Type *ML Profiler*, or a part of it, in the Search Box for GUI version, or *Silabs Machine Learning* for CLI version (`sml`).
    3. Select the latest ML Profiler by selecting the checkbox beside it. Versioning is based on semantic versioning.
    4. Accept Terms of Service and License Agreement by selecting the checkbox at the bottom.

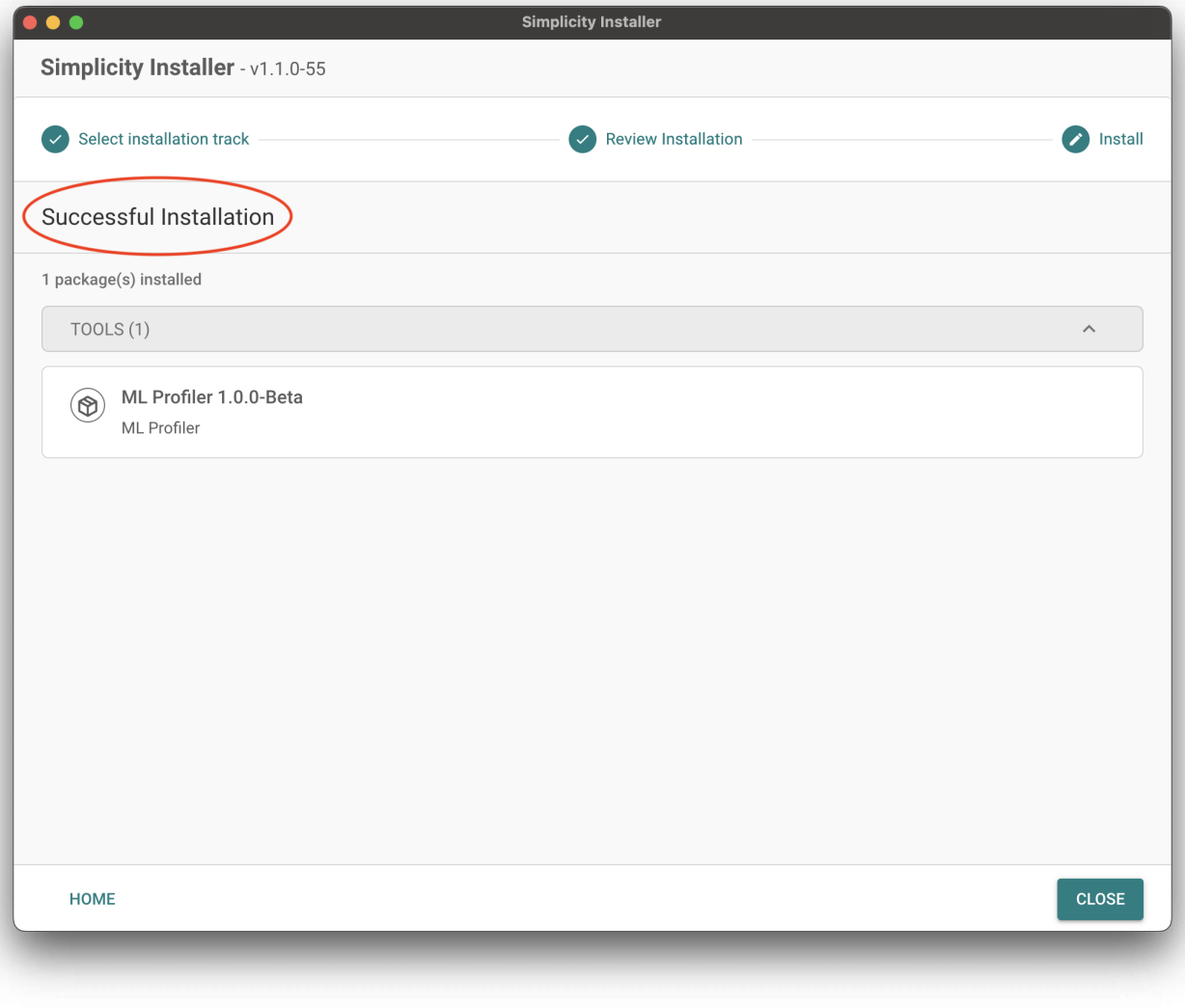
5. Click the Install button.



5. Installation should progress, as shown below.



6. Once the installation is successful, you should see the following.



7. If you installed the CLI version (`sml`), add its location to your PATH environment variable. You can find the location using `slt where sml` .

Using SLT on the command line

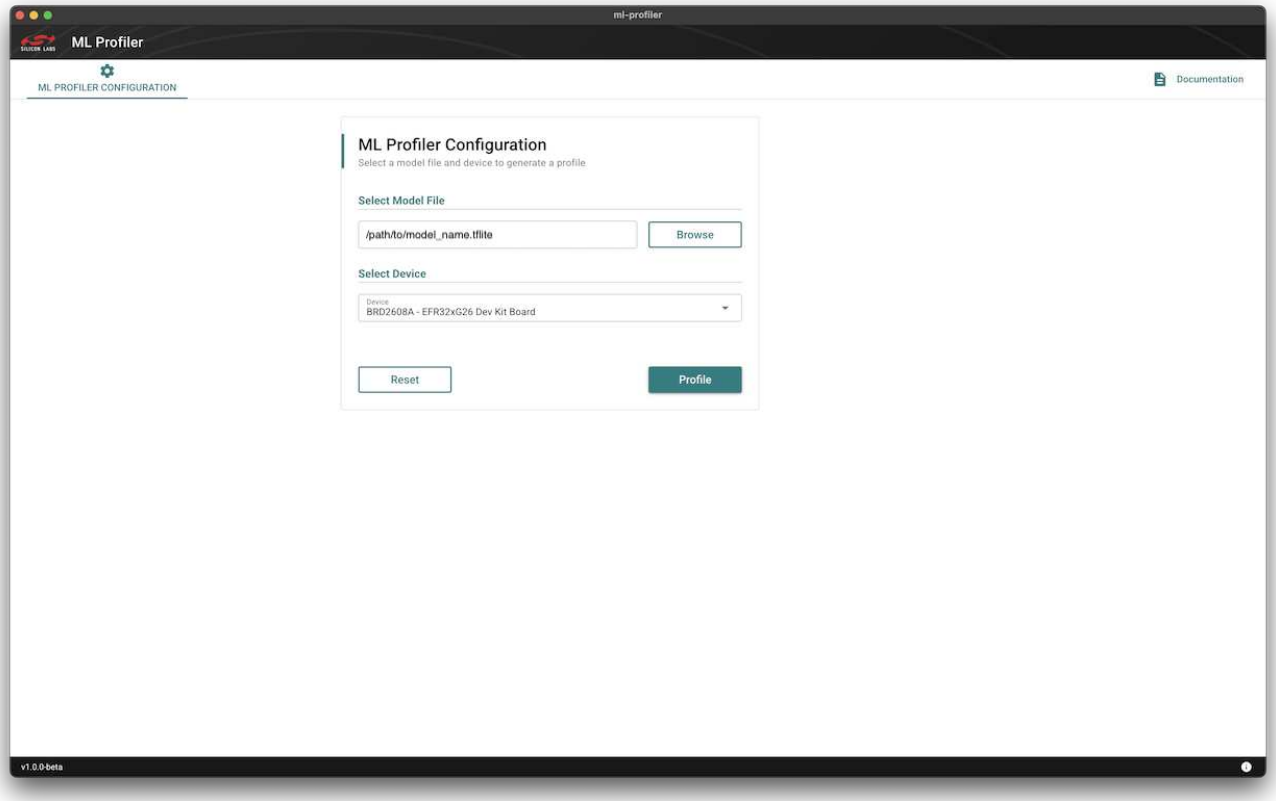
1. Download Silicon Labs Tool (SLT) using this link for your OS.
2. Unzip the downloaded zip file to your preferred location. Add this location to your PATH environment variable so that `slt` command is available globally.
3. Install ML Profiler using `slt install ml-profiler` for the GUI version or `slt install sml` for the CLI version.

Running a Profiling Session from the GUI

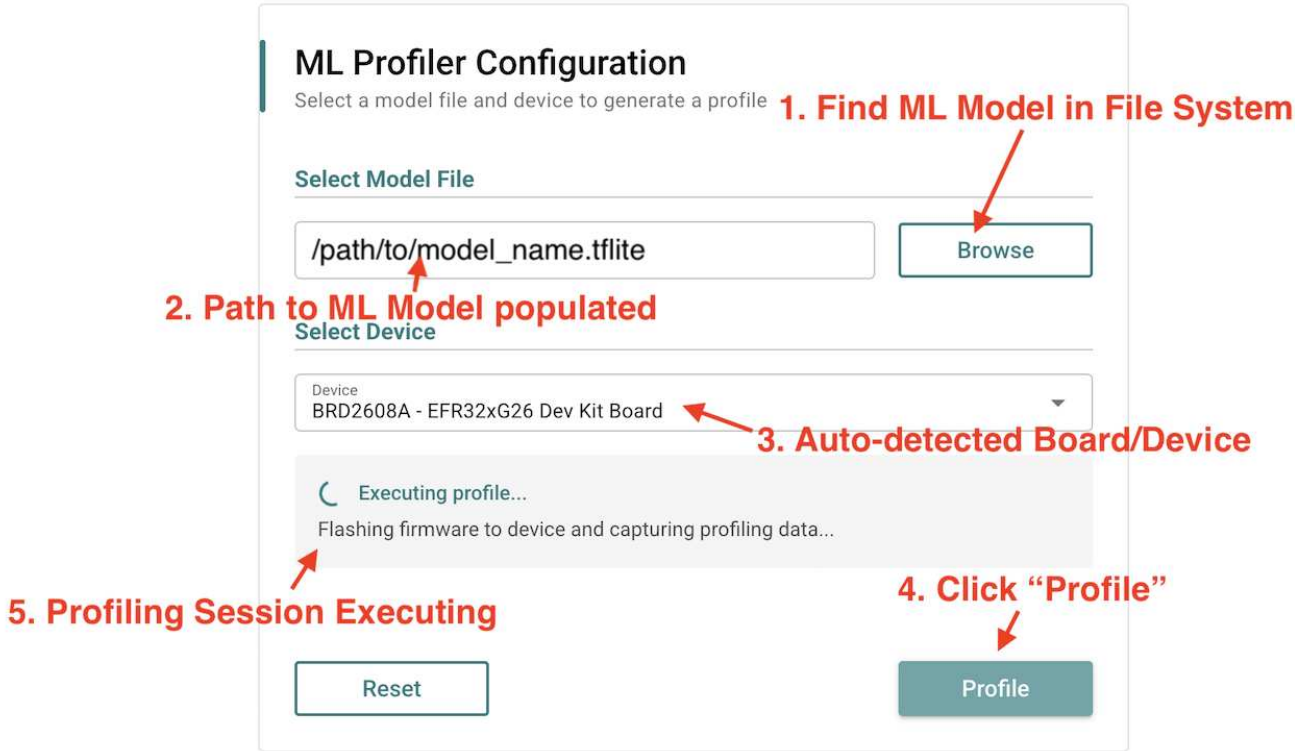
1. Launch the ML Profiler from Simplicity Studio v6
    1. Open Simplicity Studio and navigate to the Tools tab on the left panel.
    2. Hover on ML Profiler and click the Play icon that appears on the right side.
2. Or, Launch the ML Profiler from the command line:

```
slt launch ml-profiler
```

3. Step 1 or 2 should launch the ML Profiler. It is recommended to keep the window of ML Profiler maximized or in full-screen for the best user experience.



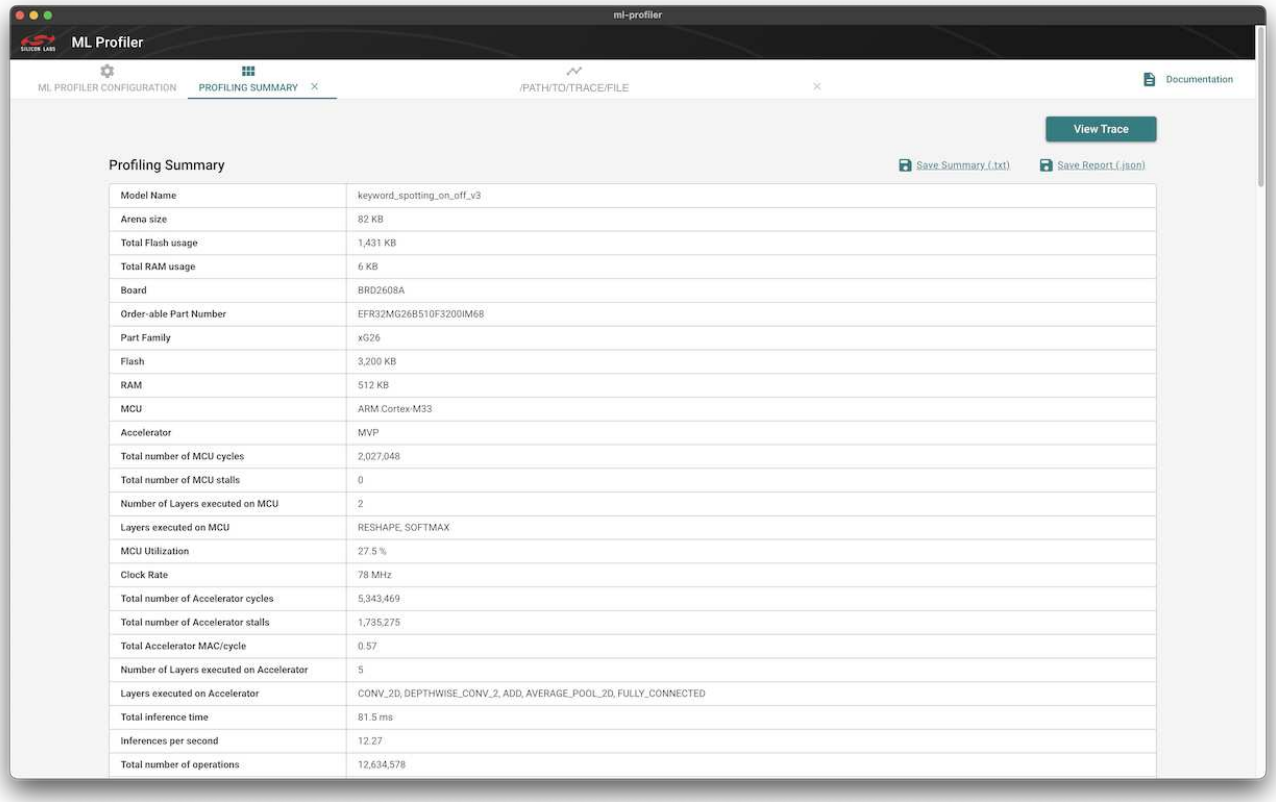
4. Connect the board you on which you want to profile your model. The board will be detected automatically.
5. Select the `.tflite` model you want to profile by clicking the Browse button and navigating your file system for the model file.
6. Click the Profile Button

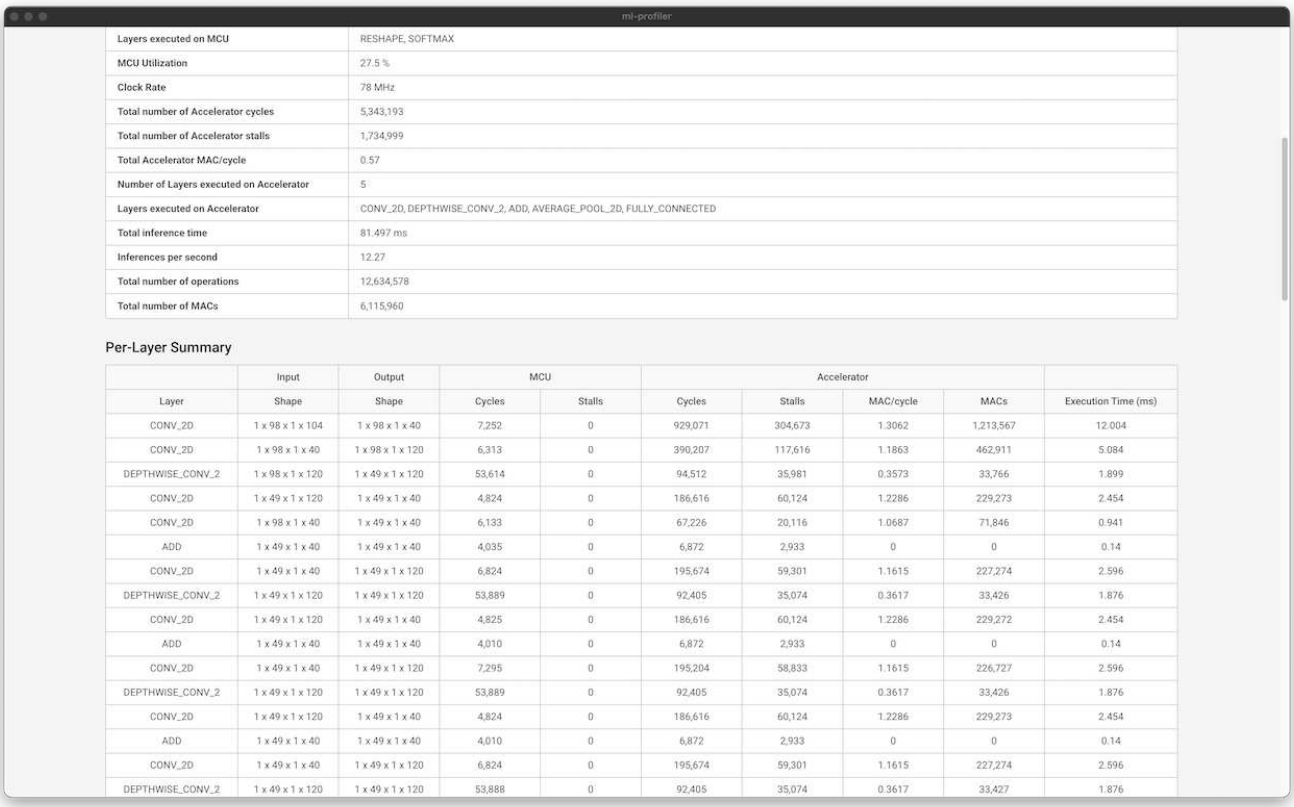


> *NOTE:* See Troubleshooting section for handling any errors.

