دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

پیاده سازی عملیاتی یادگیری ماشین روی ESP32 از صفر تا صد

محمدامین نقیان

تابستان ۱۴۰۳

Contents

۱- یادگیری ماشین چیست؟

یادگیری ماشین روشی برای نوشتن برنامه‌های کامپیوتری است. به طور خاص، روشی برای نوشتن برنامه‌هایی است که داده‌های خام را پردازش کرده و آن‌ها را به اطلاعاتی تبدیل می‌کنند که در سطح کاربردی معنادار هستند.

برای مثال، یک برنامه یادگیری ماشین ممکن است طوری طراحی شود که تشخیص دهد یک ماشین صنعتی کی خراب شده است، بر اساس داده‌های مختلف حسگرهای آن، تا بتواند به اپراتور هشدار دهد. برنامه دیگر ممکن است داده‌های صوتی خام را از یک میکروفون بگیرد و تعیین کند که آیا کلمه‌ای گفته شده است یا خیر، تا بتواند یک دستگاه هوشمند خانگی را فعال کند.

برخلاف برنامه‌های کامپیوتری معمولی، قوانین برنامه‌های یادگیری ماشین توسط یک توسعه‌دهنده تعیین نمی‌شوند. بلکه، یادگیری ماشین از الگوریتم‌های خاصی استفاده می‌کند تا قوانین را از داده‌ها بیاموزد، که به این فرآیند «آموزش» گفته می‌شود.

در یک نرم‌افزار سنتی، یک مهندس الگوریتمی را طراحی می‌کند که ورودی را می‌گیرد، قوانین مختلفی را اعمال می‌کند و یک خروجی تولید می‌کند. عملیات داخلی الگوریتم توسط مهندس برنامه‌ریزی شده و به‌طور صریح از طریق خطوط کد پیاده‌سازی می‌شود. برای پیش‌بینی خرابی در یک ماشین صنعتی، مهندس باید درک کند که کدام اندازه‌گیری‌ها در داده‌ها نشان‌دهنده یک مشکل هستند و کدی بنویسد که به‌طور هدفمند آن‌ها را بررسی کند.

این روش برای بسیاری از مسائل به خوبی کار می‌کند. برای مثال، ما می‌دانیم که آب در دمای ۱۰۰ درجه سانتی‌گراد در سطح دریا به جوش می‌آید، بنابراین نوشتن یک برنامه که بتواند بر اساس دمای فعلی و ارتفاع، پیش‌بینی کند که آیا آب در حال جوشیدن است یا خیر، آسان است. اما در بسیاری از موارد، ممکن است دانستن ترکیب دقیق عواملی که یک حالت خاص را پیش‌بینی می‌کنند، دشوار باشد. به عنوان مثال، در مورد ماشین صنعتی ما، ممکن است ترکیبات مختلفی از نرخ تولید، دما و سطح لرزش وجود داشته باشد که نشان‌دهنده یک مشکل باشد، اما از طریق مشاهده ساده داده‌ها بلافاصله مشخص نشود.

برای ایجاد یک برنامه یادگیری ماشین، یک مهندس ابتدا مجموعه بزرگی از داده‌های آموزشی را جمع‌آوری می‌کند. سپس این داده‌ها را به یک نوع خاص از الگوریتم وارد می‌کند و به الگوریتم اجازه می‌دهد تا قوانین را کشف کند. این بدان معناست که به عنوان مهندسان یادگیری ماشین، ما می‌توانیم برنامه‌هایی ایجاد کنیم که بر اساس داده‌های پیچیده پیش‌بینی‌هایی انجام دهند، بدون اینکه خودمان تمامی پیچیدگی‌ها را درک کنیم.

در طول فرآیند آموزش، الگوریتم یادگیری ماشین یک مدل از سیستم بر اساس داده‌هایی که ارائه می‌دهیم می‌سازد. سپس ما داده‌ها را از این مدل عبور می‌دهیم تا پیش‌بینی‌ها را انجام دهیم، که به این فرآیند «استنباط» گفته می‌شود.

۱-۱- یادگیری ماشین کجا می‌تواند کمک کند؟

یادگیری ماشین ابزاری عالی برای حل مسائلی است که شامل تشخیص الگوها هستند، به‌ویژه الگوهایی که پیچیده‌اند و ممکن است شناسایی آن‌ها برای یک ناظر انسانی دشوار باشد. الگوریتم‌های یادگیری ماشین در تبدیل داده‌های خام پیچیده و با پهنای باند بالا به سیگنال‌های قابل استفاده بسیار مهارت دارند، به‌خصوص زمانی که با پردازش سیگنال معمولی ترکیب شوند.

