НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення з оцінки зашумлення медичних МРТ зображень»
Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ **Андрій БРИТАН**
(підпис)

Виконав _____ **Сергій ГУЗЕНКО**
(підпис)

Київ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Гузенку Сергію Володимировичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення з оцінки зашумлення медичних МРТ зображень»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: дані моніторингу якості повітря від Open-Meteo API, нормативи ГДК згідно з ДСТУ та СанПіН, методика European AQI (EEA), метод аналізу ієрархій Сааті (AHP).
Перелік питань, що підлягають дослідженню:
1. Дослідження предметної області та вимог
2. Проектування інформаційної частини
3. Проектування та розробка програмного забезпечення системи
4. Експлуатація та впровадження системи
Перелік графічних документів (за необхідності).

Дата видачі завдання « 19 » грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Андрій БРИТАН**
(підпис)

Завдання взяв до виконання _____ **Сергій ГУЗЕНКО**
(підпис)

ЗМІСТ

ВСТУП

Магнітно-резонансна томографія (МРТ) є одним із провідних методів медичної візуалізації, що забезпечує отримання детальних зображень внутрішніх

органів і тканин людини без застосування іонізуючого випромінювання. Завдяки унікальним можливостям диференціації м'яких тканин МРТ широко використовується для діагностики захворювань головного мозку, хребта, суглобів, серця та інших органів.

Попри значні переваги, МРТ зображення неминуче піддаються різним видам шуму та артефактів, що суттєво знижує їхню діагностичну цінність. Теплові шуми електронних компонентів, артефакти руху пацієнта, ефекти хімічного зсуву, артефакти Гіббса та нерівномірність магнітного поля є основними чинниками погіршення якості зображень. Об'єктивне кількісне оцінювання якості МРТ зображень набуває особливого значення в контексті клінічної практики, науково-дослідної діяльності та розробки нових методів реконструкції зображень.

Існуючі рішення для оцінки якості МРТ зображень, такі як FSL, MRIQC та MRIcroGL, є потужними, проте орієнтовані переважно на наукові задачі й мають значний поріг входження для медичного персоналу. Більшість із них функціонують у середовищі Unix/Linux і не мають зручного графічного інтерфейсу, адаптованого для практичного використання в клінічних умовах.

Актуальність розробки зумовлена відсутністю спеціалізованого україномовного програмного забезпечення для оцінки якості МРТ зображень із підтримкою форматів DICOM та зручним графічним інтерфейсом, придатним для щоденного використання в медичних закладах України.

Метою дипломної роботи є проєктування та розробка програмної системи для автоматизованої оцінки якості зашумлених медичних МРТ зображень на платформі .NET 8 з україномовним інтерфейсом.

Для досягнення мети поставлено такі завдання:

– провести системний аналіз предметної області та особливостей МРТ зображень;

– дослідити та обрати методи кількісного оцінювання якості зображень;

– виконати проєктування архітектури програмної системи з використанням UML;

– розробити алгоритми розрахунку основних метрик якості: SNR, CNR, рівномірності поля;

– реалізувати модулі виявлення артефактів, статистичного аналізу, гістограм та карт шуму;

– розробити функціонал пакетної обробки та генерації звітів;

– провести тестування та підготувати документацію для впровадження.

Об'єктом дослідження є процеси оцінки якості цифрових медичних МРТ зображень.

Предметом дослідження є методи, алгоритми та програмні засоби кількісного аналізу якості МРТ зображень.

Практичне значення роботи полягає у створенні готового до впровадження програмного забезпечення для медичних установ, лабораторій контролю якості та дослідницьких центрів, що займаються обробкою МРТ зображень.

РОЗДІЛ 1.СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області та особливості зображень МРТ

Магнітно-резонансна томографія базується на явищі ядерного магнітного резонансу (ЯМР), відкритого Блохом та Парселлом у 1946 році. МРТ сканер генерує потужне статичне магнітне поле (типово 1,5–3,0 Тл) та радіочастотні імпульси, що призводять до прецесії ядер водню у тканинах організму. Зареєстровані сигнали піддаються математичній обробці (перетворення Фур'є) та формують тривимірний масив інтенсивностей, що відображає властивості тканин [3, 4].

МРТ зображення характеризуються низкою особливостей, що принципово відрізняють їх від звичайних фотографічних або рентгенівських зображень. По-перше, інтенсивність пікселя відображає не поглинання випромінювання, а відносну концентрацію протонів (протонна щільність) і час релаксації (T1, T2). По-друге, різні імпульсні послідовності (SE, GRE, EPI, FLAIR тощо) дають принципово різні контрастні характеристики навіть для тих самих анатомічних структур. По-третє, зображення отримується у k-просторі та піддається зворотному перетворенню Фур'є, що відкриває специфічні артефакти реконструкції [2].

Основними фізичними параметрами МРТ, що впливають на якість зображень, є:

– напруженість основного магнітного поля B0 (визначає SNR та хімічний зсув);

– однорідність магнітного поля (визначає рівномірність сигналу та наявність спотворень);

– характеристики градієнтних котушок (просторова роздільна здатність, швидкість переключення);

– конструкція радіочастотних котушок (прийомних та передавальних);

– параметри імпульсної послідовності: TR, TE, flip angle, смуга пропускання.

Шум у МРТ зображеннях обумовлений декількома незалежними джерелами. Домінуючим є тепловий (джонсонівський) шум, що виникає внаслідок теплового руху зарядів у провідникових тканинах тіла пацієнта та в електронних компонентах апаратури. При стандартних умовах сканування цей шум має нормальний розподіл (розподіл Гаусса) у k-просторі, однак після перетворення Фур'є та детектування фази в магнітудних зображеннях він трансформується у розподіл Райса (Rician distribution). Це принципово важливо для точного розрахунку SNR [7, 8].

Класифікація основних типів артефактів МРТ наведена у таблиці 1.1.

Таблиця 1.1 - Класифікація артефактів МРТ зображень

Тип артефакту	Механізм виникнення	Прояв на зображенні	Метод мінімізації
Артефакт руху (Motion)	Переміщення анатомічних структур під час сканування	Розмиття, примарні копії структур	Кардіосинхронізація, дихальна компенсація
Привид (Ghosting)	Нестаціонарні процеси в напрямку фазового кодування	Повторювані копії зображення зі зміщенням	Оптимізація TR, алгоритми зменшення
Дзвін/Гіббс (Ringing)	Усічення k-простору, обмежена смуга пропускання	Осциляції по периметру різких меж	Апаратизація k-простору, збільшення матриці
Хімічний зсув	Різниця у резонансних частотах жиру та води	Зміщення жирових структур у частотному напрямку	Пригнічення жиру, широкосмугове збудження

Класифікація основних типів артефактів, що виникають під час магнітно-резонансної томографії, дозволяє систематизувати знання про природу спотворень зображення та методи їх усунення. Одним із найпоширеніших є артефакт руху (Motion), механізм виникнення якого пов'язаний із фізичним переміщенням анатомічних структур пацієнта протягом тривалого часу сканування. Навіть мікрорухи, такі як дихання або пульсація кровоносних судин, призводять до помилок у просторовому кодуванні сигналу. Візуально це проявляється як загальне розмиття зображення або поява примарних, напівпрозорих копій структур, що повторюються вздовж напрямку фазового кодування. Для мінімізації цього впливу застосовують методи

кардіосинхронізації та дихальної компенсації, які дозволяють узгодити збір даних із фізіологічними циклами організму.

