УДК 004.8:004.9 **doi: 10.26906/SUNZ.2024.2.090**

О. Ю. Заковоротний, А. В. Хулап

Національний технічний університет “Харківський політехнічний інститут”, Харків, Україна

ОПТИМІЗАЦІЯ ОБЧИСЛЕННЯ НЕЙРОМЕРЕЖ ЗА ДОПОМОГОЮ ВИКОРИСТАННЯ ЦІЛОЧИСЕЛЬНОЇ АРИФМЕТИКИ

Анотація. Більшість пристроїв-датчиків в системах інтернету речей базуються на енергоефективних мікроконтролерах, обчислювальні ресурси яких обмежені, як і обсяги наявної пам’яті. Підвищення захищеності використання таких пристроїв за допомогою нейромереж є важливою та актуальною проблемою. В статті описана можливість використання штучних нейронних мереж у малих мікроконтролерах з обмеженими ресурсами. **Мета** даної роботи полягає у перевірці можливості обчислення нейромереж на базі цілочисельної арифметики задля зменшення часу обчислення нейромережі та виключення операцій по нормалізації даних, а також оцінка доцільності використання таких нейромереж у сфері безпеки інтернету речей у порівнянні з традиційними методами, такими як чорні та білі списки. **Отримані наступні результати:** при переході на цілочисельну арифметику, у порівнянні з плаваючою точкою, точність обчислень результату знаходиться у межах допустимої похибки навчання нейромережі, тобто не змінилася. Час виконання зменшився на 30-96%, в залежності від архітектури мікроконтролеру. Розмір програми знизився на 22-48% також в залежності від архітектури мікроконтролеру. **Висновки.** Було доказано можливість та доцільність використання нейромереж, оптимізованих для мікроконтролерів з обмеженими ресурсами. Це підвищить захищеність систем інтернету речей особливо перед загрозами автентифікації пристроїв та виявленню вторгнень. Визначено перспективи подальших досліджень.

Ключові слова: мікроконтролер, нейромережа, оптимізація, програмна реалізація.

Вступ

Інтернет речей (IoT) почав активно застосовуватися на практиці приблизно з 2008 року. Його використання стрімко зростає, і тепер IoT є частиною повсякденного життя та займає місце в багатьох домівках і на підприємствах.

Зараз, у 2024 році, вже існує 17.08 мільярдів пристроїв IoT, а також очікується, що до 2030 року ця кількість підвищиться до 30 мільярдів пристроїв [1]. Через таку кількість пристроїв постає питання безпеки як персональних даних, так і проблема використання пристроїв для заподіяння шкоди. Наприклад, атака відома як Mirai, визначає вразливі IoT-пристрої за допомогою таблиці з понад 60 поширених заводських імен користувачів і паролів [2].

Внаслідок цієї атаки на Dyn DNS виникли проблеми із доступом до багатьох веб-сайтів, зокрема: Twitter, Etsy, Github, Soundcloud, Spotify та інші. Вірус BrickerBot був призначений для запобігання зараженню пристроїв Mirai, але він, у свою чергу, знищив більше десяти мільйонів пристроїв.

Таким чином, IoT є дуже вразливим до кібератак, особливо через відсутність стандартів безпеки та вимог до безпеки [3, с.1-3]. Через низьку захищеність багатьох пристроїв IoT зловмисники знайшли багато способів атакувати подібні пристрої. Методи та засоби атаки можуть відрізнятися в залежності від самого пристрою IoT, його апаратного та програмного забезпечення, мережі, до якої підключений пристрій IoT та програми, з якою пристрій взаємодіє. На рис. 1 наведено типову схему IoT.

