

Машинное обучение на микроконтроллере STM32F407 – обнаружение вибраций

Валентин Юрзин

В статье подробно описан процесс создания модели и архитектура библиотеки MML, а также приведено прямое сравнение точности работы библиотеки от NanoEdge AI Studio и модели MML, созданной автором на целевом микроконтроллере STM32F407VET6.

Современная концепция Edge AI [1] кардинально меняет подход к мониторингу промышленного оборудования. Традиционная передача сырых данных в облако требует высокой пропускной способности сети и увеличивает задержки. Внедрение машинного обучения непосредственно на микроконтроллеры TinyML [2] позволяет анализировать сигналы локально. Это обеспечивает мгновенную реакцию системы и полную автономность датчиков.

В качестве аппаратной платформы для таких задач отлично подходят микроконтроллеры серии STM32F4. Они обладают оптимальным балансом производительности ядра ARM Cortex-M4 и имеют аппаратный блок вычислений с плавающей точкой (FPU).

Реализация нейросетевых алгоритмов на «голом железе» (Bare Metal) требует специализированных инструментов. В данной статье рассматривается практический опыт создания системы обнаружения аномальных вибраций с использованием платформы автоматизации: NanoEdge AI Studio – программной среды от STMicroelectronics для генерации оптимизированных библиотек аномалий.

NeyrinoAI – авторская программа для мониторинга работы компактных ML-моделей на микроконтроллере STM32F407.

Использование готовых систем генерации библиотек накладывает ограничения в виде закрытого исходного кода. Для решения этой проблемы автором была разработана собственная библиотека MML (Micro Machine Learning) [3].

Главная цель создания MML – обеспечение полной технологической независимости разработки и возможность гибкой настройки всей системы под конкретную инженерную задачу.

Применение ИИ и машинного обучения

Машинное обучение на микроконтроллерах – это разновидность моделей искусственного интеллекта, запускаемых непосредственно на миниатюрных электронных компонентах с ограниченными вычислительными ресурсами (микроконтроллерах или специализированных чипах), например, на STM32A407.

Модель реализуется непосредственно на микроконтроллере, обучается данными, поступающими с датчиков, и затем распознаёт полезные (обученные) данные в потоке сигнала или группы сигналов без помощи Интернета.

Рассмотрим созданные библиотеки модели MML с помощью NanoEdge AI Studio и протестируем их работу от вибраций мотора. Оценим поведение трёхкомпонентного сигнала акселерометра и метрики схожести на экране ПК в приложении NeyrinoAI, специально созданном автором для мониторинга работы моделей MML. Создадим свою модель MML и также протестируем её работу.

Программное обеспечение NanoEdge AI Studio

NanoEdge AI Studio – это бесплатное программное обеспечение от ST, позволяющее легко интегрировать искусственный интеллект в любые встроенные проекты, работающие на любом микроконтроллере Arm® Cortex®-M.

Программное обеспечение NanoEdge AI Studio, работающее локально на ПК, принимает входные данные и генерирует библиотеку, которая включает в себя модель, её предварительную обработку и функции для лёгкой интеграции в новые или существующие встроенные проекты.

Проект обнаружение аномалий с библиотекой, сгенерированной NanoEdge AI Studio, обеспечивает обучение модели для различия двух типов поведения: номинального и аномального.

Этот тип проекта также обладает уникальной возможностью переобучения непосредственно на микроконтроллере.

Для создания проекта нам потребуется набор данных, созданных с помощью NanoEdge AI или записанных с определённым форматом данных и сохранённых в виде текстового файла.

Формат записи в файл: минимум 20 строк, 255 значений в строке. Данные в строке имеют вид: значение X, значение Y, значение Z: «2 1 9 2 1 9 2 1 9 ... и т.д.», где значение X = 2; значение Y = 1; значение Z = 9.

Этапы создания библиотеки MML.

- Настройка и загрузка аномальных и стандартных сигналов.
- Тест производительности будущей модели MML.
- Валидация – оценка «лучшей библиотеки».
- Компиляция библиотеки MML.