Outputs

- Summary tab: inference time, MCU vs MVP cycles, power



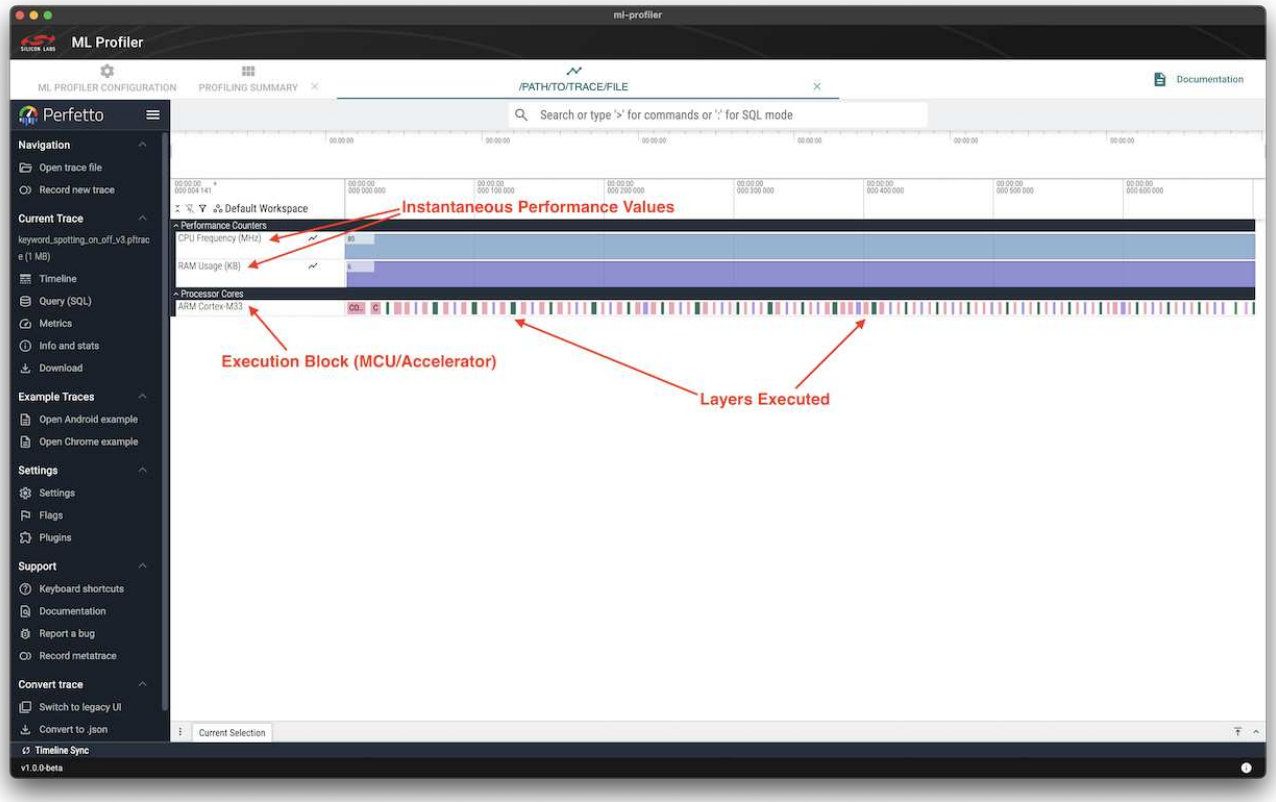


Layers executed on MCU	RESHAPE, SOFTMAX
MCU Utilization	27.5 %
Clock Rate	78 MHz
Total number of Accelerator cycles	5,343,193
Total number of Accelerator stalls	1,734,999
Total Accelerator MAC/cycle	0.57
Number of Layers executed on Accelerator	5
Layers executed on Accelerator	CONV_2D, DEPTHWISE_CONV_2, ADD, AVERAGE_POOL_2D, FULLY_CONNECTED
Total inference time	81.497 ms
Inferences per second	12.27
Total number of operations	12,634,578
Total number of MACs	6,115,960

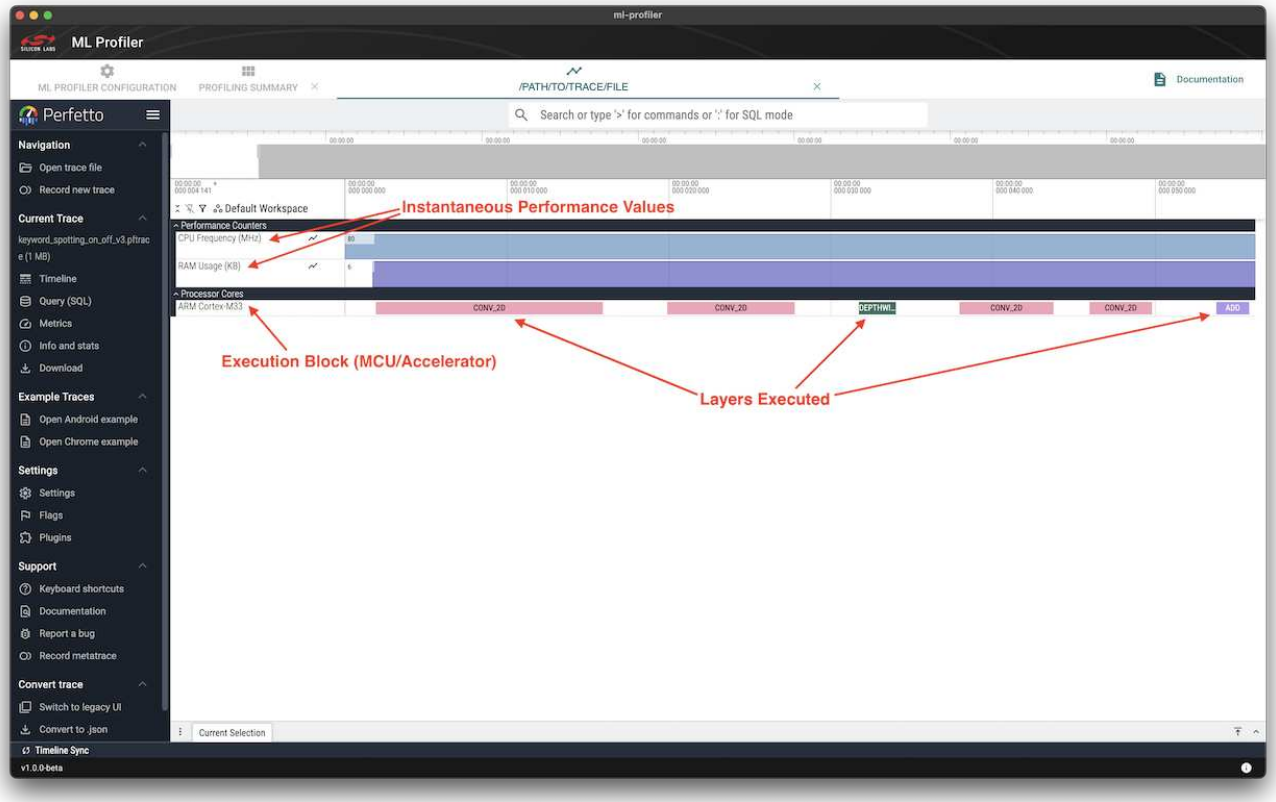
Per-Layer Summary

	Input	Output	MCU		Accelerator				
Layer	Shape	Shape	Cycles	Stalls	Cycles	Stalls	MAC/cycle	MACs	Execution Time (ms)
CONV_2D	1 x 98 x 1 x 104	1 x 98 x 1 x 40	7,252	0	929,071	304,673	1.3062	1,213,567	12.004
CONV_2D	1 x 98 x 1 x 40	1 x 98 x 1 x 120	6,313	0	390,207	117,616	1.1863	462,911	5.084
DEPTHWISE_CONV_2	1 x 98 x 1 x 120	1 x 49 x 1 x 120	53,614	0	94,512	35,981	0.3573	33,766	1.899
CONV_2D	1 x 49 x 1 x 120	1 x 49 x 1 x 40	4,824	0	186,616	60,124	1.2286	229,273	2.454
CONV_2D	1 x 98 x 1 x 40	1 x 49 x 1 x 40	6,133	0	67,226	20,116	1.0687	71,846	0.941
ADD	1 x 49 x 1 x 40	1 x 49 x 1 x 40	4,035	0	6,872	2,933	0	0	0.14
CONV_2D	1 x 49 x 1 x 40	1 x 49 x 1 x 120	6,824	0	195,674	59,301	1.1615	227,274	2.596
DEPTHWISE_CONV_2	1 x 49 x 1 x 120	1 x 49 x 1 x 120	53,889	0	92,405	35,074	0.3617	33,426	1.876
CONV_2D	1 x 49 x 1 x 120	1 x 49 x 1 x 40	4,825	0	186,616	60,124	1.2286	229,272	2.454
ADD	1 x 49 x 1 x 40	1 x 49 x 1 x 40	4,010	0	6,872	2,933	0	0	0.14
CONV_2D	1 x 49 x 1 x 40	1 x 49 x 1 x 120	7,295	0	195,204	58,833	1.1615	226,727	2.596
DEPTHWISE_CONV_2	1 x 49 x 1 x 120	1 x 49 x 1 x 120	53,889	0	92,405	35,074	0.3617	33,426	1.876
CONV_2D	1 x 49 x 1 x 120	1 x 49 x 1 x 40	4,824	0	186,616	60,124	1.2286	229,273	2.454
ADD	1 x 49 x 1 x 40	1 x 49 x 1 x 40	4,010	0	6,872	2,933	0	0	0.14
CONV_2D	1 x 49 x 1 x 40	1 x 49 x 1 x 120	6,824	0	195,674	59,301	1.1615	227,274	2.596
DEPTHWISE_CONV_2	1 x 49 x 1 x 120	1 x 49 x 1 x 120	53,888	0	92,405	35,074	0.3617	33,427	1.876

- Perfetto trace tab: time-based execution and resource traces



(Zoomed in view)



> *NOTE:* The profiler currently tracks only the ARM Cortex®M MCU processor timeline. Usage and cycle information for the Matrix Vector Processor (MVP) is instead provided in the summary tab.

Running a Profiling Session from the CLI

1. Connect the board on which you want to profile your model. The board will be detected automatically, once connected.
2. Find the "device ID" of the connected board.
    1. Linux/macOS

    ```
    $ ~/.silabs/slt/installs/archive/sdm-darwin-arm64/sdm adapter list
    👉 Total adapter count: 1
    ↳ xxxxx [ usb wstk 440339411 xxxxx 127.0.0.1 ]
    ```

    2. Windows

    ```
    PS> $HOME\.silabs\slt\installs\archive\sdm-windows-amd64\sdm.exe adapter list
    👉 Total adapter count: 1
      ↳ xxxxx [ usb wstk 440339411 xxxxx 127.0.0.1 ]
    ```

    The device ID is "440339411".
3. Run Profiling

```
sml profile /path/to/model_name.tflite 440339411
```

> *NOTE:* See Usage for more command line arguments. See Troubleshooting section for handling

> any errors.

Output

The following is an example of the output you can expect to see on the command line terminal. Usernames and other sensitive information has been stubbed.

The log below includes:

- Inference time
- MCU vs MVP cycle breakdown
- Power consumption (planned feature)

You can access:

- text summary
- detailed JSON report

```
🔍 Checking SDM server status...
✅ SDM server is connected and accessible

🔍 Checking if device exists in connected adapters...
✅ Device "440339411" found in connected adapters

🔍 Checking board support...
✅ Board "BRD2608A" is supported

🔍 Checking model file...
✅ Model file found: /path/to/model_name.tflite (534336 bytes)
🚀 Starting ML Profiler...

SDM Service connected

🚀 Starting profiling workflow...
  Device: 440339411 (BRD2608A)
  Model:  model_name

📡 Step 1: Connecting to debug channel...
  Terminal socket opened
  Debug channel connected, ready to capture packets...

⚡ Step 2: Combining model with firmware and flashing...
  Using firmware: aiml_soc_profiler_firmware_efr32_brd2608a_mvp.s37
  Combined: model_name_combined.s37 (2447 KB)
  Flashing to 440339411...
  Firmware flashed successfully

📦 Step 3: Capturing packets and generating trace...
  Captured 313 packets, building trace...
  Decoded 681 packets, generated 648 trace events

╔═══════════════════════════════════════════════════════╗
║        ML PROFILER - PROFILING SUMMARY                ║
╚═══════════════════════════════════════════════════════╝

SESSION SUMMARY
──────────────────────────────────────────────────────────
 Model Name                    model_name
 Arena size                    82 KB
...more session summary...
 Board                         BRD2608A
...truncated for this documentation...
 Total number of MACs          6,115,960

PER-LAYER SUMMARY
──────────────────────────────────────────────────────────────────────────────────────────────────────────────
           |    Input     |    Output    |        MCU        |          Acc          |          Acc          |
-----------+--------------+--------------+-------------------+-----------------------+-----------------------+---------
Layer      | Shape        | Shape        | Cycles   | Stalls | Cycles   | Stalls     | Time(ms)
-----------+--------------+--------------+----------+--------+----------+------------+---------
CONV_2D    | 1 x 98 x 1 x 104 | 1 x 98 x 1 x 40  | 7,252    | 0      | 929,071  | 304,673    | 12.004
...more per-layer summary...
DEPTHWISE_CONV_2 | 1 x 98 x 1 x 120 | 1 x 49 x 1 x 120 | 53,617   | 0      | 94,539   | 36,008     | 1.899
...truncated for this documentation...
SOFTMAX    | 1 x 3        | 1 x 3        | 3,144    | 0      | 0        | 65         | 0.04
──────────────────────────────────────────────────────────────────────────────────────────────────────────────

──────────────────────────────────────────────────────────
Generated: YYYY-MM-DDTHH:MM:SS.SSSZ

📁 Profile data saved to:
  /path/to/profiling/data/model_name-YYYY-MM-DDTHH-MM-SSZ

  Includes:
```

```
• captured-packets.json (decoded packets)
  • report.json (profiling data)
  • summary.txt (readable summary)

  📄 See summary.txt for the complete profiling summary.

🌐 Step 4: Starting Perfetto UI server...
  Perfetto server started on port 10000

✅ Profiling completed successfully!

📊 Trace URL: http://localhost:10000#!/?
url=http%3A%2F%2Flocalhost%3A10000%2Ftrace%2FL1VzZXJzL3JhYW5zYXJpLy5zaWxhYnMvbWwtcHJvZmlsZXIva2V5d29yZF9zcG90dGlu

Use --gui to open GUI after profiling.
```

Press Ctrl+C to exit the profiling session.

Usage

```
sml --help
```

```
Usage: sml [options] [command]

Silicon Labs ML tooling.

Options:
  -v, --version                Show version and exit.
  --gui                        Open GUI after command completes (if supported).
  --dry-run                    Validate and print the effective configuration, but do not
                               execute the command.
  -h, --help                   display help for command

Commands:
  profile [options] <model> <device>  Profile a machine learning model on a Silicon Labs device. Emits
                               a Perfetto-compatible trace (.pftrace) or JSON summary (.json).
  version                      Show the version number
  help [command]               display help for command
```

Understanding Performance

The profiler presents a hierarchical execution view:

Inference → Layer → Operator → Kernel

Time-aligned tracks allow correlation between:

- MCU vs MVP execution
- memory usage
- clock rate
- power consumption

Idle time per kernel helps identify when:

- accelerator overhead dominates
- memory stalls occur
- MCU execution may be more efficient

Limitations

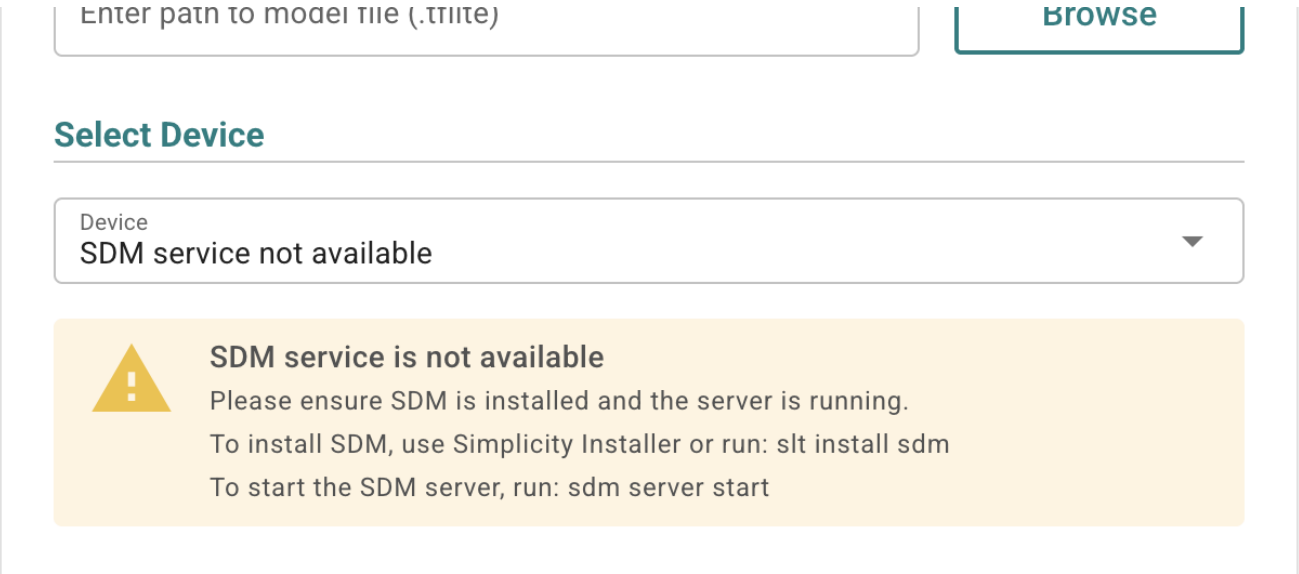
- Requires real Silicon Labs hardware, currently only supports xG24, xG26, and xG28 devices, specifically BRD2601B, BRD2608A, and BRD2705A dev kits.
- Simulator support is in development
- Does not auto-compare MCU vs MVP
- Does not measure model accuracy. This is not a target of this tool. It is geared exclusively towards execution performance analysis.

Summary

The Silicon Labs Machine Learning Model Profiler helps engineers reason about embedded ML performance by making execution behavior visible, comparable, and intuitive.

Troubleshooting

"SDM Service is not available" warning



Solution

1. Verify if Simplicity Device Manager (SDM) is installed using, `slt locate sdm`.
2. If you see no output on the console, install SDM using:
    1. `slt install sdm` through the CLI, or
    2. Simplicity Installer by following the steps mentioned in the Install using Simplicity Installer section. Search for "Simplicity Device Manager" instead of "ML Profiler".
3. Start SDM server.
    1. Linux/macOS

        ```
        ~/.silabs/slt/installs/archive/sdm-darwin-arm64/sdm server start
        ```

    2. Windows

        ```
        PS> $HOME\.silabs\slt\installs\archive\sdm-windows-amd64\sdm.exe server start
        ```

"No devices connected" message in the "Select Device" field

Solution

Connect the desired board on which you want to profile your models.