برای مثال، یک فرد معمولی ممکن است در تشخیص علائم خرابی یک ماشین با وجود ده جریان مختلف از داده‌های حسگر متراکم و پر سر و صدا دچار مشکل شود. با این حال، یک الگوریتم یادگیری ماشین می‌تواند اغلب تفاوت‌ها را شناسایی کند.

اما یادگیری ماشین همیشه بهترین ابزار برای انجام کار نیست. اگر قوانین یک سیستم به خوبی تعریف شده باشند و بتوان آن‌ها را به‌راحتی با منطق کدنویسی ثابت بیان کرد، معمولاً کار به این روش کارآمدتر است.

۱-۲- محدودیت‌های یادگیری ماشین

الگوریتم‌های یادگیری ماشین ابزارهای قدرتمندی هستند، اما ممکن است معایب زیر را داشته باشند:

- خروجی آن‌ها تخمین‌ها و تقریب‌ها هستند، نه پاسخ‌های دقیق
- اجرای مدل‌های یادگیری ماشین می‌تواند از نظر محاسباتی پرهزینه باشد
- جمع‌آوری داده‌های آموزشی می‌تواند زمان‌بر و پرهزینه باشد

ممکن است وسوسه شوید که سعی کنید یادگیری ماشین را در همه جا به کار ببرید—اما اگر بتوانید مشکلی را بدون یادگیری ماشین حل کنید، معمولاً بهتر است این کار را انجام دهید.

۲- یادگیری ماشین در سیستم‌های نهفته

پیشرفت‌های اخیر در معماری میکروپروسسورها و طراحی الگوریتم‌ها امکان اجرای وظایف پیچیده یادگیری ماشین را حتی بر روی کوچک‌ترین میکروکنترلرها فراهم کرده است. یادگیری ماشین نهفته که به عنوان TinyML نیز شناخته می‌شود، شاخه‌ای از یادگیری ماشین است که در سیستم‌های نهفته مانند این‌ها به کار گرفته می‌شود.

استفاده از یادگیری ماشین در دستگاه‌های نهفته مزایای مهمی دارد. مزایای کلیدی آن به خوبی در مخفف BLERP، که توسط جف بیر ابداع شده است، بیان شده‌اند. این مزایا عبارتند از:

- **پهنای باند — (Bandwidth)** الگوریتم‌های یادگیری ماشین روی دستگاه‌های لبه‌ای می‌توانند اطلاعات معناداری را از داده‌هایی استخراج کنند که در غیر این صورت به دلیل محدودیت پهنای باند، دسترسی به آن‌ها ممکن نیست.
- **زمان تأخیر — (Latency)** مدل‌های یادگیری ماشین روی دستگاه می‌توانند به ورودی‌ها در زمان واقعی پاسخ دهند و امکان‌پذیر ساختن کاربردهایی مانند وسایل نقلیه خودران که در صورت وابستگی به تأخیر شبکه عملی نخواهند بود.
- **اقتصاد — (Economics)** با پردازش داده‌ها بر روی دستگاه، سیستم‌های یادگیری ماشین نهفته از هزینه‌های انتقال داده از طریق شبکه و پردازش آن در فضای ابری جلوگیری می‌کنند.
- **قابلیت اطمینان — (Reliability)** سیستم‌هایی که توسط مدل‌های روی دستگاه کنترل می‌شوند ذاتاً قابل اطمینان‌تر از سیستم‌هایی هستند که به اتصال به فضای ابری وابسته‌اند.
- **حریم خصوصی — (Privacy)** هنگامی که داده‌ها در یک سیستم نهفته پردازش می‌شوند و هرگز به فضای ابری منتقل نمی‌شوند، حریم خصوصی کاربران حفظ می‌شود و احتمال سوءاستفاده از داده‌ها کمتر می‌شود.

