Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

К защите допустить:
И.О. Заведующего кафедрой информатики
_____ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

РАСПОЗНАВАНИЕ СТРОЧНЫХ И ПРОПИСНЫХ БУКВ КИРИЛЛИЦЫ И ЦИФР НА МИКРОКОНТРОЛЛЕРА ARM

БГУИР КП 1-40 04 01 027 ПЗ

Студент К. Ю. Фроленко

Руководитель А. А. Калиновская

Нормоконтролёр А. А. Калиновская

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире искусственный интеллект (ИИ) становится неотъемлемой частью различных сфер человеческой деятельности, начиная от автоматизации производственных процессов и заканчивая разработкой интеллектуальных бытовых устройств и систем развлечений. Одной из ключевых технологий ИИ является машинное зрение, и в частности распознавание рукописного текста. Эта область исследований играет важную роль в таких приложениях, как автоматизация ввода данных, создание интерактивных обучающих систем и совершенствование средств коммуникации, что делает ее крайне актуальной в условиях быстро развивающегося цифрового мира.

Распознавание рукописного текста представляет собой сложную задачу, требующую применения передовых методов машинного обучения и глубоких оптимизаций. Разнообразие почерков, стили написания символов и необходимость достижения высокой точности при ограниченных вычислительных ресурсах усиливают сложность задачи. Более того, скорость распознавания и эффективность системы играют ключевую роль в её успешном внедрении, особенно в контексте мобильных и встроенных устройств, где каждое вычисление влияет на производительность системы в целом.

Актуальность темы курсового проекта определяется растущей потребностью в автономных и энергоэффективных системах распознавания, которые могут работать на устройствах с ограниченными ресурсами. В условиях, когда доступ к облачным сервисам и мощным вычислительным системам ограничен или отсутствует, особенно в удаленных и малообеспеченных регионах, создание таких систем становится важной задачей. Внедрение локальных решений на базе микроконтроллеров позволяет значительно снизить задержки при обработке данных и повысить конфиденциальность, поскольку информация остается на устройстве, не покидая его пределов. Это особенно важно в контексте защиты данных и их безопасности.

Одной из основных сложностей при разработке таких систем является балансировка между сложностью модели и вычислительными возможностями микроконтроллера. Процессоры с архитектурой *ARM*, такие как *Cortex-M7*, хотя и обладают высокой производительностью для своего класса, существенно уступают по вычислительной мощности настольным компьютерам и облачным серверам. Это накладывает строгие ограничения на размеры нейронных сетей и объем доступной памяти, что делает задачу разработки системы еще более интересной и сложной.

Целью данной работы является разработка системы распознавания строчных и прописных букв кириллицы и цифр на микроконтроллере *ARM*, используя плату *STM32F746G-DISCO*.

Для достижения поставленной цели необходимо решить следующие задачи:

1 Изучить архитектуру вычислительной системы на основе микроконтроллера *ARM Cortex-M7* и обосновать выбор платы *STM32F746G-DISCO*.

2 Проанализировать программную платформу *TensorFlow Lite* для микроконтроллеров, её структуру, историю развития и ключевые преимущества.

3 Обосновать необходимость разработки программного продукта и определить технологии программирования, необходимые для решения поставленных задач.

4 Разработать и оптимизировать модель нейронной сети для распознавания рукописных символов кириллицы и цифр.

5 Реализовать программное обеспечение на микроконтроллере *STM32F746G-DISCO* и провести тестирование системы распознавания.

Достижение поставленной цели позволит продемонстрировать возможности выполнения сложных задач машинного обучения на устройствах с ограниченными ресурсами и оценить эффективность использования современных технологий ИИ во встраиваемых системах. Более того, успешная реализация системы подчеркнет потенциал использования нейронных сетей в микроконтроллерах для решения практических задач.

Детальная постановка задачи заключается в создании эффективной и экономичной системы распознавания рукописного текста, способной работать в режиме реального времени на микроконтроллере с ограниченными вычислительными ресурсами. Это предполагает обучение модели на базе данных рукописных символов, оптимизацию модели для выполнения на *ARM Cortex-M7* и разработку встроенного приложения для демонстрации возможностей системы.

1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

1.1 Структура и архитектура вычислительной системы

Микроконтроллеры семейства *ARM Cortex-M*, в частности модель *Cortex-M7*, являются высокопроизводительными 32-битными процессорами, специально разработанными для встраиваемых систем с ограниченными вычислительными ресурсами . В рамках данного проекта выбор пал на плату *STM32F746G-DISCO*, основанную на микроконтроллере *STM32F746NGH6* [1]. Этот микроконтроллер работает на базе ядра *Cortex-M7* с тактовой частотой до 216 МГц, что делает его подходящим для задач, требующих высокой производительности при ограниченных ресурсах.

На рисунке 1 представлена блок схема архитектуры системы *STM32F746G-DISCO*, демонстрирующая взаимодействие компонентов.

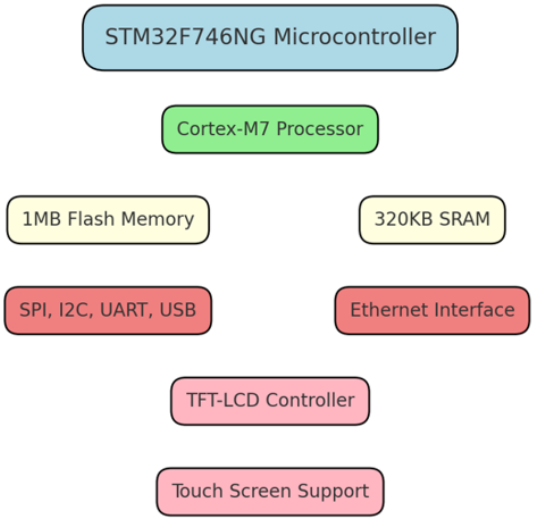


Рисунок 1 – Блок схема архитектуры системы *STM32F746G-DISCO*

Архитектура *Cortex-M7* обеспечивает отличную производительность благодаря поддержке операций с плавающей запятой и усовершенствованной системе предсказания переходов, что позволяет оптимально использовать вычислительные возможности при выполнении сложных программ. Эти особенности делают его идеальным выбором для реализации проекта по распознаванию рукописного текста, включая цифры, заглавные и прописные буквы русского алфавита, где требуется быстрая обработка данных и точное распознавание символов [2].

Кроме того, микроконтроллер *STM32F746NG* оснащен достаточным объемом памяти – 1 МБ флэш-памяти и 320 КБ оперативной памяти *SRAM*. Такой объем памяти позволяет разместить как само программное обеспечение, так и оптимизированную модель нейронной сети для эффективного выполнения задачи [3]. Благодаря поддержке различных периферийных интерфейсов, таких как *SPI*, *I2C*, *USART*, *USB* и *Ethernet*, данный

микроконтроллер может легко интегрироваться с различными внешними модулями и устройствами, что значительно расширяет возможности его применения в различных системах.

Плата *STM32F746G-DISCO* также предлагает расширенные графические возможности, включая встроенный контроллер *TFT-LCD* и поддержку сенсорного экрана, что делает её удобной для создания визуально привлекательных пользовательских интерфейсов. Это особенно полезно для демонстрации работы системы распознавания рукописного текста, предоставляя возможность наглядно отслеживать процесс распознавания и взаимодействовать с системой напрямую через экран.

1.2 История, версии и достоинства

Семейство микроконтроллеров *ARM Cortex-M*, представленное компанией *ARM Holdings* в начале 2000-х годов, быстро завоевало популярность благодаря своей эффективности и универсальности. *Cortex-M7*, выпущенный в 2014 году, стал флагманом среди высокопроизводительных микроконтроллеров [4]. Он выделяется благодаря высокой тактовой частоте и ряду усовершенствованных архитектурных решений, таких как шестиступенчатый суперскалярный конвейер и возможность выполнения нескольких инструкций за один такт, что делает его идеальным для приложений, требующих интенсивной обработки данных в реальном времени.

На рисунке 2 представлена гистограмма энергопотребления различных процессоров *ARM Cortex*, демонстрирующая невысокое энергопотребление *Cortex-M7* по сравнению с другими моделями.

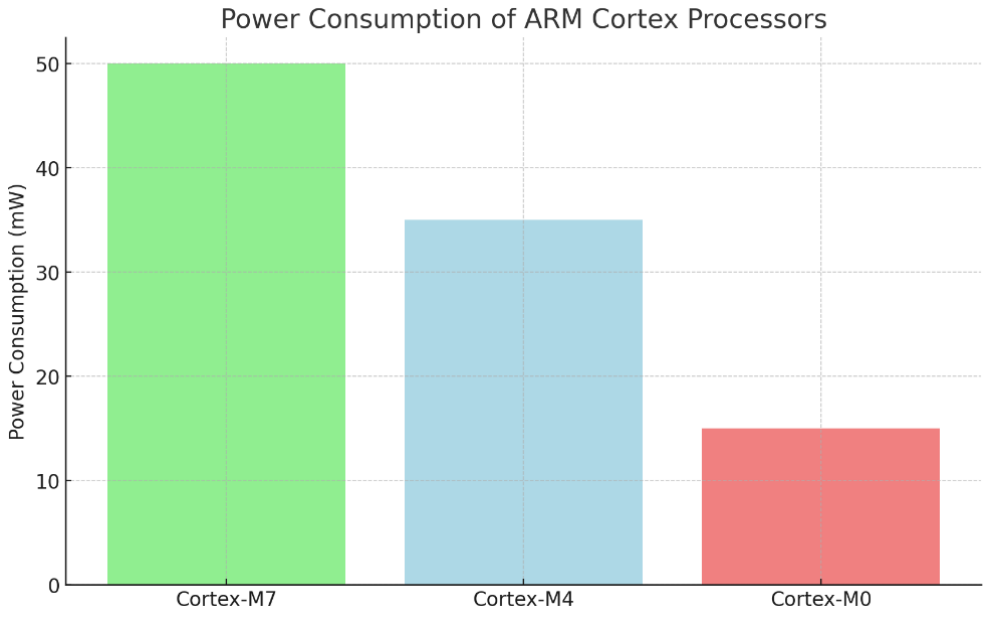


Рисунок 2 – Гистограмма энергопотребления различных процессоров *ARM Cortex*

Основные преимущества *Cortex-M7* включают высокую производительность, позволяющую выполнять сложные вычисления в реальном времени, что особенно важно для задач машинного обучения, таких как распознавание рукописного текста.