Any type of "Firmware preparation/flashing failed: Failed to combine model with firmware" error

Examples of this type of error:

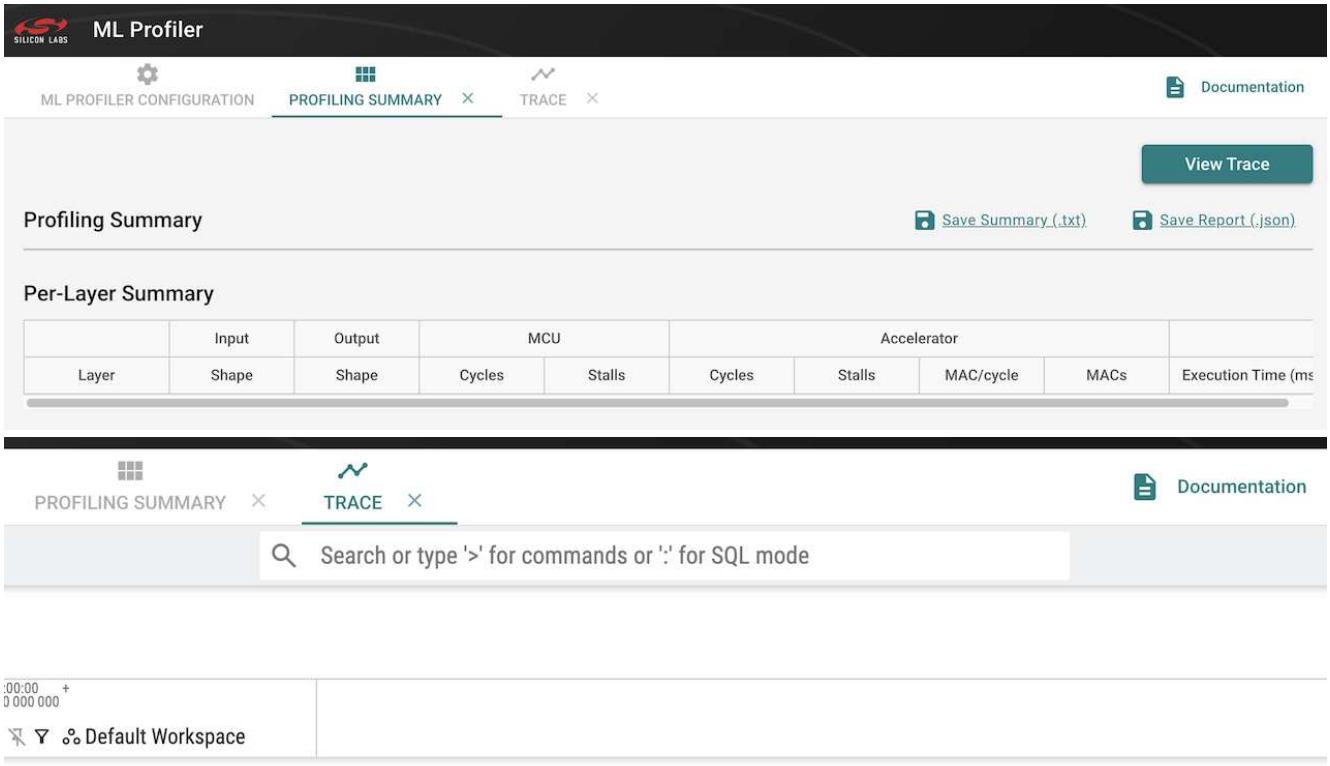
1. Firmware preparation/flashing failed: Failed to combine model with firmware: 404 Not Found: Not Found
2. Firmware preparation/flashing failed: Failed to combine model with firmware: Combine binary job failed: Error: Could not find function simpleCombineConvertBinaries. It is either typed wrong, you miss an adapter pack, or you need to upgrade one.

Solution

1. This issue is most commonly caused due to an older version of either Simplicity Device Manager or Simplicity Commander.
2. Update Simplicity Commander to v1.22+. Install using `slt install commander`. Verify using `slt locate commander`.
3. Update Simplicity Device Manager to v0.101.4+. Install using `slt install sdm`. Verify using `slt locate sdm`.

No Profiling Output

GUI



Empty workspace

CLI

```
📦 Step 3: Capturing packets and generating trace...
  Captured 43 packets, building trace...
  Decoded 0 packets, generated 0 trace events

╔═══════════════════════════════════════════════════════════╗
║          ML PROFILER - PROFILING SUMMARY                  ║
╚═══════════════════════════════════════════════════════════╝

──────────────────────────────────────────────────────────────
```

Solution

1. Verify Simplicity Device Manager Service is up: `sdm server status`. If not, invoke `sdm server start`.
2. Find the device ID, see Running a Profiling Session From the CLI section.
3. Jump into admin console of your device: `sdm terminal -a <device_id> -c admin`.

4. Verify the debug message version: `dch message version` , If the output is `Message protocol version : 3` , use Step 5 below.
5. Invoke `dch message version 2` . The output must show `Current version = 2` .
6. Execute a Profiling session again.

Machine Learning API Reference

Machine Learning API Reference

List of modules	Description
TensorFlow Lite Micro Init	Initialize the Tensorflow Lite Micro Runtime.
TensorFlow Lite Micro Debug	Additional SL utilities for logging in TensorFlow Lite Micro.
Audio Feature Generator	Extracts mel-filterbank features from an audio signal to use with machine learning audio classification applications.
Model Specific Functions	Functions generated using the name of the .tflite model file. (Currently usable only with SiWx917 platform)
Model Specific Variables and Constants	Model specific variables and constants generated using the name of the .tflite model file. (Currently, usable only with SiWx917 platform)

Microphone I2S Driver for Machine Learning

Microphone I2S Driver for Machine Learning

Microphone driver API for I2S interface on SI91x.

Typedefs

typedef void(* sl_mic_buffer_ready_callback_t)(const void *buffer, uint32_t n_frames)
Callback function indicating that the sample buffer is ready.

Functions

sl_status_t sl_ml_mic_init(uint32_t sample_rate, uint8_t channels)
Initialize the microphone.

sl_status_t sl_ml_mic_deinit(void)
De-initialize the microphone.

sl_status_t sl_ml_mic_start_streaming(void *buffer, uint32_t n_frames, sl_mic_buffer_ready_callback_t callback)
Read samples from the microphone into a sample buffer continuously.

sl_status_t sl_ml_mic_stop(void)
Stop the microphone.

Typedef Documentation

sl_mic_buffer_ready_callback_t

```
typedef void(* sl_mic_buffer_ready_callback_t) (const void *buffer, uint32_t n_frames) )(const void *buffer, uint32_t n_frames)
```

Callback function indicating that the sample buffer is ready.

Parameters

Type	Direction	Argument Name	Description
	[in]	buffer	Pointer to the sample buffer.
	[in]	n_frames	Number of audio frames in the sample buffer.

Returns

- None.

Function Documentation

sl_ml_mic_init

> sl_status_t sl_ml_mic_init (uint32_t sample_rate, uint8_t channels)

Initialize the microphone.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	sample_rate	The desired sample rate in Hz
uint8_t	[in]	channels	Number of audio channels (1 or 2)

Returns

- Returns SL_STATUS_OK on success, non-zero otherwise

sl_ml_mic_deinit

> sl_status_t sl_ml_mic_deinit (void)

De-initialize the microphone.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_ml_mic_start_streaming

> sl_status_t sl_ml_mic_start_streaming (void * buffer, uint32_t n_frames, sl_mic_buffer_ready_callback_t callback)

Read samples from the microphone into a sample buffer continuously.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	buffer	Pointer to the sample buffer to store the data. 16-bit channel data is stored consecutively, starting with ch0. This buffer shall be big enough to hold twice the n_frames because of the ping-pong operation.
uint32_t	[in]	n_frames	The number of audio frames to receive before the callback is called. Maximum value limited by DMADRV_MAX_XFER_COUNT.
sl_mic_buffer_ready_callback_t	[in]	callback	Callback is called when n_frames in the sample buffer is ready.

This function starts the microphone sampling and stops only upon calling sl_ml_mic_stop or sl_ml_mic_deinit. The buffer is used in a "ping-pong" manner meaning that one half of the buffer is used for sampling while the other half is being processed.

sl_ml_mic_stop

```
sl_status_t sl_ml_mic_stop (void )
```

Stop the microphone.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Model Specific Functions

Model Specific Functions

Functions generated using the name of the .tflite model file. These APIs are currently usable only with the SiWx917 platform.

sl_ml_<model_name>_init

`sl_status_t sl_ml_<model_name>_model_init(void)`

Create the error reporter and opcode resolver and initialize variables for the given model. `<model_name>` is derived from the file name of your `.tflite` file.

Returns

- Status of model initialization. `SL_STATUS_OK` if successful, otherwise, `SL_STATUS_FAIL` .

sl_ml_<model_name>_run

`sl_status_t sl_ml_<model_name>_model_run(void)`

Execute the model once for inference. The output is updated internally in the model.

Returns

- Status of model execution and inference. `SL_STATUS_OK` if successful, otherwise, `SL_STATUS_FAIL` .

Model Specific Variables and Constants

Model Specific Variables and Constants

Model specific variables and constants generated using the name of the .tflite model file. These APIs are currently usable only with the SiWx917 platform.

<model_name>_model

`static TfliteMicroModel <model_name>_model`

Model object with a plethora of methods to interact with the model.

<model_name>_model_status

`static sl_status_t <model_name>_model_status`

Tracks the status of `<model_name>_model` when initialized and executed.

<model_name>_model_flatbuffer

`uint8_t* <model_name>_model_flatbuffer`

Pointer to the raw flatbuffer of the model referred to by `<model_name>_model`

<model_name>_model_flatbuffer_length

`const int <model_name>_model_flatbuffer_length`

Length of the raw flatbuffer referred to by `<model_name>_model_flatbuffer`

IMU - Inertial Measurement Unit

IMU - Inertial Measurement Unit

Inertial Measurement Unit driver.

IMU example code

Basic example for looping measurement of orientation data:

```
#include "sl_imu.h"

int main( void )
{

 ...

 int16_t o_vec[3];

 sl_imu_init();

 // Configure sample rate
 sl_imu_configure(10);

 // Recalibrate gyro
 sl_imu_calibrate_gyro();

 while (true) {

  sl_imu_update();

  while (!sl_imu_is_data_ready()) {
   // wait
  }

  sl_imu_get_orientation(o_vec);

  ...

 }
}
```

Modules

IMU Fusion

Vector and Matrix Math

Direction Cosine Matrix

State Definitions

#define IMU_STATE_DISABLED 0x00
The IMU is disabled

#define IMU_STATE_READY 0x01
The IMU is fully configured and ready to take measurements

#define IMU_STATE_INITIALIZING 0x02
The IMU is being initialized

#define IMU_STATE_CALIBRATING 0x03
The IMU is being calibrated

Functions

sl_status_t sl_imu_init(sl_ssi_handle_t ssi_driver_handle)
Initialize and calibrate the IMU chip.

sl_status_t sl_imu_deinit(void)
De-initialize the IMU chip.

uint8_t sl_imu_get_state(void)
Return IMU state.

void sl_imu_update(void)
Get a new set of data from the accel and gyro sensor and updates the fusion calculation.

void sl_imu_reset(void)
Reset the fusion calculation.

void sl_imu_get_acceleration(int16_t avec[3])
Retrieve the processed acceleration data.

void sl_imu_get_orientation(int16_t ovec[3])
Retrieve the processed orientation data.

void sl_imu_get_gyro(int16_t gvec[3])
Retrieve the processed gyroscope data.

sl_status_t sl_imu_calibrate_gyro(sl_ssi_handle_t ssi_driver_handle)
Perform gyroscope calibration to cancel gyro bias.

void sl_imu_get_gyro_correction_angles(float acorr[3])
Retrieve the processed gyroscope correction angles.

void sl_imu_configure(float sampleRate, sl_ssi_handle_t ssi_driver_handle)
Configure the IMU.

bool sl_imu_is_data_ready(sl_ssi_handle_t ssi_driver_handle)
Check if new accel/gyro data is available for read.

void sl_imu_get_acceleration_raw_data(float avec[3])
Retrieve the raw acceleration data from the IMU.

void sl_imu_get_gyro_raw_data(float gvec[3])
Retrieve the raw gyroscope data from the IMU.

State Definitions Documentation

Function Documentation

sl_imu_init

> sl_status_t sl_imu_init (sl_ssi_handle_t ssi_driver_handle)

Initialize and calibrate the IMU chip.

Parameters

Type	Direction	Argument Name	Description
sl_ssi_handle_t	N/A	ssi_driver_handle	

Returns

- Returns zero on OK, non-zero otherwise

sl_imu_deinit

> sl_status_t sl_imu_deinit (void)

De-initialize the IMU chip.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns zero on OK, non-zero otherwise

sl_imu_get_state

> uint8_t sl_imu_get_state (void)

Return IMU state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- IMU state

sl_imu_update

> void sl_imu_update (void)

Get a new set of data from the accel and gyro sensor and updates the fusion calculation.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_imu_reset

```
void sl_imu_reset (void )
```

Reset the fusion calculation.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_imu_get_acceleration

```
void sl_imu_get_acceleration (int16_t avec)
```

Retrieve the processed acceleration data.

Parameters

Type	Direction	Argument Name	Description
int16_t	[out]	avec	Three dimensional acceleration vector

sl_imu_get_orientation

```
void sl_imu_get_orientation (int16_t ovec)
```

Retrieve the processed orientation data.

Parameters

Type	Direction	Argument Name	Description
int16_t	[out]	ovec	Three dimensional orientation vector

sl_imu_get_gyro

```
void sl_imu_get_gyro (int16_t gvec)
```

Retrieve the processed gyroscope data.

Parameters

Type	Direction	Argument Name	Description
int16_t	[out]	gvec	Three dimensional gyro vector

sl_imu_calibrate_gyro

```
sl_status_t sl_imu_calibrate_gyro (sl_ssi_handle_t ssi_driver_handle)
```

Perform gyroscope calibration to cancel gyro bias.

Parameters

Type	Direction	Argument Name	Description
sl_ssi_handle_t	N/A	ssi_driver_handle	

sl_imu_get_gyro_correction_angles

```
void sl_imu_get_gyro_correction_angles (float acorr)
```

Retrieve the processed gyroscope correction angles.

Parameters

Type	Direction	Argument Name	Description
float	[out]	acorr	Three dimensional gyro correction angle vector

sl_imu_configure

```
void sl_imu_configure (float sampleRate, sl_ssi_handle_t ssi_driver_handle)
```

Configure the IMU.

Parameters

Type	Direction	Argument Name	Description
float	[in]	sampleRate	The desired sample rate of the acceleration and gyro sensor
sl_ssi_handle_t	N/A	ssi_driver_handle	

sl_imu_is_data_ready

```
bool sl_imu_is_data_ready (sl_ssi_handle_t ssi_driver_handle)
```

Check if new accel/gyro data is available for read.

Parameters

Type	Direction	Argument Name	Description
sl_ssi_handle_t	N/A	ssi_driver_handle	

Returns

- True if the measurement data is ready, false otherwise

sl_imu_get_acceleration_raw_data

```
void sl_imu_get_acceleration_raw_data (float avec)
```

Retrieve the raw acceleration data from the IMU.

Parameters

Type	Direction	Argument Name	Description
float	[out]	avec	Three dimensional raw acceleration vector

sl_imu_get_gyro_raw_data

```
void sl_imu_get_gyro_raw_data (float gvec)
```

Retrieve the raw gyroscope data from the IMU.

Parameters

Type	Direction	Argument Name	Description
float	[out]	gvec	Three dimensional raw gyro vector

IMU Fusion

IMU Fusion

IMU fusion driver.

Modules

sl_imu_sensor_fusion

Typedefs

typedef struct sl_imu_sensor_fusion **sl_imu_sensor_fusion_t**
Structure to store the sensor fusion data.

Functions

void **sl_imu_fuse_accelerometer_set_sample_rate**(sl_imu_sensor_fusion_t *f, float rate)
Set the accelerometer sample rate in the sl_imu_sensor_fusion_t structure.

void **sl_imu_fuse_accelerometer_update_filter**(sl_imu_sensor_fusion_t *f, float avec[3])
Add the current accelerometer data to the accumulator.

void **sl_imu_fuse_gyro_set_sample_rate**(sl_imu_sensor_fusion_t *f, float rate)
Set the gyro sample rate and related values in the sl_imu_sensor_fusion_t structure.

void **sl_imu_fuse_gyro_update**(sl_imu_sensor_fusion_t *f, float gvec[3])
Update the fusion calculation with a new gyro data.

void **sl_imu_fuse_gyro_clear_correction_vector**(sl_imu_sensor_fusion_t *f)
Clear the gyro correction vector.

void **sl_imu_fuse_gyro_calculate_correction_vector**(sl_imu_sensor_fusion_t *f, bool accValid, bool dirValid, float dirZ)
Calculate the gyro correction vector.

void **sl_imu_fuse_new**(sl_imu_sensor_fusion_t *f)
Initialize a new sl_imu_sensor_fusion_t structure.

void **sl_imu_fuse_reset**(sl_imu_sensor_fusion_t *f)
Clear the values of the sensor fusion object.

void **sl_imu_fuse_update**(sl_imu_sensor_fusion_t *f)
Update the fusion calculation.

Typedef Documentation

sl_imu_sensor_fusion_t

```
typedef struct sl_imu_sensor_fusion sl_imu_sensor_fusion_t
```

Structure to store the sensor fusion data.

Function Documentation

sl_imu_fuse_accelerometer_set_sample_rate

> void sl_imu_fuse_accelerometer_set_sample_rate (sl_imu_sensor_fusion_t * f, float rate)

Set the accelerometer sample rate in the sl_imu_sensor_fusion_t structure.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object
float	[in]	rate	Sample rate of the accelerometer

sl_imu_fuse_accelerometer_update_filter

> void sl_imu_fuse_accelerometer_update_filter (sl_imu_sensor_fusion_t * f, float avec)

Add the current accelerometer data to the accumulator.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object
float	[in]	avec	Accelerometer vector

sl_imu_fuse_gyro_set_sample_rate

> void sl_imu_fuse_gyro_set_sample_rate (sl_imu_sensor_fusion_t * f, float rate)

Set the gyro sample rate and related values in the sl_imu_sensor_fusion_t structure.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object
float	[in]	rate	Sample rate of the gyroscope

sl_imu_fuse_gyro_update

> void sl_imu_fuse_gyro_update (sl_imu_sensor_fusion_t * f, float gvec)

Update the fusion calculation with a new gyro data.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object
float	[in]	gvec	Gyroscope vector

sl_imu_fuse_gyro_clear_correction_vector

> void sl_imu_fuse_gyro_clear_correction_vector (sl_imu_sensor_fusion_t * f)

Clear the gyro correction vector.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object

sl_imu_fuse_gyro_calculate_correction_vector

> void sl_imu_fuse_gyro_calculate_correction_vector (sl_imu_sensor_fusion_t * f, bool accValid, bool dirValid, float dirZ)

Calculate the gyro correction vector.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object
bool	[in]	accValid	True if the acceleration value is within limits
bool	[in]	dirValid	True if the direction value is valid
float	[in]	dirZ	The direction of the Z axis

sl_imu_fuse_new

> void sl_imu_fuse_new (sl_imu_sensor_fusion_t * f)

Initialize a new sl_imu_sensor_fusion_t structure.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object to be initialized

sl_imu_fuse_reset

> void sl_imu_fuse_reset (sl_imu_sensor_fusion_t * f)

Clear the values of the sensor fusion object.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object

sl_imu_fuse_update

```
void sl_imu_fuse_update (sl_imu_sensor_fusion_t * f)
```

Update the fusion calculation.

Parameters

Type	Direction	Argument Name	Description
sl_imu_sensor_fusion_t *	[inout]	f	Pointer to the sl_imu_sensor_fusion_t object

sl_imu_sensor_fusion

Structure to store the sensor fusion data.

Public Attributes

float	dcm
float	aVector
float	aAccumulator
uint32_t	aAccumulatorCount
float	aSampleRate
float	gVector
float	gSampleRate
float	gDeltaTime
float	gDeltaTimeScale
float	angleCorrection
float	orientation

Public Attribute Documentation

dcm

```
float sl_imu_sensor_fusion::dcm[3][3]
```

Direction Cosine Matrix

aVector

```
float sl_imu_sensor_fusion::aVector[3]
```

Acceleration vector

aAccumulator

```
float sl_imu_sensor_fusion::aAccumulator[3]
```

Accumulator for acceleration vector

aAccumulatorCount

```
uint32_t sl_imu_sensor_fusion::aAccumulatorCount
```

Number of vectors stored in the accumulator

aSampleRate

```
float sl_imu_sensor_fusion::aSampleRate
```

Acceleration measurement sample rate

gVector

```
float sl_imu_sensor_fusion::gVector[3]
```

Gyro vector

gSampleRate

```
float sl_imu_sensor_fusion::gSampleRate
```

Gyroscope measurement sample rate

gDeltaTime

```
float sl_imu_sensor_fusion::gDeltaTime
```

Time between gyro samples

gDeltaTimeScale

```
float sl_imu_sensor_fusion::gDeltaTimeScale
```

Rotation between gyro samples

angleCorrection

```
float sl_imu_sensor_fusion::angleCorrection[3]
```

Angle correction vector

orientation

```
float sl_imu_sensor_fusion::orientation[3]
```

Orientation vector

Vector and Matrix Math

Vector and Matrix Math

Inertial measurement unit fusion driver math routines.