۳- پیاده‌سازی یادگیری ماشین در ESP32

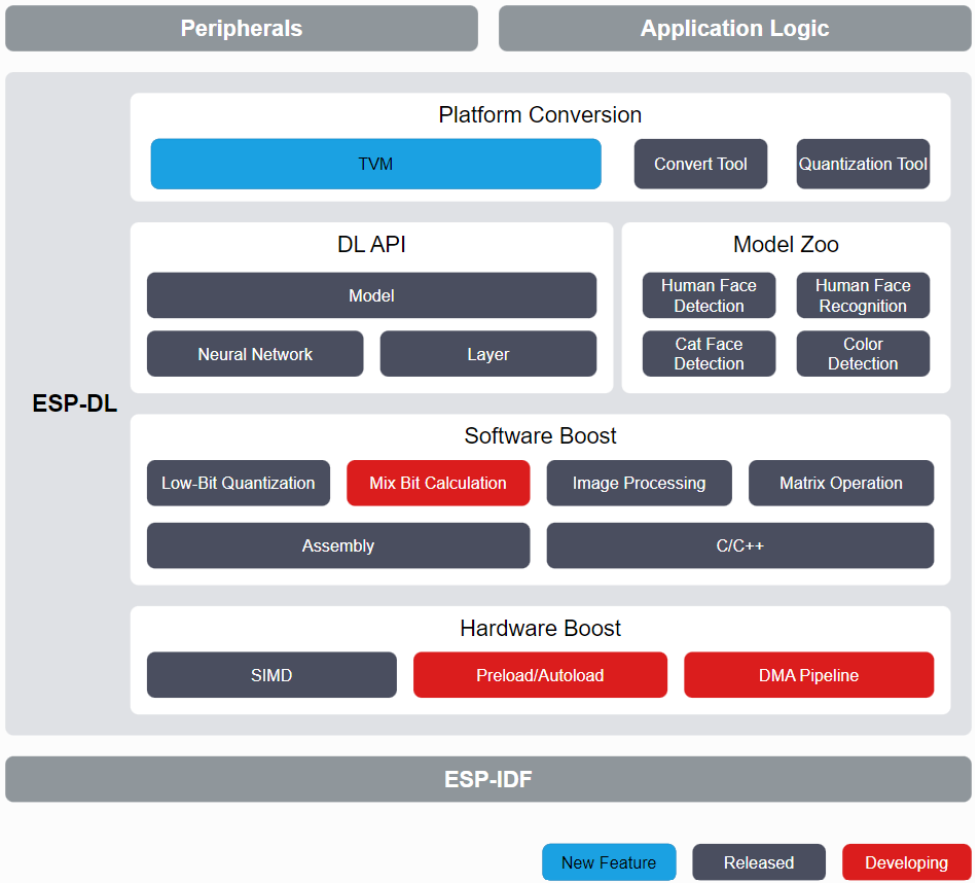
۳-۱- ESP-DL:

ESP-DL یک کتابخانه برای منابع یادگیری عمیق با عملکرد بالا است که به‌طور خاص برای پردازنده‌هایESP32 ، ESP32-S2، ESP32-S3وESP32-C3 طراحی شده است.

ESP-DL، API هایی برای استنباط شبکه‌های عصبی(NN Inference) ، پردازش تصویر، عملیات ریاضی و برخی مدل‌های یادگیری عمیق فراهم می‌کند. با استفاده ازESP-DL ، می‌توانید از سیستم-روی-چیپ‌های Espressif (SoCs) برای برنامه‌های هوش مصنوعی به‌راحتی و به‌سرعت استفاده کنید.

از آنجایی که ESP-DL به هیچ‌گونه سخت‌افزار جانبی نیاز ندارد، می‌توان از آن به عنوان یک جزء در برخی پروژه‌ها استفاده کرد. برای مثال، می‌توانید از آن به عنوان یک کامپوننت در ESP-WHO استفاده کنید، که شامل چندین مثال سطح پروژه‌ای از کاربردهای تصویری است. تصویر زیر نشان می‌دهد که ESP-DL از چه بخش‌هایی تشکیل شده است و چگونه به عنوان یک جزء در یک پروژه پیاده‌سازی می‌شود.

ESP-DL به توسعه‌دهندگان امکان می‌دهد که از امکانات یادگیری عمیق با کارایی بالا بر روی میکروکنترلرهای ESP استفاده کنند و برای پروژه‌هایی که به قابلیت‌های هوش مصنوعی نیاز دارند، یک راه‌حل کارآمد و ساده ارائه می‌دهد. این کتابخانه می‌تواند به راحتی در پروژه‌های مختلف ادغام شود و بدون نیاز به سخت‌افزار اضافی، قابلیت‌های پیشرفته‌ای را به دستگاه‌های نهفته اضافه کند.