Низкое энергопотребление делает его привлекательным для встраиваемых систем и мобильных устройств, где важна продолжительная автономная работа. Оптимизация энергопотребления вместе с высокой производительностью позволяет применять эти микроконтроллеры в широком спектре задач, от бытовых до промышленных решений [5].

Микроконтроллеры семейства *Cortex-M7* широко поддерживаются разработчиками благодаря развитой экосистеме инструментов, таких как среды разработки, библиотеки и готовые решения, что значительно упрощает процесс создания и отладки приложения. Большое сообщество разработчиков, обширная база знаний и доступные ресурсы делают разработку на базе этих микроконтроллеров максимально удобной.

Совместимость с *ARM*-стандартами и поддержка различных операционных систем реального времени обеспечивают гибкость в разработке, позволяя интегрировать *Cortex-M7* в самые разнообразные системы и проекты.

1.3 Обоснование выбора вычислительной системы

Выбор платы *STM32F746G-DISCO* для данного проекта обусловлен рядом ключевых факторов. Прежде всего, микроконтроллер *STM32F746NG*, на котором основана плата, обладает необходимой производительностью и объемом памяти, что делает его подходящим для работы с оптимизированными моделями машинного обучения [6]. Это особенно важно в рамках задачи распознавания рукописного текста, где требуется обработка сложных данных в режиме реального времени.

В таблице 1 приведено сравнение характеристик микроконтроллеров *STM32F746G*, *STM32F4* и *STM32F0*, демонстрирующее их основные отличия по параметрам, таким как процессор, объем памяти и поддерживаемые интерфейсы.

Таблица 1 – Сравнение характеристик микроконтроллеров

Parameter	*STM32F746G*	*STM32F4*	*STM32F0*
Processor	*Cortex-M7 (216 MHz)*	*Cortex-M4 (180 MHz)*	*Cortex-M0 (48 MHZ*
Flash Memory	*1MB*	*512KB*	*256KB*
SRAM	*320KB*	*192KB*	*32KB*
Interfaces	*SPI, I2C, UART, USB, Ethernet*	*SPI, I2C, UART, USB*	*SPI, I2C, UART, USB*
Display	*Cortex-M7 (216 MHz)*	*Cortex-M4 (180 MHz)*	*Cortex-M0 (48 MHZ*

Одним из значительных преимуществ *STM32F746G-DISCO* является развитая экосистема, которую поддерживает производитель *STMicroelectronic*. Наличие обширной документации, примеров кода, а также поддержка популярных инструментов разработки, таких как *STM32CubeIDE* и *Keil MDK-ARM*, значительно упрощает процесс реализации и позволяет сосредоточиться на основных задачах проекта. Такая интеграция с проверенными инструментами обеспечивает разработчикам возможность быстро приступить к созданию сложных приложений и значительно сокращает время на отладку [7].

Кроме того, плата предоставляет широкие аппаратные возможности. Встроенный 4,3-дюймовый *TFT-LCD*-дисплей с сенсорным экраном облегчает разработку и тестирование пользовательского интерфейса, что является важным аспектом для визуализации результатов распознавания и взаимодействия с системой. Такая интеграция аппаратных средств сокращает необходимость в дополнительном оборудовании, что упрощает прототипирование и демонстрацию работы системы.

Активное сообщество разработчиков, связанное с платформой *STM32*, также играет важную роль. Доступ к форумам, репозиториям с примерами кода, а также возможностям обратной связи обеспечивает разработчикам необходимые ресурсы и поддержку. Это особенно важно на этапах разработки и тестирования, когда возникают вопросы или трудности, требующие помощи со стороны сообщества.

Таким образом, плата *STM32F746G-DISCO* идеально соответствует требованиям проекта, предоставляя как высокую производительность, так и удобство разработки благодаря поддержке со стороны производителя и сообщества.

1.4 Анализ выбранной вычислительной системы

При разработке системы распознавания на базе *STM32F746G-DISCO* важно учитывать ряд специфических особенностей архитектуры микроконтроллера и обеспечить эффективное управление доступными ресурсами. Одним из ключевых аспектов является управление памятью. В условиях ограниченного объема оперативной и флэш-памяти крайне важно оптимизировать размещение как модели нейронной сети, так и программного кода [8]. Методы квантизации играют здесь ключевую роль, позволяя существенно уменьшить размер модели без значительной потери точности распознавания. Это открывает возможности для эффективного использования ресурсов микроконтроллера и повышения общей производительности системы.

Кроме того, производительность системы во многом зависит от эффективного использования аппаратных возможностей микроконтроллера. Встроенный блок операций с плавающей запятой и поддержка цифровой обработки сигналов позволяют значительно ускорить выполнение вычислений и обработку данных. Это особенно важно при работе с задачами

машинного обучения, которые требуют высокой производительности для распознавания рукописных цифр и букв в реальном времени.

Совместимость микроконтроллера с *TensorFlow Lite for Microcontrollers* – еще один важный аспект разработки. *TensorFlow Lite* предоставляет возможность интеграции оптимизированных моделей машинного обучения в микроконтроллерные приложения, что упрощает разработку и внедрение современных технологий ИИ. Использование готовых библиотек и инструментов *TensorFlow Lite* сокращает время на разработку и позволяет сосредоточиться на оптимизации и адаптации решения под конкретное оборудование.

Энергопотребление, хотя и не является критичным аспектом в данном проекте, все же заслуживает внимания. Оптимизация использования вычислительных и энергетических ресурсов способствует повышению надежности и общей эффективности системы, особенно если в дальнейшем планируется использовать решение в автономных устройствах.

Еще одной особенностью разработки под микроконтроллеры является необходимость тщательной оптимизации кода и моделей. В условиях ограниченных ресурсов важно использовать языки низкого уровня, такие как *C* и *C++*, которые позволяют более точно управлять процессом выполнения программы и ресурсами системы. Это также подразумевает использование специализированных инструментов отладки и программирования, обеспечивающих глубокую интеграцию с микроконтроллером. Такие инструменты позволяют выявлять узкие места в производительности и оптимизировать работу системы на всех этапах разработки [9].

Таким образом, успешная реализация системы распознавания на базе *STM32F746G-DISCO* требует учета множества аспектов – от правильного использования аппаратных возможностей микроконтроллера до грамотной оптимизации программного обеспечения. Баланс между производительностью, энергоэффективностью и эффективным управлением ресурсами станет ключом к созданию надежного и эффективного решения.

1.5 Заключение

Анализ архитектуры микроконтроллера *STM32F746G-DISCO* на базе процессора *Cortex-M7* подтвердил его пригодность для сложных задач машинного обучения благодаря высокой производительности, расширенным графическим возможностям и поддержке множества интерфейсов. Эта платформа обеспечивает необходимую мощность и гибкость для разработки и масштабирования систем распознавания, делая её оптимальным выбором для данного проекта.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 Структура и архитектура платформы

TensorFlow Lite для микроконтроллеров является частью обширной экосистемы *TensorFlow*, разработанной специально для выполнения моделей машинного обучения на устройствах с ограниченными вычислительными ресурсами [10]. Эта платформа предоставляет инструменты и средства, позволяющие эффективно запускать сложные модели на микроконтроллерах, таких как *STM32F746G-DISCO*, сохраняя производительность и снижая требования к аппаратным ресурсам. Она обеспечивает возможность интеграции ИИ в устройства с минимальными вычислительными мощностями.

Одним из ключевых компонентов *TensorFlow Lite* является конвертер моделей, который позволяет преобразовывать обученные модели TensorFlow в облегченный формат *TensorFlow Lite*. Этот формат специально оптимизирован для использования на мобильных и встраиваемых устройствах, что позволяет значительно сократить объем памяти, необходимый для хранения модели, и ускорить её выполнение. Использование конвертера является первым шагом в процессе внедрения модели на устройства с ограниченными ресурсами.

Следующим важным элементом является интерпретатор**,** представляющий собой легковесный движок для выполнения моделей. Интерпретатор *TensorFlow Lite* разработан для работы без необходимости в полноценной операционной системе или с минимальной её поддержкой. Это позволяет запускать модели машинного обучения непосредственно на микроконтроллере, не прибегая к использованию мощных вычислительных платформ, что является ключевым преимуществом при разработке автономных устройств и встраиваемых систем.

Также важную роль в архитектуре *TensorFlow Lite* играют оптимизированные операторы – специализированные реализации математических операций, адаптированные для эффективной работы на микроконтроллерах. Даже без поддержки аппаратного ускорения такие операторы обеспечивают приемлемую производительность при выполнении вычислений, что позволяет запускать модели машинного обучения даже на простых устройствах. Эти оптимизации позволяют максимально эффективно использовать вычислительные ресурсы микроконтроллера, что особенно важно при разработке систем, таких как распознавание рукописного текста на *STM32F746G-DISCO* [11].

Таким образом, структура платформы *TensorFlow Lite* для микроконтроллеров включает инструменты и механизмы, обеспечивающие эффективное выполнение моделей на устройствах с ограниченными ресурсами, что делает её удобным решением для встраиваемых систем.