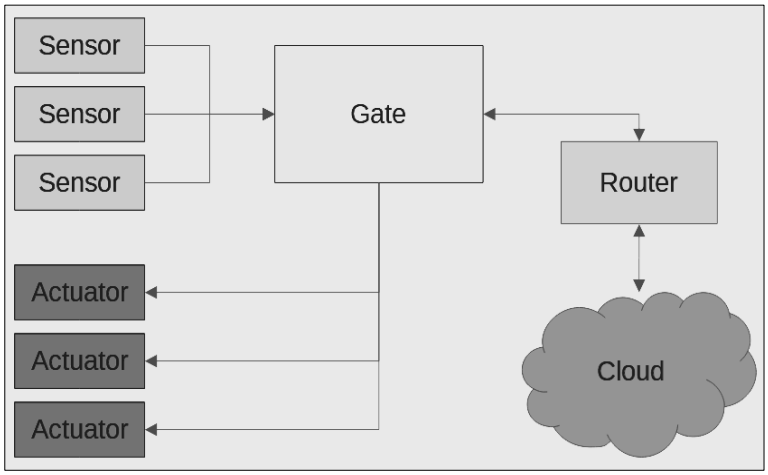

Рис. 1. Типова структура IoT

Аналіз літературних джерел [4-8] та досвід роботи фахівців IoT показує, що більшість атак відбуваються через підключення до мережевого шлюзу або хмарного серверу.

Постановка проблеми

Машинне навчання широко застосовуються для покращення безпеки мережі, наприклад: автентифі-

кація, контроль доступу і виявлення шкідливих програм. Пристрої IoT можуть застосовувати методи контрольованого навчання для виявлення зловмисного програмного забезпечення. У схемі виявлення зловмисного програмного забезпечення, яка розроблена у [9, с.5], пристрій IoT використовує K-NN і класифікатори випадкового лісу для створення моделі виявлення зловмисного програмного забезпечення. Цей пристрій фільтрує пакети TCP і вибирає необхідні параметри, включаючи номер і довжину кадру та зберігає їх в базі даних. Згідно з експериментами в [9, с.10-11], справжній позитивний показник виявлення шкідливих програм на основі K-NN і схеми на основі випадкового лісу з набором даних MalGenome становить 99,7% і 99,9% відповідно.

Пристрої IoT можуть завантажувати журнали мережевого трафіку на сервери в хмарі для виявлення зловмисного програмного забезпечення. Сервери мають більшу базу даних, вищу швидкістю обчислення, більший обсяг пам'яті та інші обчислювальні ресурси. Набір даних для відправлення у хмару залежить від якості каналу зв'язку для кожного периферійного пристрою та обсягу даних [10].

У такій схемі роботи, де центральний вузол (хмара) накопичує дані, створює та навчає необхідні конфігурації нейромережі, а пристрої IoT тільки виконують обчислення цих нейромереж, все одно постає проблема ресурсів. Більшість пристроїв IoT базуються на малих мікроконтролерах з обмеженими обсягами пам'яті програм та даних. Ресурси таких мікроконтролерів не дозволяють використовувати операційні системи та наявні бібліотеки, такі як TensorFlow або Brain.js [11]. Такі пристрої навіть не можуть проводити обчислення нейромереж у реальному часі, так як окрім самого обчислення, потрібно ще виконати операції нормалізації та перетворення даних.

Виконання усіх обчислень за допомогою цілочисельної арифметики покращує як проблему швидкодії (так як більшість систем команд мікроконтролерів створені тільки для роботи з цілими числами), так і вирішує проблему перетворення і нормалізації даних, так як абсолютна більшість параметрів мережевих протоколів є цілими числами які і так нормалізовані через розмір відповідних полів протоколу.

У зв'язку з цим, **метою дослідження є** оптимізація обчислення нейромережі, не використовуючи сторонні бібліотеки або додаткові апаратні засоби (такі як сопроцесор), а використовуючи тільки засоби з бібліотеки glibc, яка є відкритою реалізацією стандартної бібліотеки C, що дозволить значно розширити використання нейромереж у мікроконтролерах, а також дозволить уникнути потенційних проблем з ліцензіями.

Основна частина роботи

Для перевірки можливості оптимізації обчислення нейромереж було проведено наступний експеримент. Його план полягає у наступному:

1. Створити набір даних для тренування нейромережі. Буде використовуватися постійно для порівняння результатів.

2. Змоделювати нейромережу за допомогою brain.js.

3. Переробити обчислення цієї нейромережі на мові C з використанням плаваючої точки та з використанням цілочисельної арифметики.

4. Провести дослідження швидкодії обчислення отриманих нейромереж на декількох архітектурах мікроконтролерів та зробити висновок о доцільності даного підходу.

Для експерименту будемо використовувати дані з мережевого протоколу UDP. Цей метод також підходить для будь-якого комунікаційного інтерфейсу де визначені апаратна адреса, програмна адреса та розмір пакета. Разом усі ці параметри дозволяють визначити типову модель поведінки для пристроїв інтернету речей, а на основі цієї моделі - робити розпізнавання свій-чужий, в залежності від типової чи нетипової поведінки.