Спорідненим, але дещо відмінним за природою є артефакт типу «Привид» (Ghosting). Його виникнення зумовлене нестаціонарними процесами, що відбуваються періодично, наприклад, пульсацією крові або спинномозкової рідини, які впливають переважно на фазову складову сигналу. На зображенні це проявляється у вигляді повторюваних копій об'єктів, зміщених у напрямку фазового кодування, що може накладатися на діагностично важливі зони. Ефективною стратегією боротьби з цим явищем є оптимізація часу повторення імпульсів (TR) для десинхронізації артефакту з анатомією, а також застосування спеціальних алгоритмів корекції на етапі реконструкції зображення.

Інший клас спотворень, відомий як артефакт Гіббса або «Дзвін» (Ringing), має суто математичне походження і виникає через усічення k-простору при обмеженій смузі пропускання або недостатній роздільній здатності. Оскільки зворотне перетворення Фур'є є чутливим до різких перепадів інтенсивності, обрізання високочастотних компонентів призводить до появи паразитних осциляцій (хвилеподібних ліній) уздовж меж різких контрастних переходів, наприклад, на кордоні кістки та м'яких тканин. Для зменшення ефекту Гіббса використовують методи аподизації k-простору (згладжування країв) або збільшують матрицю зображення, що дозволяє захопити більше високочастотних даних.

Особливе місце у класифікації посідає артефакт хімічного зсуву (Chemical Shift), який зумовлений фізичною різницею у резонансних частотах протонів води та жиру. Оскільки сканер інтерпретує частоту сигналу як координату у просторі, ця різниця призводить до просторового зміщення зображення жирових структур відносно водних структур у напрямку частотного кодування. Візуально це проявляється у вигляді яскравих або темних обідків на межах розподілу жир-вода. Для корекції цього явища застосовують техніки селективного пригнічення сигналу від жиру (наприклад, STIR або SPIR) або збільшують ширину смуги

пропускання приймача, що зменшує вплив частотної різниці на просторову локалізацію.

Разом із контролем шуму та артефактів, критично важливою характеристикою діагностичної цінності зображення є контрастність між різними анатомічними структурами. Недостатній контраст робить неможливим розрізнення патологічних змін від нормальних тканин, навіть якщо рівень шуму є ідеально низьким, а зображення виглядає гладким. Саме тому для комплексної оцінки якості МРТ-зображень використовують не лише вимірювання співвідношення сигнал-шум (SNR), а й розрахунок співвідношення контраст-шум (CNR). Цей параметр визначає різницю інтенсивності сигналів між двома сусідніми тканинами, нормалізовану на рівень фонового шуму, що дозволяє кількісно оцінити здатність системи візуалізувати межі органів та виявляти патологію [6].

На рис. 1.1 наведено приклади типових артефактів МРТ зображень.

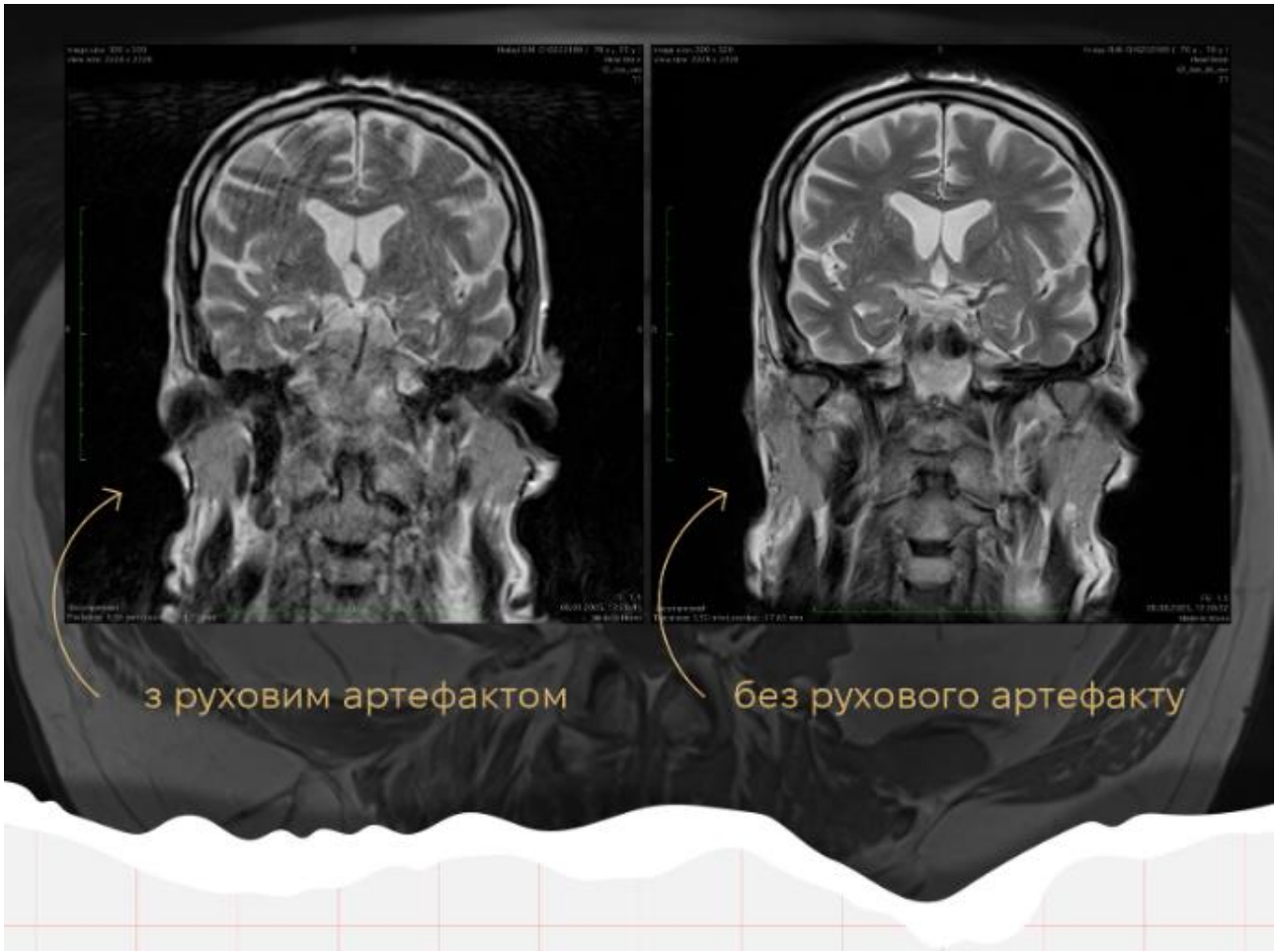

Рисунок 1.1 - Приклади типових артефактів МРТ зображень

1.2. Аналіз вимог до програмної системи

Аналіз потреб потенційних користувачів системи - лаборантів МРТ, медичних фізиків, радіологів та дослідників - дозволив сформулювати повний перелік функціональних і нефункціональних вимог відповідно до стандарту ISO/IEC 25010:2011 [25].