Functions

void sl_imu_normalize_angle(float *a)
Normalize the angle (-PI < angle <= PI)

void sl_imu_matrix_multiply(float c[3][3], float a[3][3], float b[3][3])
Multiply two 3x3 matrices.

void sl_imu_vector_normalize_angle(float v[3])
Normalize the angle of a vector.

void sl_imu_vector_zero(float v[3])
Set all elements of a vector to 0.

void sl_imu_vector_scale(float v[3], float scale)
Scale a vector by a factor.

void sl_imu_vector_scalar_multiplication(float r[3], float v[3], float scale)
Multiply a vector by a scalar.

void sl_imu_vector_add(float r[3], float a[3], float b[3])
Add two vectors.

void sl_imu_vector_subtract(float r[3], float a[3], float b[3])
Subtract vector b from vector a.

float sl_imu_vector_dot_product(float a[3], float b[3])
Calculate the dot product of two vectors.

void sl_imu_vector_cross_product(float r[3], float a[3], float b[3])
Calculate the cross product of two vectors.

Function Documentation

sl_imu_normalize_angle

```
void sl_imu_normalize_angle (float * a)
```

Normalize the angle (-PI < angle <= PI)

Parameters

Type	Direction	Argument Name	Description
float *	N/A	a	The angle to be normalized

sl_imu_matrix_multiply

> void sl_imu_matrix_multiply (float c, float a, float b)

Multiply two 3x3 matrices.

Parameters

Type	Direction	Argument Name	Description
float	[out]	c	The multiplication result, AB
float	[in]	a	Input vector A
float	[in]	b	Input vector B

sl_imu_vector_normalize_angle

> void sl_imu_vector_normalize_angle (float v)

Normalize the angle of a vector.

Parameters

Type	Direction	Argument Name	Description
float	N/A	v	The vector, which contains angles to be normalized

sl_imu_vector_zero

> void sl_imu_vector_zero (float v)

Set all elements of a vector to 0.

Parameters

Type	Direction	Argument Name	Description
float	N/A	v	The vector to be cleared

sl_imu_vector_scale

> void sl_imu_vector_scale (float v, float scale)

Scale a vector by a factor.

Parameters

Type	Direction	Argument Name	Description
float	N/A	v	The vector to be scaled
float	[in]	scale	The scale factor

sl_imu_vector_scalar_multiplication

> void sl_imu_vector_scalar_multiplication (float r, float v, float scale)

Multiply a vector by a scalar.

Parameters

Type	Direction	Argument Name	Description
float	[out]	r	The multiplied vector
float	[in]	v	The vector to be multiplied
float	[in]	scale	The scalar multiplicator value

sl_imu_vector_add

> void sl_imu_vector_add (float r, float a, float b)

Add two vectors.

Parameters

Type	Direction	Argument Name	Description
float	[out]	r	The vectorial sum of the vectors, a+b
float	[in]	a	The first vector
float	[in]	b	The second vector

sl_imu_vector_subtract

> void sl_imu_vector_subtract (float r, float a, float b)

Subtract vector b from vector a.

Parameters

Type	Direction	Argument Name	Description
float	[out]	r	The vectorial difference, a-b
float	[in]	a	Vector a
float	[in]	b	Vector b

sl_imu_vector_dot_product

> float sl_imu_vector_dot_product (float a, float b)

Calculate the dot product of two vectors.

Parameters

Type	Direction	Argument Name	Description

Type	Direction	Argument Name	Description
float	[in]	b	The second vector

Returns

- The dot product

sl_imu_vector_cross_product

> void sl_imu_vector_cross_product (float r, float a, float b)

Calculate the cross product of two vectors.

Parameters

Type	Direction	Argument Name	Description
float	[out]	r	The cross product
float	[in]	a	The first vector
float	[in]	b	The second vector

Direction Cosine Matrix

Direction Cosine Matrix

Unit DCM matrix related routines.

Functions

void sl_imu_dcm_reset(float dcm[3][3])
Set the elements of the DCM matrix to the corresponding elements of the identity matrix.

void sl_imu_dcm_reset_z(float dcm[3][3])
DCM reset, Z direction.

void sl_imu_dcm_normalize(float dcm[3][3])
Normalize the DCM matrix.

void sl_imu_dcm_rotate(float dcm[3][3], float ang[3])
Rotate the DCM matrix by a given angle.

void sl_imu_dcm_get_angles(float dcm[3][3], float ang[3])
Calculate the Euler angles (roll, pitch, yaw) from the DCM matrix.

Function Documentation

sl_imu_dcm_reset

> void sl_imu_dcm_reset (float dcm)

Set the elements of the DCM matrix to the corresponding elements of the identity matrix.

Parameters

Type	Direction	Argument Name	Description
float	N/A	dcm	DCM matrix

sl_imu_dcm_reset_z

> void sl_imu_dcm_reset_z (float dcm)

DCM reset, Z direction.

Parameters

Type	Direction	Argument Name	Description
float	N/A	dcm	DCM matrix

sl_imu_dcm_normalize

> void sl_imu_dcm_normalize (float dcm)

Normalize the DCM matrix.

Parameters

Type	Direction	Argument Name	Description
float	N/A	dcm	DCM matrix

sl_imu_dcm_rotate

> void sl_imu_dcm_rotate (float dcm, float ang)

Rotate the DCM matrix by a given angle.

Parameters

Type	Direction	Argument Name	Description
float	[inout]	dcm	DCM matrix
float	[in]	ang	Rotation angle

sl_imu_dcm_get_angles

> void sl_imu_dcm_get_angles (float dcm, float ang)

Calculate the Euler angles (roll, pitch, yaw) from the DCM matrix.

Parameters

Type	Direction	Argument Name	Description
float	[in]	dcm	DCM matrix
float	[out]	ang	An array containing the Euler angles

Audio Feature Generator

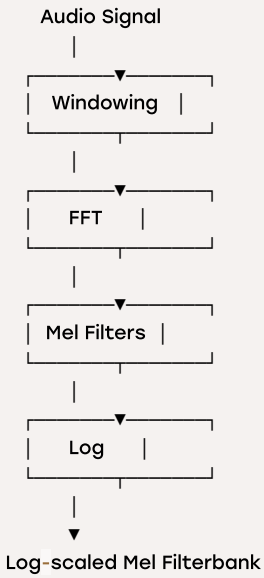
Audio Feature Generator

The audio feature generator extracts mel-filterbank features from an audio signal to use with machine learning audio classification applications using a microphone as an audio source.

Feature Generation

The Mel scale replicates the behavior of the human ear, which has a higher resolution for lower frequencies and is less discriminative of the higher frequencies. To create a mel filterbank, a number of filters are applied to the signal, where the pass-band of the lower channel filters is narrow and increases towards higher frequencies.

The audio signal is split into short overlapping segments using a window function (Hamming). The Fast Fourier Transform (FFT) is applied to each segment to retrieve the frequency spectrum and then the power spectrum of the segment. The filterbank is created by applying a series of mel-scaled filters to the output. Finally, the log is applied to the output to increase the sensitivity between the lower channels.



The feature array is generated by stacking filterbanks of sequential segments together to form a spectrogram. The array is sorted such that the first element is the first channel of the oldest filterbank.

Usage

sl_ml_audio_feature_generation_init() initializes the frontend for feature generation based on the configuration in sl_ml_audio_feature_generation_config.h. It also initializes and starts the microphone in streaming mode, which places the audio samples into a ring-buffer.

If used together with the Flatbuffer Converter Tool and a compatible TensorFlow Lite model, the configuration is pulled from the TensorFlow Lite model by default. Set the configuration option `SL_ML_AUDIO_FEATURE_GENERATION_MANUAL_CONFIG_ENABLE` to override this behavior and use manually-configured options from the configuration header.

The features are generated when sl_ml_audio_feature_generation_update_features() is called. The feature generator then updates the features for as many new segments of audio as possible, starting from the last

time the function was called up until the current time. The new features are appended to the feature buffer, replacing the oldest features such that the feature array always contains the most up to date features.

Note that if the audio buffer is not large enough to hold all audio samples required to generate features between calls to sl_ml_audio_feature_generation_update_features(), audio data will simply be overwritten. The generator will not return an error. The audio buffer must therefore be configured to be large enough to store all new sampled data between updating features.

To retrieve the generated features, either sl_ml_audio_feature_generation_get_features_raw(), sl_ml_audio_feature_generation_get_features_quantized(), or sl_ml_audio_feature_generation_fill_tensor() must be called.

Example

When used with TensorFlow Lite Micro, the audio feature generator can be used to fill a tensor directly by using sl_ml_audio_feature_generation_fill_tensor(). However, the model has to be trained using the same feature generator configurations as used for inference, configured in sl_ml_audio_feature_generation_config.h.

```
#include "sl_tflite_micro_init.h"
#include "sl_ml_audio_feature_generation.h"

void main(void)
{
 sl_ml_audio_feature_generation_init();

 while(1){
  sl_ml_audio_feature_generation_update_features();

  if(do_inference){
   sl_ml_audio_feature_generation_fill_tensor(sl_tflite_micro_get_input_tensor());
   sl_tflite_micro_get_interpreter()->Invoke();
  }

  ...

 }
}
```

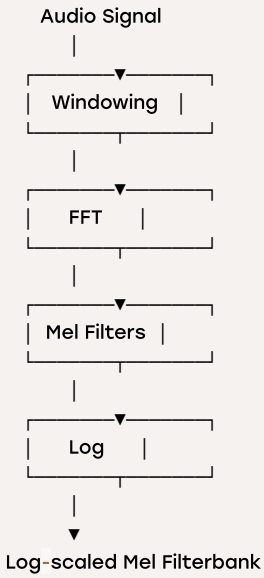
Note that updating features and retrieving them can be performed independently. Updating features should be done often enough to avoid overwriting the audio buffer while retrieving them only needs to be done prior to inference.

The audio feature generator extracts mel-filterbank features from an audio signal to use with machine learning audio classification applications using a microphone as an audio source.

Feature Generation

The Mel scale replicates the behavior of the human ear, which has a higher resolution for lower frequencies and is less discriminative of the higher frequencies. To create a mel filterbank, a number of filters are applied to the signal, where the pass-band of the lower channel filters is narrow and increases towards higher frequencies.

The audio signal is split into short overlapping segments using a window function (Hamming). The Fast Fourier Transform (FFT) is applied to each segment to retrieve the frequency spectrum and then the power spectrum of the segment. The filterbank is created by applying a series of mel-scaled filters to the output. Finally, the log is applied to the output to increase the sensitivity between the lower channels.



The feature array is generated by stacking filterbanks of sequential segments together to form a spectrogram. The array is sorted such that the first element is the first channel of the oldest filterbank.

Usage

sl_ml_audio_feature_generation_init() initializes the frontend for feature generation based on the configuration in sl_ml_audio_feature_generation_config_si91x.h. It also initializes and starts the microphone in streaming mode, which places the audio samples into a ring-buffer.

If used together with the Flatbuffer Converter Tool and a compatible TensorFlow Lite model, the configuration is pulled from the TensorFlow Lite model by default. Set the configuration option `SL_ML_AUDIO_FEATURE_GENERATION_MANUAL_CONFIG_ENABLE` to override this behavior and use manually-configured options from the configuration header.

The features are generated when sl_ml_audio_feature_generation_update_features() is called. The feature generator then updates the features for as many new segments of audio as possible, starting from the last time the function was called up until the current time. The new features are appended to the feature buffer, replacing the oldest features such that the feature array always contains the most up to date features.

Note that if the audio buffer is not large enough to hold all audio samples required to generate features between calls to sl_ml_audio_feature_generation_update_features(), audio data will simply be overwritten. The generator will not return an error. The audio buffer must therefore be configured to be large enough to store all new sampled data between updating features.

To retrieve the generated features, either sl_ml_audio_feature_generation_get_features_raw(), sl_ml_audio_feature_generation_get_features_quantized(), or sl_ml_audio_feature_generation_fill_tensor() must be called.

Example

When used with TensorFlow Lite Micro, the audio feature generator can be used to fill a tensor directly by using sl_ml_audio_feature_generation_fill_tensor(). However, the model has to be trained using the same feature generator configurations as used for inference, configured in sl_ml_audio_feature_generation_config_si91x.h.

```
#include "sl_tflite_micro_init.h"
#include "sl_ml_audio_feature_generation_si91x.h"

void main(void)
{
 sl_ml_audio_feature_generation_init();

 while(1){
  sl_ml_audio_feature_generation_update_features();

  if(do_inference){
   sl_ml_audio_feature_generation_fill_tensor(sl_tflite_micro_get_input_tensor());
   sl_tflite_micro_get_interpreter()->Invoke();
  }

  ...

 }
}
```

Note that updating features and retrieving them can be performed independently. Updating features should be done often enough to avoid overwriting the audio buffer while retrieving them only needs to be done prior to inference.

Typedefs

typedef void(* sl_ml_audio_feature_generation_mic_callback_t)(void *arg, const int16_t *data, uint32_t n_frames)
Microphone callback function type for audio feature generation.

Variables

int16_t * sl_ml_audio_feature_generation_audio_buffer
External audio buffer for audio feature generation.

Functions

sl_status_t sl_ml_audio_feature_generation_init()
Set up the microphone as an audio source for feature generation and initialize the frontend for feature generation.

sl_status_t sl_ml_audio_feature_generation_frontend_init()
Initialize microfrontend according to the configuration in sl_ml_audio_feature_generation_config.h.

sl_status_t sl_ml_audio_feature_generation_update_features()
Update the feature buffer with the missing feature slices since the last call to this function.

sl_status_t sl_ml_audio_feature_generation_get_features_raw(uint16_t *buffer, size_t num_elements)
Retrieve the features as type uint16 and copy them to the provided buffer.

sl_status_t sl_ml_audio_feature_generation_fill_tensor(TfLiteTensor *input_tensor)
Fill a TensorFlow tensor with feature data of type int8.

int sl_ml_audio_feature_generation_get_new_feature_slice_count()
Return the number of new or unfetched feature slices that have been updated since the last call to sl_ml_audio_feature_generation_get_features_raw or sl_ml_audio_feature_generation_fill_tensor.

int sl_ml_audio_feature_generation_get_feature_buffer_size()
Return the feature buffer size.

void	sl_ml_audio_feature_generation_reset() Reset the state of the audio feature generator.
sl_status_t	sl_ml_audio_feature_generation_init_with_buffer(int16_t *buffer, int n_frames) Initialize audio feature generation with a user-provided buffer.
sl_status_t	sl_ml_audio_feature_generation_get_features_raw_float32(float *buffer, size_t num_elements) Retrieve the features as type float32 and copy them to the provided buffer.
sl_status_t	sl_ml_audio_feature_generation_get_features_scaled(float *buffer, size_t num_elements, float scaler) Retrieve the features, scales them by the given scaler, and fills float buffer.
sl_status_t	sl_ml_audio_feature_generation_get_features_mean_std_normalized(float *buffer, size_t num_elements) Retrieve the features, normalizes them by centering about their mean and standard deviation and fills float buffer.
sl_status_t	sl_ml_audio_feature_generation_activity_detected() Return if the activity detection block detected activity in the audio stream.
sl_status_t	sl_ml_audio_feature_generation_set_mic_callback(sl_ml_audio_feature_generation_mic_callback_t callback, void *arg) Set the microphone callback for audio feature generation.

Typedef Documentation

sl_ml_audio_feature_generation_mic_callback_t

```
typedef void(* sl_ml_audio_feature_generation_mic_callback_t) (void *arg, const int16_t *data, uint32_t n_frames) )(void *arg, const int16_t *data, uint32_t n_frames)
```

Microphone callback function type for audio feature generation.

Parameters

Type	Direction	Argument Name	Description
	N/A	arg	User-defined argument passed to the callback.
	N/A	data	Pointer to the audio data buffer.
	N/A	n_frames	Number of audio frames in the buffer.

Variable Documentation

sl_ml_audio_feature_generation_audio_buffer

```
int16_t* sl_ml_audio_feature_generation_audio_buffer
```

External audio buffer for audio feature generation.

Function Documentation

sl_ml_audio_feature_generation_init

```
sl_status_t sl_ml_audio_feature_generation_init ()
```

Set up the microphone as an audio source for feature generation and initialize the frontend for feature generation.

Returns

- SL_STATUS_OK for success SL_STATUS_FAIL

sl_ml_audio_feature_generation_frontend_init

```
sl_status_t sl_ml_audio_feature_generation_frontend_init ()
```

Initialize microfrontend according to the configuration in sl_ml_audio_feature_generation_config.h.

Initialize microfrontend according to the configuration in sl_ml_audio_feature_generation_config_si91x.h.

Returns

- SL_STATUS_OK for success SL_STATUS_FAIL

sl_ml_audio_feature_generation_update_features

```
sl_status_t sl_ml_audio_feature_generation_update_features ()
```

Update the feature buffer with the missing feature slices since the last call to this function.

To retrieve the features, call sl_ml_audio_feature_generation_get_features_raw or sl_ml_audio_feature_generation_fill_tensor.

Note

- This function needs to be called often enough to ensure that the audio buffer isn't overwritten.

Returns

- SL_STATUS_OK for success SL_STATUS_EMPTY No new slices were calculated

sl_ml_audio_feature_generation_get_features_raw

```
sl_status_t sl_ml_audio_feature_generation_get_features_raw (uint16_t * buffer, size_t num_elements)
```

Retrieve the features as type uint16 and copy them to the provided buffer.

Parameters

Type	Direction	Argument Name	Description
uint16_t *	[out]	buffer	Pointer to the buffer to store the feature data

Note

- This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_audio_feature_generation_fill_tensor

```
sl_status_t sl_ml_audio_feature_generation_fill_tensor (TfLiteTensor * input_tensor)
```

Fill a TensorFlow tensor with feature data of type int8.

Parameters

Type	Direction	Argument Name	Description
TfLiteTensor *	[in]	input_tensor	The input tensor to fill with features.

The int8 values are derived by quantizing the microfrontend output, expected to be in the range 0 to 670, to signed integer numbers in -128 to 127 range.

Note

- This function overwrites the entire input tensor.
- Supports tensors of type kTfLiteInt8.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER Tensor type or size does not correspond with configuration

sl_ml_audio_feature_generation_get_new_feature_slice_count

```
int sl_ml_audio_feature_generation_get_new_feature_slice_count ()
```

Return the number of new or unfetched feature slices that have been updated since the last call to sl_ml_audio_feature_generation_get_features_raw or sl_ml_audio_feature_generation_fill_tensor.

Returns

- The number of unfetched feature slices

sl_ml_audio_feature_generation_get_feature_buffer_size

```
int sl_ml_audio_feature_generation_get_feature_buffer_size ()
```

Return the feature buffer size.