В результате NanoEdge AI создаёт библиотечный файл «libneai.a» с расширением *.a. Файл «libneai.a» – это откомпилированная библиотека, в которой невозможно изменить или отредактировать код. Этот файл необходимо встроить в проект для использования функций машинного обучения.

Более подробная информация о работе с ПО NanoEdge AI Studio доступна здесь [4].

Следует отметить, что перестроить или скорректировать созданную модель уже нельзя.

Настройка модели происходит только на этапе сбора сигналов путём изменения амплитуды значений осей X, Y, Z.

Программное обеспечение NeyrinoAI

Проверку и сравнение поведения сгенерированных моделей будем про-

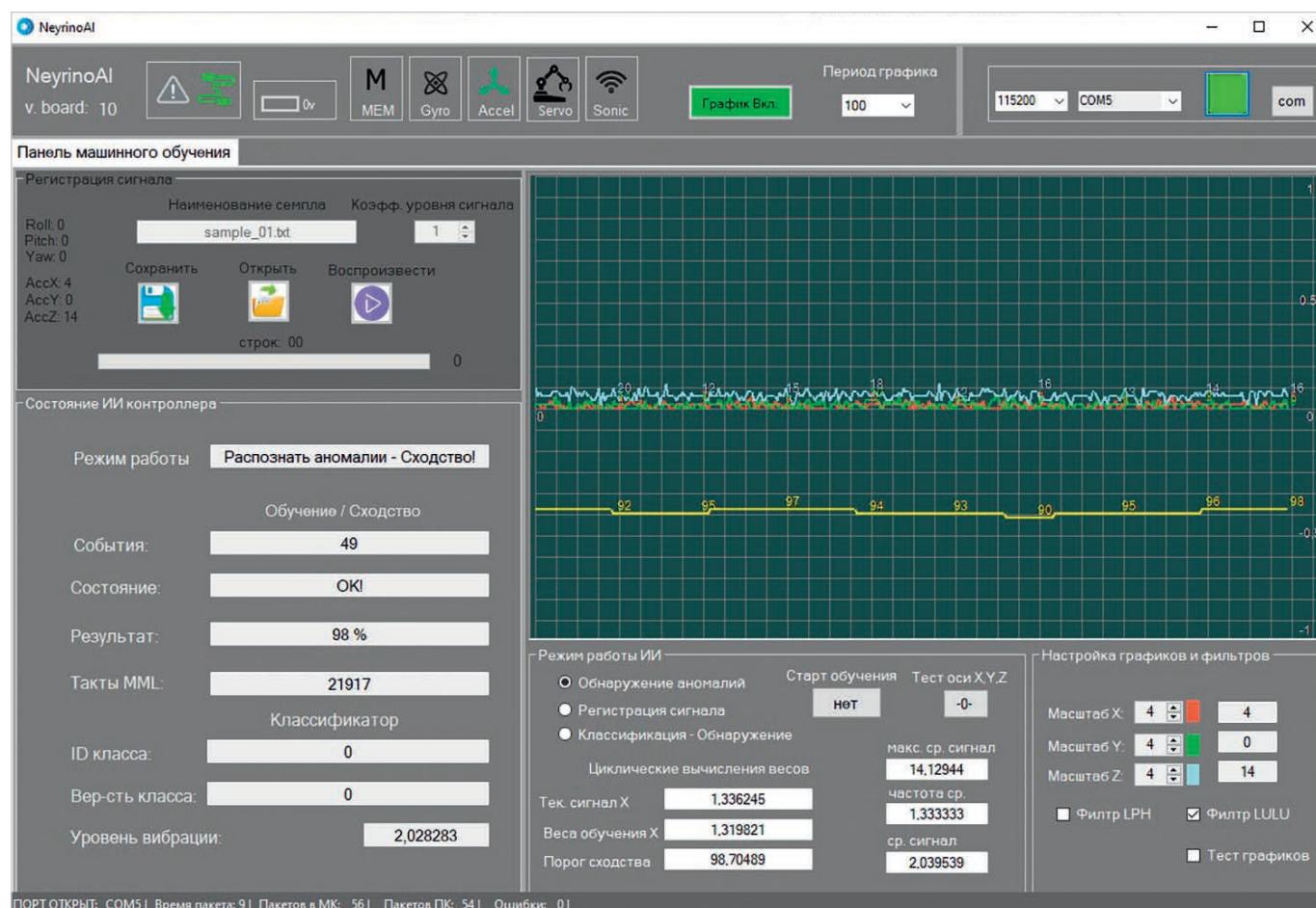


Рис. 1. Визуальное оформление NeyrinoAI

водить с помощью программного обеспечения NeyrinoAI, созданного для мониторинга работы MML. Визуальное оформление NeyrinoAI представлено на рис. 1.

