УДК 004.932.2:004.056.55:004.451 DOI https://doi.org/10.35546/kntu2078-4481.2025.4.2.20

В. С. СИЧКОВ
асистент кафедри інженерії програмного забезпечення
Національний університет «Одеська політехніка»
ORCID: 0009-0009-3709-2029

ПОРІВНЯННЯ АЛГОРИТМІВ ВИЗНАЧЕННЯ ЧАСТОТНИХ СКЛАДОВИХ СИГНАЛУ В МОБІЛЬНИХ СИСТЕМАХ

У статті проведено систематичний аналіз алгоритмів визначення частотних складових цифрового сигналу та їхньої ефективної реалізації у мобільному середовищі Android. Розглянуто класичні підходи до спектрального аналізу, зокрема швидке перетворення Фур'є (FFT), алгоритм Герцеля (Goertzel) та автокореляційні методи (YIN, MPM), із позиції їхньої придатності для роботи в режимі реального часу на пристроях із обмеженими обчислювальними ресурсами. Показано, що в умовах мобільної архітектури головною проблемою є баланс між точністю та швидкодією.

З метою усунення цих обмежень розроблено нативну реалізацію обчислювально інтенсивних етапів спектрального аналізу на мові C++ із використанням Android NDK. Ключові елементи архітектури включають бібліотеку KissFFT для виконання швидкого перетворення Фур'є, технологію ARM NEON для векторної оптимізації операцій множення та додавання, а також бібліотеку Oboe, що забезпечує низьколатентний обмін аудіоданими без участі віртуальної машини. У роботі наведено принципи інтеграції між Java та C++ через JNI, мінімізацію копіювання даних і створення власного пулу буферів для уникнення пауз збирача сміття під час обробки сигналів.

Експериментальна частина передбачала функціональне та навантажувальне тестування. Отримані результати підтвердили, що система здатна функціонувати у режимі реального часу із загальною затримкою не більше 12 мс, що відповідає вимогам професійних музичних застосунків.

Узагальнено сформульовано рекомендації щодо вибору алгоритму залежно від цільового призначення: FFT для спектрального аналізу й візуалізації, Goertzel для вибіркової частотної детекції, автокореляційні методи для точного визначення основного тону. Доведено, що поєднання нативної обробки, SIMD-оптимізацій та алгоритмічної гнучкості забезпечує оптимальний баланс між точністю, стабільністю та енергоефективністю. Запропонований підхід може бути використаний у подальшому для розроблення мобільних аудіотюнерів, аналізаторів спектра, систем автоматичного розпізнавання звуку й навчальних засобів для музикантів.

Ключові слова: *інженерія програмного забезпечення для мобільних застосунків, спектральний аналіз сигналів, оптимізація продуктивності; Android NDK, системи реального часу.*

V. S. SYCHKOV
Assistant at the Software Engineering Department
Odesa Polytechnic National University
ORCID: 0009-0009-3709-2029

COMPARISON OF ALGORITHMS FOR DETERMINING THE FREQUENCY COMPONENTS OF A SIGNAL IN MOBILE SYSTEMS

The article presents a systematic analysis of algorithms for determining the frequency components of a digital signal and their efficient implementation in the Android mobile environment. Classical approaches to spectral analysis, including the Fast Fourier Transform (FFT), Goertzel algorithm, and autocorrelation methods (YIN, MPM), are considered in terms of their suitability for real-time operation on devices with limited computational resources. It is shown that within the constraints of mobile architecture, the main challenge lies in balancing accuracy and computational speed.

To overcome these limitations, a native implementation of computationally intensive stages of spectral analysis was developed in C++ using the Android NDK. The key architectural elements include the KissFFT library for performing fast Fourier transforms, ARM NEON technology for vector optimization of multiplication and addition operations, and the Oboe library, which provides low-latency audio exchange without involving the virtual machine. The paper describes the principles of integration between Java and C++ via JNI, minimization of data copying, and the creation of a custom buffer pool to avoid garbage collector pauses during signal processing.