Слід зауважити, що всі пакети прийняті центральним мікроконтролером від хмари, будуть мати MAC-адресу роутера. Але, для зв'язку з сенсорами (sensor) та приводами (actuator), де зв'язок іде напряму, ці дані є актуальними.

Для створення набору даних була написана програма на мові C, яка за допомогою генератора довільних чисел створювала набір даних у двох форматах: .json для використання у brain.js та .h для використання у програмах на мові C. Усього було створено 10000 записів, які імітують реальну модель використання. На основі цих даних та за допомогою бібліотеки Brain.js було створено та навчено 2 нейромережі прямого розповсюдження, тобто без зворотного зв'язку. Такі нейромережі найкраще підходять для задач розпізнавання образів та класифікації даних. Перша нейромережа була обрана з мінімальною кількістю нейронів у внутрішньому шарі (4 - по кількості входів). Друга - навпаки, з максимально доцільною кількістю (два шари: 8 та 16 нейронів відповідно). Як активатор було обрано сигмовидну функцію. Вона нелінійна і також є предметом для дослідження швидкодії та оптимізації, на відміну від лінійної функції-випрямляча (rectifier, ReLU) яка не представляє жодної проблеми при обчисленні.

На рис. 2 зображена нейромережа з 4 нейронами у внутрішньому шарі, як її представляє бібліотека Brain.js.

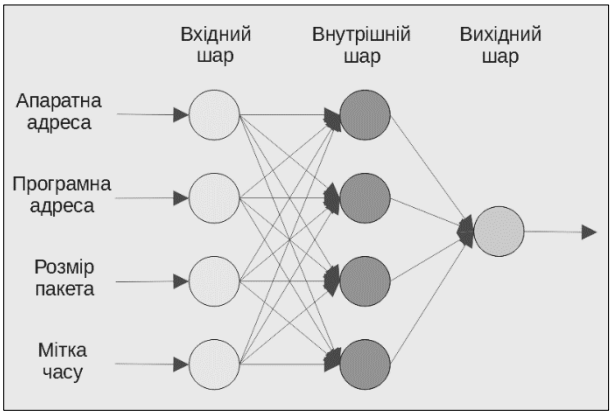

Рис. 2. Структура нейромережі з 4 нейронами у внутрішньому шарі

Слід зазначити, що вхідний шар не використовується для обчислень, а тільки для встановлення значень вхідних параметрів.

Бібліотека Brain.js дозволяє зберігати навчені нейромережі у .json форматі, тому відомі вагові коефіцієнти кожного нейрона. Також Brain.js є відкритою бібліотекою, тому програмний код функції _runInputSigmoid() також відомий і її можливо перекодувати на мові С [12]. Окрім переходу на цілочисленні обчислення, був також проведений експеримент з типом даних float. На відміну від 64-бітного типу double, який використовується у Brain.js, тип float має розмір 32 біта. Це повинно значно скоротити як час обчислення, так і обсяг пам'яті, необхідний для зберігання конфігурації нейромережі.

При обчисленні за допомогою цілих чисел використовувалося 32-бітне слово у форматі наведеному на рис. 3. Якщо спрощено, то це звичайні знакові 32-бітні цілі числа (тип int32_t визначений у stdint.h), де старший байт - це ціла частина та знак, а 3 молодші байти - це дрібна частина. Наприклад, число 1.5 буде виглядати як 0x01800000, а максимальне число буде 127.99999994 (0x7FFFFFFF).

1 біт	7 біт	24 біта
знак	ціла частина	дрібна частина

Рис. 3. Формат даних з псевдоплаваючою точкою

Обчислення при перетворенні робиться на етапі формування таблиці коефіцієнтів нейромережі та не виконується у реальному часі, тобто немає затрат на конвертацію даних. Також плюсом використання цілих чисел є відсутність необхідності виконувати операції ділення чи множення під час нормалізації даних. Ці дані є параметрами мережевих протоколів, максимальні значення задані розміром відповідних полів заголовків протоколів, і кратні 2. Тому при нормалізації здебільшого використовується операція зсуву, що теж підвищує швидкодію.