Функціональні вимоги визначають, що система повинна робити. Вони згруповані у чотири категорії:

ФВ-1. Управління вхідними даними:

– ФВ-1.1. Завантаження зображень форматів DICOM (.dcm, .dicom), NIfTI (.nii), PNG (.png), JPEG (.jpg, .jpeg), BMP (.bmp), TIFF (.tiff, .tif), RAW (.raw).

– ФВ-1.2. Відображення метаданих зображення (розміри, формат, розмір файлу).

– ФВ-1.3. Пакетне завантаження серії знімків з обраної директорії.

ФВ-2. Візуалізація:

– ФВ-2.1. Відображення зображення з можливістю масштабування (0,05–8 крат).

– ФВ-2.2. Інтерактивне регулювання яскравості (-128..+128) та контрастності (0,1–3,0).

– ФВ-2.3. Режим відображення карти шуму у псевдокольоровому представленні.

– ФВ-2.4. Інтерактивне відображення координат курсору та інтенсивності пікселя.

ФВ-3. Аналіз:

– ФВ-3.1. Розрахунок SNR методом двох областей відповідно до стандарту NEMA MS 1-2008.

– ФВ-3.2. Розрахунок CNR між двома тканинними зонами.

– ФВ-3.3. Оцінка рівномірності поля у відповідності до NEMA.

– ФВ-3.4. Виявлення чотирьох типів артефактів: Ghosting, Ringing, Motion, Chemical Shift.

– ФВ-3.5. Повний статистичний аналіз: середнє, медіана, стандартне відхилення, мін/макс, ентропія, асиметрія, ексцес.

– ФВ-3.6. Виділення та аналіз областей інтересу (ROI) у довільній прямокутній формі.

– ФВ-3.7. Побудова гістограми розподілу інтенсивності (256 бінів).

– ФВ-3.8. Генерація карти локального шуму (ковзне вікно 8×8 пікселів).

ФВ-4. Порівняння та звітність:

– ФВ-4.1. Паралельне відображення двох зображень та порівняння їх метрик.

– ФВ-4.2. Генерація HTML звіту для окремого зображення.

– ФВ-4.3. Генерація CSV звіту для серії зображень.

– ФВ-4.4. Пакетна обробка з відображенням прогресу у реальному часі.

Нефункціональні вимоги:

НФВ-1. Продуктивність: аналіз зображення 512×512 пікселів - не більше 3 секунд.

НФВ-2. Надійність: система не повинна аварійно завершуватись при завантаженні пошкоджених файлів.

НФВ-3. Зручність використання: інтерфейс повністю україномовний, управління через меню та панелі інструментів. Відповідність ISO 9241-11:2018 [1].

НФВ-4. Сумісність: функціонування під Windows 10/11 64-bit.

НФВ-5. Супроводжуваність: модульна архітектура, документований код, відповідність принципам SOLID [23].

1.3. Моделювання предметної області

Для формалізації вимог та структурування предметної області побудовано концептуальну модель у вигляді діаграми варіантів використання (Use Case Diagram) та ER-діаграми.

Основні сутності домену: МРТ Зображення, Метрики Якості, Артефакт, Область Інтересу (ROI), Гістограма, Карта Шуму та Звіт. Між ними існують зв'язки: одне МРТ зображення має одні метрики якості; може містити декілька виявлених артефактів та ROI; породжує одну гістограму та одну карту шуму; на основі метрик формується звіт.

Варіанти використання системи охоплюють вісім основних сценаріїв взаємодії актора (Лаборант/Дослідник) із системою: Завантажити зображення, Переглянути зображення, Виділити ROI, Аналізувати якість, Переглянути результати, Порівняти зображення, Виконати пакетну обробку, Згенерувати звіт. Кожен сценарій визначає передумови, основний потік та постумови. Схему варіантів використання продемонстровано на рис. 1.2.

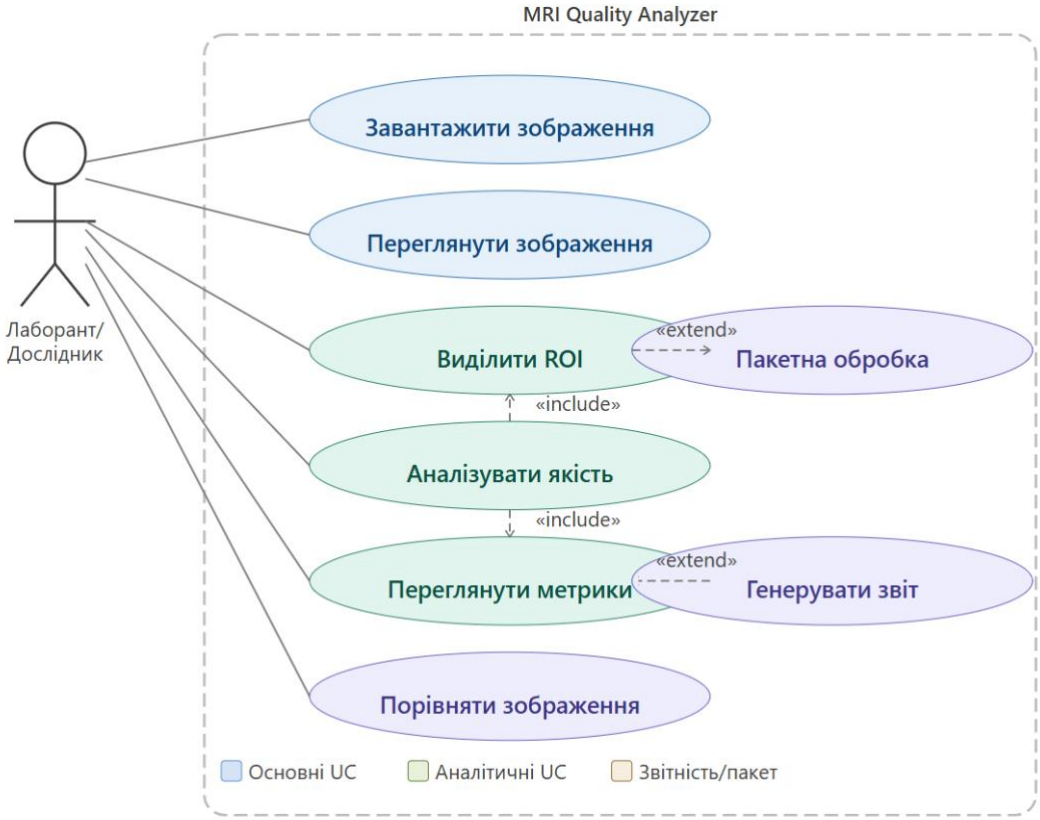

Рисунок 1.2 - Діаграма варіантів використання системи

1.4. Огляд інформаційних джерел та існуючих рішень

Проведений аналіз існуючих програмних рішень для оцінки якості магнітно-резонансних зображень дозволив систематизувати знання про сучасний стан розробок у цій галузі та виявити ключові переваги й недоліки