The experimental part included both functional and stress testing. The obtained results confirmed that the system can operate in real-time mode with a total latency not exceeding 12 ms, which meets the requirements of professional music applications.

General recommendations have been formulated for selecting the appropriate algorithm depending on the target task: FFT for spectral analysis and visualization, Goertzel for selective frequency detection, and autocorrelation methods for precise fundamental tone estimation. It has been demonstrated that combining native signal processing, SIMD optimization, and algorithmic flexibility ensures an optimal balance between accuracy, stability, and energy efficiency. The proposed approach can be further applied to the development of mobile audio tuners, spectrum analyzers, automatic sound recognition systems, and educational tools for musicians.

Key words: *software engineering for mobile applications; signal spectral analysis; performance optimization; Android NDK; real-time systems.*

Постановка проблеми

У сучасних мобільних пристроях дедалі частіше реалізуються функції, які раніше були притаманні спеціалізованим вимірювальним приладам: спектроаналізаторам, тюнерам або системам акустичного контролю. Завдяки розвитку процесорних архітектур ARM та появі високопродуктивних аудіоінтерфейсів мобільні системи здатні виконувати складні обчислення цифрової обробки сигналів (Digital Signal Processing, DSP) у реальному часі. Водночас умови мобільного середовища створюють низку технічних обмежень: обмежена обчислювальна потужність, вплив системного збору сміття (Garbage Collector) у віртуальній машині Android, енергоспоживання, а також вимоги до затримки сигналу, що не повинна перевищувати 10–20 мілісекунд у професійних застосунках.

Одним із ключових завдань цифрової обробки сигналів є визначення частотних складових звукового потоку, тобто аналіз його гармонічної структури. Від точності та швидкодії цього процесу залежить коректність роботи великої кількості застосунків (від музичних тюнерів і систем розпізнавання мовлення до медичних сенсорів і діагностичних комплексів). Традиційно для цього використовуються алгоритми швидкого перетворення Фур'є (FFT), автокореляційні методи та алгоритм Герцеля (Goertzel), проте їх ефективність значною мірою залежить від платформи реалізації та оптимізації програмного коду.

На рівні Java або Kotlin середовище Android обмежує швидкодію таких алгоритмів через проміжну інтерпретацію байт-коду та періодичну зупинку виконання програми під час очищення пам'яті. Це створює аудіоартефакти («клацання», пропуски кадрів) і робить неможливим стабільний аналіз сигналів у реальному часі. Тому постає потреба у перенесенні критичних обчислень на нативний рівень із використанням Android NDK та мови C++, що дозволяє отримати прямий доступ до апаратних ресурсів, у тому числі SIMD-інструкцій ARM NEON, і суттєво знизити затримку.

Проблема полягає у виборі та порівнянні алгоритмів, здатних забезпечити оптимальний баланс між точністю спектрального аналізу, швидкодією обчислень і енергетичною ефективністю у мобільних системах. Вирішення цієї задачі передбачає не лише аналітичне порівняння математичних властивостей алгоритмів FFT, Goertzel та Autocorrelation, а й експериментальну перевірку їх ефективності у середовищі Android при реалізації через NDK.

Аналіз останніх досліджень і публікацій

Останнє десятиліття характеризується зміщенням акценту з «чисто алгоритмічних» досліджень до системних інженерних підходів, що враховують продуктивність, латентність та енергоефективність у мобільних DSP-системах. На рівні апаратної архітектури доведено, що продуктивність мобільних процесорів ARM значною мірою залежить від ефективності векторних інструкцій (SIMD), які визначають швидкість обчислення спектральних алгоритмів. У роботі [1, с. 1] показано, що коректна реалізація векторної обробки дозволяє зменшити час виконання DSP-завдань і підвищити енергоефективність мобільних пристроїв.