У ході роботи програми використовується 3 основних операції, додавання, множення та обчислення сигмовидної функції:

- Операція додавання не потребує будь-яких додаткових перетворень та виконується як додавання двох 32-бітних знакових цілих чисел, для більшості 32-бітних архітектур це буде виконуватися за 1 цикл процесора.
- Операція множення виконується як множення 32 біта на 32 біта з результатом у 64 бітах та зі зсувом цього результату на 24 біта вліво. У архітектурі ARM множення за допомогою інструкції SMULL займає 3 цикла процесора, зсув також займе декілька циклів [13]. Для інших архітектур, наприклад AVR, дану операцію можна переписати на асемблері щоб уникнути непотрібних операцій з бібліотеки С, тим самим значно скоротивши час виконання. Дуже важливо, що у даному випадку один із множників завжди не більший за 1 (результат або нормалізації вхідних даних, або результат сигмовидної функції), тому переповнення не виникає.
- При обчисленні сигмовидної функції був обраний найшвидший варіант з можливих - табличний. Він можливий тільки для цілих чисел, тому що аргумент використовується як індекс у таблиці. Так як сигмоїда — це *S*-подібна функція, то у таблиці можна зберігати тільки частину результатів. Для даного випадку було обрано зберігати 256 значень у діапазоні від 0 до 7, тобто 3 цілих розрядів та 5 дрібних. Така таблиця займає рівно 1 кБ, що є більшим ніж розмір коду для обчислення експоненти з математичної бібліотеки С. Але, зважаючи на великі обсяги пам'яті програм у сучасних мікроконтролерах та значне скорочення часу виконання, це не є критичним недоліком.

Було проведено моделювання роботи нейромереж з використанням цілочисельної арифметики, а також з використанням типу даних float. На даному етапі не виконувалася оцінка швидкодії, а виконувалося тільки порівняння результатів. Тому моделювання виконувалося на ПК. Аналіз проводився методом подачі на вхід нейромережі тих же самих даних для навчання та порівняння значень вихідних нейронів між собою.

Порівняння показало практично повну ідентичність результатів, різниця вихідних значень знаходиться у межах похибки. У табл. 1 приведено порівняння результатів роботи еталонної нейромережі Brain.js та нейромереж, оптимізованих за допомогою мови С. Вказані мінімальне, максимальне та середнє відхилення значень вихідного нейрона на наборі у 10000 тестових даних.

Таблиця 1 – **Відхилення результатів роботи нейромережі від еталонних**

Відхилення	**Нейромережа [4]**		**Нейромережа [8, 16]**	
	Між Brain.js та Float	**Між Brain.js та Integer**	**Між Brain.js та Float**	**Між Brain.js та Integer**
Мінімальне	0.000002999	0	0.000015999	0
Максимальне	0.017194478	0.053537817	0.035021356	0.1162835
Середнє	0.000616933	0.00102014	0.002131325	0.00963141

Набагато більшій вплив має інтерпретація результатів, тобто що вважати 1 (свій), а що – 0 (чужій).

Різні порогові значення значно змінюють бінарній результат, але, знов таки, усі отримані результати майже ідентичні. Тому у цій частині експерименту було зроблено висновок про можливість оптимізації обчислення нейромереж та доцільність наступного етапу експерименту - оцінки часу виконання роз-

рахунку. У табл. 2 наведені результати розпізнавання свій-чужий в залежності від порогів визначення результату роботи нейромережі (значення вихідного нейрона).

Таблиця 2 – **Порівняння результатів роботи нейромереж**

Параметри визначення	Тип результату	Нейромережа [4]			Нейромережа [8, 16]		
		Brain	Float	Int	Brain	Float	Int
“1” - [1; 0.9) “0” - (0.1, 0]	Не Визначено	132	130	133	1189	1189	1199
	Кількість помилок (фальш-позитивні / фальш-негативні)	37 (37 / 0)	37 (37 / 0)	37 (37 / 0)	79 (26 / 53)	79 (26 / 53)	77 (27 / 50)
“1” - [1; 0.5] “0” - (0.5; 0]	Кількість помилок (фальш-позитивні / фальш-негативні)	46 (46 / 0)	47 (47 / 0)	47 (47 / 0)	157 (33 / 124)	158 (33 / 125)	157 (34 / 123)

Заключною частиною експерименту була оцінка швидкодії виконання обчислень та визначення приросту продуктивності. Для тесту був використаний мікроконтролер STM32L476 виробництва ST Electronics на базі ядра Cortex-M4. Цей мікроконтролер є дуже популярним серед пристроїв інтернету речей через низьке енергоспоживання, високу продуктивність та розвинуту інфраструктуру. Ядро ARM Cortex також є дуже широко поширеним, тому результати можуть бути застосовані і для великої кількості інших мікроконтролерів. Також було використане моделювання для архітектури AVR виробництва Microchip (ATMEL) за допомогою симулятора.