Панель машинного обучения содержит следующие параметры.

1. Окно графиков визуализации данных акселерометра и коэффициента схожести.
2. Группа параметров «Режим работы ИИ контроллера». Здесь происходит переключение режимов работы приложения:
 - обнаружение аномалий, регистрация сигнала, классификация;
 - текущее значение сигнала на оси X в модели MML;
 - веса [5] обучения оси X, соответствующие циклу обучения;
 - порог сходства;
 - кнопки «Старт обучения» и «Тест оси», значения пиков сигнала всех осей, средней частоты и общее среднеквадратичное значение осей.
3. Группа параметров «Состояние ИИ контроллера».
 - Здесь отображаются: события модели, состояние, результат, количество тактов в MML.

- если работает классификатор сигнала, отображается ID класса и вероятность класса.

4. Группа параметров «Регистрация сигнала».

- Здесь происходит управление созданием файлов с записанными значениями осей X, Y, Z в формате приложения NanoEdge AI Studio, которые приведены выше.

5. В группе «Настройка графиков и фильтров» можно изменить масштабирование сигналов в окне графиков и управлять работой фильтров в самом контроллере.

- Дополнительные фильтры: LPH – низкой частоты, LULU – фильтр от резких случайных выбросов – работают после получения сырых данных с акселерометра и не относятся к модели MML.

Для запуска модели MML используется плата контроллера Neyrino, созданного на основе микроконтроллера STM32F407VET6, который имеет 512 Кбайт флеш-памяти и 192 Кбайт оперативной памяти.

На плате контроллера установлен акселерометр L3G4200D, подключённый по интерфейсу I²C.

Программное обеспечение для STM32F407 будет создаваться в среде разработки IAR system.

Внутренняя организация программы для микроконтроллера STM32F407VET6.

Задача 1. Поток USB (асинхронный, событийный).

Функция: передача потоковых пакетов с ПК на плату контроллера и обратно. Размер пакета 64 байта.

Задача 2. Поток акселерометра (высокий приоритет, постоянный). Периодичность: каждые 20 миллисекунд (в зависимости от требований модели).

Функция: считывает «сырые» данные (X, Y, Z) через интерфейс I²C.

Оптимально запускать задачу по аппаратному прерыванию от вывода INT акселерометра (Data Ready), чтобы не тратить ресурсы на постоянный опрос.

Задача 3. Поток модели MML (средний приоритет). Периодичность: каждые 20 миллисекунд (в зависимости от требований модели). Запускается сразу после получения новых данных от акселерометра.

Функция: принимает буфер данных, выполняет математическую обработку (фильтрация, вычисления, обуче-

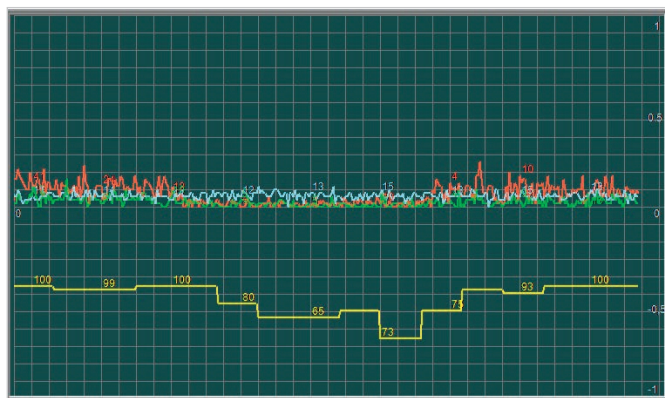


Рис. 2. Модель MML NanoEdge AI Studio в работе



Рис. 3. Модель NanoEdge AI Studio, обученная на сигнале оси X = 55 и значениях шума на осях Y, Z

ние и распознавание MML). Результат (например, состояние оборудования) записывает в регистры модели.