Водночас на рівні програмної реалізації перехід від керованого середовища Java до нативного C++ через Android NDK забезпечує контроль над «сирими» аудіобуферами та використання SIMD-інструкцій без затримок збирача сміття. У дослідженні [2, с. 110] продемонстровано, що такий підхід знижує затримку аудіосигналу з 26,7 мс до 2,7 мс, а також скорочує енергоспоживання приблизно на 40 %. Подібні висновки підтверджують автори [3, с. 35], які реалізували прискорену потокову обробку даних у реальному часі на Android із використанням GNU Radio та апаратних SIMD-інструкцій.

З точки зору алгоритмів аналізу спектра метод FFT залишається універсальним для отримання повного частотного спектра, тоді як алгоритм Goertzel доцільний для вибіркового виявлення окремих частотних складових. У роботі [4, с. 1] наведено приклади застосування Goertzel-фільтрації для мобільних IoT-пристроїв із мінімальними обчислювальними витратами. Подальші дослідження [5, с. 45] підтвердили, що поєднання ознак, сформованих алгоритмом Goertzel, з класифікаторами k-NN підвищує точність розпізнавання DTMF-тонів у шумовому середовищі.

Окремий напрям розвитку стосується нейронних моделей аудіообробки, інтегрованих у мобільні застосунки. Автори [6, с. 33] показали можливість виконання кількох аудіоефектів у реальному часі безпосередньо на смартфоні за допомогою рушіїв ONNX Runtime та RTNeural, що доводить життєздатність гібридних систем DSP + NN у мобільних умовах. У сфері радіосигнальної DSP Liu та ін. У дослідженні [7, с. 1801] запропонувано інтегровану модель зв'язку та сенсингу (ISAC) на основі стисненого вимірювання, яка перевищує класичний FFT за точністю оцінювання частот і швидкостей у багатоканальних системах.

Порівняльні дослідження [8, с. 1] продемонстрували, що реалізація FFT на C++/NDK є у 2–5 разів швидшою за Java-версію, а також стабільнішою щодо затримки GC (garbage collector). Практичні експерименти [9, с. 41] підтвердили ці висновки на рівні системного профілювання WiNTECH'20: нативна обробка забезпечує постійну латентність, відсутність джитеру та зменшення навантаження CPU.

На завершення, у роботі [10, с. 1] систематизовано методи оптимізації FFT для мобільних пристроїв та показано, що застосування попередньо обчислених таблиць коефіцієнтів і векторизації дозволяє скоротити час обробки до 50% без погіршення точності. Наведений аналіз досліджень підтверджує доцільність застосування комплексного підходу до реалізації алгоритмів визначення частотних складових у мобільних системах, який поєднує апаратну оптимізацію на рівні SIMD, ефективне нативне програмування мовою C++ та використання гібридних моделей обробки сигналів.

Формулювання мети дослідження

Метою даного дослідження є порівняльний аналіз алгоритмів визначення частотних складових цифрового сигналу в умовах обмежених ресурсів мобільних пристроїв та оцінювання ефективності їхньої програмної реалізації із застосуванням Android NDK.

Для досягнення цієї мети передбачено:

– дослідити сучасні підходи до спектрального аналізу цифрових сигналів (FFT, Goertzel, автокореляційні методи) у контексті мобільних систем;

– виявити закономірності впливу архітектурних обмежень Android-пристроїв (кероване середовище ART, збирач сміття, відсутність SIMD-інструкцій у Java) на точність і швидкодію алгоритмів;

– розробити та протестувати нативні реалізації обчислювально інтенсивних етапів (зокрема FFT) мовою C++ із використанням бібліотек KissFFT, NEON та Oboe;

– виконати експериментальну верифікацію розробленого рішення за критеріями точності, стабільності, споживання ресурсів і часових затримок;

– сформулювати рекомендації щодо вибору оптимального алгоритму та способу його реалізації для мобільних аудіосистем реального часу.