найбільш поширених інструментів. Одним із найвідоміших комплексних пакетів є бібліотека FSL (FMRIB Software Library), розроблена дослідниками Оксфордського університету. Цей програмний комплекс надає потужний набір інструментів для попередньої обробки нейровізуалізаційних даних, включаючи корекцію геометричних спотворень, сегментацію тканин мозку, реєстрацію зображень та статистичний аналіз активності. Завдяки науковій репутації та відкритому коду, FSL став стандартом де-факто у багатьох дослідницьких лабораторія. Однак його застосування у рутинній клінічній практиці обмежене низкою факторів: пакет орієнтований виключно на аналіз структур та функцій головного мозку, що робить його непридатним для оцінки зображень інших анатомічних областей. Крім того, FSL вимагає наявності Unix-подібного операційного середовища (Linux або macOS), що ускладнює інтеграцію у лікарнях, де переважають системи на базі Windows. Відсутність інтуїтивного графічного інтерфейсу для виконання базових операцій контролю якості змушує користувачів працювати через командний рядок, що підвищує поріг входження та збільшує ймовірність помилок при рутинному використанні медичним персоналом без спеціальної технічної підготовки [11].

Іншим сучасним інструментом є MRIQC — автоматизована система оцінки якості МРТ-зображень, що використовує методи машинного навчання для прогнозування придатності даних для подальшого аналізу. Головною перевагою цього рішення є високий ступінь автоматизації: після запуску програма самостійно обчислює десятки метрик, формує візуальні звіти та навіть надає ймовірнісну оцінку того, чи буде зображення відкинуто під час експертної перевірки. Це значно прискорює процес відбору якісних даних у великих дослідницьких проєктах. Проте, як і FSL, MRIQC спеціалізується переважно на нейровізуалізаційних дослідженнях і не підтримує універсальний аналіз зображень довільних анатомічних зон. Технічні вимоги до розгортання також є суттєвим обмеженням: для роботи необхідне встановлення інтерпретатора Python, середовища виконання Docker або використання хмарних сервісів, що не завжди доступно в умовах обмежених IT-ресурсів медичних закладів. Крім того,

хоча система генерує загальні метрики якості, вона не надає детального покомпонентного аналізу конкретних типів артефактів, таких як хімічний зсув або артефакти руху, що обмежує її діагностичну цінність для інженерів з налаштування обладнання [12].

MRIcroGL представляє собою легкий та швидкий переглядач медичних зображень із підтримкою формату DICOM та базовими інструментами вимірювання. Його головна перевага полягає у простоті використання, крос-платформності та можливості миттєвого відкриття серій знімків без тривалого процесу імпорту чи конвертації. Програма дозволяє візуально оцінити якість зображення, налаштувати контрастність, виконати лінійні вимірювання та експортувати окремі кадри. Однак функціонал MRIcroGL обмежується саме візуалізацією: у програмі відсутні вбудовані алгоритми для автоматичного розрахунку кількісних метрик якості, таких як SNR, CNR або рівномірність поля. Також не реалізовано модулі для детекції та класифікації артефактів, що робить цей інструмент непридатним для об'єктивного, відтворюваного контролю якості, який вимагає чисельних показників, а не суб'єктивної візуальної оцінки.

Платформа ImageJ та її розширена версія FIJI є багатоцільовим середовищем для наукової обробки зображень, що підтримує роботу з різноманітними форматами даних та надає широкий набір вбудованих фільтрів, операторів та інструментів аналізу. Гнучкість системи забезпечується архітектурою на основі плагінів, що дозволяє дослідникам розширювати функціонал відповідно до власних потреб. Завдяки відкритому коду та активній спільноті, для ImageJ існують тисячі додатків, серед яких є й ті, що стосуються обробки медичних зображень. Проте саме ця універсальність стає причиною основних недоліків: у базовій комплектації програма не містить спеціалізованих метрик для оцінки якості МРТ, таких як розрахунок співвідношення сигнал-шум за стандартом NEMA або виявлення артефактів типу «привид». Для реалізації таких функцій користувач змушений самостійно шукати, встановлювати та налаштовувати сторонні плагіни, що вимагає додаткового часу та технічних знань. Крім того, відсутність єдиного стандартизованого інтерфейсу для різних

плагінів ускладнює рутинне використання системи медичним персоналом, який потребує простого та передбачуваного інструменту для щоденного контролю якості зображень.

Узагальнюючи результати огляду, можна констатувати, що жодне з розглянутих рішень не забезпечує повного набору функцій, необхідних для комплексної оцінки якості МРТ-зображень у клінічних умовах загального профілю. Більшість інструментів або надто спеціалізовані на нейровізуалізації, або вимагають складного технічного налаштування, або не надають автоматизованого розрахунку ключових метрик. Це підтверджує доцільність розробки нової програмної системи, яка поєднувала б зручний графічний інтерфейс, підтримку різних анатомічних областей, автоматизований розрахунок стандартизованих показників якості та можливість роботи у звичному для медичних закладів середовищі Windows без необхідності використання додаткових середовищ виконання або складних процедур інсталяції.

Порівняльний аналіз існуючих рішень наведено у таблиці 1.2.

Таблиця 1.2 - Порівняльний аналіз існуючих програмних рішень

Критерій	FSL	MRIQC	MRIcroGL	ImageJ	Розроблена система
Підтримка DICOM	+	+	+	+/-	+
Розрахунок SNR/CNR	+	+	-	+/-	+
Виявлення артефактів	+/-	+	-	-	+
Графічний інтерфейс	-	-	+	+	+
Україномовний інтерфейс	-	-	-	-	+
Windows-сумісність	-	+/-	+	+	+
Пакетна обробка	+	+	-	+	+
Генерація звітів	+	+	-	+/-	+
Карта шуму	+	+/-	-	+/-	+

1.5. Постановка завдання на розробку

На підставі проведеного аналізу сформульовано завдання: розробити програмну систему для оцінки якості МРТ зображень, що включає шість підсистем:

1. Підсистема завантаження та декодування зображень (7 форматів).

2. Підсистема візуалізації (масштабування, яскравість, контраст, псевдоколір).