Returns

- Size of the feature buffer

sl_ml_audio_feature_generation_reset

```
void sl_ml_audio_feature_generation_reset ()
```

Reset the state of the audio feature generator.

sl_ml_audio_feature_generation_init_with_buffer

```
sl_status_t sl_ml_audio_feature_generation_init_with_buffer (int16_t * buffer, int n_frames)
```

Initialize audio feature generation with a user-provided buffer.

Parameters

Type	Direction	Argument Name	Description
int16_t *	[in]	buffer	Pointer to the audio buffer to use for feature generation.
int	[in]	n_frames	Number of audio frames in the buffer.

Returns

- SL_STATUS_OK on success, SL_STATUS_FAIL otherwise.

sl_ml_audio_feature_generation_get_features_raw_float32

```
sl_status_t sl_ml_audio_feature_generation_get_features_raw_float32 (float * buffer, size_t num_elements)
```

Retrieve the features as type float32 and copy them to the provided buffer.

Parameters

Type	Direction	Argument Name	Description
float *	[out]	buffer	Pointer to the buffer to store the feature data
size_t	[in]	num_elements	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.

Note

- This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_audio_feature_generation_get_features_scaled

```
sl_status_t sl_ml_audio_feature_generation_get_features_scaled (float * buffer, size_t num_elements, float scaler)
```

Retrieve the features, scales them by the given scaler, and fills float buffer.

Parameters

Type	Direction	Argument Name	Description
float *	[out]	buffer	Pointer to the buffer to store the scaled feature data
size_t	[in]	num_elements	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.
float	[in]	scaler	The scaling factor to apply to each feature value.

buffer = (float)uint16_features_data * scaler

Note

- This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_audio_feature_generation_get_features_mean_std_normalized

```
sl_status_t sl_ml_audio_feature_generation_get_features_mean_std_normalized (float * buffer, size_t num_elements)
```

Retrieve the features, normalizes them by centering about their mean and standard deviation and fills float buffer.

Parameters

Type	Direction	Argument Name	Description
float *	[out]	buffer	Pointer to the buffer to store the normalized feature data
size_t	[in]	num_elements	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.

buffer = ((float)uint16_features_data - mean(uint16_features_data)) / std(uint16_features_data)

Note

- This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_audio_feature_generation_activity_detected

```
sl_status_t sl_ml_audio_feature_generation_activity_detected ()
```

Return if the activity detection block detected activity in the audio stream.

sl_ml_audio_feature_generation_update_features() must be periodically called to detect activity.

Note

- SL_ML_FRONTEND_ACTIVITY_DETECTION_ENABLE must be 1 to use this API.
- The internal state is reset after calling this API. i.e. If this API returns SL_STATUS_OK, then calling this API again will return SL_STATUS_IN_PROGRESS until new activity is detected.

Returns

- SL_STATUS_OK is activity was detected SL_STATUS_IN_PROGRESS is no activity detected SL_STATUS_NOT_AVAILABLE if the activity detection block is not enabled

sl_ml_audio_feature_generation_set_mic_callback

```
sl_status_t sl_ml_audio_feature_generation_set_mic_callback
(sl_ml_audio_feature_generation_mic_callback_t callback, void * arg)
```

Set the microphone callback for audio feature generation.

Parameters

Type	Direction	Argument Name	Description
sl_ml_audio_feature_generation_mic_callback_t	N/A	callback	Function pointer to the microphone callback.
void *	N/A	arg	User-defined argument to pass to the callback.

Returns

- sl_status_t Status code indicating success or failure.

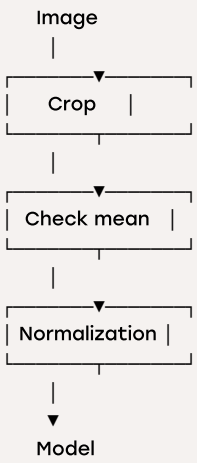
Image Feature Generator

Image Feature Generator

The image feature generator process input image captured from camera and extracts required features to use with machine learning image classification applications using a camera as an image source.

Feature Generation

The captured image pre-processed using crop and mean-std normalization based on model configuration used from application.



Usage

sl_ml_image_feature_generation_init() initializes the feature generation based on the configuration in sl_ml_image_feature_generation_config.h. It also initializes and starts the camera in streaming mode, which places the image samples into a ping-pong-buffer.

The image is pre-processed and filled in input tensor when sl_ml_image_feature_generation_fill_tensor() is called.

To retrieve the generated features sl_ml_image_feature_generation_fill_tensor() must be called.

Example

When used with TensorFlow Lite Micro, the image feature generator can be used to fill a tensor directly by using sl_ml_image_feature_generation_fill_tensor(). However, the model has to be trained using the same feature generator configurations as used for inference, configured in sl_ml_image_feature_generation_config.h.

```
#include "sl_tflite_micro_init.h"
#include "sl_ml_image_feature_generation.h"

void main(void)
{
 sl_ml_image_feature_generation_init();

 while(1){ *
  if(do_inference){
   sl_ml_image_feature_generation_fill_tensor(sl_tflite_micro_get_input_tensor());
   sl_tflite_micro_get_interpreter()->Invoke();
  }

  ...

 }
}
```

Functions

sl_status_t **sl_ml_image_feature_generation_init**()
Set up the camera as an image source for feature generation.

sl_status_t **sl_ml_image_feature_generation_fill_tensor**(TfLiteTensor *input_tensor)
Fill a TensorFlow tensor with feature data. Data type of input image for model will be selected based on model configuration.

sl_status_t **sl_ml_initialize_arducam_camera**()
Set up the aurducam as an image source for feature generation and initialize the camera with configuration.

void **sl_ml_dump_image**(const uint8_t *image_data, uint32_t image_length)
Set up the JLink stream as output node. Image data streamed over JLink to python interface.

void **sl_ml_retrieve_next_camera_image**(uint8_t **image_data, uint32_t *image_size)
Retrieve image from ping-pog buffer captured from aurducam. Store image data to destination pointer.

void **sl_ml_image_crop_lut_init**()
Initializes Look up table for croping image.

sl_status_t **sl_ml_image_feature_generation_get_image_scaled**(uint8_t *image_data, float *buffer, size_t num_elements, float scaler)
Retrieve the features, scales them by the given scaler, and fills float buffer.

sl_status_t **sl_ml_get_image_mean**(uint8_t *in_cam_image, float *mean, float *mean2, size_t num_elements)
Computes mean and mean square of input image.

sl_status_t **sl_ml_image_feature_generation_get_image_mean_std_normalized**(uint8_t *image_data, float *buffer, float mean, float mean2, size_t num_elements)
Retrieve the features, normalizes them by centering about their mean and standard deviation and fills float buffer.

sl_status_t **sl_ml_image_feature_generation_get_image_raw_float32**(uint8_t *image_data, float *buffer, size_t num_elements)
Retrieve the features as type float32 and copy them to the provided buffer.

void **sl_ml_image_feature_generation_reset**()
Reset the state of the image feature generator.

void sl_ml_convert_rgb565_to_gray(uint8_t *img)
Function to convert image color space to gray.

Function Documentation

sl_ml_image_feature_generation_init

```
sl_status_t sl_ml_image_feature_generation_init ()
```

Set up the camera as an image source for feature generation.

Returns

- SL_STATUS_OK for success SL_STATUS_FAIL

sl_ml_image_feature_generation_fill_tensor

```
sl_status_t sl_ml_image_feature_generation_fill_tensor (TfLiteTensor * input_tensor)
```

Fill a TensorFlow tensor with feature data. Data type of input image for model will be selected based on model configuration.

Parameters

Type	Direction	Argument Name	Description
TfLiteTensor *	[in]	input_tensor	The input tensor to fill with features.

Note

- This function overwrites the entire input tensor.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER Tensor type or size does not correspond with configuration

sl_ml_initialize_arducam_camera

```
sl_status_t sl_ml_initialize_arducam_camera ()
```

Set up the aurducam as an image source for feature generation and initialize the camera with configuration.

Returns

- SL_STATUS_OK for success SL_STATUS_FAIL

sl_ml_dump_image

```
void sl_ml_dump_image (const uint8_t * image_data, uint32_t image_length)
```

Set up the JLink stream as output node. Image data streamed over JLink to python interface.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	N/A	image_data	
uint32_t	N/A	image_length	

Returns

- SL_STATUS_OK for success SL_STATUS_FAIL

sl_ml_retrieve_next_camera_image

```
void sl_ml_retrieve_next_camera_image (uint8_t ** image_data, uint32_t * image_size)
```

Retrieve image from ping-pog buffer captured from aurducam. Store image data to destination pointer.

Parameters

Type	Direction	Argument Name	Description
uint8_t **	N/A	image_data	
uint32_t *	N/A	image_size	

sl_ml_image_crop_lut_init

```
void sl_ml_image_crop_lut_init ()
```

Initializes Look up table for croping image.

sl_ml_image_feature_generation_get_image_scaled

```
sl_status_t sl_ml_image_feature_generation_get_image_scaled (uint8_t * image_data, float * buffer,
size_t num_elements, float scaler)
```

Retrieve the features, scales them by the given scaler, and fills float buffer.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	image_data	Pointer to the buffer to store the scaled feature data
float *	[in]	buffer	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.
size_t	[in]	num_elements	The scaling factor to apply to each feature value.
float	N/A	scaler	

buffer = (float)image_data * scaler

Note

This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_get_image_mean

```
sl_status_t sl_ml_get_image_mean (uint8_t * in_cam_image, float * mean, float * mean2, size_t num_elements)
```

Computes mean and mean square of input image.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	in_cam_image	Pointer to the mean to store the mean value of image.
float *	[out]	mean	Pointer to the mean2 to store the mean square value of image.
float *	[in]	mean2	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.
size_t	N/A	num_elements	

mean = ((float)image_data / (width*length)

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_image_feature_generation_get_image_mean_std_normalized

```
sl_status_t sl_ml_image_feature_generation_get_image_mean_std_normalized (uint8_t * image_data, float * buffer, float mean, float mean2, size_t num_elements)
```

Retrieve the features, normalizes them by centering about their mean and standard deviation and fills float buffer.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	image_data	Pointer to the buffer to store the normalized feature data
float *	[in]	buffer	float mean, mean value of image.
float	[in]	mean	float mean2, mean square value of image.
float	[in]	mean2	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.
size_t	N/A	num_elements	

buffer = ((float)image_data - mean(image_data)) / std(image_data)

- This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_image_feature_generation_get_image_raw_float32

```
sl_status_t sl_ml_image_feature_generation_get_image_raw_float32 (uint8_t * image_data, float * buffer, size_t num_elements)
```

Retrieve the features as type float32 and copy them to the provided buffer.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	image_data	Pointer to the buffer to store the feature data
float *	[in]	buffer	The number of elements corresponding to the size of the buffer; If this is not large enough to store the entire feature buffer the function will return with an error.
size_t	N/A	num_elements	

Note

- This function overwrites the entire buffer.

Returns

- SL_STATUS_OK for success SL_STATUS_INVALID_PARAMETER num_elements too small

sl_ml_image_feature_generation_reset

```
void sl_ml_image_feature_generation_reset ()
```

Reset the state of the image feature generator.

sl_ml_convert_rgb565_to_gray

```
void sl_ml_convert_rgb565_to_gray (uint8_t * img)
```

Function to convert image color space to gray.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	img	Pointer holding reference to image data.

Microphone

Microphone

Sound level driver for PDM and I2S microphones.

Microphone example code

Basic example for looping measurement of sound level:

```
#include "sl_mic.h"

#define MIC_SAMPLE_RATE          44100
#define MIC_SAMPLE_BUFFER_SIZE   1024
#define MIC_N_CHANNELS           2

 static int16_t buffer[MIC_SAMPLE_BUFFER_SIZE * MIC_N_CHANNELS];

 int main( void )
 {

  ...
  float sound_level_0;
  float sound_level_1;
  uint32_t n_samples = 1024;

  // Initialize microphone with sample rate and number of channels
  sl_mic_init(MIC_SAMPLE_RATE, MIC_N_CHANNELS);

  while(true){

   // Read samples from the microphone
   sl_mic_get_n_samples(buffer, n_samples);

   while (!sl_mic_sample_buffer_ready()) {
    // Wait until sample buffer ready
   }

   // Calculate sound level
   sl_mic_calculate_sound_level(&sound_level_0, buffer, n_samples, 0);
   sl_mic_calculate_sound_level(&sound_level_1, buffer, n_samples, 1);

  }

  ...

 }
```

Typedefs

typedef void(* sl_mic_buffer_ready_callback_t)(const void *buffer, uint32_t n_frames)

Callback function indicating that the sample buffer is ready.

Functions

sl_status_t	sl_mic_init(uint32_t sample_rate, uint8_t channels) Initialize the microphone.
sl_status_t	sl_mic_deinit(void) De-initialize the microphone.
sl_status_t	sl_mic_get_n_samples(void *buffer, uint32_t n_frames) Read samples from the microphone into a sample buffer.
sl_status_t	sl_mic_start_streaming(void *buffer, uint32_t n_frames, sl_mic_buffer_ready_callback_t callback) Read samples from the microphone into a sample buffer continuously.
sl_status_t	sl_mic_start(void) Start the microphone.
sl_status_t	sl_mic_stop(void) Stop the microphone.
bool	sl_mic_sample_buffer_ready(void) Check whether the sample buffer is ready.
sl_status_t	sl_mic_calculate_sound_level(float *sound_level, const int16_t *buffer, uint32_t n_frames, uint8_t channel) Calculate the dBSPL value for a channel from a sample buffer.

Typedef Documentation

sl_mic_buffer_ready_callback_t

```
typedef void(* sl_mic_buffer_ready_callback_t) (const void *buffer, uint32_t n_frames) )(const void *buffer, uint32_t n_frames)
```

Callback function indicating that the sample buffer is ready.

Parameters

Type	Direction	Argument Name	Description
	[in]	buffer	Pointer to the sample buffer.
	[in]	n_frames	Number of audio frames in the sample buffer.

Returns

- None.

Function Documentation

sl_mic_init

```
sl_status_t sl_mic_init (uint32_t sample_rate, uint8_t channels)
```

Initialize the microphone.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	sample_rate	The desired sample rate in Hz
uint8_t	[in]	channels	Number of audio channels (1 or 2)

Returns

- Returns SL_STATUS_OK on success, non-zero otherwise

sl_mic_deinit

```
sl_status_t sl_mic_deinit (void )
```

De-initialize the microphone.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_mic_get_n_samples

```
sl_status_t sl_mic_get_n_samples (void * buffer, uint32_t n_frames)
```

Read samples from the microphone into a sample buffer.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	buffer	Pointer to the sample buffer to store the data. 16-bit channel data is stored consecutively, starting with ch0
uint32_t	[in]	n_frames	The number of the audio frames to get

This function starts the microphone sampling and stops the sampling after reading the desired number of samples. Call sl_mic_sample_buffer_ready to check when the samples are ready in the buffer.

sl_mic_start_streaming

```
sl_status_t sl_mic_start_streaming (void * buffer, uint32_t n_frames, sl_mic_buffer_ready_callback_t callback)
```

Read samples from the microphone into a sample buffer continuously.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	buffer	Pointer to the sample buffer to store the data. 16-bit channel data is stored consecutively, starting with ch0. This buffer shall be big enough to hold twice the n_frames because of the ping-pong operation.

Type	Direction	Argument Name	Description
uint32_t	[in]	n_frames	The number of audio frames to receive before the callback is called. Maximum value limited by DMADRV_MAX_XFER_COUNT.
sl_mic_buffer_ready_callback_t	[in]	callback	Callback is called when n_frames in the sample buffer is ready.

This function starts the microphone sampling and stops only upon calling sl_mic_stop or sl_mic_deinit. The buffer is used in a "ping-pong" manner meaning that one half of the buffer is used for sampling while the other half is being processed.

sl_mic_start

```
sl_status_t sl_mic_start (void )
```

Start the microphone.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_mic_stop

```
sl_status_t sl_mic_stop (void )
```

Stop the microphone.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_mic_sample_buffer_ready

```
bool sl_mic_sample_buffer_ready (void )
```

Check whether the sample buffer is ready.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_mic_calculate_sound_level

sl_status_t sl_mic_calculate_sound_level (float * sound_level, const int16_t * buffer, uint32_t n_frames, uint8_t channel)

Calculate the dBSPL value for a channel from a sample buffer.

Parameters

Type	Direction	Argument Name	Description
float *	[out]	sound_level	The calculated sound level
const int16_t *	[in]	buffer	Buffer to calculate sound level from. Must contain 16-bit samples, starting with channel 0
uint32_t	[in]	n_frames	Number of audio frames to use when calculating sound level
uint8_t	[in]	channel	The channel to get the sound level for

TensorFlow Lite Micro Debug

TensorFlow Lite Micro Debug

Additional SL utilities for logging in TensorFlow Lite Micro.

Functions

void sl_tflite_micro_enable_debug_log(bool enable)
Enable or disable debug logging.

bool sl_tflite_micro_is_debug_log_enabled(void)
Check if debug logging is enabled.

Function Documentation

sl_tflite_micro_enable_debug_log

> void sl_tflite_micro_enable_debug_log (bool enable)

Enable or disable debug logging.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Whether the debug logging should be enabled or disabled.

sl_tflite_micro_is_debug_log_enabled

> bool sl_tflite_micro_is_debug_log_enabled (void)

Check if debug logging is enabled.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Whether debug logging is enabled.

TensorFlow Lite Micro Init

TensorFlow Lite Micro Init

The TensorFlow Lite Micro Init functions are autogenerated from the flatbuffer file provided in the configuration, which includes an opsResolver generated according to the included operators in the flatbuffer. Helper functions to access the input and output tensors are also provided.

Functions

sl_status_t	sl_tflite_micro_estimate_arena_size(const tflite::Model *model, const tflite::MicroOpResolver &opcode_resolver, size_t *estimated_size) Estimate the arena size for a given model.
uint8_t *	sl_tflite_micro_allocate_tensor_arena(size_t arena_size, uint8_t **tensor_arena) Dynamically allocate a buffer that can be used for the tensor arena.
tflite::ErrorReporter *	sl_tflite_micro_get_error_reporter() Get a pointer to the TensorFlow Lite Micro error reporter created by the init function.
tflite::MicroInterpreter *	sl_tflite_micro_get_interpreter() Get a pointer to the TensorFlow Lite Micro interpreter created by the init function.
TfLiteTensor *	sl_tflite_micro_get_input_tensor() Get a pointer to the input tensor, set by the init function.
TfLiteTensor *	sl_tflite_micro_get_output_tensor() Get a pointer to the output tensor, set by the init function.
tflite::MicroOpResolver &	sl_tflite_micro_opcode_resolver() Get a pointer to the opcode resolver for the flatbuffer given by the configuration.
void	sl_tflite_micro_init(void) Create the error reporter and opcode resolver and initialize variables for the flatbuffer given by the configuration.

Function Documentation

sl_tflite_micro_estimate_arena_size

> sl_status_t sl_tflite_micro_estimate_arena_size (const tflite::Model * model, const tflite::MicroOpResolver & opcode_resolver, size_t * estimated_size)

Estimate the arena size for a given model.

Parameters

Type	Direction	Argument Name	Description
const tflite::Model *	[in]	model	Pointer to the model to estimate the arena size for.
const tflite::MicroOpResolver &	[in]	opcode_resolver	The opcode resolver to use for the model.
size_t *	[out]	estimated_size	The estimated size of the arena, as output.