2.2 История, версии и достоинства

TensorFlow был представлен компанией *Google* в 2015 году как мощная и открытая платформа для разработки и внедрения моделей машинного обучения [12]. Впоследствии была разработана облегченная версия – *TensorFlow Lite*, предназначенная для мобильных и встраиваемых систем, предлагая решение для задач, где объем памяти и вычислительной мощности ограничены. Она значительно упростила внедрение ИИ в мобильные и встроенные системы.

Версия *TensorFlow Lite* для микроконтроллеров расширила возможности этой платформы, предоставив разработчикам инструменты для выполнения моделей даже на самых простых устройствах с минимальными вычислительными мощностями. Одним из ключевых достоинств *TensorFlow Lite* для микроконтроллеров является компактность. Платформа специально разработана с минимальным размером двоичного кода и моделей, что делает её идеальной для работы на устройствах с ограниченным объемом ресурсов и памяти.

Еще одно преимущество – это производительность, обеспечиваемая за счет оптимизации под специфические архитектуры микроконтроллеров. Благодаря этим оптимизациям разработчики могут достичь приемлемой скорости выполнения моделей даже без наличия мощных вычислительных ресурсов или поддержки аппаратного ускорения. Это делает *TensorFlow Lite* отличным решением для приложений, таких как системы распознавания рукописного текста.

Гибкость платформы выражается в поддержке широкого спектра моделей машинного обучения. *TensorFlow Lite* для микроконтроллеров также предлагает возможности кастомизации, позволяя адаптировать модели под конкретные задачи и архитектуры. Это открывает множество вариантов использования, от простых систем автоматизации до более сложных приложений, таких как распознавание объектов или обработка естественного языка.

Сообщество и поддержка *TensorFlow Lite* для микроконтроллеров значительно облегчают процесс разработки программного обеспечения с использованием ИИ. Активное развитие платформы, поддержка со стороны *Google* и наличие обширной документации и примеров позволяют разработчикам быстро освоить платформу и интегрировать её в свои проекты. Большое количество обучающих материалов, доступные репозитории с кодом и форумы помогают решить возникающие вопросы и проблемы в процессе разработки.

Таким образом, *TensorFlow Lite* для микроконтроллеров предоставляет мощные и удобные инструменты. Эти инструменты нужны для внедрения машинного обучения на устройствах с ограниченными ресурсами, сочетая компактность, производительность, гибкость и широкую поддержку сообщества.

2.3 Обоснование выбора платформы

Выбор *TensorFlow Lite* для микроконтроллеров обусловлен несколькими ключевыми факторами, которые делают эту платформу оптимальным решением для проектов, требующих работы с машинным обучением на устройствах с ограниченными ресурсами [13]. Она позволяет разработчикам легко переносить уже обученные модели на микроконтроллеры, что значительно сокращает время разработки и адаптации. Важной особенностью является возможность работы в режиме реального времени, что критично для задач, связанных с распознаванием и обработкой данных на встраиваемых системах.

Прежде всего, важным преимуществом является совместимость с основной платформой *TensorFlow*, что позволяет использовать всю экосистему этой платформы для обучения и разработки моделей. Разработчики могут обучать модели на мощных вычислительных системах с использованием широкого набора инструментов *TensorFlow*, а затем легко переносить их на микроконтроллеры, используя *TensorFlow Lite*. Это значительно упрощает процесс внедрения моделей на устройства с ограниченными ресурсами.

Оптимизация для ограниченных ресурсов также является важным фактором. *TensorFlow Lite* предлагает инструменты и методы для уменьшения размера моделей, такие как квантизация, которая позволяет снизить требования к памяти без значительных потерь в точности. Кроме того, платформа оптимизирована для быстрого выполнения моделей, что особенно актуально для микроконтроллеров с ограниченной вычислительной мощностью.

Платформа предоставляет поддержку необходимого функционала для работы с моделями машинного обучения. *TensorFlow Lite* содержит множество встроенных операторов, необходимых для выполнения базовых математических операций и обработки данных. В случае необходимости функциональность можно расширить, добавляя персонализированные операторы для специфических задач, что делает платформу гибкой и адаптивной для различных проектов.

Также важным преимуществом является простота интеграции. *TensorFlow Lite* легко встраивается в существующие проекты на языках *C* и *C++*, что делает её совместимой с большинством микроконтроллерных платформ. Совместимость с популярными инструментами разработки, такими как *STM32CubeIDE* и другими средами, значительно упрощает процесс интеграции и отладки. Это позволяет быстро внедрить *TensorFlow Lite* в проект и сократить время на разработку.

Таким образом, выбор *TensorFlow Lite* для микроконтроллеров продиктован её совместимостью с основной платформой *TensorFlow*, оптимизацией под ограниченные ресурсы, поддержкой необходимого функционала и простотой интеграции в существующие системы.

2.4 Анализ программного обеспечения для написания программы

При разработке программного обеспечения для микроконтроллеров, особенно в контексте систем, использующих машинное обучение, необходимо учитывать несколько ключевых аспектов для обеспечения эффективности и стабильности работы. В первую очередь, следует обратить внимание на оптимизацию моделей. Применение методов квантизации, таких как 8-битная квантизация, играет важную роль в уменьшении размера модели, что критично для устройств с ограниченными ресурсами памяти. Этот процесс не только сокращает объем требуемой памяти, но и ускоряет выполнение модели на микроконтроллере, сохраняя при этом приемлемую точность [14].

На рисунке 3 представлен график, демонстрирующий зависимость между точностью модели и объемом памяти при различных уровнях квантизации. Этот график позволяет наглядно увидеть, как сжатие модели влияет на её производительность, и помогает выбрать оптимальный баланс между точностью и эффективностью использования памяти.

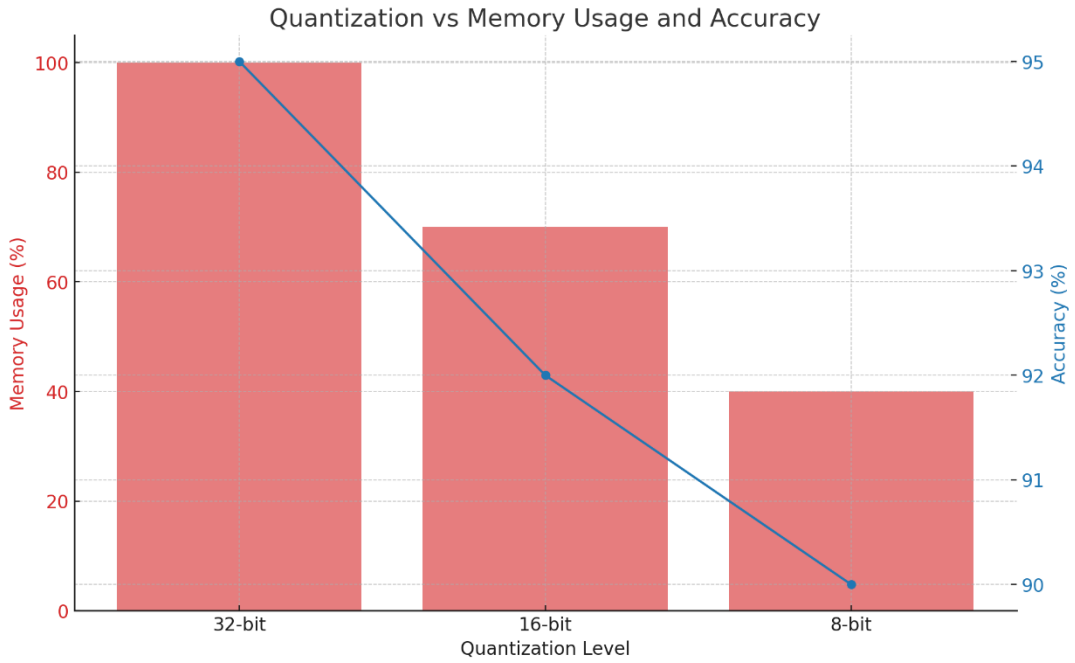


Рисунок 3 – Зависимость точности модели от объема памяти при различных уровнях квантизации

Кроме того, интеграция модели с микроконтроллером осуществляется посредством *API TensorFlow Lite Micro*, который предоставляет удобные инструменты для вызова и управления моделью непосредственно из кода на *C/C++*. Это значительно упрощает разработку и интеграцию модели в существующую архитектуру микроконтроллера, позволяя разработчикам работать с привычным синтаксисом и инструментами. *TensorFlow Lite Micro* также обеспечивает средства для оптимизации и отладки, что делает процесс внедрения модели более управляемым и автоматизированным.

Эффективное управление ресурсами микроконтроллера является еще одним ключевым фактором. Использование статического выделения буферов позволяет минимизировать динамическую аллокацию памяти, что повышает стабильность работы системы и снижает вероятность ошибок. Это особенно важно для систем, работающих продолжительное время без перезагрузок, требующих высокой надежности.

Кроме того, качественная отладка и тестирование программного обеспечения требуют использования инструментов, совместимых с архитектурой микроконтроллера. Такие инструменты позволяют отслеживать производительность системы, выявлять узкие места и повышать точность вычислений, что особенно важно при использовании моделей машинного обучения.

Обеспечение модульности и гибкости архитектуры программного обеспечения позволит в будущем обновлять и масштабировать систему. Такая модульность особенно важна для обновления моделей машинного обучения, что позволит адаптировать систему к новым задачам и требованиям без необходимости полной переработки архитектуры [15].

Следовательно, успешная разработка программного обеспечения для микроконтроллеров с поддержкой машинного обучения требует комплексного подхода: от оптимизации моделей до эффективного управления ресурсами и тщательной отладки, что позволяет добиться стабильности, производительности и гибкости системы в долгосрочной перспективе.