۲-۳- Edge Impulse:

EDGE IMPULSE

Edge Impulse یک پلتفرم جامع برای توسعه، آموزش، و استقرار مدل‌های یادگیری ماشین بر روی دستگاه‌های نهفته(embedded devices) و دستگاه‌های لبه(edge devices) است. این پلتفرم به توسعه‌دهندگان و مهندسان این امکان را می‌دهد که با استفاده از داده‌های حسگرهای مختلف، مدل‌های یادگیری ماشین را به‌راحتی ایجاد و پیاده‌سازی کنند.

۱-۲-۳- ویژگی‌ها و قابلیت‌های Edge Impulse:

- **پشتیبانی از انواع سخت‌افزارها** :Edge Impulse از طیف گسترده‌ای از سخت‌افزارهای نهفته مانندArduino ، Raspberry Pi، ESP32 و بسیاری از میکروکنترلرها و سیستم‌های لبه پشتیبانی می‌کند. این پشتیبانی به کاربران اجازه می‌دهد که پروژه‌های خود را بر روی پلتفرم‌های مختلف اجرا کنند.
- **جمع‌آوری و پردازش داده** :این پلتفرم ابزارهای متنوعی برای جمع‌آوری، برچسب‌گذاری و پیش‌پردازش داده‌ها ارائه می‌دهد. این داده‌ها می‌توانند شامل داده‌های سنسورها مانند شتاب‌سنج، ژیروسکوپ، میکروفون، دوربین و دیگر حسگرها باشند.

- **آموزش مدل‌های یادگیری ماشین:** Edge Impulse محیطی برای آموزش مدل‌های یادگیری ماشین فراهم می‌کند. این محیط می‌تواند به‌صورت آنلاین و از طریق مرورگر مورد استفاده قرار گیرد. کاربران می‌توانند از الگوریتم‌های مختلف یادگیری ماشین و یادگیری عمیق استفاده کنند و مدل‌های خود را بر اساس نیازهای پروژه بهینه‌سازی کنند.
- **استقرار مدل‌ها** :پس از آموزش، مدل‌های یادگیری ماشین می‌توانند به‌راحتی بر روی دستگاه‌های نهفته استقرار یابند. Edge Impulseبهینه‌سازی‌های لازم برای اجرای مدل‌ها در محیط‌های با منابع محدود را فراهم می‌کند، به‌طوری‌که مدل‌ها بتوانند با کارایی بالا و مصرف انرژی کم عمل کنند.

۲-۲-۳- کاربردهایEdge Impulse :

- **تشخیص صوت**: تشخیص کلمات کلیدی، تشخیص دستورات صوتی، تشخیص صدای محیط.
- **پردازش تصویر**: تشخیص اشیا، تشخیص چهره، طبقه‌بندی تصاویر.
- **تشخیص حرکت و فعالیت**: استفاده از داده‌های شتاب‌سنج و ژیروسکوپ برای تشخیص حرکات و فعالیت‌ها.
- **نظارت صنعتی**: نظارت بر وضعیت ماشین‌آلات، تشخیص ناهنجاری‌ها و پیش‌بینی خرابی‌ها.

به طور کلی، Edge Impulse یک ابزار قدرتمند و آسان برای استفاده است که به توسعه‌دهندگان امکان می‌دهد هوش مصنوعی را به دستگاه‌های نهفته و لبه اضافه کنند، بدون اینکه نیاز به تخصص عمیق در یادگیری ماشین یا تجربه‌ی گسترده در برنامه‌نویسی داشته باشند.

۳-۳- TensorFlow Lite

TensorFlow Lite برای میکروکنترلرها به‌گونه‌ای طراحی شده است که مدل‌های یادگیری ماشین را بر روی میکروکنترلرها و دیگر دستگاه‌هایی که تنها چند کیلوبایت حافظه دارند، اجرا کند. هسته‌ی اصلی این برنامه در تنها ۱۶ کیلوبایت حافظه بر روی یک میکروکنترلر Arm Cortex M3 جای می‌گیرد و می‌تواند بسیاری از مدل‌های پایه را اجرا کند. این برنامه نیازی به پشتیبانی سیستم عامل، کتابخانه‌های استاندارد C یا C++، یا تخصیص دینامیک حافظه ندارد.