Задача 4. Поток MODBUS-RTU (средний/низкий приоритет).

Функция: обрабатывает запросы от мастер-устройства по UART. Читает данные из общей карты регистров MML и отправляет обратно.

Важно: поток USB работает асинхронно по отношению к другим задачам программы.

Скорость потока пакетов составляет 60–50 пакетов в секунду, что составляет от 16 до 20 миллисекунд на пакет. В пакетах передаются данные для визуального отображения на экране ПК.

Модель MML – обнаружение вибраций

На рис. 2 представлена работа модели MML NanoEdge AI Studio. На экране четырёхканального осциллографа отображаются сигналы с трёх измерительных осей акселерометра X, Y, Z и результирующий вектор схожести:

- ось X – красного цвета;
- ось Y – зелёного цвета;
- ось Z – светло-голубого цвета.

Схожесть сигналов – жёлтый цвет (0–100%). Значение более 90% – можно считать, что модель MML распознала совокупный сигнал.

Сигналы на экране разнесены по оси Y на несколько пикселей для лучшего наблюдения, чтобы они как можно меньше перекрывали друг друга.

Методика измерений.

- Движение графиков происходит справа налево.
- В начале осциллограммы идёт суммарный сигнал из трёх осей. Это часть сигналов, на которую обучена модель MML.
- В средней части происходит изменение сигналов от воздействия ви-

браций или программной подмены значений. Последовательное программное изменение значений осей применяется для визуального отклика модели.

- В конце графиков суммарный сигнал осей восстанавливается до обученного моделью уровня.

Таким образом, на одном экране можно наблюдать реакцию модели на обученный и альтернативный сигнал.

Время движения графиков справа налево занимает 40 секунд.

Этого достаточно для того, чтобы оценить поведение модели MML.

Следует отметить, что значения схожести появляются с некоторой задержкой по отношению к изменению сигналов осей X, Y, Z. Это происходит потому, что программный сбор сигналов акселерометра в буфер для последующего анализа методом БПФ [6] занимает примерно 1,3 с и отстаёт по времени от изменения сигналов по осям: $20 \text{ мс} \times 64 = 1280 \text{ мс}$, где:

- 20 мс – это период опроса датчика акселерометра;
- 64 – количество значений окна для анализа FFT-преобразований [6].

В качестве источника вибраций будем использовать вентилятор от блока питания ПК, который имеет небольшой радиальный люфт в роторе.

Будем располагать плату контроллера с датчиком на вентиляторе в разных точках для того, чтобы получить разные значения на осях акселерометра.

На рис. 2 модель NanoEdge AI Studio обучена на сигналах с некоторым преобладанием значений на оси X. Затем происходит остановка вращения вентилятора, и суммарный сигнал на графике не превышает амплитуды шума состояния покоя.

Модель правильно определяет схожесть и различие сигналов.

На рис. 3 модель NanoEdge AI Studio обучена на сигнале оси X = 55 и значениях шума на осях Y, Z.

При изменении значения на оси X = шум, а на оси Y = 55 модель не реагирует на изменения сигналов. То же происходит и при изменении значений на осях Z = 55, Y = шум. Только при общем снижении амплитуд сигналов модель MML показывает отсутствие схожести. Модель неправильно определяет схожесть и различие сигналов.

На рис. 4 модель NanoEdge AI Studio обучена на сигнале оси Y = 55 и значениях шума на осях X, Z. Поведение модели такое же, как и на рис. 3. Модель не распознаёт равные взаимные изменения на осях. Только при общем снижении амплитуд сигналов модель MML показывает отсутствие схожести. Модель неправильно определяет схожесть и различие сигналов.