Returns

- SL_STATUS_OK if the estimation was successful.

sl_tflite_micro_allocate_tensor_arena

```
uint8_t * sl_tflite_micro_allocate_tensor_arena (size_t arena_size, uint8_t ** tensor_arena)
```

Dynamically allocate a buffer that can be used for the tensor arena.

Parameters

Type	Direction	Argument Name	Description
size_t	[in]	arena_size	The size of the arena to allocate.
uint8_t **	[out]	tensor_arena	Pointer to the allocated tensor arena buffer.

Returns

- A pointer to the allocated base buffer. Returns nullptr if the allocation failed. The base buffer is used for freeing the allocated memory.

sl_tflite_micro_get_error_reporter

```
tflite::ErrorReporter * sl_tflite_micro_get_error_reporter ()
```

Get a pointer to the TensorFlow Lite Micro error reporter created by the init function.

Returns

- A pointer to the error reporter.

sl_tflite_micro_get_interpreter

```
tflite::MicroInterpreter * sl_tflite_micro_get_interpreter ()
```

Get a pointer to the TensorFlow Lite Micro interpreter created by the init function.

Returns

- A pointer to the interpreter.

sl_tflite_micro_get_input_tensor

```
TfLiteTensor * sl_tflite_micro_get_input_tensor ()
```

Get a pointer to the input tensor, set by the init function.

Returns

- A pointer to the input tensor.

sl_tflite_micro_get_output_tensor

```
TfLiteTensor * sl_tflite_micro_get_output_tensor ()
```

Get a pointer to the output tensor, set by the init function.

Returns

- A pointer to the output tensor.

sl_tflite_micro_opcode_resolver

```
tflite::MicroOpResolver & sl_tflite_micro_opcode_resolver ()
```

Get a pointer to the opcode resolver for the flatbuffer given by the configuration.

Returns

- The address to the opcode resolver.

sl_tflite_micro_init

```
void sl_tflite_micro_init (void )
```

Create the error reporter and opcode resolver and initialize variables for the flatbuffer given by the configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

MVP Math Library API Reference

MVP Math Library API

Introduction

Some of the Silicon Labs parts have included what is called the Matrix Vector Processor (MVP). The MVP is a hardware accelerator designed to execute certain operations faster than the CPU. In particular, vector and matrix operatons are usually good candidates for acceleration. The purpose with this math library is to offer functions that utilize the MVP to speed up the operations.

The MVP supports 8 bit integers and 16 bit floating point numbers, and this library contains only functions that can be accelerated on the MVP.

Similar or identical functions can be found in the ARM CMSIS-DSP library.

For more information on the MVP hardware, please refer to the reference manual for the actual part.

Memory alignment

The MVP will read and write to the system memory without using the CPU. When using 16-bit floating point variables, there are certain rules to follow:

- Scalar variables (single 16-bit values) must be aligned to an even address; 0, 2, 4, etc.
- Complex variables (double 16-bit values) must be aligned to a 4-byte word address; 0, 4, 8, etc.

NOTE:

If functions that are working on scalar values get 4-byte word addresses for all of the function arguments, it will in many cases be able to process two values per cycle. That will nearly double the speed of the operation, and is highly recommended.

Using the library

The library is delivered as source code. It can easily be used by including the `math_mvp` component to the project and just calling the functions.

The library requires that the MVP driver has been initialized by calling the `sli_mvp_init()` function. If the project has included the `sl_system`, this function will be called automatically, else it is to be called by the user application before any of the other mvp math functions are used.

The library is divided into a number of functions in one of the categories:

- Matrix functions
- Vector functions

Each of the library functions has a separate header file, but for convenience, the `sl_math_mvp.h` file contains everything the application needs to use the library.

```
#include "sl_math_mvp.h"

static float16_t input_a[256] SL_ATTRIBUTE_ALIGN(4);
static float16_t input_b[256] SL_ATTRIBUTE_ALIGN(4);
static float16_t output[256] SL_ATTRIBUTE_ALIGN(4);

void example()
{
 sl_status_t status;
 status = sl_math_mvp_vector_fill_f16(input_a, 1.0f, 6);
 ...
 status = sl_math_mvp_vector_add_f16(input_a, input_b, output, 6);
 ...
 sl_math_matrix_init_f16(&matrix_a, 3, 4, input_a);
 sl_math_matrix_init_f16(&matrix_b, 4, 3, input_b);
 sl_math_matrix_init_f16(&matrix_z, 3, 3, output);
 status = sl_math_mvp_matrix_mult_f16(&matrix_a, &matrix_b, &matrix_z);
 ...
}
```

Please take a look at the `math_mvp_demo` example in the Simplicity SDK. It shows various examples of how to use the functions.

Error handling

In general all math functions in the library will return one of these codes defined in `sl_status.h` :

```
#define SL_STATUS_OK                ((sl_status_t)0x0000)  // No error.
#define SL_STATUS_FAIL              ((sl_status_t)0x0001)  // Generic error.
#define SL_STATUS_INVALID_PARAMETER ((sl_status_t)0x0021)  // Generic invalid argument or consequence of invalid argument.
```

`SL_STATUS_INVALID_PARAMETER` is returned when the call was made with invalid input parameters.

`SL_STATUS_FAIL` is returned when execution on the MVP processor led to unrecoverable errors.

`SL_STATUS_OK` is returned when MVP program execution succeeded. In this case execution may have generated math exceptions such as overflow, underflow or inifinty etc.

An MVP program can produce these errors/exceptions:

```
#define SL_STATUS_COMPUTE_MATH_FAULT      ((sl_status_t)0x1511)  // MATH Critical fault
#define SL_STATUS_COMPUTE_MATH_NAN        ((sl_status_t)0x1512)  // MATH NaN encountered
#define SL_STATUS_COMPUTE_MATH_INFINITY   ((sl_status_t)0x1513)  // MATH Infinity encountered
#define SL_STATUS_COMPUTE_MATH_OVERFLOW   ((sl_status_t)0x1514)  // MATH numeric overflow
#define SL_STATUS_COMPUTE_MATH_UNDERFLOW  ((sl_status_t)0x1515)  // MATH numeric underflow
```

To investigate which (if any) math exceptions occured use:

```
sl_status_t sl_math_mvp_get_error(sl_status_t *error_code, char *error_message, uint32_t buffer_length);
```

As several exception may be present, you can call this function repeatedly to retrieve error codes. The function will return `SL_STATUS_OK` when a valid error code is returned via `*error_code` , until `SL_STATUS_NOT_FOUND` is returned.

```
void example()
{
 sl_status_t error_code;

 while (sl_math_mvp_get_error(&error_code, NULL, 0) == SL_STATUS_OK) {
  printf("Error code is: %X\n", error_code);
 }
}
```

If you add the `status_string` component to your application, the error function will also return descriptive error strings for the different exceptions. This might be handy for debugging.

```
void example()
{
 char error_string[100];
 sl_status_t error_code;

 while (sl_math_mvp_get_error(&error_code, error_string, sizeof(error_string)) == SL_STATUS_OK) {
  printf("Error code / message is: %X / %s\n", error_code, error_string);
 }
}
```

NOTE:

Errors and exceptions will accumulate across calls of MVP Math functions. This means that in many cases it will be sufficient to check for errors/exceptions after a sequence of MVP Math functions have been executed. The following function is handy to make sure all errors/exceptions are cleared before commencing on a long sequence of MVP Math function calls:

```
void sl_math_mvp_clear_errors(void);
```

Vector functions

Vector functions

Functions

sl_status_t sl_math_mvp_complex_vector_conjugate_f16(float16_t *input, float16_t *output, size_t num_elements)
Conjugates the elements of a complex data vector.

sl_status_t sl_math_mvp_complex_vector_dot_product_f16(float16_t *input_a, float16_t *input_b, size_t num_elements, float16_t *output)
Computes the dot product of two complex vectors. The vectors are multiplied element-by-element and then summed.

sl_status_t sl_math_mvp_complex_vector_magnitude_squared_f16(const float16_t *input, float16_t *output, size_t num_elements)
Computes the magnitude squared of the elements of a complex data vector.

sl_status_t sl_math_mvp_complex_vector_mult_real_f16(const float16_t *input_a, const float16_t *input_b, float16_t *output, size_t num_elements)
Multiply a vector of complex f16 by a vector of real f16.

sl_status_t sl_math_mvp_complex_vector_mult_f16(const float16_t *input_a, const float16_t *input_b, float16_t *output, size_t num_elements)
Multiply complex f16 vectors.

sl_status_t sl_math_mvp_vector_abs_f16(float16_t *input, float16_t *output, size_t num_elements)
Computes the absolute value of a vector on an element-by-element basis.

sl_status_t sl_math_mvp_vector_add_f16(const float16_t *input_a, const float16_t *input_b, float16_t *output, size_t num_elements)
Add two vectors of 16 bit floats.

sl_status_t sl_math_mvp_vector_add_i8(const int8_t *input_a, const int8_t *input_b, int8_t *output, size_t num_elements)
Add two vectors of signed 8 bit integers.

sl_status_t sl_math_mvp_clamp_i8(int8_t *data, size_t num_elements, int8_t min, int8_t max)
Clamp all signed 8 bit integers in a vector to a certain range.

sl_status_t sl_math_mvp_vector_clip_f16(const float16_t *input, float16_t *output, float16_t low, float16_t high, size_t num_elements)
Element-by-element clipping of a value.

sl_status_t sl_math_mvp_vector_copy_f16(const float16_t *input, float16_t *output, size_t num_elements)
Copy one 16 bit float vector into another.

sl_status_t sl_math_mvp_vector_dot_product_f16(const float16_t *input_a, const float16_t *input_b, size_t num_elements, float16_t *output)
Computes the dot product of two vectors. The vectors are multiplied element-by-element and then summed.

sl_status_t sl_math_mvp_vector_fill_f16(float16_t *output, const float16_t value, size_t num_elements)
Fills a constant value into a floating-point vector.

sl_status_t	sl_math_mvp_vector_mult_f16(const float16_t *input_a, const float16_t *input_b, float16_t *output, size_t num_elements) Elementwise multiply two vectors of 16 bit floats.
sl_status_t	sl_math_mvp_vector_negate_f16(const float16_t *input, float16_t *output, size_t num_elements) Negate a vector of 16 bit floats.
sl_status_t	sl_math_mvp_vector_offset_f16(const float16_t *input, const float16_t offset, float16_t *output, size_t num_elements) Adds a constant offset to each element of a vector.
sl_status_t	sl_math_mvp_vector_scale_f16(const float16_t *input, float16_t scale, float16_t *output, size_t num_elements) Scale a vector of 16-bits floats with a float16 scale.
sl_status_t	sl_math_mvp_vector_sub_f16(const float16_t *input_a, const float16_t *input_b, float16_t *output, size_t num_elements) Subtract two vectors of 16 bit floats.

Function Documentation

sl_math_mvp_complex_vector_conjugate_f16

> sl_status_t sl_math_mvp_complex_vector_conjugate_f16 (float16_t * input, float16_t * output, size_t num_elements)

Conjugates the elements of a complex data vector.

Parameters

Type	Direction	Argument Name	Description
float16_t *	[in]	input	Input vector.
float16_t *	[in]	output	Output Vector.
size_t	[in]	num_elements	The number of complex elements in the vectors.

Maximum vector length is 1M (2^20), and all vectors must be 4-byte aligned.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_complex_vector_dot_product_f16

> sl_status_t sl_math_mvp_complex_vector_dot_product_f16 (float16_t * input_a, float16_t * input_b, size_t num_elements, float16_t * output)

Computes the dot product of two complex vectors. The vectors are multiplied element-by-element and then summed.

Parameters

Type	Direction	Argument Name	Description
float16_t *	[in]	input_a	Input vector a.
float16_t *	[in]	input_b	Input vector b.

Type	Direction	Argument Name	Description
size_t	[in]	num_elements	The number of complex elements in the input vectors.
float16_t *	[out]	output	Dot product result.

Maximum vector length is 1M (2^20), and all vectors must be 4-byte aligned.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_complex_vector_magnitude_squared_f16

> sl_status_t sl_math_mvp_complex_vector_magnitude_squared_f16 (const float16_t * input, float16_t * output, size_t num_elements)

Computes the magnitude squared of the elements of a complex data vector.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input	Input vector.
float16_t *	[in]	output	Output Vector.
size_t	[in]	num_elements	The number of complex elements in the input vector and the number of scalar elements in the output vector.

The input vector shall point to the source that is a vector of complex numbers and the output vector shall point to a vector where the result will be written.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_complex_vector_mult_real_f16

> sl_status_t sl_math_mvp_complex_vector_mult_real_f16 (const float16_t * input_a, const float16_t * input_b, float16_t * output, size_t num_elements)

Multiply a vector of complex f16 by a vector of real f16.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input_a	The complex vector, input A.
const float16_t *	[in]	input_b	The real vector, input B.
float16_t *	[out]	output	The complex output vector, output Z.
size_t	[in]	num_elements	The number of elements in the input and output vectors.

This function will perform the following operation: Z = A * B. Both vectors must be of same length. If both vector buffers are 4-byte aligned, the function will operate twice as fast using MVP complex processing. Maximum vector length is 1M (2^20) elements in the 4-byte aligned case, and 512K when oen or more of the complex vectors are 2-byte aligned.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_complex_vector_mult_f16

```
sl_status_t sl_math_mvp_complex_vector_mult_f16 (const float16_t * input_a, const float16_t * input_b, float16_t * output, size_t num_elements)
```

Multiply complex f16 vectors.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input_a	Complex input vector A.
const float16_t *	[in]	input_b	Complex input vector B.
float16_t *	[out]	output	Complex output vector.
size_t	[in]	num_elements	The number of complex elements in the input and output vectors.

This function will multiply two complex vectors. It is assumed that both input vectors, and the output vector have same length. All input and output buffers must be 4-byte aligned. Maximum vector length is 1M (2^20) elements.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_abs_f16

```
sl_status_t sl_math_mvp_vector_abs_f16 (float16_t * input, float16_t * output, size_t num_elements)
```

Computes the absolute value of a vector on an element-by-element basis.

Parameters

Type	Direction	Argument Name	Description
float16_t *	[in]	input	Input vector.
float16_t *	[in]	output	Output Vector.
size_t	[in]	num_elements	The number of elements in the vectors.

The output vector can be the same as or differnt to the input vector. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_add_f16

sl_status_t sl_math_mvp_vector_add_f16 (const float16_t * input_a, const float16_t * input_b, float16_t * output, size_t num_elements)

Add two vectors of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input_a	First input vector, input A.
const float16_t *	[in]	input_b	Second input vector, input B.
float16_t *	[out]	output	Output vector, output Z.
size_t	[in]	num_elements	The number of elements in the vectors.

This function will perform the following operation: Z = A + B. All vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_add_i8

sl_status_t sl_math_mvp_vector_add_i8 (const int8_t * input_a, const int8_t * input_b, int8_t * output, size_t num_elements)

Add two vectors of signed 8 bit integers.

Parameters

Type	Direction	Argument Name	Description
const int8_t *	[in]	input_a	First input vector, input A.
const int8_t *	[in]	input_b	Second input vector, input B.
int8_t *	[out]	output	Output vector, output Z.
size_t	[in]	num_elements	The number of elements in the vectors.

All vectors must be of the same length. This function will perform the following operation: Z = A + B. The add operation is performing a saturation add, which means that the operation will never overflow or underflow. When adding two elements would overflow (>127) then the result will be 127. When adding two elements would underflow (<-128) then the result will be -128.

Returns

- SL_STATUS_OK.

sl_math_mvp_clamp_i8

sl_status_t sl_math_mvp_clamp_i8 (int8_t * data, size_t num_elements, int8_t min, int8_t max)

Clamp all signed 8 bit integers in a vector to a certain range.

Parameters

Type	Direction	Argument Name	Description
int8_t *	[inout]	data	Vector with data values.
size_t	[in]	num_elements	The number of elements in the vector.
int8_t	[in]	min	Minimum value, after operation no elements will be < min.
int8_t	[in]	max	Maximum value, after operation no elements will be > max.

Given a min/max value, this function will make sure that none of the element in the input vector will be < min or > max. If any elements are < min then the value will be modified to min. If any elements are > max then value will be modified to max.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_clip_f16

```
sl_status_t sl_math_mvp_vector_clip_f16 (const float16_t * input, float16_t * output, float16_t low, float16_t high, size_t num_elements)
```

Element-by-element clipping of a value.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input	Input vector, input A.
float16_t *	[out]	output	Output vector, output Z.
float16_t	[in]	low	Lower bound.
float16_t	[in]	high	Higher bound.
size_t	[in]	num_elements	Length of input and output vectors.

This function will do an element-by-element clipping of a value. The value is constrained between 2 bounds. Both vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_copy_f16

```
sl_status_t sl_math_mvp_vector_copy_f16 (const float16_t * input, float16_t * output, size_t num_elements)
```

Copy one 16 bit float vector into another.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input	Input vector, input A.
float16_t *	[out]	output	Output vector, output Z.
size_t	[in]	num_elements	Length of input and output vectors.

This function will perform the following operation: Z = A. All vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_dot_product_f16

```
sl_status_t sl_math_mvp_vector_dot_product_f16 (const float16_t * input_a, const float16_t * input_b,
size_t num_elements, float16_t * output)
```

Computes the dot product of two vectors. The vectors are multiplied element-by-element and then summed.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input_a	Input vector a.
const float16_t *	[in]	input_b	Input vector b.
size_t	[in]	num_elements	The number of elements in the input vectors.
float16_t *	[out]	output	The result.

All vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Note

- Depending on the function arguments, the MVP implementation can calculate the dot product in different ways that may effect the rounding errors. If the same input vectors are calculated with different memory alignmen, the results may not be identical.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_fill_f16

```
sl_status_t sl_math_mvp_vector_fill_f16 (float16_t * output, const float16_t value, size_t num_elements)
```

Fills a constant value into a floating-point vector.

Parameters

Type	Direction	Argument Name	Description
float16_t *	[in]	output	Vector to fill.
const float16_t	[in]	value	Fill value.
size_t	[in]	num_elements	Length of the output vector.

Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_mult_f16

```
sl_status_t sl_math_mvp_vector_mult_f16 (const float16_t * input_a, const float16_t * input_b, float16_t * output, size_t num_elements)
```

Elementwise multiply two vectors of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input_a	First input vector, input A.
const float16_t *	[in]	input_b	Second input vector, input B.
float16_t *	[out]	output	Output vector, output Z.
size_t	[in]	num_elements	Length of all input and output vectors.

This function will perform the following operation: Z[i] = A[i] * B[i]. All vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_negate_f16

```
sl_status_t sl_math_mvp_vector_negate_f16 (const float16_t * input, float16_t * output, size_t num_elements)
```

Negate a vector of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input	Input vector.
float16_t *	[out]	output	Output vector.
size_t	[in]	num_elements	Length of input and output vectors.

This function will perform the following operation: Z = - A. Vectors must be of equal length. If both vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case. In-place negation is supported (input and output reference same buffer).

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_offset_f16

```
sl_status_t sl_math_mvp_vector_offset_f16 (const float16_t * input, const float16_t offset, float16_t * output, size_t num_elements)
```

Adds a constant offset to each element of a vector.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input	Input vector.
const float16_t	[in]	offset	Offset value.
float16_t *	[in]	output	Output vector.
size_t	[in]	num_elements	The number of elements in the input and output vectors.

Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_scale_f16

```
sl_status_t sl_math_mvp_vector_scale_f16 (const float16_t * input, float16_t scale, float16_t * output,
size_t num_elements)
```

Scale a vector of 16-bits floats with a float16 scale.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input	The input vector.
float16_t	[in]	scale	The value by which to scale the vector.
float16_t *	[out]	output	Output vector, output Z.
size_t	[in]	num_elements	Length of input and output vectors.

This function will perform the following operation: Z[i] = A[i] * scale. All vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_vector_sub_f16

```
sl_status_t sl_math_mvp_vector_sub_f16 (const float16_t * input_a, const float16_t * input_b, float16_t *
output, size_t num_elements)
```

Subtract two vectors of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const float16_t *	[in]	input_a	First input vector, input A.
const float16_t *	[in]	input_b	Second input vector, input B.
float16_t *	[out]	output	Output vector, output Z.
size_t	[in]	num_elements	Length of all input and output vectors.

This function will perform the following operation: Z = A - B. All vectors must be of equal length. If all vector buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum vector length is 1M (2^20), and 2M in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

Matrix functions

Matrix functions

Functions

void	sl_math_matrix_init_f16(sl_math_matrix_f16_t *matrix, size_t num_rows, size_t num_cols, float16_t *data) Matrix initialization.
sl_status_t	sl_math_mvp_complex_matrix_mult_f16(const sl_math_matrix_f16_t *input_a, const sl_math_matrix_f16_t *input_b, sl_math_matrix_f16_t *output) Multiply two matrices of complex 16 bit floats.
sl_status_t	sl_math_mvp_matrix_add_f16(const sl_math_matrix_f16_t *input_a, const sl_math_matrix_f16_t *input_b, sl_math_matrix_f16_t *output) Add two matrices of 16 bit floats.
sl_status_t	sl_math_mvp_matrix_mult_f16(const sl_math_matrix_f16_t *input_a, const sl_math_matrix_f16_t *input_b, sl_math_matrix_f16_t *output) Multiply two matrices of 16 bit floats.
sl_status_t	sl_math_mvp_matrix_scale_f16(const sl_math_matrix_f16_t *input, float16_t scale, sl_math_matrix_f16_t *output) Scale each element in a float16 matrix by a float16 scale.
sl_status_t	sl_math_mvp_matrix_sub_f16(const sl_math_matrix_f16_t *input_a, const sl_math_matrix_f16_t *input_b, sl_math_matrix_f16_t *output) Subtract two matrices of 16 bit floats.
sl_status_t	sl_math_mvp_matrix_transpose_f16(const sl_math_matrix_f16_t *input, sl_math_matrix_f16_t *output) Transpose a matrix.
sl_status_t	sl_math_mvp_complex_matrix_transpose_f16(const sl_math_matrix_f16_t *input, sl_math_matrix_f16_t *output) Transpose a complex f16 matrix.
sl_status_t	sl_math_mvp_matrix_vector_mult_f16(const sl_math_matrix_f16_t *input_a, const float16_t *input_b, float16_t *output) Multiply a matrix with a vector, both of 16 bit floats.

Function Documentation

sl_math_matrix_init_f16

> void sl_math_matrix_init_f16 (sl_math_matrix_f16_t * matrix, size_t num_rows, size_t num_cols, float16_t * data)

Matrix initialization.

Parameters

Type	Direction	Argument Name	Description
sl_math_matrix_f16_t *	[in]	matrix	Pointer to a matrix.

Type	Direction	Argument Name	Description
size_t	[in]	num_rows	The number of rows in the matrix.
size_t	[in]	num_cols	The number of cols in the matrix.
float16_t *	[in]	data	A pointer to the matrix data.

sl_math_mvp_complex_matrix_mult_f16

sl_status_t sl_math_mvp_complex_matrix_mult_f16 (const sl_math_matrix_f16_t * input_a, const sl_math_matrix_f16_t * input_b, sl_math_matrix_f16_t * output)

Multiply two matrices of complex 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input_a	First input matrix, input A.
const sl_math_matrix_f16_t *	[in]	input_b	Second input matrix, input B.
sl_math_matrix_f16_t *	[out]	output	Output matrix, output Z.

The number of columns of the first matrix must be equal to the number of rows of the second matrix. Also the output matrix row count must match matrix A row count and output matrix column count must match matrix B column count. All input and output matrix data buffers must be 4-byte aligned (a complex f16 element occupies 4 bytes of storage). Maximum matrix size is 1024 x 1024 which is 1M (2^20) complex f16 elements. Maximum column and row size is 1024 elements.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_matrix_add_f16

sl_status_t sl_math_mvp_matrix_add_f16 (const sl_math_matrix_f16_t * input_a, const sl_math_matrix_f16_t * input_b, sl_math_matrix_f16_t * output)

Add two matrices of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input_a	First input matrix, input A.
const sl_math_matrix_f16_t *	[in]	input_b	Second input matrix, input B.
sl_math_matrix_f16_t *	[out]	output	Output matrix, output Z.

This function will perform the following operation: Z = A + B. All matrices must have equal dimensions. If all matrice buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum matrices size is 1M (2^20) elements, and 2M elements in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_matrix_mult_f16

sl_status_t sl_math_mvp_matrix_mult_f16 (const sl_math_matrix_f16_t * input_a, const sl_math_matrix_f16_t * input_b, sl_math_matrix_f16_t * output)

Multiply two matrices of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input_a	First input matrix, input A.
const sl_math_matrix_f16_t *	[in]	input_b	Second input matrix, input B.
sl_math_matrix_f16_t *	[out]	output	Output matrix, output Z.

This function will perform the following operation: Z = A * B (matrix multiplication). The number of columns of the first matrix must be equal to the number of rows of the second matrix. If the input is 4 bytes aligned, and the number of columns in matrix B is divisible by 2, it will be 2x faster.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_matrix_scale_f16

sl_status_t sl_math_mvp_matrix_scale_f16 (const sl_math_matrix_f16_t * input, float16_t scale, sl_math_matrix_f16_t * output)

Scale each element in a float16 matrix by a float16 scale.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input	Input matrix.
float16_t	[in]	scale	Scale value.
sl_math_matrix_f16_t *	[out]	output	Output matrix.

This function will multiply each element in the input matrix by a scale, and write the result to the output matrix. The input and output matrices must be the same size. Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_matrix_sub_f16

sl_status_t sl_math_mvp_matrix_sub_f16 (const sl_math_matrix_f16_t * input_a, const sl_math_matrix_f16_t * input_b, sl_math_matrix_f16_t * output)

Subtract two matrices of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input_a	First input matrix, input A.

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input_b	Second input matrix, input B.
sl_math_matrix_f16_t *	[out]	output	Output matrix, output Z.

This function will perform the following operation: Z = A - B. All matrices must have equal dimensions. If all matrice buffers are 4-byte aligned, the function will operate twice as fast using MVP parallel processing. Maximum matrices size is 1M (2^20) elements, and 2M elements in the 4-byte aligned case.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_matrix_transpose_f16

> sl_status_t sl_math_mvp_matrix_transpose_f16 (const sl_math_matrix_f16_t * input, sl_math_matrix_f16_t * output)

Transpose a matrix.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input	Input matrix.
sl_math_matrix_f16_t *	[out]	output	output matrix.

This function will fill the output matrix with the transposed version of the input matrix. The maximum value for the rows and cols argument is 1024.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_complex_matrix_transpose_f16

> sl_status_t sl_math_mvp_complex_matrix_transpose_f16 (const sl_math_matrix_f16_t * input, sl_math_matrix_f16_t * output)

Transpose a complex f16 matrix.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input	Input matrix.
sl_math_matrix_f16_t *	[out]	output	output matrix.

This function will fill the output matrix with the transposed version of the input matrix. The maximum value for the rows and cols argument is 1024. Matrix input and output data buffers must be 4-byte aligned.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

sl_math_mvp_matrix_vector_mult_f16

sl_status_t sl_math_mvp_matrix_vector_mult_f16 (const sl_math_matrix_f16_t * input_a, const float16_t * input_b, float16_t * output)

Multiply a matrix with a vector, both of 16 bit floats.

Parameters

Type	Direction	Argument Name	Description
const sl_math_matrix_f16_t *	[in]	input_a	The input matrix.
const float16_t *	[in]	input_b	The input vector.
float16_t *	[out]	output	The output vector.

This function will perform the following operation: Z = A * b (matrix vector multiplication). The vector must be equal in length to the number of columns in matrix A. The output vector will be equal in length to the number of rows in matrix A.

Returns

- SL_STATUS_OK on success. On failure, an appropriate sl_status_t errorcode is returned.

Utility functions

Utility functions

Functions

void sl_math_mvp_clear_errors(void)
Clear all Math exception and fault flags.

sl_status_t sl_math_mvp_get_error(sl_status_t *error_code, char *error_message, uint32_t buffer_length)
Get a Math exception or fault errorcode and optional descriptive error message.

Function Documentation

sl_math_mvp_clear_errors

```
void sl_math_mvp_clear_errors (void )
```

Clear all Math exception and fault flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_math_mvp_get_error

```
sl_status_t sl_math_mvp_get_error (sl_status_t * error_code, char * error_message, uint32_t buffer_length)
```

Get a Math exception or fault errorcode and optional descriptive error message.

Parameters

Type	Direction	Argument Name	Description
sl_status_t *	[out]	error_code	The assigned sl_status_t errorcode.
char *	[inout]	error_message	A descriptive error message string. Input a NULL pointer to skip the error message.
uint32_t	[in]	buffer_length	The size of the error_message buffer.

Intended use of this function is to call it repeatedly to iterate over existing errors and get errorcodes and optional error message strings.

Returns

- Return SL_STATUS_OK when an error is present, this indicates that error_code and optionally error_message output parameters are valid. Return SL_STATUS_NOT_FOUND when no error to report.

Sample Applications

Sample Applications Overview

The following applications demonstrate the use of the TensorFlow Lite for Microcontrollers framework with the Simplicity SDK.

Voice Control Light

This application demonstrates a neural network with TensorFlow Lite for Microcontrollers to detect the spoken words "on" and "off" from audio data recorded on the microphone in a Micrium OS kernel task.

The detected keywords are used to control an LED on the board. The audio data is sampled continuously and preprocessed using the Audio Feature Generator component. Inference is run every 200 ms on the past ~1 s of audio data.

This sample application uses the Flatbuffer Converter Tool to add the .tflite file to the application binary.

Z3SwitchWithVoice

This application combines voice detection with Zigbee 3.0 to create a voice-controlled switch node that can be used to toggle a light node. The application uses the same model as Voice Control Light to detect the spoken keywords "on" and "off". Upon detection, the switch node sends On/Off commands over the Zigbee network.

This sample application uses the Flatbuffer Converter Tool to add the .tflite file to the application binary.

TensorFlow Lite Micro - Hello World

This application demonstrates a model trained to replicate a sine function and use the inference results to fade an LED. The application is originally written by TensorFlow, but has been ported to the Simplicity SDK.

The model is approximately 2.5 KB. The entire application takes around 157 KB flash and 15 KB RAM. This application uses large amounts of flash memory because it does not manually specify which operations are used in the model and, as a result, compiles all kernel implementations.

The application illustrates a minimal inference application and serves as a good starting point for understanding the TensorFlow Lite for Microcontrollers model interpretation flow.

This sample application uses a fixed model contained in `hello_world_model_data.cc` .

TensorFlow Lite Micro - Micro Speech

This application demonstrates a 20 KB model trained to detect simple words from speech data recorded from a microphone. The application is originally written by TensorFlow, but has been ported to the Simplicity SDK.

This application uses around 100 KB flash and 37 KB of RAM. Around 10 KB of the RAM usage is related to FFT frontend and to store audio data. With a clock speed of 38.4 MHz and using the optimized kernel implementations, the inference time on ~1 s of audio data is approximately 111 ms.

This application illustrates the process of generating features from audio data and doing detections in real time. It also demonstrates how to manually specify which operations are used in the network, which saves a significant amount of flash.

This sample application uses a fixed model contained in `micro_speech_model_data.cc` .

TensorFlow Lite Micro - Magic Wand

This application demonstrates a 10 KB model trained to recognize various hand gestures using an accelerometer to detect the motion. The detected gestures are printed to the serial port. The application is originally written by TensorFlow, but has been ported to the Simplicity SDK.

This application uses around 104 KB flash and 25 KB of RAM. This application demonstrates how to use accelerometer data as inference input and also shows how to manually specify which operations are used in the network, which saves a significant amount of flash.

This sample application uses the Flatbuffer Converter Tool to add the .tflite file to the application binary.

TensorFlow Model Profiler

This application is designed to profile a TensorFlow Lite Micro model on Silicon Labs hardware. The model used by the application is provided by a TensorFlow Lite flatbuffer file called model.tflite in the config/tflite subdirectory. The profiler will measure the number of CPU clock cycles and elapsed time in each layer of the model when performing an inference. It will also produce a summary when inference is done. The input layer of the model is filled with all zeroes before performing a single inference. Profiling results are transmitted over VCOM.

To run the application with a different .tflite model, you can replace the file called model.tflite with a new TensorFlow Lite Micro flatbuffer file. This new file must also be called "model.tflite" and be placed inside the config/tflite subdirectory to be picked up by the sample application. After the model has been replaced, regenerate the project.

To load and perform inference on a TensorFlow Lite Micro model, allocate a number of bytes to a "tensor arena" to hold state needed by the TensorFlow Lite Micro. The size of this tensor arena depends on the size of the model and the number of operators. The TensorFlow Model Profiler application can be used to measure the amount of RAM needed by the tensor arena to load the specific TensorFlow Lite Micro model. This is measured by dynamically allocating RAM for the tensor arena and reporting the number of bytes needed on VCOM. The number of bytes needed for the tensor arena can later be used to statically allocate memory when the model is used in a different application.

This sample application uses the Flatbuffer Converter Tool to add the .tflite file to the application binary.

Rock-Paper-Scissors (Image Classification)

Image classification is one of the most important applications of deep learning and Artificial Intelligence. Image classification refers to assigning labels to images based on certain characteristics or features present in them. The algorithm identifies these features and uses them to differentiate between different images and assign labels to them.

This application uses TensorFlow Lite for Microcontrollers to run image classification machine learning models to classify hand gestures from image data captured from ArduCAM camera. The detection is visualized using the LED's on the board and the classification results are written to the VCOM serial port.

For more information, refer to the Image Classifier documentation.

Image Classifier

Image Classifier

Image classification is one of the most important applications of deep learning and Artificial Intelligence. Image classification refers to assigning labels to images based on certain characteristics or features present in them. The algorithm identifies these features and uses them to differentiate between different images and assign labels to them.

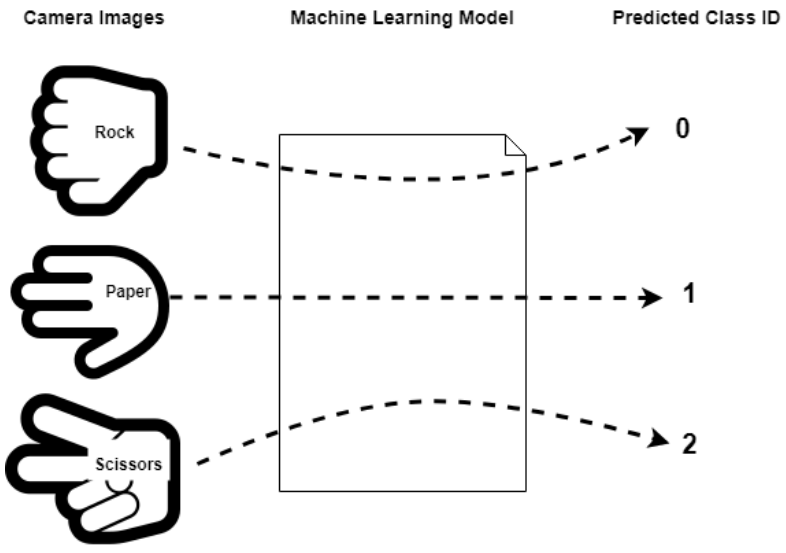
This application uses TensorFlow Lite for Microcontrollers to run image classification machine learning models to classify hand gestures from image data captured from ArduCAM camera. The detection is visualized using the LED's on the board and the classification results are written to the VCOM serial port.

This sample application uses the Flatbuffer Converter Tool to add the .tflite file to the application binary.

Class Labels

- rock: Images of a person's hand making a "rock" gesture
- paper: Images of a person's hand making a "paper" gesture
- scissors: Images of a person's hand making a "scissors" gesture
- unknown: Random images not containing any of the above

Following figure describes class labels with respect to hand pose.



Required Hardware and Setup

- Silicon Labs EFR series boards BRD2601B or BRD2608A.
- Berg Strip Connectors with Jumper wires (minimum 8 count) any one of following combination.
  - Male-Berg strip with female to female jumper wires.
  - Female-Berg strip with male to female jumper wires.
- ArduCAM camera module.

Pin Configuration

Following table shows pin connections between ArduCAM and Development kit.

ArduCAM Pin	Board Expansion Header Pin
GND	1
VCC	18
CS	10
MOSI	4
MISO	6
SCK	8
SDA	16
SCL	15

Required Software

- Simplicity studio v5 with
  - simplicity_sdk
  - aiml-extension

Dumping Images to PC

This application uses JLink to stream image data to a Python script located at aiml-extension/tools/image-visualization. To get started, refer to the instructions provided in the readme.md file. Once the application binary is flashed onto the development board, you can launch the visualization tool to view incoming image streams and optionally save them to your local PC.

Application Notes

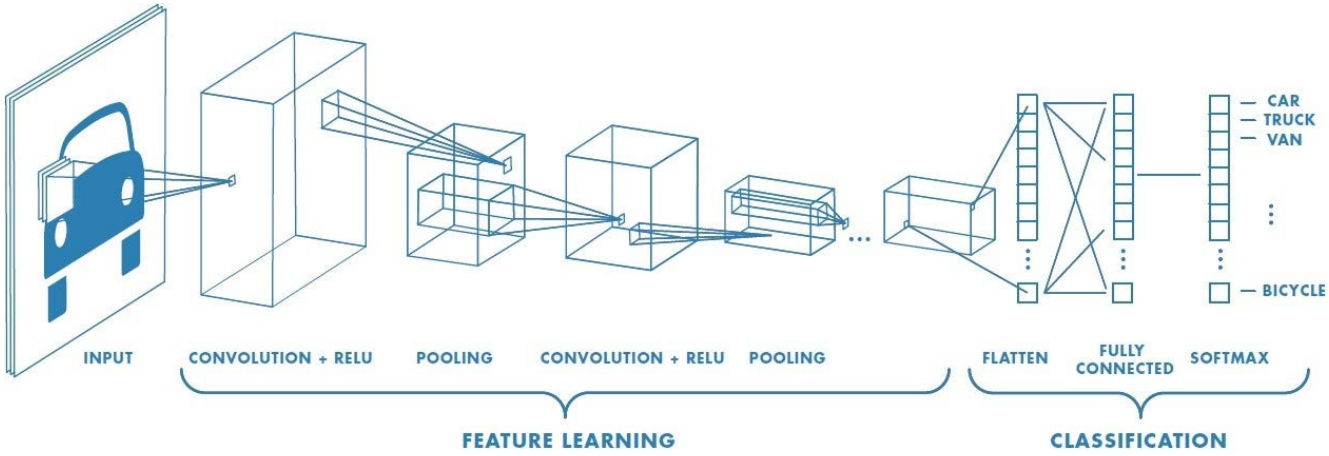
- Camera Input: This application utilizes images captured using an ArduCAM camera.
- Lighting Conditions: The model was trained on well-lit images. To achieve optimal results, conduct experiments in good lighting. Adjust the SL_ML_IMAGE_MEAN_THRESHOLD parameter in the .slcp configuration file to set the minimum mean intensity required for image processing.
- Recommended Distance: Maintain approximately 0.5 meters between the camera and the subject's hand during experimentation.
- Background Setup: Use a plain background (preferably white or black) to improve detection accuracy and consistency.