۱-۳-۳- پلتفرم‌های پشتیبانی‌شده در TensorFlow Lite

TensorFlow Lite برای میکروکنترلرها با زبان C++ 17 نوشته شده و به یک پلتفرم ۳۲ بیتی نیاز دارد. این ابزار به‌طور گسترده با بسیاری از پردازنده‌هایی که بر اساس معماری سری Arm Cortex-M ساخته شده‌اند، آزمایش شده و به سایر معماری‌ها از جمله ESP32 نیز پورت شده است. این چارچوب به‌عنوان یک کتابخانه برای Arduino در دسترس است و می‌تواند پروژه‌هایی را برای محیط‌های توسعه‌ای مانند Mbed تولید کند. این ابزار متن‌باز است و می‌توان آن را در هر پروژه C++ 17 گنجاند.

بردهای توسعه‌ی زیر پشتیبانی می‌شوند:

-Arduino Nano 33 BLE Sense

-SparkFun Edge

-STM32F746 Discovery kit

-Adafruit EdgeBadge

-Adafruit TensorFlow Lite for Microcontrollers Kit

-Adafruit Circuit Playground Bluefruit

-Espressif ESP32-DevKitC

-Espressif ESP-EYE

-Wio Terminal: ATSAMD51

-Himax WE-I Plus EVB Endpoint AI Development Board

-Synopsys DesignWare ARC EM Software Development Platform

-Sony Spresense

۴- مثال "Hello World"

مثال "Hello World" به‌منظور نمایش اصول اولیه استفاده از TensorFlow Lite برای میکروکنترلرها طراحی شده است. در این مثال، مدلی آموزش داده و اجرا می‌شود که تابع سینوسی را بازتولید می‌کند؛ یعنی، یک عدد به‌عنوان ورودی دریافت کرده و مقدار سینوسی آن عدد را به‌عنوان خروجی ارائه می‌دهد. وقتی این مدل روی میکروکنترلر پیاده‌سازی می‌شود، پیش‌بینی‌های آن برای چشمک زدنLED ها یا کنترل یک انیمیشن استفاده می‌شود.

روند کار از ابتدا تا انتها شامل مراحل زیر است:

۱. **آموزش مدل (با پایتون):** یک فایل پایتون برای آموزش، تبدیل و بهینه‌سازی مدل جهت استفاده روی دستگاه.

۲. **اجرای استنتاج(با17 C++):** یک تست واحد کامل که استنتاج را روی مدل با استفاده از کتابخانه C++ اجرا می‌کند.

آموزش یک مدل

توجه: شما می‌توانید این بخش را نادیده بگیرید و از مدل آموزش‌دیده‌ای که در کد مثال موجود است، استفاده کنید.

برای آموزش مدل "Hello World" جهت تشخیص موج سینوسی، از فایل train.py استفاده کنید.

دستور زیر را برای آموزش مدل اجرا کنید:

```
bazel build tensorflow/lite/micro/examples/hello_world:train
bazel-bin/tensorflow/lite/micro/examples/hello_world/train --save_tf_model
--save_dir=/tmp/model_created/
```

اجرای استنتاج

۱. اضافه کردن هدرهای کتابخانه

برای استفاده از کتابخانه TensorFlow Lite برای میکروکنترلرها، باید فایل‌های هدر زیر را در کد خود اضافه کنید:

```
#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"
```

`micro_mutable_op_resolver.h`: عملیات‌هایی را فراهم می‌کند که توسط مفسر برای اجرای مدل استفاده می‌شود.

`micro_error_reporter.h`: اطلاعات اشکال‌زدایی را خروجی می‌دهد.

`micro_interpreter.h`: شامل کدی است که برای بارگذاری و اجرای مدل‌ها استفاده می‌شود.

`schema_generated.h`: اسکیما یا ساختار قالب فایل مدل TensorFlow Lite FlatBuffer را شامل می‌شود.

`version.h`: اطلاعات مربوط به نسخه‌ی اسکیما TensorFlow Lite را فراهم می‌کند.