Наша модель MML

Цель:

- различать данные, генерируемые во время нормальной работы машины (двигателя), и данные, указывающие на наличие проблем. Для этого требуется генерировать и анализировать оба типа данных: данные, отражающие нормальную работу или состояние покоя, и данные, полученные в моменты возникновения аномалий в работе объекта наблюдения;
- необходимо объединить данные осей X, Y, Z без потери их уникальности, сохранив независимость каждой из них.

Обычное сложение уничтожает данные. Когда мы складываем значения трёх осей арифметически, мы стираем границы между пространственны-

ми измерениями. Это приводит к трём критическим проблемам:

- взаимная компенсация (эффект маскировки): если по оси X значение равно +10, а по оси Y равно -10, их сумма даст 0. Для системы это будет выглядеть как полное отсутствие активности, хотя на самом деле произошло сильное движение по двум направлениям;
- потеря вектора (подмена источника): сумма $X = 10$, $Y = 0$, $Z = 0$ равна 10. Сумма $X = 0$, $Y = 10$, $Z = 0$ тоже равна 10. Получив на выходе число 10, алгоритм не сможет определить, по какой именно оси (или осям) пришёл сигнал;
- физическая некорректность: с точки зрения геометрии сложение координат разных осей напрямую нарушает правила векторных пространств. Для решения проблемы суммирования трёх осей необходимо:
- вычисление модуля вектора, чтобы получить единое общее значение (интенсивность, силу) и гарантировать, что показатели одной оси не будут подменяться другими при сохранении весов [5];
- исключение компенсации: возведение в квадрат убирает знаки минус. Любое отклонение по любой из осей всегда увеличивает итоговый результат;
- равноправие осей: каждая ось вносит свой независимый вклад в «энергию» общего сигнала. Значения не поглощают друг друга;
- структурирование данных в коде, чтобы избежать случайной подмены осей на этапе вычислений. Данные должны быть изолированы на уровне архитектуры;
- в качестве дополнительного бонуса рассчитаем пиковые значения амплитуд и частот сигналов осей.

Рекомендации по созданию библиотеки.

1. Не следует использовать динамическое выделение памяти (malloc/free), поскольку в микроконтроллерах, особенно при реализации задач TinyML/ML, использование динамической памяти обычно не рекомендуется.
2. Для STM32F407 (ядро Cortex-M4 с аппаратным модулем вычислений с плавающей запятой FPU) оптимальным решением является использование официальной и глубоко оптимизированной библиотеки CMSIS-DSP. Она выполняет FFT [6] в

несколько раз быстрее алгоритмов, написанных без CMSIS-DSP, благодаря применению SIMD-инструкций процессора.

Преимущество данного подхода для STM32F407 – экономия тактов CPU: библиотека CMSIS-DSP написана разработчиками ARM на ассемблере с использованием инструкций SIMD, что позволяет обрабатывать 64 точки за микросекунды.

Алгоритм нашей модели MML с использованием инструкции ARM CMSIS-DSP.

// Некоторые глобальные объявления:
#define LERN 10 // Количество циклов обучения

#define LEN_WINDOWS 64 // Длина окна (степень двойки для FFT)

#define NUM_FEATURES 3 // Количество извлекаемых признаков

// Структура нашей модели MML

```
typedef struct {
    float weights_AxisX[LERN] __attribute__((aligned(4))); // Веса оси X
    float weights_AxisY[LERN] __attribute__((aligned(4))); // Веса оси Y
    float weights_AxisZ[LERN] __attribute__((aligned(4))); // Веса оси Z
    float weights_sqr[LERN] __attribute__((aligned(4))); // Веса среднего значения сигнала
    float bias; // Смещение
} MML_Model_t;
```

Три основные функции:

1. Инициализация математических модулей – anomalydetection_init(void).
2. Обучение модели – anomalydetection_learn(float *inBuff).
3. Распознавание схожести – anomalydetection_detect(float *inBuff, uint8_t *similarity).

В функции инициализации происходит:

- инициализация и заполнение всех массивов нулями;
- инициализация структуры FFT (RFFT) из библиотеки CMSIS-DSP.

В функции обучения модели:

- происходит режим сбора данных – по 64 отсчёта для трёх осей X, Y, Z с акселерометра;
- оконное преобразование Фурье (STFT) – 64 значения в окне для каждой оси;
- извлекаются признаки частоты, пиковой амплитуды и общей энергии сигнала для трёх осей;
- обучение модели (сохранение веса для каждой оси X, Y, Z).