Викладення основного матеріалу дослідження

Аналіз частотних складових цифрового сигналу є фундаментальною задачею цифрової обробки сигналів, від якої залежить точність розпізнавання звуків, нот або шумів у мобільних застосунках. У контексті сучасних мобільних систем найпоширенішими підходами до спектрального аналізу є швидке перетворення Фур'є, алгоритм Герцеля та методи часової області на основі автокореляції.

Метод FFT вважається універсальним інструментом для визначення спектрального складу сигналу. Його основна перевага – можливість отримати повну картину частотних компонентів, що дозволяє не лише визначати основну частоту, але й аналізувати гармоніки та шумові складові. Разом із тим цей метод є обчислювально складним, що накладає обмеження на швидкодію в мобільних пристроях. Для реальних застосунків FFT зазвичай комбінують з оптимізаціями, такими як попередньо обчислені таблиці коефіцієнтів, інтерполяція максимуму амплітуди для підвищення точності та застосування віконних функцій для зменшення спектрального витікання.

Алгоритм Goertzel відрізняється від FFT тим, що обчислює не весь спектр, а лише окремі його частини. Це робить його ефективним для вибіркового аналізу сигналів, наприклад, для розпізнавання DTMF-тонів або контролю окремих гармонік. Його реалізація проста, потребує мінімум пам'яті та забезпечує низьке споживання ресурсів. Проте, коли необхідно аналізувати широкий частотний діапазон, метод поступається FFT за швидкодією. У мобільних пристроях алгоритм Goertzel є оптимальним вибором для задач із вузьким частотним фокусом або для локального уточнення результатів, отриманих за допомогою FFT.

Автокореляційні методи (зокрема YIN і MPM) належать до підходів часової області, що виявляють періодичність сигналу без прямого переходу до частотного спектра. Вони відзначаються високою точністю у визначенні основної частоти звуку, особливо для живих музичних інструментів і голосу, оскільки менш чутливі до гармонічних збурень і шуму. Основний недолік цих методів – підвищена обчислювальна складність, яка може бути зменшена за рахунок комбінування з FFT (через теорему Вінера–Хінчина).

У мобільних середовищах ефективність кожного з методів залежить від компромісу між точністю, швидкодією та доступними ресурсами. FFT залишається базовим алгоритмом для повноспектрального аналізу, Goertzel – для вузькосмугового моніторингу або вибіркової детекції, а автокореляційні методи – для задач високої точності визначення тону. Таким чином, у контексті Android-платформи доцільним є комбінування цих підходів: використання FFT для швидкої оцінки спектра, Goertzel – для уточнення окремих гармонік, а автокореляційних методів – для стабілізації результату в шумових умовах.

Таким чином, сучасні мобільні системи цифрової обробки сигналів орієнтуються на гібридні архітектури, де класичні алгоритми (FFT, Goertzel, ACF) поєднуються з оптимізаціями на рівні коду та апаратного прискорення. Це дозволяє забезпечити необхідну точність визначення частотних складових при мінімальних часових затримках, що є ключовим для професійних аудіозастосунків реального часу.

Реалізація алгоритмів цифрової обробки сигналів у мобільному середовищі має низку специфічних обмежень, зумовлених особливостями архітектури Android. Найбільш критичними чинниками, що впливають на точність і стабільність роботи спектральних методів, є кероване середовище виконання ART, робота Garbage Collector, відсутність прямого доступу до SIMD-інструкцій (NEON) у Java, а також особливості багатоядерних процесорів ARM.

Кероване середовище ART забезпечує автоматичне керування пам'яттю, але це супроводжується періодичними паузами збирача сміття. У задачах обробки аудіо в реальному часі навіть короткочасна затримка викликає порушення рівномірності обробки кадрів, так званий джиттер. Це призводить до втрати стабільності спектральних оцінок, «стрибань» піків частот на FFT-графіках і появи помилок при визначенні основного тону. У середовищі Java неможливо повністю контролювати момент спрацювання збирача сміття, тому такі затримки є невідворотними.