Було виконано заміри часу виконання нейромережі на типу даних double (64 біт, еталон), а також float та integer (32 біт). Для типів даних з плаваючою комою були виконані розрахунки як з використанням експоненціальної функції як із стандартної бібліотеки математики libm, так і з використанням її спрощеного варіанта.

У табл. 3 приведені заміри часу одного повного обчислення різних варіантів нейромережі на процесорі Cortex-M4 з частотою 80 МГц, вимірювання виконувалося за допомогою осцилографа.

У табл. 4 приведені такі ж самі результати, але для процесора архітектури AVR, який працює на частоті 16 МГц. Обчислення для типу даних double не виконувалися, тому що стандартна бібліотека С для цієї архітектури не розрізняє типи double та float – вони обидва мають розмір 32 біта.

Виходячи з даних, наведених у табл. 1 та 3, можливо зробити висновок, що при переході на цілочисельну арифметику, у порівнянні з плаваючою точкою, точність обчислень результату знаходиться у межах допустимої похибки навчання нейромережі, тобто не змінюється (згідно табл. 1, максимальне середнє відхилення від еталону менше 1%).

Час виконання зменшився приблизно у 30 разів, в залежності від архітектури нейромережі (табл. 3 та 4, час обчислення).

Таблиця 3 – **Результати роботи на процесорі Cortex-M4, 80 МГц**

Тип даних	Час обчислення, нейромережа [4], мкс	Час обчислення, нейромережа [8,16], мкс	Використання пам'яті програм, байт	Використання пам'яті даних, байт
double, sigmoid	688	3240	7828	9644
double, rough_sigmoid	428	2144	6800	9564
float, sigmoid	116.8	257	7132	5036
float, rough_sigmoid	94.4	163.6	6496	4956
integer, table sigmoid	22.64	103.2	6100	4956

Таблиця 4 – **Результати роботи на процесорі AVR, 16 МГц**

Тип даних	Час обчислення, нейромережа [4], мкс	Час обчислення, нейромережа [8,16], мкс	Використання пам'яті програм, байт	Використання пам'яті даних, байт
float, sigmoid	14906	80002	7444	4802
float, rough_sigmoid	10558	61152	7152	4802
integer, table sigmoid	8154	55669	7932	4802

Розмір програми знизився пропорційно типу даних тому що значна частка - це зберігання конфігурації нейромережі).

Також при переході від 64-бітового типу даних до 32-бітового, обсяг пам'яті, необхідний для зберігання конфігурації нейромережі, аналогічно зменшується у 2 рази (табл. 3, використання пам'яті даних).

Висновки та перспективи подальших досліджень

Для архітектури Cortex-M4, обчислення за 100 мкс (найскладніша з розглянутих конфігурацій) робить можливим використання нейромереж для розпізнавання свій-чужий у реальному часі.

Архітектура AVR непридатна для таких операцій, для неї повинні використовуватися табличні методи, наприклад чорні та білі списки. За допомогою оптимізації обчислення множення 32 біта на 32 біта з результатом у 64 бітах на мові асемблера, можна досягти ще значнішого зниження часу виконання, але все одно він залишиться неприйнятним.

Виходячи з отриманих результатів, нейромережі можливо використовувати у малих мікроконтролерах з обмеженими ресурсами. Це відкриває перспективи подальших досліджень у сфері штучного інтелекту для захисту інтернету речей. За допомогою нейромереж можна створювати поведінкові моделі використання пристроїв, та на їх основі проводити розпізнавання свій-чужий, виявляти аномальну поведінку та виявляти вторгнення.

Наступні дослідження будуть стосуватися вибору набору параметрів мережевих протоколів, як то IP-адреса, порт, розмір пакета та інше, а також створення початкового універсального набору даних, який пришвидшить подальше навчання системи працюючою вже згідно власної моделі.