В функции распознавания схожести:

- происходит режим сбора данных – по 64 отсчёта для трёх осей X, Y, Z с акселерометра;

- оконное преобразование Фурье (STFT) – 64 значения в окне для каждой оси;

- расчёт степени сходства на основе среднеквадратичной ошибки (RMSE/RMS).

В функции обучения модели и распознавания схожести используется одна функция: функция сбора данных – 64 отсчёта для трёх осей X, Y, Z с акселерометра и оконное преобразование Фурье (STFT) – 64 значения в окне для каждой оси.

```
MML_apply_stft_3axis( float *inBuff,
float32_t *outFeatures, float32_t *outAxis_XYZ, int window_size);
```

Разберём её более подробно.

- Из общего входного отсчёта, состоящего из 64 значений для осей X, Y, Z, выделим массивы для каждой из осей X, Y, Z соответственно.

```
standard_x[i]; // семпл – эталон оси X.
```

```
standard_y[i]; // семпл – эталон оси Y.
```

```
standard_z[i]; // семпл – эталон оси Z.
```

- Расчёт математических признаков спектра. Ищем доминирующую частоту.

```
arm_max_f32(&magnitude_buffer[1],
(LEN_WINDOWS / 2) - 1, &max_magnitude, &max_index);
```

- Используем векторную функцию CMSIS для поиска максимума амплитуды.

- Считаем среднюю мощность спектра через векторное суммирование CMSIS.

```
arm_mean_f32(magnitude_buffer, LEN_WINDOWS / 2, &sum_magnitude);
```

```
// Получаем признаки сигнала для каждой оси для оси X, Y, Z.
```

```
features[0] = max_magnitude;
```

```
// Пиковая амплитуда движения
features[1] = max_index; // Доминирующая частота (шаг/вибрации)
```

```
features[2] = sum_magnitude;
```

```
// Общая энергия сигнала.
```

- Вычисляем RMS каждого эталона, чтобы понять его масштаб/энергию. Алгоритм использует RMS (извлекает квадратный корень), возвращая ошибку к исходной физической величине. Это делает сравнение с эталоном линейным и более интуитивным.

```
arm_rms_f32(standard_x, LEN_WINDOWS, &ref_rms);
```

```
arm_rms_f32(standard_y, LEN_WINDOWS, &ref_rms);
```



Рис. 4. Модель NanoEdge AI Studio, обученная на сигнале оси Y = 55 и значениях шума на осях X, Z

```
arm_rms_f32(standard_z, LEN_
WINDOWS, &ref_rms);
```

- Для запоминания весов сигнала используем: экспоненциальное скользящее среднее (EMA), простейший низкочастотный фильтр (IIR-фильтр) или бегущее среднее.

Формулу можно написать так:

$$S = 0,5 \times S(n-1) + 0,5 \times X(n).$$

Тут происходит сглаживание пиков: если в последовательности чисел произойдёт резкий скачок (например, шло 10, 10, 10 и вдруг пришло 100), среднее значение не подскочит до 100 сразу, а будет плавно подбираться к нему: $10 \rightarrow 55 \rightarrow 77,5 \rightarrow 88,75$.

Таким образом, в значениях масса весов нет резких скачков.

- Если функция работает в режиме обучения, сохраняем рассчитанные веса сигнала:
`my_model.weights_AxisX[couLernAI];`
`my_model.weights_AxisY[couLernAI];`
`my_model.weights_AxisZ[couLernAI];`
иначе – игнорируем сохранение весов.
- На выходе функции получаем признаки и веса сигнала.

В функции распознавания схожести используется алгоритм расчёта степени сходства на основе среднеквадратичной ошибки (RMSE/RMS) или метрики сходства по нормированной среднеквадратичной ошибке (NRMSE).

Этот алгоритм применяется, когда критически важны и форма, и абсолютные значения (например, совпадение физических сигналов, напряжений или аудиосигналов).

Он подходит для сравнения сигналов с датчиков или других физических процессов.