Крім того, значний вплив має межа між керованим і нативним кодом. При передачі масивів даних між Java та C++ через JNI часто відбувається приховане копіювання пам'яті, що збільшує латентність і створює нерівномірність у потоковій обробці. Це проявляється у випадкових коливаннях часу обробки кадрів, через що результати алгоритмів FFT або Goertzel можуть змінюватися навіть за однакових вхідних даних.

Додатковим фактором є динамічне перемикання потоків між «великими» і «малими» ядрами (big.LITTLE-архітектура) та частотне масштабування (DVFS). Android оптимізує споживання енергії, переміщуючи обчислювальні процеси між ядрами різної продуктивності. Це створює ще один тип джиттеру, що впливає на сталу затримку аудіообробки. У разі нагрівання пристрою система може знижувати частоту процесора, що тимчасово збільшує час обробки сигналу.

Усі ці фактори безпосередньо позначаються на двох головних характеристиках – точності та швидкодії. Нерівномірний час виконання викликає зміщення оцінених частотних піків, тоді як недостатня пропускна здатність без SIMD призводить до грубішої частотної дискретизації або скорочення буферів. Для мінімізації впливу цих обмежень критичні частини алгоритму доцільно реалізовувати мовою C++ через Android NDK із використанням бібліотек KissFFT та NEON-інструкцій. Нативна обробка дозволяє уникнути зайвих копій даних, усунути затримки, пов'язані зі збирачем сміття, та стабілізувати час виконання.

Нативна реалізація обчислювально інтенсивних етапів спектрального аналізу виконана на C++ із прямим доступом до аудіобуферів та векторних інструкцій процесора. Аудіотракт побудований на базі бібліотеки Oboe, яка забезпечує низьколатентний обмін аудіоданими. Під час ініціалізації застосунок створює вхідний потік із пріоритетом реального часу та мінімальним розміром буфера. Зворотний виклик onAudioReady виконується без динамічних алокацій, блокувань чи синхронізації з Java. Вхідні вибірки надходять безпосередньо до кільцевого буфера в нативному середовищі, тоді як Java/Kotlin використовується лише для графічного інтерфейсу та візуалізації результатів.

Конвеєр обробки складається з кількох послідовних етапів: попередньої фільтрації (віконні функції Hann або Blackman, видалення частот 50/60 Гц та високочастотного шуму), виконання FFT, розрахунку енергетичних показників і виявлення домінантних гармонік. Для обчислення FFT застосовано бібліотеку KissFFT, що має компактний код і не потребує складної конфігурації. Конфігураційні структури створюються один раз під час запуску програми для типових розмірів блоків (1024, 2048 та 4096 семплів), що дозволяє уникнути повторного виділення пам'яті. Дані зберігаються у вирівняних масивах типу float і kiss_fft_cpx, що підвищує ефективність векторизації при компіляції. Після отримання спектра використовується суббінне уточнення максимуму (параболічна інтерполяція або фазовий метод), завдяки чому досягається точність визначення частоти до десятих герца без збільшення розміру вікна й затримки.

Для перевірки ефективності та стабільності реалізованого рішення проведено експериментальну верифікацію, спрямовану на оцінку його придатності для роботи в режимі реального часу. Тестування виконувалося у двох режимах – функціональному та навантажувальному.

У функціональному режимі на вхід подавалися контрольні сигнали: синусоїди частотою 110, 440 та 1000 Гц, а також акустичний запис гри на фортепіано. Для кожного випадку фіксувалися абсолютна похибка визначення частоти та стабільність результатів у часі. У «тихих» умовах перевірялося поводження системи за наявності фонового мережевого шуму 50/60 Гц.

Навантажувальне тестування тривало 60 хвилин безперервної обробки звукового потоку. У процесі фіксувалися споживання пам'яті, стабільність частоти кадрів, температура пристрою, а також вплив системного масштабування частоти (DVFS) і теплового обмеження (thermal throttling) на латентність обчислень.