Convolutional Neural Networks

The type of machine learning model used in this application is Convolutional Neural Network (CNN).

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

A typical CNN can be visualized as follows:

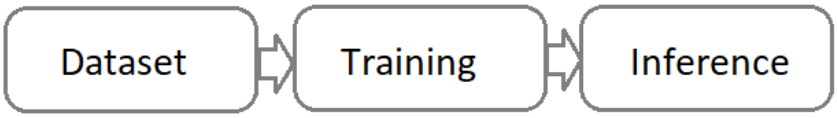


A typical CNN is comprised of multiple layers. A given layer is basically a mathematical operation that operates on multi-dimensional arrays (a.k.a tensors). The layers of a CNN can be split into two core phases:

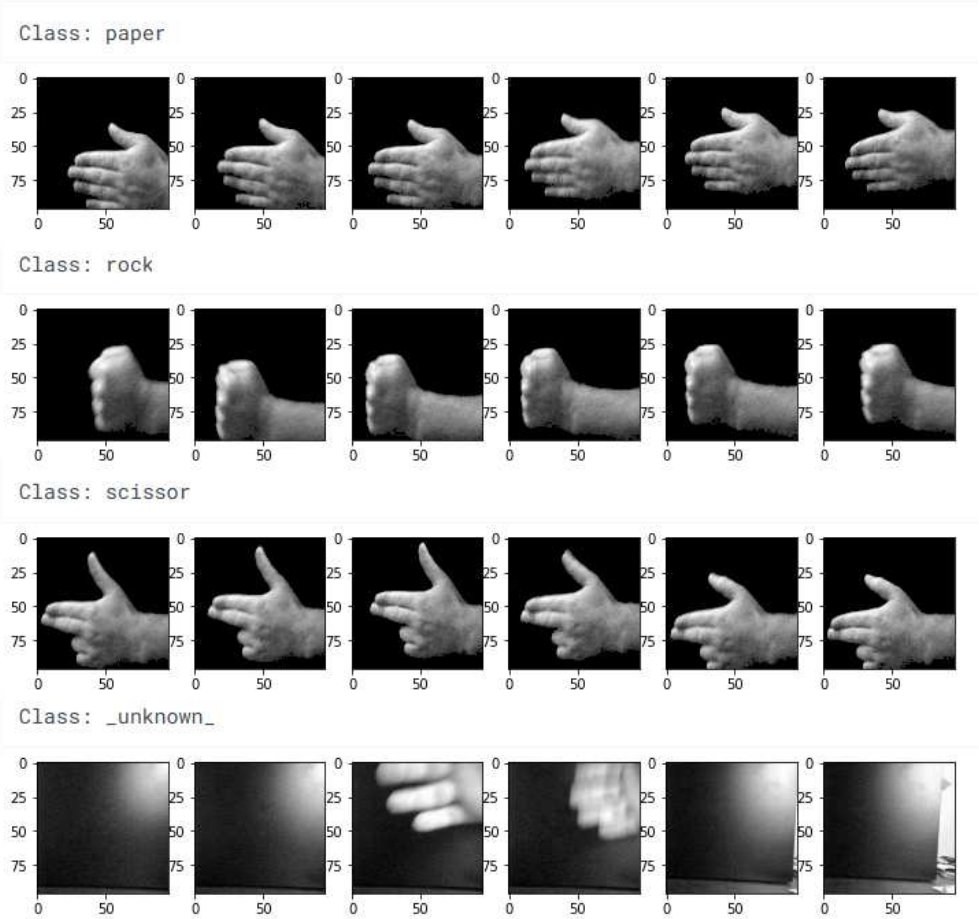
- Feature Learning: This uses Convolutional layers to extract “features” from the input image
- Classification: This takes the flatten “feature vector” from the feature learning layers and uses “fully connected” layer(s) to make a prediction on which class the input image belongs

Deep Learning Pipeline

A deep learning pipeline typically consists of three primary stages: dataset collection, model training, and inference as follows.



Some sample data used for training the model in this application is illustrated in the figure below.



Data Preprocessing

The preprocessing stage involves specific data normalization settings, managed by the ml_image_feature_generation component.

samplewise_center = True samplewise_std_normalization = True

norm_img = (img - mean(img)) / std(img)

These settings normalize each input image individually using the formula. This helps to ensure the model is not as dependent on camera and lighting variations.

Model Details

The model summary presents detailed information about each layer along with the number of parameters involved. This breakdown outlines the architecture used to perform image classification task.

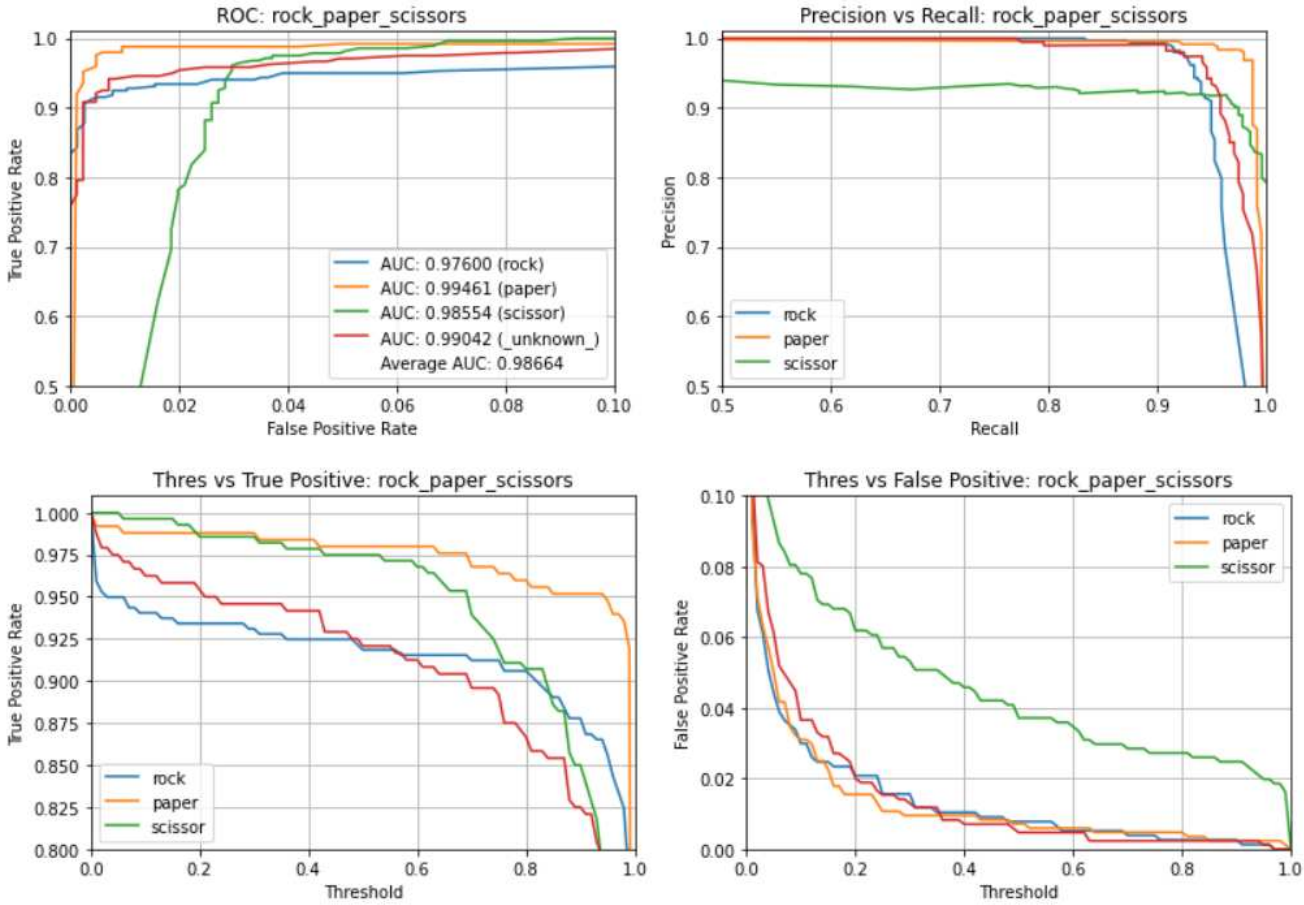
```
+-------+-----------------+--------------------+-----------------+-----------------------------------------------------+
| Index | OpCode          | Input(s)           | Output(s)       | Config                                              |
+-------+-----------------+--------------------+-----------------+-----------------------------------------------------+
| 0     | quantize        | 84x84x1 (float32)  | 84x84x1 (int8)  | BuiltinOptionsType=0                                |
| 1     | conv_2d         | 84x84x1 (int8)     | 82x82x16 (int8) | Padding:valid stride:1x1 activation:relu            |
|       |                 | 3x3x1 (int8)       |                 |                                                     |
|       |                 | 16 (int32)         |                 |                                                     |
| 2     | max_pool_2d     | 82x82x16 (int8)    | 41x41x16 (int8) | Padding:valid stride:2x2 filter:2x2 activation:none |
| 3     | conv_2d         | 41x41x16 (int8)    | 39x39x16 (int8) | Padding:valid stride:1x1 activation:relu            |
|       |                 | 3x3x16 (int8)      |                 |                                                     |
|       |                 | 16 (int32)         |                 |                                                     |
| 4     | max_pool_2d     | 39x39x16 (int8)    | 19x19x16 (int8) | Padding:valid stride:2x2 filter:2x2 activation:none |
| 5     | conv_2d         | 19x19x16 (int8)    | 17x17x32 (int8) | Padding:valid stride:1x1 activation:relu            |
|       |                 | 3x3x16 (int8)      |                 |                                                     |
|       |                 | 32 (int32)         |                 |                                                     |
| 6     | max_pool_2d     | 17x17x32 (int8)    | 8x8x32 (int8)   | Padding:valid stride:2x2 filter:2x2 activation:none |
| 7     | reshape         | 8x8x32 (int8)      | 2048 (int8)     | BuiltinOptionsType=0                                |
|       |                 | 2 (int32)          |                 |                                                     |
| 8     | fully_connected | 2048 (int8)        | 32 (int8)       | Activation:relu                                     |
|       |                 | 2048 (int8)        |                 |                                                     |
|       |                 | 32 (int32)         |                 |                                                     |
| 9     | fully_connected | 32 (int8)          | 4 (int8)        | Activation:none                                     |
|       |                 | 32 (int8)          |                 |                                                     |
|       |                 | 4 (int32)          |                 |                                                     |
| 10    | softmax         | 4 (int8)           | 4 (int8)        | BuiltinOptionsType=9                                |
| 11    | dequantize      | 4 (int8)           | 4 (float32)     | BuiltinOptionsType=0                                |
+-------+-----------------+--------------------+-----------------+-----------------------------------------------------+
Total MACs: 5.870 M
Total OPs: 12.050 M
Name: rock_paper_scissors
```

Model Evaluation

The fundamental approach to model evaluation involves feeding test samples—new, unseen data not used during training—into the model and comparing its predictions against the actual expected values. If every prediction is correct, the model achieves 100% accuracy; each incorrect prediction lowers the overall accuracy.

The figure below illustrates key performance indicators (KPIs) achieved using the Rock-Paper-Scissors dataset, including:

Accuracy: Proportion of correct predictions over total predictions ROC Curve: Graphical representation of true positive rate vs. false positive rate Recall: Measure of the model's ability to identify all relevant instances Precision: Proportion of true positives among all predicted positives



Overall accuracy: 95.037% Class accuracies:

- paper = 98.394%
- scissor = 97.500%
- rock = 92.476%
- unknown = 92.083% Average ROC AUC: 98.664% Class ROC AUC:
- paper = 99.461%
- unknown = 99.042%
- scissor = 98.554%
- rock = 97.600%

Additional Topics

Third Party Tooling and Partner Solutions

Tooling

TensorFlow

The SiSDK supports TensorFlow Lite for Microcontrollers (TFLM) natively. The developer can create a quantized `.tflite` model file using the TensorFlow environment directly and integrate it into the GDSK.

Getting Started

- Developing a Model Using TensorFlow and Keras

Silicon Labs Machine Learning Toolkit (MLTK) (Deprecated)

The Machine Learning Toolkit (MLTK) was created for these reasons to help the Tensorflow developer. It is a set of python scripts designed to follow the typical machine learning workflow for Silicon Labs embedded devices.

> Note:
>
> - This package is made available as an open source, self-serve, community supported, reference package with a comprehensive set of online documentation. There are no Silicon Labs support services for this software at this time.
> - These scripts are a reference implementations for the use case covered by the documented tutorials. Support for other use cases is the responsibility of the developer.

Getting Started

- Create a Keyword Spotting Model for an SiSDK example. (GitHub.io) - MLTK Tutorial

Integrating the Model

The trained model, whether using TensorFlow directly or the MLTK, is represented in a `.tflite` file. To add the `.tflite` file to your embedded application, see developing an inference application.

There are several SiSDK examples that include machine learning and show how to run inference using pre-trained models. The pre-trained models can be replaced with a model trained from one of the methods above.

Getting Started

- Start with any one of these Sample Applications that are provided with the SiSDK.

Partner Toolchains

These tools curate the end-to-end workflow for creating a machine learning application specific model and accompanying embedded software to include in your application.

They offer:

- An end-to-end coverage of the machine learning workflow
- An easy-to-follow developer experience
- Extra value added features like AutoML, or object detection

Requires:

- Understanding of machine learning concepts (date collection, training, verification, inference, confusion matrix, etc.).
- Ability to collect data that represents the real world scenarios for your product deployment.

The following tools fit well with developing ML as a feature for a Silicon Labs-based embedded application.

Edge Impulse

Edge Impulse is ushering in the future of embedded machine learning by empowering developers to create and optimize solutions with real-world data. They are making the process of building, deploying, and scaling embedded ML applications easier and faster than ever, unlocking massive value across every industry, with millions of developers making billions of devices smarter.

Getting Started

Using Edge Impulse Studio with the xG24 Dev Kit

SensiML

SensiML offers cutting edge AutoML embedded code generation software for implementing AI at the IoT edge. With SensiML Analytics Toolkit, developers have an AI development tool which supports rapid data collection, labeling, feature extraction, ML classification and optimized firmware code generation. Automation built into the tool drastically reduces development time and cost, allowing projects ranging from single users to large teams to generate optimized edge AI sensor algorithms in a fraction of the time that would have otherwise been required with hand coding.

Getting Started

Using SensiML Analytics Toolkit with the xG24 Dev Kit

Micro.AI

MicroAI's AtomML is an Edge-native, self-correcting, semi-supervised learning engine that aggregates data from internal device sensors, to tune itself to create a behavioral profile of the asset, which then detects and acts upon abnormal behavior. AtomML brings big infrastructure intelligence into a single piece of equipment or device. Unlike traditional AI-driven asset management solutions that rely on the cloud, AtomML is deployed directly to smart devices and sensors. AtomML operates within the small environment of the device itself, providing a more efficient method for asset analytics and generating real-time alerts. AtomML brings an intimate, local approach to asset management for producing a host of operational efficiencies.

Getting Started

Using MicroAI AtomML with the xG24 Dev Kit

Partner Solutions

These tools do not require any knowledge of machine learning to take full advantage of their features. They are designed for specific use cases.

Wake-word/Voice Commands

Sensory creates a safer and superior UX through vision and voice technologies. Sensory technologies are widely deployed in consumer electronics applications including mobile phones, automotive, wearables, toys, IoT, PCs, medical products, and various home electronics. Sensory’s product line includes TrulyHandsfree voice

control, TrulySecure biometric authentication, and TrulyNatural large vocabulary natural language embedded speech recognition. Sensory's technologies have shipped in over three billion units of leading consumer products.

Getting Started

Using Sensory Truly Hands Free with the xG24 Dev Kit

Choosing a Machine Learning Tool Based on Use Case

This section provides getting started links for tools available or suggested, based on use case. The links provided include examples, demos, and tutorials.

Sensor Signal Processing

Sensor signal processing is the use of low data rate sensors, like accelerometers, gyroscopes, air quality sensors, temperature sensors, pressure sensors, and so on. These use cases can be found in many types of markets, such as preventive maintenance, medical devices, or smoke detectors.

- TensorFlow Magic Wand Example: SiSDK Platform Example
- Predictive Maintenance Tutorial (PDF)
- Air Quality Anomaly Detection Demo: Micro.ai Demo (video 1.5 min)
- BLE PowerPoint Gesture Controller Demo: Neuton Demo (video 3 min)
- BLE PowerPoint Gesture Controller Example: Neuton GitHub example
- Smartwatch Touch Free Control: Neuton Demo (video 1 min)
- Package Condition Monitoring: Neuton Demo (video 2.5 min)

Audio Pattern Matching

Audio pattern matching uses a microphone to detect different sounds. The types of sounds detected can be a wide range of non-speech related sounds, such as squeaky bearings, jingling keys, breaking glass, water running, animal sounds, and so on.

- Creating an Acoustic Smart Home Door Lock Demo: SensiML Tutorial (5 part blog)
- Introduction to Building an Audio Classifier Demo: Edge Impulse Tutorial Introduction (video 10 min)
- Industrial Predictive Maintenance: Edge Impulse Tutorial (video 70 min), with pre-work instructions (PDF), start on page 12
- Audio classifier Example: MLTK Example

Voice Command

Voice commands is a specific subset of audio patterns that are the recognition of a small set of spoken words. This is also referred to as keyword spotting. It can be used in a variety of use cases, such as waking up a voice service (i.e., Alexa), or using voice activation for smart home devices or appliances.

- Zigbee Voice Controlled Light Switch Example: SiSDK Zigbee Example
- Voice Controlled LED Example: SiSDK Platform Example
- Create a Voice Command Model for a SiSDK Example: MLTK Tutorial
  - This tutorial can be used for either example above to create a model (i.e., `.tflite` file).
- Building a Voice Recognition Demo: Edge Impulse Tutorial (video 90 min)
- Voice Command/Wake-word Example: Sensory Truly Hands Free Example

Low-Resolution Vision

Low resolution vision uses a camera with resolution around 100x100 to detect objects for presence detection, people counting, video wake-up or more.

- People Flow and Counting Example: SiSDK Platform Example

- People Counting Demo: Edge Impulse Example
- Build a Rock, Paper, Scissors Detector Demo: MLTK Tutorial
- Build a Fingerprint Authenticator Demo: MLTK Tutorial

Summary of Machine Learning Tools

Below is a table that summarizes our different tool options organized by developer skills and target use cases.

	ML EXPERT	ML EXPLORER	ML SOLUTIONS
LOW-RATE SENSORS	TensorFlow	EDGE IMPULSE SensiML	Micro.ai Anomaly Detection
AUDIO PATTERN MATCHING	TensorFlow	EDGE IMPULSE SensiML	
VOICE COMMANDS	TensorFlow	EDGE IMPULSE SensiML	sensory Voice Command
LOW-RESOLUTION VISION	TensorFlow	EDGE IMPULSE	