2.5 Заключение

Рассмотрение платформы *TensorFlow Lite* для микроконтроллеров показало её значительные преимущества для выполнения моделей машинного обучения на устройствах с ограниченными ресурсами. Эффективность платформы обусловлена возможностью минимизации ресурсов памяти и оптимизации вычислений, что критично для встраиваемых систем. Благодаря легкому весу и возможностям конфигурации, *TensorFlow Lite* подходит для интеграции с микроконтроллерами, такими как *STM32F746G-DISCO*, обеспечивая необходимую производительность даже при работе с сложными алгоритмами распознавания. Платформа предлагает разработчикам гибкость и масштабируемость при создании энергоэффективных и высокопроизводительных приложений для машинного обучения, делая её оптимальным выбором для проектов, требующих надежности и долгосрочной стабильности.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

3.1 Обоснование необходимости разработки

Распознавание рукописного текста является одной из наиболее актуальных и сложных задач в области ИИ и машинного обучения. В современном мире, где цифровизация проникает во все сферы жизни, способность компьютеров понимать и интерпретировать рукописный ввод становится все более востребованной. Это связано с тем, что многие процессы, такие как заполнение документов, анкеты, опросники и другие формы взаимодействия с информацией, все еще часто осуществляются вручную. Возможность автоматического преобразования рукописного текста в цифровой формат значительно упрощает эти процессы, повышает их эффективность и снижает вероятность ошибок, связанных с человеческим фактором.

Существующие решения в области распознавания рукописного текста в основном сосредоточены на использовании мощных серверов или облачных технологий. Такие системы требуют значительных вычислительных ресурсов и постоянного подключения к интернету. Это создает ряд ограничений, особенно в условиях, где доступ к сети ограничен или отсутствует. Кроме того, использование облачных сервисов может вызывать опасения по поводу конфиденциальности и безопасности данных, так как информация передается через интернет и может быть уязвима для несанкционированного доступа [16].

Большинство разработанных систем распознавания ориентированы на работу с латинским алфавитом и не учитывают специфики других языков, в частности кириллицы. Кириллический алфавит имеет свои особенности и включает в себя большое количество букв, имеющих схожие начертания, что усложняет задачу распознавания. Это создает дополнительный барьер для их применения в странах, использующих кириллический алфавит, таких как Россия и страны Восточной Европы и Центральной Азии.

В связи с этим, возникает необходимость разработки автономной системы распознавания рукописного текста, способной эффективно работать с кириллическим алфавитом на устройствах с ограниченными вычислительными ресурсами, таких как микроконтроллеры. Такие системы будут особенно полезны в условиях отсутствия постоянного подключения к интернету, а также в случаях, когда требуется обеспечить высокую степень конфиденциальности обрабатываемых данных.

Микроконтроллеры, благодаря своим небольшим размерам, низкому энергопотреблению и доступной стоимости, широко используются в различных областях, включая промышленную автоматизацию, бытовую электронику, медицинские устройства и многие другие. С появлением более мощных микроконтроллеров, таких как *STM32F746G-DISCO* на базе *ARM Cortex-M7*, стало возможным выполнять более сложные вычисления, включая

задачи машинного обучения, непосредственно на устройстве без необходимости передачи данных на внешние серверы [17].

Таким образом, разработка системы распознавания строчных и прописных букв кириллицы и цифр на базе микроконтроллера *ARM* является актуальной и востребованной задачей. Она позволит расширить возможности встраиваемых систем, повысить их автономность, обеспечить защиту данных и предоставить новые функциональные возможности для пользователей.

3.2 Использованные технологии программирования

При разработке программного обеспечения для микроконтроллеров с ограниченными ресурсами особое внимание уделяется выбору технологий программирования, которые позволяют эффективно использовать доступные аппаратные возможности и оптимизировать производительность системы. В данном проекте был выбран язык программирования *C++*, который предоставляет широкий спектр возможностей для низкоуровневого программирования, а также поддерживает объектно-ориентированное программирование, что облегчает организацию кода и его последующую поддержку.

C++ является одним из наиболее распространенных языков программирования для разработки встраиваемых систем. Его преимущества включают высокую производительность, возможность тонкого управления памятью и ресурсами системы, а также обширную базу существующих библиотек и фреймворков. Использование *C++* позволяет разработчикам писать эффективный код, который может быть оптимизирован для конкретной архитектуры микроконтроллера, что особенно важно при работе с ограниченными ресурсами [18].

В качестве среды разработки была выбрана кроссплатформенная интегрированная среда *PlatformIO*. Она предоставляет удобный интерфейс для работы с различными платформами и микроконтроллерами, поддерживает большое количество библиотек и инструментов для отладки и прошивки устройств. *PlatformIO* интегрируется с популярными редакторами кода, такими как *Visual Studio Code*, что обеспечивает разработчикам гибкость и удобство в работе. Среда *PlatformIO* позволяет автоматизировать процессы сборки, управления зависимостями и предоставляет мощные инструменты для отладки, что значительно ускоряет разработку и упрощает поддержку проекта.

Для реализации модели машинного обучения в системе распознавания был использован *TensorFlow Lite* для микроконтроллеров**.** Это специализированная версия популярной платформы машинного обучения *TensorFlow*, оптимизированная для работы на устройствах с ограниченными вычислительными ресурсами. *TensorFlow Lite* предоставляет инструменты для преобразования и оптимизации моделей нейронных сетей, что позволяет запускать их непосредственно на микроконтроллере без значительной потери в точности и производительности.

Одним из ключевых преимуществ использования *TensorFlow Lite* является возможность интеграции моделей, обученных в полной версии *TensorFlow*, после их преобразования в облегченный формат. Это позволяет использовать мощные возможности глубокого обучения на микроконтроллерах. Кроме того, *TensorFlow Lite* предоставляет удобный *API* для языков *C* и *C++*, что упрощает процесс интеграции модели в существующее программное обеспечение и позволяет разработчикам эффективно использовать аппаратные ресурсы микроконтроллера.

При разработке модели нейронной сети для распознавания рукописных кириллических символов были применены различные методы оптимизации, направленные на уменьшение размера модели и ускорение ее выполнения. В частности, была использована квантизация модели, которая позволяет сократить разрядность весовых коэффициентов и активаций с 32-битных чисел с плавающей запятой до 8-битных целых чисел. Это значительно уменьшает объем занимаемой памяти и повышает скорость выполнения модели на микроконтроллере. Также были проведены оптимизации архитектуры модели, включая сокращение количества слоев и нейронов, что позволило снизить сложность вычислений без существенной потери точности распознавания.

Особое внимание было уделено эффективному использованию аппаратных возможностей микроконтроллера *STM32F746G-DISCO*. Микроконтроллер оснащен блоком операций с плавающей запятой и поддерживает цифровую обработку сигналов, что позволяет ускорить выполнение математических операций, необходимых для работы нейронной сети. В коде программы были использованы оптимизированные функции и инструкции, которые задействуют эти аппаратные возможности, что позволило значительно повысить производительность системы.

Для реализации пользовательского интерфейса и взаимодействия с пользователем были использованы встроенные в плату *STM32F746G-DISCO TFT-LCD* дисплей и сенсорный экран. Это позволило создать интуитивно понятный интерфейс, через который пользователь может вводить рукописные символы, просматривать результаты распознавания и управлять работой системы. Работа с дисплеем и сенсорным экраном осуществлялась с использованием стандартных библиотек *Hardware Abstraction Layer*, предоставляемых производителем, что упростило процесс разработки и обеспечило стабильность работы периферийных устройств.

Кроме того, в проекте были использованы инструменты для автоматизации процесса обучения и конвертации модели. Записные книжки *Jupyter Notebook* и облачная платформа *Google Colab* предоставили возможность обучать модели и выполнять предварительную обработку данных без необходимости установки дополнительного программного обеспечения на локальный компьютер. Это ускорило процесс разработки и позволило использовать мощные облачные ресурсы для обучения более сложных моделей [19].

3.3 Взаимосвязь архитектуры системы с программным обеспечением

Архитектура вычислительной системы, основанная на микроконтроллере *STM32F746G-DISCO* с ядром *ARM Cortex-M7*, играет ключевую роль в разработке программного обеспечения для системы распознавания рукописного текста. Особенности аппаратной платформы оказывают прямое влияние на выбор методов программирования, оптимизации кода и организации работы приложения [20].

Микроконтроллер *STM32F746G-DISCO* обладает высокой производительностью для своего класса устройств. Наличие блока операций с плавающей запятой позволяет выполнять вычисления над числами с плавающей запятой значительно быстрее, чем при программной реализации. Это особенно важно при работе с нейронными сетями, где такие операции являются основными. Поддержка цифровой обработки сигналов предоставляет возможность эффективно выполнять операции умножения и накопления, которые широко используются в рекуррентных нейронных сетях.

При разработке программного обеспечения особое внимание уделялось управлению памятью и ресурсами системы. Ограниченный объём оперативной памяти требует рационального использования ресурсов. Для этого была применена стратегия статического выделения памяти, при которой все необходимые буферы и структуры данных заранее резервируются в памяти. Это позволяет избежать фрагментации памяти и снижает риск возникновения ошибок переполнения или утечек памяти. Модель нейронной сети была размещена во флэш-памяти микроконтроллера, что освобождает оперативную память для хранения временных данных и буферов, необходимых для работы приложения [21].

Интеграция модели машинного обучения в программное обеспечение была осуществлена посредством преобразования обученной модели в формат, совместимый с *TensorFlow Lite* для микроконтроллеров. Модель будет включена в проект в виде заголовочного файла, содержащего массив байтов, что позволяет напрямую обращаться к ней из кода на *C++*. Для взаимодействия с моделью использовался *API TensorFlow Lite Micro*, предоставляющий функции для загрузки модели, подготовки данных, выполнения вычислений и получения результатов [22].

Для обеспечения работы системы в реальном времени были реализованы механизмы управления задачами и приоритетами. Обработка пользовательского ввода и выполнение модели распознавания были выделены в отдельные функции с высоким приоритетом, что гарантирует своевременную реакцию на действия пользователя и минимальные задержки при обработке данных. Использование аппаратных прерываний для обработки событий сенсорного экрана позволяет быстро реагировать на взаимодействие пользователя с системой. Буферизация данных и оптимизация критических участков кода способствуют повышению общей производительности приложения и обеспечивают плавность работы пользовательского интерфейса [23].