За результатами експериментів середній час обробки блоку обсягом 1024 вибірки становив приблизно 0,5 мс при використанні бібліотеки KissFFT та 0,2 мс після активації оптимізацій NEON. Розкид затримок залишався стабільним у межах сотих часток мілісекунди навіть за тривалої роботи. Це дозволяє повністю задовольнити вимоги до музичних застосунків реального часу, де сумарна затримка (від отримання сигналу до відображення результату на екрані) не повинна перевищувати 10–20 мс.

Додатково відзначено стабільне споживання оперативної пам'яті (~45 МБ) та відсутність витоків, що підтверджує ефективність управління ресурсами в нативному середовищі. Процесорне навантаження не перевищувало 15% навіть під час одночасного виконання декількох спектральних розрахунків.

Підсумкові результати підтвердили, що нативна реалізація з використанням Oboe, KissFFT і NEON забезпечує оптимальний баланс між точністю, швидкодією та стабільністю. Отримана архітектура повністю відповідає вимогам до мобільних систем спектрального аналізу реального часу та може бути застосована у професійних тюнерах, аудіоаналізаторах і навчальних інструментах для музикантів.

На основі проведеного аналізу та експериментальної верифікації можна сформулювати узагальнені рекомендації щодо вибору алгоритмів і технологічних засобів реалізації для забезпечення стабільної роботи аудіосистем реального часу на платформі Android.

По-перше, вибір алгоритму спектрального аналізу має ґрунтуватися на цільовому призначенні застосунку. Для задач візуалізації спектра, визначення гармонічного складу сигналу або побудови спектрограм доцільно застосовувати FFT, яке забезпечує повну картину частотних компонентів. У випадках, коли потрібно визначати наявність або амплітуду окремих частот (наприклад, при розпізнаванні DTMF-тонів або моніторингу контрольних сигналів), ефективним рішенням є алгоритм Goertzel. Для завдань визначення основної частоти звуку, особливо в умовах шуму або для живих інструментів, найкращі результати демонструють автокореляційні методи (YIN, MPM), що забезпечують високу точність, хоча й вимагають більших обчислювальних ресурсів.

По-друге, при реалізації алгоритмів у мобільному середовищі необхідно враховувати обмеження архітектури Android, зокрема вплив керованого середовища ART та збирача сміття. З метою усунення затримок і коливань часу виконання доцільно переносити критичні етапи спектральних обчислень на нативний рівень. Реалізація мовою C++ із використанням Android NDK дозволяє отримати доступ до векторних інструкцій процесора (ARM NEON), що суттєво знижує час обробки аудіоблоків і забезпечує стабільну продуктивність. Оптимальним рішенням є використання бібліотеки KissFFT, яка поєднує швидкодію, простоту інтеграції та невеликі вимоги до пам'яті.

Для аудіовводу й виводу найефективнішим підходом є використання бібліотеки Oboe, яка автоматично підбирає найкращий аудіодрайвер (AAudio або OpenSL ES) і забезпечує мінімальну латентність без додаткових накладних витрат. У процесі обробки даних варто уникати динамічного виділення пам'яті, блокувань потоків і зайвих переходів між Java і C++ через JNI. Передача даних має здійснюватися без копіювання, використовуючи DirectByteBuffer або GetPrimitiveArrayCritical із мінімальним часом утримання посилань.

Для зменшення впливу шумів і покращення точності визначення частоти рекомендовано використовувати віконні функції Hann або Blackman, нотовий фільтр (notch filter) для зменшення перешкод 50/60 Гц та високочастотний фільтр (high-pass filter) для стабілізації сигналу в умовах тиші. Додаткове застосування порогової обробки дозволяє уникнути хибних спрацьовувань при слабких сигналах або відсутності звуку.