۲. اضافه کردن هدر مدل

مفسر TensorFlow Lite برای میکروکنترلرها، مدل را به‌صورت یک آرایه C++ انتظار دارد. این مدل در فایل‌های `model.h` و `model.cc`تعریف شده است. برای اضافه کردن هدر مدل، خط زیر را در کد خود وارد کنید:

```
#include "tensorflow/lite/micro/examples/hello_world/model.h"
```

۳. اضافه کردن هدر فریمورک تست واحد

برای ایجاد یک تست واحد (Unit Test)، باید هدر فریمورک تست واحد TensorFlow Lite برای میکروکنترلرها را با اضافه کردن خط زیر به کد خود اضافه کنید:

```
#include "tensorflow/lite/micro/testing/micro_test.h"
```

تست با استفاده از ماکروهای زیر تعریف می‌شود:

```
TF_LITE_MICRO_TESTS_BEGIN
TF_LITE_MICRO_TEST(LoadModelAndPerformInference) {
  . // add code here
  .
}
TF_LITE_MICRO_TESTS_END
```

در ادامه، کدی که در این ماکروها قرار می‌گیرد را بررسی می‌کنیم.

۴. راه‌اندازی لاگینگ

برای راه‌اندازی لاگینگ، یک اشاره‌گر `tflite::ErrorReporter` با استفاده از یک نمونه‌ی `tflite::MicroErrorReporter` ایجاد می‌شود:

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

این متغیر به مفسر پاس داده می‌شود که به آن اجازه می‌دهد لاگ‌ها را بنویسد. از آنجا که میکروکنترلرها اغلب مکانیزم‌های مختلفی برای لاگینگ دارند، پیاده‌سازی `tflite::MicroErrorReporter` به گونه‌ای طراحی شده که می‌توان آن را برای دستگاه خاص شما سفارشی‌سازی کرد.

۵. بارگذاری یک مدل

در کد زیر، مدل با استفاده از داده‌هایی از یک آرایه‌ی `char` به نام `g_model` که در `model.h` تعریف شده، ایجاد می‌شود. سپس مدل را بررسی می‌کنیم تا مطمئن شویم نسخه‌ی اسکیما با نسخه‌ای که استفاده می‌کنیم سازگار است:

```
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);
}
```

۶. ایجاد عملیات resolver

یک نمونه از `MicroMutableOpResolver` تعریف می‌شود. این resolver توسط مفسر برای ثبت و دسترسی به عملیات‌هایی که مدل استفاده می‌کند، استفاده می‌شود:

```
using HelloWorldOpResolver = tflite::MicroMutableOpResolver<1>;

TfLiteStatus RegisterOps(HelloWorldOpResolver& op_resolver) {
  TF_LITE_ENSURE_STATUS(op_resolver.AddFullyConnected());
  return kTfLiteOk;
}

HelloWorldOpResolver op_resolver;
TF_LITE_ENSURE_STATUS(RegisterOps(op_resolver));
```

`MicroMutableOpResolver` به یک پارامتر template نیاز دارد که نشان‌دهنده‌ی تعداد عملیات‌هایی است که ثبت خواهند شد. تابع `RegisterOps` این عملیات‌ها را در resolver ثبت می‌کند.

۷. تخصیص حافظه

ما نیاز داریم که مقداری حافظه برای آرایه‌های ورودی، خروجی، و واسطها از پیش تخصیص دهیم. این کار با یک آرایه‌ی `uint8_t` به اندازه‌ی `tensor_arena_size` انجام می‌شود:

```
const int tensor_arena_size = 2 * 1024;
uint8_t tensor_arena[tensor_arena_size];
```

اندازه‌ی مورد نیاز بستگی به مدل شما دارد و ممکن است نیاز به آزمون و خطا برای تعیین دقیق آن باشد.

۸. ایجاد مفسر

یک نمونه‌ی `tflite::MicroInterpreter` ایجاد می‌کنیم و متغیرهای ایجاد شده قبلی را به آن پاس می‌دهیم:

```
tflite::MicroInterpreter interpreter(model, resolver, tensor_arena,
tensor_arena_size, error_reporter);
```

۹. تخصیص تنسورها

به مفسر اطلاع می‌دهیم که حافظه را از `tensor_arena` برای تنسورهای مدل تخصیص دهد:

```
interpreter.AllocateTensors;()
```

۱۰. اعتبارسنجی شکل ورودی

با فراخوانی `input(0)`. روی نمونه `MicroInterpreter`، می‌توانیم به یک اشاره‌گر به تنسور ورودی مدل دسترسی پیدا کنیم. در اینجا، 0 نمایانگر اولین (و تنها) تنسور ورودی است:

```
  // Obtain a pointer to the model's input tensor
  TfLiteTensor* input = interpreter.input(0);
```