- Находим поэлементную разницу векторов ($pRef[i] - pInput[i]$); результат складывается во временный

```
byRef diff_buffer arm_sub_f32(pRef,
pInput, diff_buffer, len_size).
```

- Вычисляем RMS (среднеквадратичную ошибку) полученной разности. Алгоритм использует RMS (извлекает квадратный корень), возвращая ошибку к исходной физической величине. Это делает сравнение с эталоном линейным и более интуитивным. Функция внутри возводит элементы в квадрат, берёт среднее и извлекает корень `arm_rms_f32(diff_buffer, len_size, &rmse)`.
- Вычисляем RMS самого эталона, чтобы понять его масштаб/энергию: `arm_rms_f32(pRef, len_size, &ref_rms)`.
- Перевод величины ошибки в процент схожести:
$$\text{float32_t error_percent} = (\text{rmse} / \text{ref_rms}) * 100.0f;$$
$$\text{float32_t similarity} = 100.0f - \text{error_percent}.$$
На выходе функции получаем значение схожести от 0% до 100%.
Рекомендации по компиляции модели.

1. При компиляции проекта необходимо указать основной макрос в настройках проекта IAR, например, `ARM_MATH_CM4` для STM32F4.
2. Нулевое выделение памяти во время работы программы. Все массивы с типом (Float – вещественных чисел) должны быть объявлены статически и выровнены по границе 4 байт (`_attribute__((aligned(4)))`), что необходимо для корректной и быстрой работы шины FPU.
3. При компиляции со знаком «High» следует отключить пункт «Static Clustering» (статическая кластеризация). Этот метод оптимизации размещения данных в памяти, при котором глобальные и статические переменные, используемые внутри

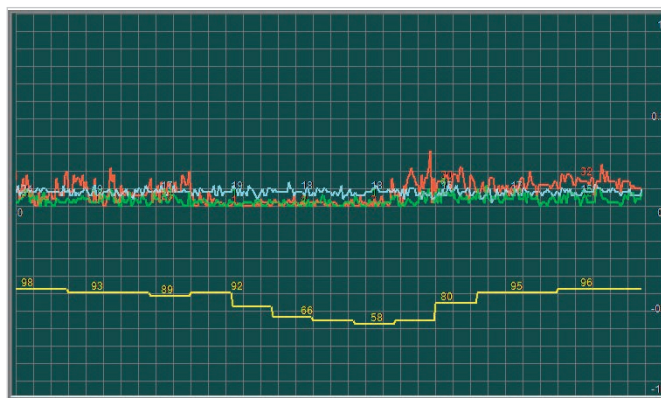


Рис. 5. Модель, обученная на сигналах с некоторым преобладанием значений на оси X

одной функции или модуля, группируются (кластеризуются) как можно ближе друг к другу в структуре RAM.

При использовании этого метода со знаком «High» функции ARM DSP в модели перестают работать.

Результаты измерений нашей модели MML

На рис. 5 наша модель обучена на сигналах с некоторым преобладанием значений на оси X. Затем происходит остановка вращения вентилятора, и суммарный сигнал на графике не превышает амплитуды шума состояния покоя. Модель правильно определяет схожесть и различие сигналов.

На рис. 6 наша модель обучена на сигналах оси X = 55 и значениях шума на осях Y, Z. При изменениях значения на оси X = шум, а на оси Y = 55 модель реагирует снижением процента схожести на изменения сигналов. При изменении значений на осях Z = 55, Y = шум также модель снижает процент схожести на изменения сигналов. При общем снижении амплитуд сигналов модель MML показывает отсутствие схожести. Модель правильно определяет схожесть и различие сигналов.

На рис. 7 наша модель обучена на сигнале оси $Y = 55$ и значениях шума на осях X, Z . Модель распознаёт равные, взаимные изменения на осях, снижая процент схожести. При общем снижении амплитуд сигналов модель также показывает отсутствие схожести. Модель правильно определяет схожесть и различие сигналов.