Експериментальні результати підтвердили, що оптимальна комбінація KissFFT + NEON + Oboe забезпечує середній час обробки одного блоку (1024 семпли) на рівні 0,2–0,5 мс при стабільній затримці системи не більше 10–12 мс. Це дозволяє дотримуватись вимог до аудіосистем реального часу, які використовуються в музичних тюнерах, спектроаналізаторах і програмах-настроювачах.

Таким чином, найбільш раціональним рішенням для сучасних мобільних аудіосистем є реалізація алгоритмів спектрального аналізу у вигляді гібридного підходу, де обчислювально інтенсивні етапи виконуються в нативному середовищі C++ з апаратною оптимізацією, а допоміжні – у високорівневій частині Android-застосунку. Такий підхід забезпечує баланс між точністю, швидкодією та енергоефективністю, що є ключовими критеріями якості для систем реального часу.

Висновки

У ході дослідження виконано порівняльний аналіз алгоритмів визначення частотних складових цифрового сигналу та оцінено їхню ефективність у мобільному середовищі Android. Встановлено, що традиційні реалізації на рівні Java не забезпечують необхідної стабільності та швидкодії для систем реального часу через обмеження керованого середовища та відсутність апаратної векторизації. Перенесення обчислювально інтенсивних етапів на нативний рівень C++ із використанням бібліотек **KissFFT**, **NEON** та **Oboe** дозволило досягти значного підвищення продуктивності – скорочення часу обробки блоку до 0,2–0,5 мс та забезпечення затримки менше 12 мс.

Експериментальна верифікація підтвердила стабільність, відсутність витоків пам'яті й високу точність визначення частоти (похибка не перевищує ±0,1 Гц). Отримані результати доводять доцільність застосування нативних технологій і оптимізованих алгоритмів у задачах спектрального аналізу аудіосигналів. Розроблені підходи можуть бути використані для створення високоточних мобільних тюнерів, музичних аналізаторів та інших систем обробки звуку в реальному часі.

Список використаної літератури

1. Khadem A., Fujiki D., Talati N., Mahlke S., Das R. Vector-Processing for Mobile Devices: Benchmark and Analysis // *Proceedings of the 2023 IEEE International Symposium on Workload Characterization (IISWC 2023)*. New York : IEEE, 2023. P. 1–12. DOI 10.1109/IISWC59245.2023.00036.

2. Lee J., Park H., Kwon S. та ін. Energy-Efficient Low-Latency Audio on Android // *Journal of Systems and Software.* 2019. Vol. 157. P. 110–123. DOI 10.1016/j.jss.2019.03.013.

3. Bloessl B., Baumgärtner L., Hollick M. Hardware-Accelerated Real-Time Stream Data Processing on Android with GNU Radio // *Proceedings of the 14th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH '20).* New York : ACM, 2020. P. 35–40. DOI 10.1145/3411276.3412184.

4. Lota J., Demosthenous A. Low Computational Sensing with Goertzel Filtering for Mobile Industrial IoT Devices // *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2020).* Seville : IEEE, 2020. P. 1–5. DOI 10.1109/ISCAS45731.2020.9180896.

5. Maity A., Prakasam P., Bhargava S. Robust Dual-Tone Multi-Frequency Tone Detection Using k-Nearest Neighbour Classifier for a Noisy Environment // *Applied Computing and Informatics.* 2025. Vol. 21, No. 2. P. 45–59. DOI 10.1108/ACI-10-2020-0105.

6. Hoopes J., Chalmers B., Zappi V. Neural Audio Processing on Android Phones // *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx 2024).* Huddersfield : University of Huddersfield Press, 2024. P. 33–40. URL: https://www.dafx.de/paper-archive/2024/papers/DAFx24_paper_78.pdf.

7. Liu H., Wei Z., Chen X. Integrated Sensing and Communication Signal Processing Based on Compressed Sensing Over Unlicensed Spectrum Bands // *IEEE Transactions on Cognitive Communications and Networking.* 2024. Vol. 10, No. 3. P. 1801–1816. DOI 10.1109/TCCN.2024.3391307.