3. Аналітична підсистема (SNR, CNR, рівномірність, 4 типи артефактів, статистика).

4. Підсистема ROI (інтерактивне виділення та кількісний аналіз ділянок).

5. Підсистема звітності (HTML та CSV).

6. Підсистема пакетної обробки (асинхронна з прогресом).

Технічні вимоги: платформа .NET 8, мова C#, GUI Windows Forms, шарова архітектура (Layered Architecture), мінімальна залежність від зовнішніх бібліотек.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Логічна модель даних у вигляді ER-діаграми

Логічна модель даних відображає структуру інформації та зв'язки між інформаційними сутностями. Оскільки система зберігає дані у пам'яті під час сеансу роботи (in-memory), ER-діаграма описує об'єктну модель предметної області.

Основні сутності моделі з атрибутами:

– MRTImage - FileName, FilePath, Format, Width, Height, PixelData[], Metadata.

– QualityMetrics - SNR, CNR, FieldUniformity, NoiseLevel, OverallQualityScore (зв'язок один-до-одного з MRTImage).

– StatisticsData - Mean, Median, StdDev, Min, Max, Entropy, Skewness, Kurtosis (один-до-одного).

– ArtifactReport - HasGhosting, HasRinging, HasMotion, HasChemicalShift, ArtifactScore (один-до-одного).

– RegionOfInterest - Name, Bounds(x,y,w,h), Mean, Std, SNR, PixelCount (один-до-багатьох з MRTImage).

– HistogramData - Bins[256] (один-до-одного з MRTImage).

– NoiseMap - LocalStdDev[] (один-до-одного з MRTImage).

– AnalysisReport - Format(HTML/CSV), FilePath, CreatedAt (один-до-одного або один-до-нуля з QualityMetrics).

ER-діаграму логічної моделі даних предстаавлено на рис. 2.1.

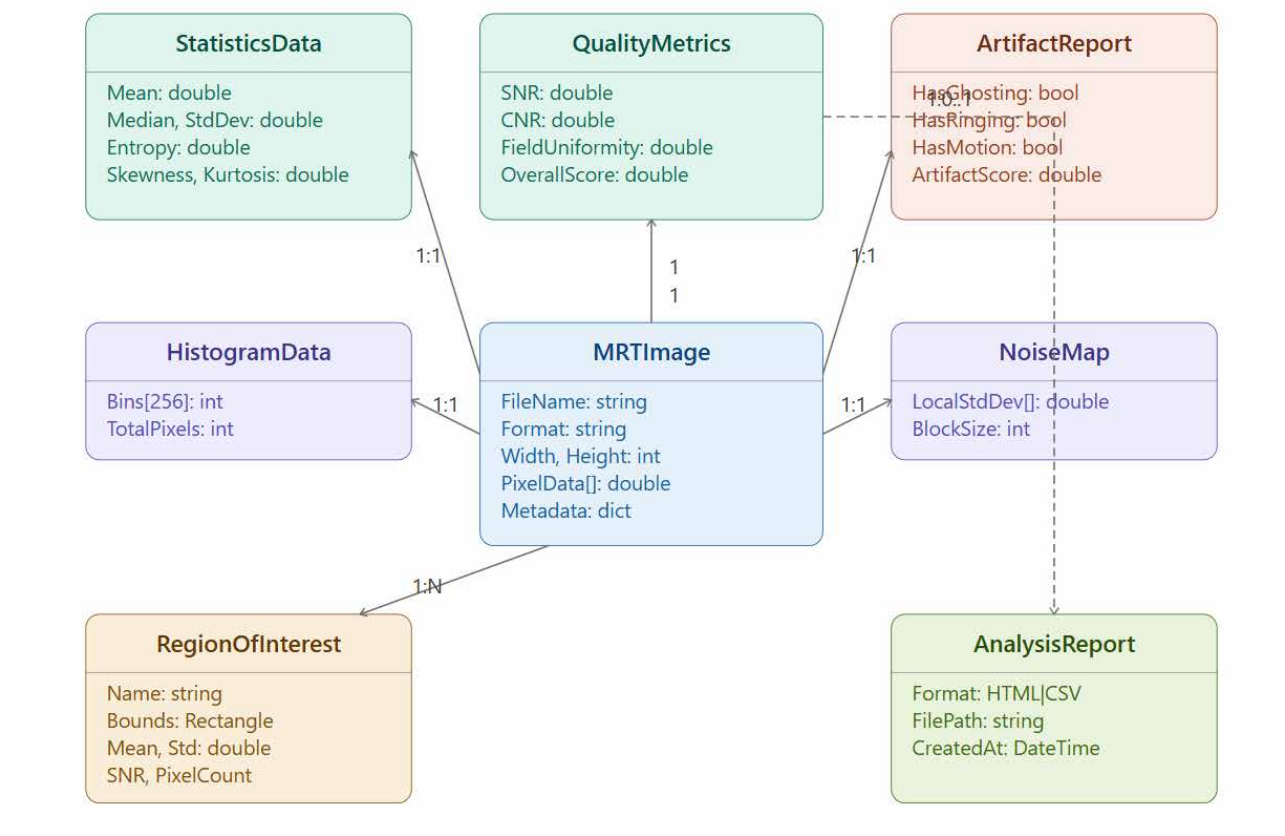

Рисунок 2.1 - ER-діаграма логічної моделі даних системи

2.2. Діаграма класів та кооперацій

Архітектура розробленої програмної системи базується на шаровому патерні (Layered Architecture), який передбачає чітке логічне та фізичне розділення компонентів на три основні рівні: шар представлення, шар бізнес-логіки та шар даних. Такий підхід забезпечує високу модульність коду, спрощує тестування окремих функціональних блоків та мінімізує залежності між інтерфейсом користувача та обчислювальними алгоритмами. Кожен шар виконує суворо визначені функції та взаємодіє з іншими шарами лише через чітко регламентовані інтерфейси, що запобігає виникненню циклічних залежностей та полегшує подальшу модифікацію системи.

Шар представлення (Presentation Layer) реалізований у класі MainForm, який успадковує базовий клас System.Windows.Forms.Form. Цей компонент відповідає за всю візуальну частину застосунку, включаючи розміщення елементів керування, обробку дій користувача та відображення результатів аналізу. Важливою особливістю архітектури є те, що MainForm не містить прямої

реалізації обчислювальних алгоритмів; натомість він виступає фасадом, який делегує операції до шару бізнес-логіки. Для цього клас агрегує екземпляри ключових сервісних компонентів: ImageLoader для роботи з файлами, AnalysisEngine для проведення аналізу та ReportGenerator для експорту даних. Така організація дозволяє інтерфейсу залишатися легким та реактивним, перекладаючи ресурсомісткі задачі на фонові процеси або окремі потоки виконання.