سپس این تنسور را بررسی می‌کنیم تا اطمینان حاصل کنیم که شکل و نوع آن مطابق انتظار ما است:

```
// Make sure the input has the properties we expect
TF_LITE_MICRO_EXPECT_NE(nullptr, input);
// The property "dims" tells us the tensor's shape. It has one element for
// each dimension. Our input is a 2D tensor containing 1 element, so
"dims"
// should have size 2.
TF_LITE_MICRO_EXPECT_EQ(2, input->dims->size);
// The value of each element gives the length of the corresponding tensor.
// We should expect two single element tensors (one is contained within
the
// other).
TF_LITE_MICRO_EXPECT_EQ(1, input->dims->data[0]);
TF_LITE_MICRO_EXPECT_EQ(1, input->dims->data[1]);
// The input is a 32 bit floating point value
TF_LITE_MICRO_EXPECT_EQ(kTfLiteFloat32, input->type);
```

مقدار `kTfLiteFloat32` یکی از انواع داده‌های TensorFlow Lite است که در `common.h` تعریف شده است.

۱۱. فراهم کردن یک مقدار ورودی

برای دادن ورودی به مدل، محتویات تنسور ورودی را به صورت زیر تنظیم می‌کنیم:

```
input->data.f[0] = 0.;
```

در این مثال، یک مقدار با نقطه شناور که برابر 0 است را وارد می‌کنیم.

۱۲. اجرای مدل

برای اجرای مدل، می‌توانیم تابع `Invoke()` را روی نمونه `tflite::MicroInterpreter` فراخوانی کنیم:

```
TfLiteStatus invoke_status = interpreter.Invoke();
if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed\n");
}
```

می‌توانیم مقدار بازگشتی که از نوع `TfLiteStatus` است را بررسی کنیم تا متوجه شویم آیا اجرا موفق بوده است یا خیر. مقادیر ممکن برای `TfLiteStatus` که در `common.h` تعریف شده‌اند، شامل `kTfLiteOk` و `kTfLiteError` هستند.

کد زیر بررسی می‌کند که آیا مقدار برابر `kTfLiteOk` است، به این معنی که استنتاج با موفقیت انجام شده است:

```
TF_LITE_MICRO_EXPECT_EQ(kTfLiteOk, invoke_status);
```

۱۳. به دست آوردن خروجی

می‌توان تنسور خروجی مدل را با فراخوانی `output(0)` روی `tflite::MicroInterpreter` به دست آورد، که در اینجا `۰` نمایانگر اولین (و تنها) تنسور خروجی است.

در این مثال، خروجی مدل یک مقدار نقطه شناور است که درون یک تنسور ۲ بعدی قرار دارد:

```
TfLiteTensor* output = interpreter.output(0);
TF_LITE_MICRO_EXPECT_EQ(2, output->dims->size);
TF_LITE_MICRO_EXPECT_EQ(1, input->dims->data[0]);
TF_LITE_MICRO_EXPECT_EQ(1, input->dims->data[1]);
TF_LITE_MICRO_EXPECT_EQ(kTfLiteFloat32, output->type);
```

می‌توانیم مقدار را مستقیماً از تنسور خروجی خوانده و بررسی کنیم که آیا مطابق انتظار ما است:

```
// Obtain the output value from the tensor
float value = output->data.f[0];
// Check that the output value is within 0.05 of the expected value
TF_LITE_MICRO_EXPECT_NEAR(0., value, 0.05);
```

۱۴. اجرای دوباره استنتاج

باقی‌مانده کد چندین بار دیگر استنتاج را اجرا می‌کند. در هر بار، مقداری را به تنسور ورودی اختصاص می‌دهیم، مفسر را اجرا می‌کنیم، و نتیجه را از تنسور خروجی می‌خوانیم:

```
input->data.f[0] = 1.;
interpreter.Invoke();
value = output->data.f[0];
TF_LITE_MICRO_EXPECT_NEAR(0.841, value, 0.05);

input->data.f[0] = 3.;
interpreter.Invoke();
value = output->data.f[0];
TF_LITE_MICRO_EXPECT_NEAR(0.141, value, 0.05);

input->data.f[0] = 5.;
interpreter.Invoke();
value = output->data.f[0];
TF_LITE_MICRO_EXPECT_NEAR(-0.959, value, 0.05);
```