8. Danielsson A. Comparing Android Runtime with Native: Fast Fourier Transform on Android : Master's Thesis. – Stockholm : KTH Royal Institute of Technology, 2017. 78 p. URL: https://www.diva-portal.org/smash/get/diva2:1109257/FULLTEXT01.pdf.

9. Bloessl B., Baumgärtner L., Hollick M. Accelerated Processing for Wireless Applications on Android Using GNU Radio // *ACM WiNTECH '20 Companion Proceedings.* New York : ACM, 2020. P. 41–47. DOI 10.1145/3411276.3412190.

10. Sugawara K. Efficient FFT Algorithms for Mobile Devices : Master's Thesis. Espoo : Aalto University, School of Electrical Engineering, 2016. 72 p. URL: https://aaltodoc.aalto.fi/handle/123456789/19783.

References

1. Khadem, A., Fujiki, D., Talati, N., Mahlke, S., & Das, R. (2023). *Vector-processing for mobile devices: Benchmark and analysis.* In *Proceedings of the 2023 IEEE International Symposium on Workload Characterization (IISWC 2023)* (pp. 1–12). IEEE. https://doi.org/10.1109/IISWC59245.2023.00036

2. Lee, J., Park, H., Kwon, S., Kim, M., & Cho, H. (2019). *Energy-efficient low-latency audio on Android. Journal of Systems and Software, 157*, 110–123. https://doi.org/10.1016/j.jss.2019.03.013

3. Bloessl, B., Baumgärtner, L., & Hollick, M. (2020). *Hardware-accelerated real-time stream data processing on Android with GNU Radio.* In *Proceedings of the 14th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH '20)* (pp. 35–40). ACM. https://doi.org/10.1145/3411276.3412184

4. Lota, J., & Demosthenous, A. (2020). *Low computational sensing with Goertzel filtering for mobile industrial IoT devices.* In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2020)* (pp. 1–5). IEEE. https://doi.org/10.1109/ISCAS45731.2020.9180896

5. Maity, A., Prakasam, P., & Bhargava, S. (2025). *Robust dual-tone multi-frequency tone detection using k-nearest neighbour classifier for a noisy environment. Applied Computing and Informatics, 21*(2), 45–59. https://doi.org/10.1108/ACI-10-2020-0105

6. Hoopes, J., Chalmers, B., & Zappi, V. (2024). *Neural audio processing on Android phones.* In *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx 2024)* (pp. 33–40). University of Huddersfield Press. https://www.dafx.de/paper-archive/2024/papers/DAFx24_paper_78.pdf

7. Liu, H., Wei, Z., & Chen, X. (2024). *Integrated sensing and communication signal processing based on compressed sensing over unlicensed spectrum bands. IEEE Transactions on Cognitive Communications and Networking, 10*(3), 1801–1816. https://doi.org/10.1109/TCCN.2024.3391307

8. Danielsson, A. (2017). *Comparing Android runtime with native: Fast Fourier transform on Android* (Master's thesis). KTH Royal Institute of Technology. https://www.diva-portal.org/smash/get/diva2:1109257/FULLTEXT01.pdf

9. Bloessl, B., Baumgärtner, L., & Hollick, M. (2020). *Accelerated processing for wireless applications on Android using GNU Radio.* In *ACM WiNTECH '20 Companion Proceedings* (pp. 41–47). ACM. https://doi.org/10.1145/3411276.3412190

10. Sugawara, K. (2016). *Efficient FFT algorithms for mobile devices* (Master's thesis). Aalto University, School of Electrical Engineering. https://aaltodoc.aalto.fi/handle/123456789/19783

Дата першого надходження рукопису до видання: 18.11.2025
Дата прийнятого до друку рукопису після рецензування: 16.12.2025
Дата публікації: 31.12.2025