СПИСОК ЛІТЕРАТУРИ

1. Duarte F. Number of IoT Devices. 2024. URL: https://explodingtopics.com/blog/number-of-iot-devices
2. Ragan S. Here are the 61 passwords that powered the Mirai IoT botnet. Csoonline, 2016. URL: https://www.csoonline.com/article/3126924/here-are-the-61-passwords-that-powered-the-mirai-iot-botnet.html
3. Vorakulpipat C., Rattanalerdnusorn E., Thaenkaew P., Hai H.D. Recent challenges, trends, and concerns related to IoT security: an evolutionary study. IEEE, 2018. DOI: https://doi.org/10.23919/ICACT.2018.8323774
4. Abdul-Ghani H.A., Konstantas D., Mahyoub M. A comprehensive IoT attacks survey based on a building-blocked reference mode. *International Journal of Advanced Computer Science and Applications*, Vol. 9, No. 3, 2018. С. 355–373. URL: https://doi.org/10.14569/IJACSA.2018.090349
5. Заковоротний О.Ю., Орлова Т.О. Порівняльний аналіз хмарних та туманних середовищ Інтернету речей. *Системи управління, навігації та зв'язку*, 2023, випуск 2(72). С. 152–154. DOI: https://doi.org/10.26906/SUNZ. 2023.3.152
6. Melamed T. An active man-in-the-middle attack on Bluetooth smart devices. Safety and Security Eng., Vol. 8, No. 2, 2018. С. 200–211. URL: https://www.witpress.com/Secure/ejournals/papers/SSE080202f.pdf
7. De Donno M., N Dragoni N., Giaretta A. Analysis of DDoS-capable IoT malwares. IEEE, 2017. DOI: https://doi.org/10.15439/2017F288
8. Cekerevac Z., Dvorak Z., Prigoda L., Cekerevac P. Internet of things and the man-in-the-middle attacks–security and economic risks. MEST Journal, 2017. 11 с. DOI: https://doi.org/10.12709/mest.05.05.02.03
9. Narudin F.A., Feizollah A., Anuar N.B.,Gani A. Evaluation of machine learning classifiers for mobile malware detection. Soft Computing, 2016. DOI: https://doi.org/10.1007/s00500-014-1511-6
10. Xiao L ., Li Y.,Huang X, Du X.J. Cloud-based malware detection game for mobile devices with offloading. IEEE, 2017. DOI: https://doi.org/10.1109/TMC.2017.2687918
11. Brain.js: GPU accelerated Neural networks in JavaScript. URL: https://brain.js.org/#/ (дата звернення 20.03.2024).
12. Githib: Some cool AI tools. Now the community carries the torch. URL: https://github.com/BrainJS/brain.js/blob/master/src/neural-network.ts
13. Cortex-M3 Devices Generic User Guide. URL: https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set

Received (Надійшла) 19.02.2024
Accepted for publication (Прийнята до друку) 24.04.2024

Optimization of neural network computation using integer arithmetic

Oleksandr Zakovorotnyi, Andrii Khulap

Abstract. Most of the sensor devices in the Internet of Things systems are based on energy-efficient microcontrollers, the computing resources of which are limited, as well as the amount of available memory. Increasing the security of the use of such devices with the help of neural networks is an important and urgent problem. The article describes the possibility of using artificial neural networks in small microcontrollers with limited resources. **The purpose of this work** is to check the possibility of calculating neural networks based on integer arithmetic to reduce the time of calculating a neural network and eliminate data normalization operations, as well as to evaluate the feasibility of using such neural networks in the field of security of the Internet of Things in comparison with traditional methods, such as black lists and white lists. **The following results were obtained:** when switching to integer arithmetic, compared to floating point, the accuracy of the result calculations is within the permissible error of neural network training, that is, it has not changed. Execution time decreased by 30-96%, depending on the architecture of the microcontroller. The program size is reduced by 22-48%, also depending on the microcontroller architecture. **Conclusions:** the possibility and expediency of using neural networks optimized for microcontrollers with limited resources was proved. This will increase the security of Internet of Things systems, especially against device authentication threats and intrusion detection. Prospects for further research are determined.

Keywords: microcontroller, neural network, optimization, software implementation.