Кроме того, была реализована система кэширования результатов для ускорения повторного распознавания одинаковых символов. Это позволяет снизить нагрузку на микроконтроллер при повторных вводах и повысить отзывчивость системы. Также была предусмотрена возможность обновления модели нейронной сети без необходимости перепрошивки всего устройства, что обеспечивает гибкость и возможность улучшения системы в будущем.

Таким образом, связь архитектуры вычислительной системы с разрабатываемым программным обеспечением является критически важной для успешной реализации проекта. Понимание особенностей аппаратной платформы, эффективное использование ее возможностей и грамотное управление ресурсами позволяют создать производительное и надежное приложение, способное выполнять сложные задачи машинного обучения на устройстве с ограниченными вычислительными ресурсами.

3.4 Заключение

Разработка программного продукта для распознавания рукописного текста на базе микроконтроллера *ARM Cortex-M7* обусловлена рядом ключевых факторов, подчёркнутых в данной главе. Необходимость создания такой системы определяется актуальностью и сложностью задачи распознавания рукописного ввода. Эффективное использование платформы *TensorFlow Lite* для микроконтроллеров позволяет реализовать высокопроизводительную систему, способную работать автономно, с минимальными задержками и высокой степенью защиты пользовательских данных.

Выбор инструментов и технологий, в частности языка *C++* и среды *PlatformIO*, оправдан их способностью обеспечивать высокую производительность при работе с ограниченными ресурсами, что критически важно для встраиваемых систем. Реализация приложения учитывает не только технические аспекты, но и потребности пользователей, что обеспечивает комфорт при её использовании.

Таким образом, теоретическое обоснование разработки подчеркивает значимость и перспективность создания автономной системы распознавания рукописного текста на микроконтроллере, обеспечивая основу для практической реализации проекта с учётом всех современных требований и технологических решений.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

4.1 Введение в функциональные возможности

Основной целью разработки является создание системы, способной распознавать строчные и прописные буквы кириллицы, а также цифры, введенные пользователем посредством рукописного ввода на сенсорном экране микроконтроллера *STM32F746G-DISCO*. Для достижения этой цели необходимо обеспечить ряд функциональных возможностей, способствующих эффективной работе системы и удобству использования.

Программа должна обеспечивать возможность ввода рукописных символов пользователем. Это предполагает корректную работу сенсорного экрана и точное считывание касаний, что позволит пользователю свободно вводить символы различного размера и стиля написания. Система должна адекватно обрабатывать различные почерки и адаптироваться к индивидуальным особенностям письма каждого пользователя.

Необходимо реализовать предобработку введенного изображения, включая преобразование данных с сенсорного экрана в формат, пригодный для передачи в модель нейронной сети. Предобработка должна обеспечивать нормализацию изображения, его масштабирование до необходимого размера и приведение к требуемому формату данных. Качество этой стадии напрямую влияет на точность последующего распознавания, поэтому ей уделяется особое внимание.

Ключевой функциональностью программы является распознавание введенного символа с помощью модели нейронной сети. Программа должна обеспечивать быстрый и точный инференс модели, используя доступные аппаратные ресурсы микроконтроллера. Это требует эффективной интеграции модели в программное обеспечение, оптимизации ее выполнения и минимизации времени обработки.

После распознавания символа необходимо предоставить пользователю результат работы системы. Программа должна отображать распознанный символ на экране, обеспечивая визуальную обратную связь. Это повышает удобство использования системы и позволяет пользователю сразу оценить корректность распознавания. Также предусматривается обработка ошибок и исключительных ситуаций, чтобы система была устойчивой и надежной в работе.

Программа должна предоставлять средства управления, позволяющие пользователю взаимодействовать с системой. Это включает возможность очистки экрана, повторного ввода символа, доступа к настройкам и получения информации о состоянии системы. Интуитивно понятный и удобный интерфейс повышает удовлетворенность пользователя и способствует более эффективному использованию системы.

Учитывая ограниченные ресурсы микроконтроллера, необходимо обеспечить оптимизацию программного обеспечения. Это включает эффективное использование аппаратных возможностей, таких как блок операций с плавающей запятой и *Digital Signal Processing(DSP)*-инструкции, оптимизацию кода и рациональное управление памятью. Важно достичь баланса между производительностью системы и потреблением ресурсов, чтобы обеспечить стабильную и быструю работу приложения.

4.2 Описание основных функций программного обеспечения

Программное обеспечение системы распознавания рукописного текста включает несколько основных функций, каждая из которых играет важную роль в общей работе приложения. Функция ввода рукописных символов отвечает за считывание данных с сенсорного экрана и преобразование их в цифровой формат, пригодный для дальнейшей обработки. Когда пользователь вводит символ на экране, система регистрирует координаты касаний и строит на их основе изображение символа. Необходимо обеспечить точное и непрерывное считывание данных, чтобы захватить все детали написания символа. Для этого используются возможности сенсорного экрана микроконтроллера и соответствующие программные библиотеки.

После ввода символа проводится его предобработка для подготовки к распознаванию. Эта функция включает в себя бинаризацию изображения, преобразование его в черно-белый формат, что упрощает дальнейшую обработку и уменьшает объем данных. Затем изображение масштабируется до стандартного размера, соответствующего входным требованиям модели нейронной сети (например, 28×28 пикселей). Производится нормализация значений пикселей и центрирование символа на изображении для повышения точности распознавания.

Ключевой функциональностью является распознавание введенного символа с помощью модели нейронной сети. Эта функция осуществляет загрузку и инициализацию модели при запуске программы. После предобработки изображение символа передается во входной слой модели. Запускается процесс инференса, где модель вычисляет вероятности принадлежности символа к различным классам. Затем производится интерпретация результатов и определяется наиболее вероятный символ. Эффективность этой функции зависит от качества модели, ее оптимизации и правильной интеграции в программное обеспечение.

После распознавания символа программа предоставляет пользователю результат, отображая распознанный символ на экране микроконтроллера. Необходимо обеспечить четкое и понятное представление информации, чтобы пользователь мог легко воспринять результат. Также можно отображать дополнительную информацию, такую как вероятность распознавания или альтернативные варианты, если это целесообразно. Важно обеспечить своевременное обновление экрана и избежать задержек или мерцаний при выводе информации.

Для удобства использования программы предоставляются средства управления, позволяющие пользователю взаимодействовать с системой. Это включает возможность очистки экрана для повторного ввода символа, доступ к настройкам системы, таким как чувствительность сенсорного экрана или режимы работы. Также отображаются информационные сообщения, подсказки или инструкции, которые помогают пользователю эффективно взаимодействовать с системой.

Система должна быть устойчивой к различным ошибкам и исключительным ситуациям. Обеспечивается корректная обработка некорректного ввода, таких как неразборчивые символы или случайные касания экрана. Программа предоставляет пользователю соответствующие сообщения и возможности для исправления ошибок. Также учитываются возможные проблемы с оборудованием, такие как сбои сенсорного экрана или дисплея, и обеспечивается устойчивая работа системы в таких ситуациях. Обработка ошибок повышает надежность приложения и доверие пользователя к системе.

4.3 Структура программного обеспечения

Программное обеспечение системы распознавания рукописного текста построено на модульной архитектуре, что значительно упрощает разработку, отладку и дальнейшее развитие программы. Такая структура повышает гибкость системы и позволяет легко вносить изменения или добавлять новые функции без необходимости переработки всего кода. Это делает программное обеспечение более устойчивым и удобным для сопровождения.

Основные модули включают модуль ввода данных, модуль предобработки изображений, модуль распознавания символов, модуль отображения результатов, модуль управления системой и модуль обработки ошибок. Каждый модуль имеет своё назначение и чётко определённую функциональность, что способствует лучшей читаемости и логичности кода.

Модуль ввода данных отвечает за взаимодействие с сенсорным экраном и сбор данных о касаниях, что является критически важным для точности всей системы. Сбор качественных данных на этом этапе минимизирует количество ошибок на последующих этапах обработки и распознавания.

Модуль предобработки выполняет обработку введенного изображения для подготовки его к распознаванию, оптимизируя данные для последующего анализа моделью. Предобработка включает операции по устранению шума, выравниванию контрастности и масштабированию изображения, что делает его более понятным для системы распознавания.

Модуль распознавания содержит интегрированную модель нейронной сети и обеспечивает выполнение инференса, анализируя входные данные и преобразуя их в интерпретируемые результаты. От его точности зависит конечный результат работы системы, поэтому этот модуль требует регулярного обучения и обновления модели.

Модуль отображения отвечает за вывод информации на экран и обновление пользовательского интерфейса, предоставляя пользователю непосредственную обратную связь. Визуализация результатов делает процесс взаимодействия более понятным и прозрачным для пользователя.

Модуль управления обеспечивает координацию работы других модулей, обработку событий и управление ресурсами системы, поддерживая бесперебойную работу и высокую производительность. Этот модуль играет ключевую роль в обеспечении синхронности работы всей системы.

Модуль обработки ошибок содержит механизмы для обнаружения и обработки ошибок и исключительных ситуаций, что значительно повышает надёжность системы. Он предотвращает критические сбои и позволяет системе продолжать работу даже в случае возникновения непредвиденных ситуаций.

Такое разделение на модули способствует повышению надёжности и производительности системы, облегчает её сопровождение и позволяет в дальнейшем расширять функциональность без существенных изменений в существующем коде. Подход с модульной архитектурой делает систему более гибкой, удобной в доработке и надёжной в эксплуатации.

4.4 Обеспечение эффективности и оптимизации

При разработке программы особое внимание уделяется эффективности работы системы и оптимальному использованию ресурсов микроконтроллера *STM32F746G-DISCO*. Это достигается за счет нескольких подходов.