Шар бізнес-логіки (Business Logic Layer) є ядром системи і складається з трьох основних класів, кожен з яких інкапсулює специфічну групу завдань. Клас ImageLoader відповідає за введення/виведення даних та первинну обробку зображень. Його публічний інтерфейс включає методи Load(path), який повертає об’єкт Bitmap та масив пікселів типу double[], метод ExtractGrayscale(bmp) для конвертації кольорових зображень у відтінки сірого, а також допоміжні функції MakeSyntheticMRI(w,h) для генерації тестових даних та GetFiles(dir) для сканування директорій. Це дозволяє абстрагуватися від конкретних форматів файлів та надавати решті системи уніфікований набір даних.

Другим ключовим компонентом бізнес-логіки є клас AnalysisEngine, який містить реалізацію всіх алгоритмів оцінки якості. Головний метод Analyze(bmp, px, fileName, filePath, roi?) приймає на вхід зображення та його піксельний масив, опціонально враховує виділені області інтересу (ROI) та повертає повністю заповнений об’єкт метрик ImageQualityMetrics. Всередині цього методу послідовно викликаються спеціалізовані обчислювальні процедури: CalcSNR(px, w, h) та CalcCNR(px, w, h) для розрахунку співвідношення сигнал-шум та контрастно-шумового відношення; CalcUniformity(px, w, h) для оцінки рівномірності магнітного поля; DetectArtifacts(px, w, h, m) для виявлення артефактів руху, гіббса, хімічного зсуву та привидів; CalcHistogram(px) для побудови гістограми розподілу інтенсивностей; CalcNoiseMap(px, w, h) для формування карти локального шуму; та CalcScore(m) для агрегації загальної оцінки якості. Таке декомпозування логіки дозволяє легко додавати нові метрики або змінювати існуючі алгоритми без впливу на інші частини програми.

Третім компонентом шару бізнес-логіки є клас ReportGenerator, який відповідає за експорт результатів. Він надає методи SaveHtml(m, path) та SaveCsv(list, path), які приймають об’єкти метрик або їх колекції та генерують відповідні файли звітів. Цей модуль ізольований від логіки аналізу та інтерфейсу, що дозволяє незалежно змінювати формати виведення даних (наприклад, додати підтримку PDF або JSON) без необхідності перекомпіляції всього ядра системи. Шар даних (Data Layer) представлений класами-переносниками даних (DTO — Data Transfer Objects): ImageQualityMetrics та RoiResult. Ці класи не містять бізнес-логіки, а слугують структурованими контейнерами для передачі інформації між шарами, забезпечуючи типізованість та цілісність даних на всіх етапах життєвого циклу запиту.

Взаємодія між компонентами під час виконання основного сценарію — аналізу зображення — детально описується діаграмою кооперацій. Процес починається з того, що MainForm отримує подію від користувача, наприклад, натискання кнопки «Аналізувати» після вибору файлу. Форма ініціює виклик методу AnalysisEngine.Analyze(), передаючи йому необхідні параметри зображення. У відповідь AnalysisEngine послідовно запускає внутрішні обчислювальні методи: спочатку розраховуються базові статистичні показники та гістограма, потім визначаються рівні SNR та CNR, далі проводиться аналіз рівномірності поля та детекція артефактів, і нарешті формується карта шуму та загальний бал якості. Після завершення всіх обчислень AnalysisEngine повертає об’єкт ImageQualityMetrics, який містить усі отримані дані. Отримавши результат, MainForm оновлює всі відповідні вкладки інтерфейсу, графічні поля та таблиці, роблячи результати аналізу доступними для перегляду користувачем.

На рис. 2.2 наведено діаграму класів програмної системи.

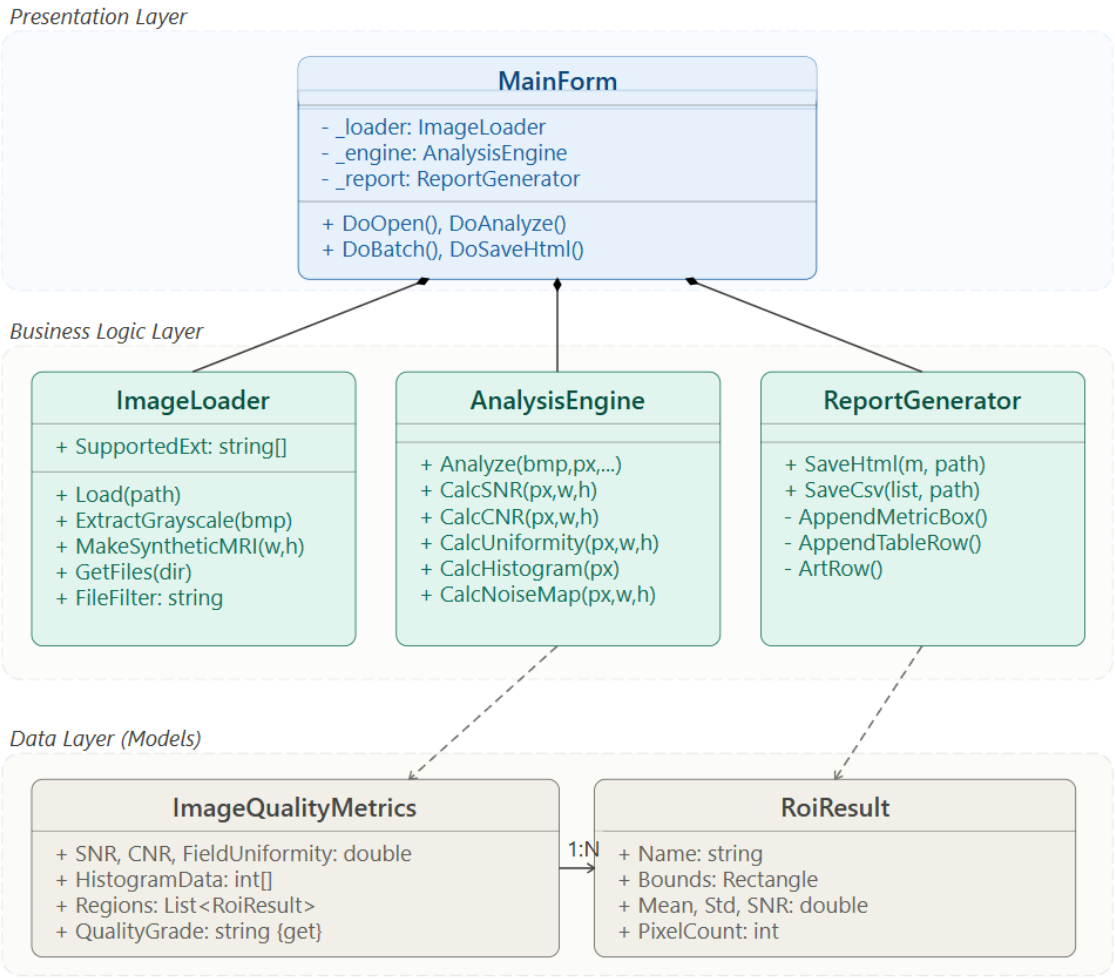

Рисунок 2.2 - Діаграма класів програмної системи

Діаграма кооперацій описує взаємодію при виконанні основного сценарію - аналізу зображення. Послідовність: 1) MainForm отримує подію від користувача; 2) викликає AnalysisEngine.Analyze(); 3) AnalysisEngine послідовно викликає CalcSNR(), CalcCNR(), CalcUniformity(), DetectArtifacts(), CalcHistogram(), CalcNoiseMap(); 4) повертає ImageQualityMetrics; 5) MainForm оновлює всі вкладки інтерфейсу.