На рис. 8 более сложный пример. Наша модель обучена на сигналах с некоторым преобладанием значений на осях X, Y. Затем в общий спектр сигнала последовательно добавляются



Рис. 6. Модель, обученная на сигнале оси X = 55 и значениях шума на осях Y, Z

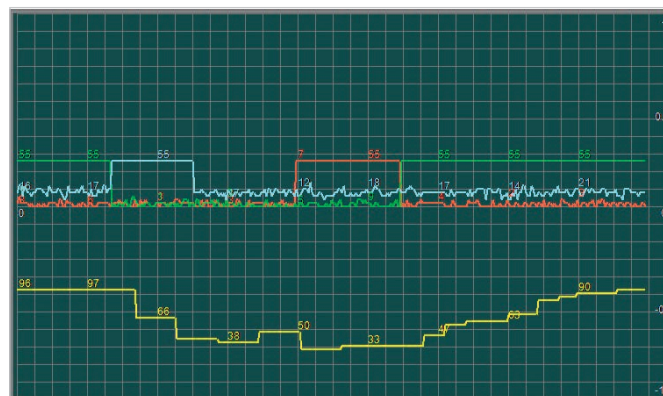


Рис. 7. Модель, обученная на сигнале оси Y = 55 и значениях шума на осях X, Z



Рис. 8. Модель, обученная на сигналах с некоторым преобладанием значений на осях X, Y



Рис. 9. Модель, обученная на сигналах X = 40, Y = 55, Z = 25

значения на осях X, Y, Z амплитудой 55. В результате модель показывает схожесть только по краям представленной осциллограммы. Во всех остальных случаях происходит снижение процента схожести.

На рис. 9 наша модель обучена на сигналах X = 40, Y = 55, Z = 25. Затем следуют значения шума на осях X, Y, Z. Далее последовательно изменяются значения на осях X = 55, Y = Z = шум, Y = 55, X = Z = шум. В результате модель точно реагирует на обученные сигналы, которые располагаются по краям графиков осей.

Выводы

Созданная нами модель MML обладает некоторой степенью избирательности, когда обучается на конкретный сигнал. Фактически модель запоминает определённый код из трёх сигналов и впоследствии способна на него реагировать (распознавать).

Применение ИИ на микроконтроллерах даёт ряд преимуществ: машинное обучение позволяет анализировать накапливаемые данные и находить в них закономерности. Эти закономерности могут быть основой новых бизнес-моделей.

Если есть возможность отслеживать и анализировать поведение какой-либо машины, то можно прогнозировать её аварийное состояние ещё до того, как она будет ломаться.

Одним из наиболее критических сценариев применения TinyML является предиктивное обслуживание (Predictive Maintenance) промышленных моторов и подшипниковых узлов на основе анализа вибраций. Таким образом, можно вовремя находить аномалии в любой работающей системе и выполнять превентивное техническое обслуживание.

Термины и полезные ссылки

1. Edge AI (периферийный искусственный интеллект) – это запуск алгоритмов и моделей ИИ прямо на локальных устройствах.
2. TinyML (Tiny Machine Learning) – это сфера искусственного интеллекта, которая занимается запуском моделей машинного обучения на микроконтроллерах, умных датчиках и чипах IoT (Интернета вещей), которые потребляют минимальное количество энергии.

3. Модель MML (микромашинное обучение) – это подход, направленный на создание точных и эффективных моделей с использованием минимального количества данных, вычислительной мощности и программного кода вместо гигантских нейросетей.
4. URL: https://wiki.st.com/stm32mcu/wiki/AI:How_to_Build_an_Anomaly_Detection_Project_for_Predictive_Maintenance_with_NanoEdge_AI_Studio.
5. Веса (Weights): числа, которые показывают важность связи между нейронами. Они усиливают или ослабляют сигнал, который идёт от одного слоя к другому. Смещение (Bias): дополнительное число, которое сдвигает график функции. Оно помогает модели точнее подстраиваться под данные, даже если все входные сигналы равны нулю.
6. FFT (Fast Fourier Transform – быстрое преобразование Фурье) – это быстрый вычислительный алгоритм для перевода сигнала или данных из временной (пространственной) области в частотную. Алгоритм сокращает число операций, что сильно ускоряет расчёты.