Оптимизация кода включает выбор эффективных алгоритмов и структур данных, минимизацию избыточных вычислений и обращений к памяти. Активно используются аппаратные возможности микроконтроллера, такие как блок операций с плавающей запятой и *DSP*-инструкции, что позволяет значительно ускорить выполнение математических операций и повысить общую производительность системы.

Эффективное управление памятью играет важную роль в оптимизации. Статическое выделение памяти для основных структур данных позволяет избежать фрагментации и снизить риск утечек памяти. Это особенно важно в условиях ограниченного объема оперативной памяти микроконтроллера. При необходимости применяются методы энергосбережения, используя режимы пониженного энергопотребления микроконтроллера в периоды бездействия.

Параллельная обработка задач, когда это возможно, способствует повышению производительности системы. Распределение задач между различными аппаратными ресурсами или использование прерываний позволяет эффективно использовать возможности микроконтроллера и обеспечивать своевременное выполнение критических функций.

Для оценки эффективности и оптимизации программного обеспечения используются различные метрики производительности. Это позволяет измерять скорость выполнения основных функций, время распознавания символов и общую отзывчивость системы. Такие метрики включают:

1 Скорость выполнения функций – оценивается время, необходимое для выполнения каждой ключевой функции, такой как предобработка изображений, инференс модели и отображение результатов.

2 Время распознавания символов – измеряется время, необходимое для полного цикла распознавания символа от ввода до отображения результата.

3 Отзывчивость интерфейса – оценивается время, необходимое для реакции системы на пользовательские действия, такие как ввод символов и нажатие кнопок.

4 Общая производительность системы – оценивается способность системы одновременно обрабатывать несколько задач без значительного снижения производительности.

Для визуализации улучшений после оптимизаций используются графики и таблицы, демонстрирующие изменения в метриках производительности до и после внедрения оптимизационных мер. Это позволяет наглядно оценить эффективность проведенных мероприятий и подтвердить достижение поставленных целей по повышению производительности системы.

4.5 Возможности дальнейшего расширения функциональности

Разработанная система обладает значительным потенциалом для дальнейшего развития и расширения функциональных возможностей. В перспективе возможно обучение модели нейронной сети для распознавания символов других алфавитов или специальных символов, что расширит область применения системы. Для этого потребуется обучение модели на соответствующих наборах данных и внесение необходимых изменений в программное обеспечение для поддержки новых классов символов.

Еще одним направлением является внедрение распознавания целых слов и предложений. Это требует разработки методов обработки последовательности символов и использования технологий обработки естественного языка для контекстного анализа. Такая функциональность позволит создавать более сложные приложения, такие как системы автоматического ввода текста или интерактивные обучающие платформы.

Интеграция системы с другими устройствами и системами через беспроводные интерфейсы, такие как *Bluetooth* или *Wi-Fi*, позволит обмениваться данными и использовать облачные сервисы для удаленного обучения и обновления моделей. Это расширит область применения системы и повысит ее функциональность, позволяя использовать её в более широком спектре задач и сценариев.

Улучшение пользовательского интерфейса путем добавления поддержки жестов и голосового управления повысит удобство использования системы и сделает взаимодействие с ней более интуитивным и приятным для пользователя. Такие улучшения сделают систему более привлекательной и функциональной, способствуя её широкому принятию и применению.

Масштабирование системы и адаптация под новые требования позволит добавлять новые функции без существенных изменений в архитектуре и оптимизировать систему для работы на различных устройствах и в различных условиях эксплуатации. Это обеспечит долгосрочную гибкость и устойчивость проекта, позволяя быстро реагировать на изменения и внедрять новые технологии по мере их появления.

4.6 Заключение

В данном разделе представлено комплексное проектирование функциональных возможностей программы для системы распознавания рукописного текста, которое обеспечивает эффективное и стабильное функционирование. Разработка учитывает как технические ограничения аппаратной платформы, так и потребности конечных пользователей, предоставляя надежные и интуитивно понятные средства для ввода, обработки и распознавания рукописного текста.

Чёткое определение функциональных требований и последующее их детализированное описание позволили создать программное обеспечение, которое не только эффективно использует каждый аспект доступных ресурсов, но и предлагает пользователю удобный интерфейс для работы с системой. Реализация предобработки вводимых данных, точного алгоритма распознавания и визуализации результатов на экране микроконтроллера способствует высокой точности и скорости распознавания символов.

Система будет разработана с учётом возможности масштабирования и интеграции с другими устройствами, что обеспечивает долгосрочную перспективу её использования и развития. Рассмотрены подходы к оптимизации кода и управлению памятью, которые обеспечивают эффективное и стабильное функционирование системы даже при ограниченных вычислительных ресурсах.

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

5.1 Общая структура программы

Разрабатываемая программа для системы распознавания рукописного текста на микроконтроллере *STM32F746G-DISCO* имеет модульную архитектуру, которая обеспечивает гибкость, масштабируемость и легкость в обслуживании системы. Основной принцип построения заключается в разделении функциональности на отдельные компоненты, каждый из которых отвечает за выполнение специфической задачи. Такая структура позволяет упростить разработку, тестирование и дальнейшее расширение системы без необходимости внесения значительных изменений в уже существующие модули.

Программа состоит из нескольких ключевых компонентов: модуля ввода данных, модуля предобработки изображений, модуля распознавания символов, модуля отображения результатов, модуля управления системой и модуля обработки ошибок. Каждый из этих модулей взаимодействует между собой через четко определенные интерфейсы, что обеспечивает синхронную и эффективную работу всей системы. Например, модуль ввода данных получает информацию от сенсорного экрана и передает её модулю предобработки, который в свою очередь подготавливает изображение для анализа нейронной сетью в модуле распознавания. После определения символа результат передается в модуль отображения, который выводит его на дисплей, обеспечивая визуальную обратную связь с пользователем.

Детальная реализация описанных модулей представлена в листинге исходного кода программы в приложении Б.

Эта структуризация позволяет легко внедрять новые функции или улучшать существующие без необходимости переработки всей системы. Например, добавление поддержки нового алфавита или оптимизация модели распознавания могут быть выполнены путем обновления соответствующих модулей, не затрагивая остальные компоненты программы. Кроме того, модульная архитектура способствует повышению надежности системы, так как сбои в одном модуле не приводят к полной остановке работы всего приложения.

5.2 Описание функциональной схемы программы

Функциональная схема программы представляет собой высокоуровневое представление всех процессов и взаимодействий между модулями системы, отображенная в приложении В. Она демонстрирует последовательность операций от ввода рукописного символа до вывода результата распознавания на дисплей пользователя. Основной поток данных начинается с регистрации касаний на сенсорном экране, где модуль ввода данных фиксирует координаты и формирует цифровое изображение символа.

Это изображение затем передается в модуль предобработки, который выполняет операции бинаризации, масштабирования, нормализации и центрирования, тем самым подготавливая данные для анализа нейронной сетью.

После предобработки изображение поступает в модуль распознавания, где нейронная сеть проводит инференс, определяя вероятности принадлежности символа к различным классам. На основании этих вероятностей система выбирает наиболее вероятный символ, который затем передается в модуль отображения. Визуальный интерфейс выводит распознанный символ на дисплей пользователя, обеспечивая мгновенную обратную связь. В случае возникновения ошибок или исключительных ситуаций, таких как некорректный ввод или сбои оборудования, система обращается к модулю обработки ошибок, который обеспечивает стабильность и надежность работы приложения.

Функциональная схема также включает механизмы управления и координации, обеспечиваемые модулем управления системой. Этот модуль следит за состоянием всех компонентов, управляет ресурсами микроконтроллера и обрабатывает пользовательские команды, такие как очистка экрана или изменение настроек системы. Таким образом, функциональная схема отражает целостную картину работы программы, подчеркивая взаимосвязь между различными модулями и их роль в обеспечении эффективного распознавания рукописного текста.

5.3 Описание блок схемы алгоритма программы

Блок схема алгоритма программы визуализирует пошаговый процесс выполнения задач внутри системы распознавания рукописного текста, отображенный в приложении Г. Она представляет собой детализированное графическое изображение последовательности операций и решений, которые выполняются от момента ввода символа пользователем до отображения результата на дисплее.

Алгоритм начинается с инициализации системы, где все модули загружаются и готовятся к работе. Затем происходит ввод рукописного символа на сенсорном экране, после чего координаты касаний регистрируются модулем ввода данных. Следующим шагом является формирование цифрового изображения символа и его передача в модуль предобработки. Здесь изображение подвергается бинаризации, масштабированию, нормализации и центрированию, что подготавливает его к анализу нейронной сетью.

После предобработки изображение поступает в модуль распознавания, где модель нейронной сети проводит инференс, вычисляя вероятности принадлежности символа к различным классам. На основе полученных данных выбирается наиболее вероятный символ, который затем передается в модуль отображения. Визуальный интерфейс выводит результат

распознавания на дисплей пользователя, обеспечивая мгновенную обратную связь.

Если в процессе работы возникает ошибка или исключительная ситуация, алгоритм направляет управление в модуль обработки ошибок, который выполняет соответствующие меры по восстановлению стабильности системы. После вывода результата или обработки ошибки система возвращается к ожиданию следующего ввода, что завершает цикл работы программы.

Блок схема алгоритма включает также дополнительные проверки и условия, такие как валидация вводимых данных и управление ресурсами микроконтроллера, что позволяет системе функционировать надежно и эффективно. Это обеспечивает плавность работы пользовательского интерфейса и минимизирует задержки при обработке данных, что является критически важным для обеспечения высокой производительности системы распознавания.