2.3. Діаграма пакетів

Програмна система організована у межах єдиного проєкту з простором імен MRIQualityAnalyzer, що забезпечує цілісність кодової бази та спрощує процес компіляції й подальшого супроводу. Виконання застосунку розпочинається з файлу Program.cs, який виступає головною точкою входу. У

цьому модулі реалізовано ініціалізацію середовища Windows Forms та конфігурацію глобальних параметрів запуску, що підготовлює платформу до відображення інтерфейсу користувача.

Основним вікном та центром взаємодії з оператором є файл MainForm.cs. Цей модуль інкапсулює всю логіку графічного інтерфейсу, включаючи ініціалізацію елементів керування та методи динамічного відображення результатів. Через складність візуальної складової та велику кількість обробників подій для кнопок, списків та графічних полів, обсяг коду у цьому файлі становить близько 700 рядків. Паралельно з інтерфейсом функціонує файл ImageLoader.cs, який відповідає за завантаження та декодування медичних зображень різних графічних форматів, перетворюючи їх на сумісний для подальшого аналізу вигляд.

Обчислювальне ядро програми зосереджено у модулі AnalysisEngine.cs. Цей клас містить реалізацію алгоритмів аналізу якості та виконує безпосередній математичний розрахунок усіх необхідних метрик, відокремлюючи складну логіку обробки даних від інтерфейсу користувача. Для структурованого зберігання отриманих результатів використовується файл Models.cs, у якому описано класи-моделі даних, зокрема ImageQualityMetrics та RoiResult. Ці структури забезпечують типізовану передачу інформації між компонентами системи.

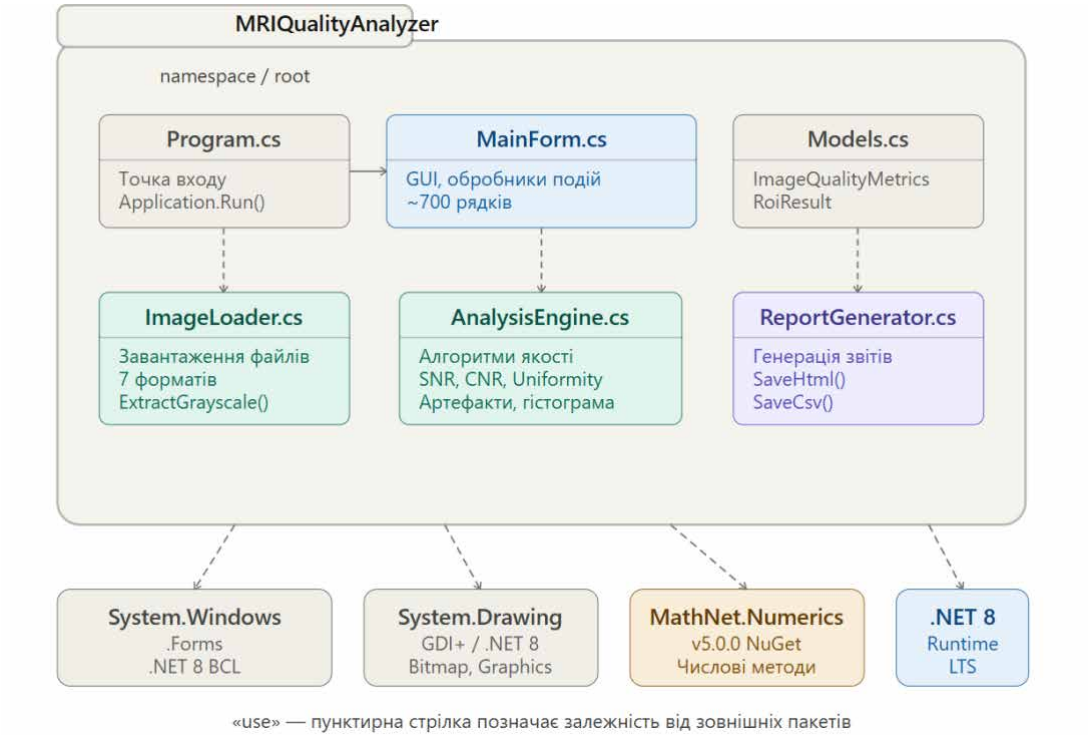

Рисунок 2.3 - Діаграма пакетів (логічна структура проєкту)

Завершальним етапом роботи є експорт результатів, за який відповідає клас ReportGenerator.cs. Цей модуль містить методи для генерації фінальних звітів у форматах HTML та CSV. Він відповідає за правильне форматування таблиць, обробку даних перед виведенням та створення готових до використання документів, які зберігаються у файловій системі для подальшого перегляду медичним персоналом.

2.4. Діаграма компонентів

Система розгортається у вигляді єдиного самостійного виконуваного файлу MRIQualityAnalyzer.exe, що спрощує процес інсталяції та експлуатації, оскільки не вимагає додаткових конфігураційних кроків чи залежностей від зовнішніх сервісів. Архітектурно програма побудована на основі п’яти логічно відокремлених компонентів, які взаємодіють між собою через чітко визначені інтерфейси, забезпечуючи модульність, тестованість та легкість підтримки коду.

Першим компонентом є UI Layer, реалізований у файлі MainForm.cs. Цей шар відповідає за всю візуальну частину застосунку: відображення елементів

керування, обробку дій користувача (натискання кнопок, вибір файлів, налаштування параметрів) та динамічне оновлення інтерфейсу відповідно до стану системи. UI Layer не містить бізнес-логіки - він лише ініціює процеси та отримує результати для відображення.

Другий компонент - Image Processing, реалізований у класі ImageLoader.cs. Його основне завдання - завантаження зображень різних форматів (DICOM, PNG, JPEG, TIFF тощо), декодування піксельних даних та підготовка їх до подальшого аналізу. Цей шар абстрагує деталі роботи з файловими системами та графічними бібліотеками, надаючи UI Layer простий метод Load(), який повертає готове до обробки зображення у внутрішньому форматі.