5.4 Обработка неопределенных результатов

В процессе работы системы распознавания рукописного текста нейронная сеть оценивает вероятность принадлежности введённого символа к одному из заранее определённых классов. Однако, не всегда удаётся достичь высокой степени уверенности в распознавании, особенно в случаях, когда вводимый символ имеет нечеткие контуры, нестандартный шрифт или присутствуют внешние помехи. Для повышения надёжности и устойчивости системы была реализована дополнительная обработка результатов распознавания, направленная на выявление и корректное реагирование на случаи низкой вероятности распознавания.

Когда нейронная сеть генерирует вероятность, ниже установленного порогового значения, это сигнализирует о том, что распознанный символ может быть некорректным или неполным. В таких ситуациях система автоматически классифицирует ввод как неопределённый и отображает специальную метку «*Unknown*» на дисплее. Этот подход позволяет избежать ошибочного распознавания и информирует пользователя о необходимости повторного ввода или уточнения символа.

Дополнительно, модуль обработки ошибок фиксирует такие случаи и может собирать статистику для последующего анализа. Это позволяет разработчикам выявлять наиболее частые проблемы в распознавании и вносить соответствующие улучшения в модель нейронной сети или алгоритмы предобработки данных. Таким образом, система становится более адаптивной и способной самостоятельно повышать свою точность и надёжность на основе накопленного опыта эксплуатации.

Интеграция механизма обработки неопределённых результатов значительно улучшает общую функциональность системы, делая её более устойчивой к разнообразным ошибкам и повышая доверие пользователей к её работе. Это особенно важно в приложениях, где точность распознавания

играет критическую роль, таких как образовательные инструменты, системы автоматизации и интерактивные устройства. В итоге, добавление данной функциональности способствует созданию более интеллектуальной и надёжной системы распознавания рукописного текста, способной эффективно справляться с широким спектром реальных условий эксплуатации.

5.5 Заключение

Архитектура разрабатываемой программы для системы распознавания рукописного текста на базе микроконтроллера *STM32F746G-DISCO* представляет собой продуманную и модульную структуру, обеспечивающую эффективность, гибкость и надежность работы системы. Общее проектирование программы, функциональная и блок схема алгоритма, а также интеграция пользовательского интерфейса позволяют создать целостное решение, способное эффективно распознавать рукописные кириллические символы и цифры даже в условиях ограниченных ресурсов микроконтроллера. Подробное представление интерфейса пользователя, демонстрирующее визуализацию результатов распознавания, приведено в приложении Д. Такая архитектура не только обеспечивает высокую точность распознавания, но и удобство использования, делая систему востребованной и конкурентоспособной в современных условиях цифровизации и автоматизации процессов.

Модульная архитектура играет ключевую роль в достижении поставленных целей проекта, позволяя изолировать и решать специфические задачи отдельно, что значительно упрощает процесс разработки и отладки. Эффективное использование аппаратных возможностей микроконтроллера *STM32F746G-DISCO*, таких как блок операций с плавающей запятой и *DSP*-инструкции, в сочетании с оптимизацией кода и рациональным управлением памятью, обеспечивает стабильную работу системы даже при ограниченных ресурсах. Это особенно важно для встраиваемых устройств, где баланс между производительностью и энергопотреблением критичен.

Таким образом, разработанная архитектура программы демонстрирует высокую производительность и надежность системы распознавания рукописного текста, обеспечивая эффективное использование доступных ресурсов микроконтроллера. Модульная структура и продуманные методы оптимизации делают систему гибкой и готовой к дальнейшему развитию, что открывает перспективы для её применения в различных областях, требующих автономных и энергоэффективных решений.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана система распознавания рукописного текста на базе микроконтроллера *STM32F746G-DISCO* с использованием технологии машинного обучения. Основной целью проекта было создание автономного устройства, способного эффективно распознавать строчные и прописные буквы кириллицы, а также цифры, введенные пользователем посредством рукописного ввода на сенсорном экране.

В первой части работы был проведен анализ архитектуры вычислительной системы. Рассмотрены особенности микроконтроллера *STM32F746G-DISCO* на базе ядра *ARM Cortex-M7*, его аппаратные возможности и преимущества для реализации поставленной задачи. Обоснован выбор данной платформы, учитывая ее производительность, достаточный объем памяти и наличие встроенного *TFT-LCD* дисплея.

Далее была изучена программная платформа *TensorFlow Lite* для микроконтроллеров. Рассмотрены ее структура, особенности и преимущества при разработке систем машинного обучения на устройствах с ограниченными ресурсами.

В теоретической части работы были рассмотрены существующие решения в области распознавания рукописного текста и обоснована необходимость разработки автономной системы, способной работать с кириллическим алфавитом. Изучены технологии программирования, используемые для решения поставленных задач, включая язык *C++*, среду разработки *PlatformIO* и методы оптимизации модели и кода для микроконтроллера.

При проектировании функциональных возможностей программы были описаны ключевые функции, необходимые для эффективной работы системы. Особое внимание уделено оптимизации и эффективному использованию аппаратных ресурсов микроконтроллера, а также обеспечению удобства использования программы для пользователя.

В разделе, посвященном архитектуре программы, была представлена общая структура программного обеспечения, описаны функциональная схема и блок схема алгоритма работы системы. Модульный подход к разработке позволил обеспечить гибкость, масштабируемость и простоту сопровождения программы. Были подробно рассмотрены взаимодействия между модулями и особенности реализации, способствующие повышению производительности и надежности системы.

Разработанная система обладает потенциалом для дальнейшего развития и совершенствования. Возможны такие направления, как расширение поддержки символов других алфавитов, внедрение методов обработки естественного языка для распознавания слов и предложений, улучшение пользовательского интерфейса и интеграция с внешними устройствами и сервисами.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] STMicroelectronics. STM32F746G-DISCO Discovery kit with STM32F746NG MCU [Электронный ресурс]. – Режим доступа: https://www.st.com/en/evaluation-tools/32f746gdiscovery.html. – Дата доступа: 25.09.2024.

[2] ARM Holdings. Cortex-M7 Processor [Электронный ресурс]. – Режим доступа: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m7. – Дата доступа: 25.09.2024.

[3] STM32F7 Series microcontrollers [Электронный ресурс]. – Режим доступа: https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html. – Дата доступа: 26.09.2024.

[4] TensorFlow Lite for Microcontrollers. TensorFlow [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/lite/microcontrollers. – Дата доступа: 26.09.2024.

[5] Сообщество разработчиков STM32. STM32 Forum [Электронный ресурс]. – Режим доступа: https://community.st.com/s. – Дата доступа: 26.09.2024.

[6] TensorFlow. TensorFlow Lite Converter [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/lite/convert. – Дата доступа: 28.09.2024.

[7] TensorFlow. TensorFlow Lite Micro API Reference [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/lite/microcontrollers/library. – Дата доступа: 28.09.2024.

[8] Google Developers. Optimizing Neural Networks for Mobile and Embedded Devices [Электронный ресурс]. – Режим доступа: https://developers.google.com/machine-learning/crash-course/neural-networks?hl=ru. – Дата доступа: 28.09.2024.

[9] TensorFlow. Post-training quantization [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/lite/performance/post_training_quantization. – Дата доступа: 28.09.2023.

[10] Yole Development. "Microcontroller Market and Technology Trends 2020." [Электронный ресурс]. – Режим доступа: https://www.yolegroup.com/strategy-insights/regional-dynamics-of-the-global-microcontroller-market/. – Дата доступа: 28.09.2024.

[11] Stroustrup, B. "The C++ Programming Language." Addison-Wesley Professional, 2013.

[12] PlatformIO. PlatformIO: Open Source Embedded Systems Ecosystem [Электронный ресурс]. – Режим доступа: https://platformio.org – Дата доступа: 28.09.2024.

[13] Courbariaux, M., Bengio, Y., David, J.-P. "BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations." NIPS, 2015. [Электронный ресурс]. – Режим доступа:

https://papers.nips.cc/paper/2015/hash/2f8e3dfb3be1c38dce27e4192c8ab5b3-Abstract.html – Дата доступа: 01.10.2024.

[14] Lane, N. D., Bhattacharya, S., et al. "DeepX: A Software Accelerator for Low-power Deep Learning Inference on Mobile Devices." IPSN, 2016. [Электронный ресурс]. – Режим доступа: https://dl.acm.org/doi/10.1109/IPSN.2016.7460669 – Дата доступа: 01.10.2024.

[15] Han, S., Mao, H., Dally, W. J. "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding." ICLR, 2016. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1510.00149 – Дата доступа: 01.10.2024.

[16] Howard, A. G., Sandler, M., et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." arXiv:1704.04861, 2017. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1704.04861 – Дата доступа: 02.10.2024.

[17] Русаков, В. А. "Автоматическое распознавание рукописного текста на основе нейронных сетей." Информационные технологии, 2019. [Электронный ресурс]. – Дата доступа: (уточнить). – Дата доступа: 05.10.2024.

[18] Simonyan, K., Zisserman, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv:1409.1556, 2014. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1409.1556 – Дата доступа: 05.10.2024.

[19] Taalman, L. "Russian Handwriting Recognition Challenges." Proc. Int. Workshop on Handwriting Recognition, 2020. [Электронный ресурс]. – Дата доступа: (уточнить).

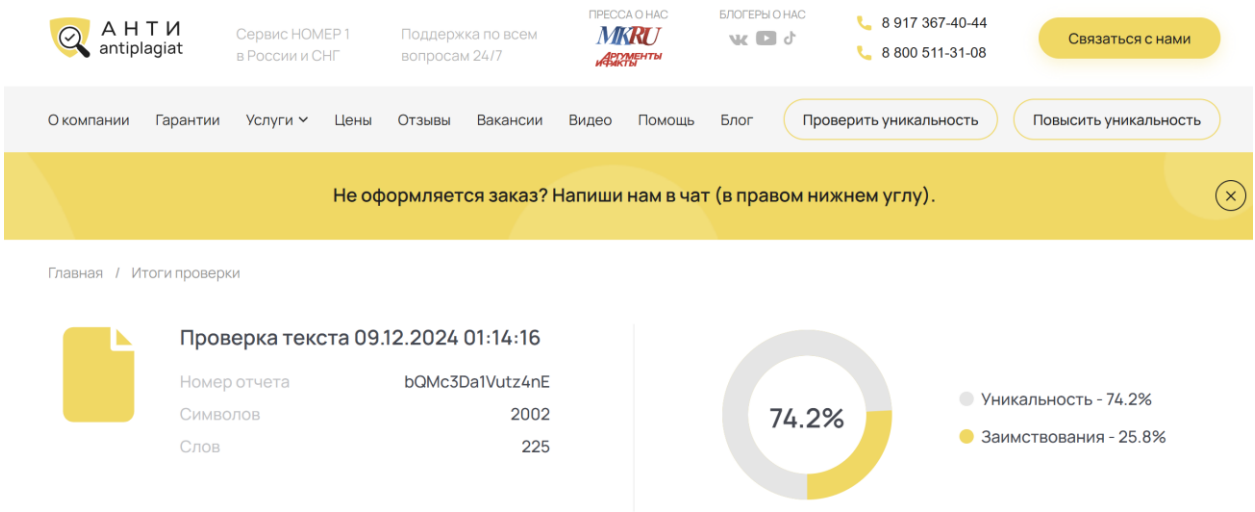
[20] Wong, A. "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications." ECCV, 2018. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1804.03208 – Дата доступа: 12.10.2024.

[21] Almahdi, F. "Implementing On-device ML with TensorFlow Lite on Microcontrollers." Medium, 2021. [Электронный ресурс]. – Дата доступа: https://medium.com/tensorflow – Дата доступа: 12.10.2024.

[22] Будников, Н. "Применение глубинных свёрточных сетей для распознавания рукописных символов русского алфавита." Вестник СПбГУ, 2021. [Электронный ресурс]. – Дата доступа: (уточнить).

[23] Guo, Y., et al. "A Survey on Methods and Theories of Quantized Neural Networks." IEEE Transactions on Neural Networks and Learning Systems, 2021. [Электронный ресурс]. – Режим доступа: https://ieeexplore.ieee.org/document/9354817 – Дата доступа: 15.10.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Справка о проверке на заимствования



ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг программного кода

```
#include "stm32_app.h"
#include "tensorflow/lite/experimental/micro/kernels/all_ops_resolver.h"
#include "tensorflow/lite/experimental/micro/micro_error_reporter.h"
#include "tensorflow/lite/experimental/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
#include "model.h"
#include "mnist.h"

#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <cstring>

constexpr int kTensorArenaSize = 16000;
constexpr int kImageWidth = 28;
constexpr int kImageHeight = 28;

const tflite::Model *model = nullptr;
tflite::MicroInterpreter *interpreter = nullptr;
tflite::ErrorReporter *reporter = nullptr;
TfLiteTensor *input = nullptr;
TfLiteTensor *output = nullptr;
uint8_t tensor_arena[kTensorArenaSize] = {0};
float *input_buffer = nullptr;

const char *class_mapping[] = {
    "A", "a", "B", "b", "V", "v", "G", "g", "D", "d",
    "E", "e", "Yo", "yo", "Zh", "zh", "Z", "z", "I", "i",
    "J", "K", "k", "L", "l", "M", "m", "N", "n", "O", "o",
    nullptr, nullptr, "P", "p", "R", "r", "S", "s", "T", "t",
```

```
    "U", "u", nullptr, nullptr, "F", "f", "H", "h", "C", "c",
    "Ch", "ch", "Sh", "sh", "Shch", "shch", "HardSign", "Yi", "SoftSign",
"Eh", "eh",
    "Yu", "yu", "Ya", "ya",
    "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};


void CheckPointer(void *ptr, const char *error_message)
{
    if (ptr == nullptr)
    {
        if (reporter)
        {
            reporter->Report(error_message);
        }
        while (true)
        {
        }
    }
}


void bitmap_to_float_array(float *dest, const unsigned char *bitmap)
{
    int pixel = 0;
    int bytes_per_row = (kImageWidth + 7) / 8;
    for (int y = 0; y < kImageHeight; y++)
    {
        for (int x = 0; x < kImageWidth; x++)
        {
            int byte_index = x / 8;
            int bit_index = x % 8;
            dest[pixel] = (bitmap[y * bytes_per_row + byte_index] >> (7 -
bit_index)) & 0x1 ? 1.0f : 0.0f;
            pixel++;
        }
    }
}
```

```
void draw_input_buffer()
{
    CheckPointer(input_buffer, "Input buffer is nullptr in
draw_input_buffer");

    clear_display();
    for (int y = 0; y < kImageHeight; y++)
    {
        for (int x = 0; x < kImageWidth; x++)
        {
            draw_pixel(x + 16, y + 3, input_buffer[y * kImageWidth + x] > 0 ?
0xFFFFFFFF : 0xFF000000);
        }
    }
}

void setup()
{
    srand(static_cast<unsigned int>(time(nullptr)));

    static tflite::MicroErrorReporter error_reporter_instance;
    reporter = &error_reporter_instance;
    CheckPointer(reporter, "Failed to initialize error reporter");
    reporter->Report("Initializing model...");

    model = tflite::GetModel(tf_model);
    CheckPointer((void *)model, "Failed to load model");

    if (model->version() != TFLITE_SCHEMA_VERSION)
    {
        reporter->Report("Model schema mismatch. Expected %d but got %d",
TFLITE_SCHEMA_VERSION, model->version());
        while (true)
            ;
    }

    static tflite::ops::micro::AllOpsResolver resolver;
```

```
    static tflite::MicroInterpreter static_interpreter(model, resolver,
tensor_arena, kTensorArenaSize, reporter);
    interpreter = &static_interpreter;
    CheckPointer(interpreter, "Failed to initialize interpreter");

    if (interpreter->AllocateTensors() != kTfLiteOk)
    {
        reporter->Report("Failed to allocate tensors");
        while (true)
            ;
    }

    input = interpreter->input(0);
    CheckPointer(input, "Input tensor is nullptr");

    output = interpreter->output(0);
    CheckPointer(output, "Output tensor is nullptr");

    input_buffer = input->data.f;
    CheckPointer(input_buffer, "Input buffer is nullptr after assignment");
}

void loop()
{
    const int num_test_images = (sizeof(test_images) /
sizeof(test_images[0]));

    if (num_test_images == 0)
    {
        reporter->Report("No test images found.");
        while (true)
            ;
    }

    int random_index = rand() % num_test_images;
    bitmap_to_float_array(input_buffer, test_images[random_index]);
    draw_input_buffer();
```

```
    if (interpreter->Invoke() != kTfLiteOk)
    {
        reporter->Report("Invoke failed");
        while (true)
            ;
    }

    CheckPointer(output, "Output tensor is nullptr before accessing data.f");

    float *result = output->data.f;
    CheckPointer(result, "Output data.f is nullptr");

    CheckPointer(output->dims, "Output dims is nullptr");
    CheckPointer(output->dims->data, "Output dims data is nullptr");

    int output_size = output->dims->data[1];
    int predicted_class = std::distance(result, std::max_element(result,
result + output_size));

    const char *predicted_label = nullptr;
    if (predicted_class >= 0 && predicted_class <
static_cast<int>(sizeof(class_mapping) / sizeof(class_mapping[0])))
    {
        predicted_label = class_mapping[predicted_class];
    }

    char resultText[256];
    if (predicted_label != nullptr)
    {
        snprintf(resultText, sizeof(resultText), "It looks like a: %s",
predicted_label ? predicted_label : "unknown");
    }
    else
    {
        snprintf(resultText, sizeof(resultText), "Failed to recognize");
    }
```

```
    draw_text(resultText, 0xFF0000FF);
    delay(1000);
}
#include "stm32_app.h"

static void SystemClock_Config( void );

void delay( int n )
{
  HAL_Delay( n );
}

void clear_display()
{
  BSP_LCD_Clear( LCD_COLOR_WHITE );
}

void draw_text( char* text, uint32_t color )
{
  BSP_LCD_SetTextColor( color );
  BSP_LCD_SetFont( &Font20 );
  BSP_LCD_DisplayStringAt( 0, 4, (uint8_t*) text, CENTER_MODE );
}

void draw_pixel( uint32_t x, uint32_t y, uint32_t color )
{
  BSP_LCD_SetTextColor( color );
  BSP_LCD_FillRect( x * 8, y * 8, 8, 8 );
}

int main( void )
{
  SCB_EnableICache();
  SCB_EnableDCache();
```

```
  HAL_Init();

  SystemClock_Config();

  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit( 0, LCD_FB_START_ADDRESS );
  BSP_LCD_DisplayOn();
  BSP_LCD_SelectLayer( 0 );

  setup();
  while (1)
  {
    loop();
  }
}

void SysTick_Handler(void)
{
  HAL_IncTick();
}

void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  HAL_StatusTypeDef ret = HAL_OK;

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 432;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 9;
```

```
  ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
  if(ret != HAL_OK)
  {
    while(1) { ; }
  }

  /* Activate the OverDrive to reach the 216 MHz Frequency */
  ret = HAL_PWREx_EnableOverDrive();
  if(ret != HAL_OK)
  {
    while(1) { ; }
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and
PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);
  if(ret != HAL_OK)
  {
    while(1) { ; }
  }
}
```

ПРИЛОЖЕНИЕ В
(обязательное)
Функциональная схема алгоритма, реализующего программное средство

ПРИЛОЖЕНИЕ Г
(обязательное)
Блок схема алгоритма, реализующего программное средства

ПРИЛОЖЕНИЕ Д
(обязательное)
Графический интерфейс пользователя

ПРИЛОЖЕНИЕ Е
(обязательное)
Ведомость курсового проекта