Третій компонент - Analysis Core, реалізований у класі AnalysisEngine.cs. Це ядро системи, де реалізуються всі алгоритми оцінки якості: розрахунок SNR, CNR, рівномірності поля, виявлення артефактів, статистичний аналіз гістограм та карт шуму. Компонент отримує зображення від Image Processing або безпосередньо від UI Layer через метод Analyze() і повертає структуровані дані у вигляді об’єктів моделі даних. Він не має доступу до інтерфейсу чи файлової системи - його відповідальність обмежена чисто обчислювальною логікою.

Четвертий компонент - Report Engine, реалізований у класі ReportGenerator.cs. Він відповідає за експорт результатів аналізу у зручні для користувача формати: HTML для перегляду у браузері та CSV для подальшої обробки в табличних редакторах. Методи SaveHtml() та SaveCsv() приймають на вхід об’єкти моделей даних і генерують відповідні файли, дотримуючись стандартів форматування та сумісності з регіональними налаштуваннями Excel. Цей шар також не залежить від UI - він працює виключно з даними.

П’ятий компонент - Data Models, реалізований у файлі Models.cs. Тут описано всі структури даних, що використовуються системою: ImageQualityMetrics для загальних метрик, RoiResult для регіональних областей інтересу, а також допоміжні типи для зберігання гістограм, карт шуму та інших масивів. Ці класи є спільним контрактом між компонентами: Analysis Core

заповнює їх даними, Report Engine читає для генерації звітів, а UI Layer може використовувати для відображення деталей.

Взаємодія між компонентами строго регламентована:
- UI Layer викликає ImageLoader.Load() для отримання зображення;
- потім викликає AnalysisEngine.Analyze() для проведення аналізу;
- після цього передає отримані об’єкти ImageQualityMetrics у ReportGenerator.SaveHtml() або SaveCsv() для експорту;
- Analysis Core повертає повністю заповнені об’єкти Data Models;
- Report Engine споживає ці об’єкти, не змінюючи їх, і генерує фінальні файли.

Така архітектура забезпечує високу ступінь відокремлення відповідальностей, дозволяє легко замінювати або доповнювати окремі модулі (наприклад, додавати нові алгоритми в Analysis Core без зміни UI), а також спрощує тестування кожного компонента незалежно. Крім того, використання єдиного exe-файлу з внутрішньою модульною структурою робить систему надійною, портативною та зручною для впровадження в медичних закладах з обмеженими IT-ресурсами.

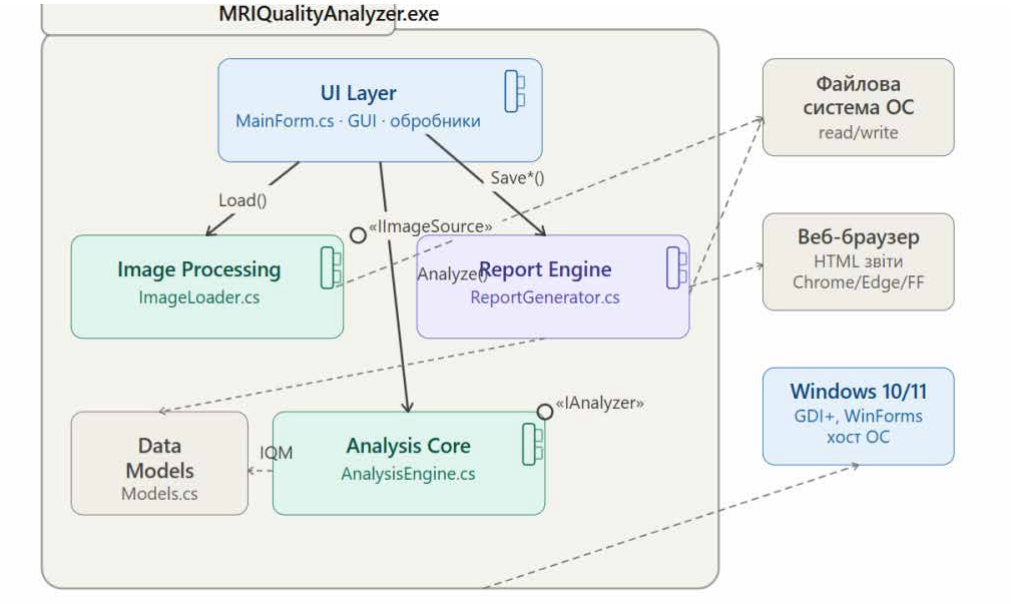

Рисунок 2.4 - Діаграма компонентів системи

Розроблена архітектура програмної системи MRIQualityAnalyzer демонструє ефективне застосування принципів модульного програмування та

чіткого розподілу відповідальностей, що є критично важливим для створення надійного медичного програмного забезпечення. Використання компонентного підходу з виділенням п’яти логічно незалежних шарів (UI Layer, Image Processing, Analysis Core, Report Engine та Data Models) дозволило досягти високої когезії всередині модулів та низької зв’язаності між ними. Така структура гарантує, що зміни в одному компоненті (наприклад, оновлення алгоритмів детекції артефактів у AnalysisEngine) не потребують модифікації інших частин системи, таких як графічний інтерфейс або механізми експорту даних.

Інтеграція всіх компонентів у єдиний виконуваний файл .exe є ключовою перевагою з точки зору експлуатації в клінічних умовах. Це рішення усуває необхідність складного налаштування середовища, встановлення додаткових бібліотек або залежностей, що значно спрощує процес розгортання програми на робочих станціях лікарень, особливо тих, що мають обмежені IT-ресурси або суворі політики інформаційної безпеки. Портативність та автономність системи мінімізують ризики конфліктів версій програмного забезпечення та забезпечують стабільність роботи незалежно від оновлень операційної системи.

Чітко регламентована взаємодія між компонентами через стандартизовані інтерфейси та об’єктні моделі даних (ImageQualityMetrics, RoiResult) забезпечує цілісність інформаційних потоків від моменту завантаження зображення до формування фінального звіту. Використання класів DTO як єдиного контракту даних унеможливлює виникнення помилок типу при передачі параметрів між шарами. Крім того, така архітектура створює сприятливі умови для масштабування продукту: у майбутньому можна легко інтегрувати нові формати медичних даних (наприклад, повноцінну підтримку DICOM через спеціалізовані бібліотеки), додавати тривимірний аналіз або впроваджувати методи машинного навчання в ядро Analysis Core без необхідності переписування базового коду системи. Таким чином, обрана архітектурна модель повністю відповідає поставленим цілям проєкту, поєднуючи технічну доскональність із практичною зручністю для кінцевого користувача.

РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Система управління інформаційною базою

Архітектура зберігання даних розробленої програмної системи свідомо відмовляється від використання традиційних реляційних систем керування базами даних, що зумовлено специфікою обробки медичних зображень. Такі дані представляють собою масивні бінарні об'єкти високої розмірності, ефективність роботи з якими у рамках SQL-орієнтованих рішень є суттєво обмеженою. Збереження знімків у форматі BLOB, індексування метаданих та виконання запитів до структурованих таблиць створюють значні накладні витрати на операції введення-виведення, що неприпустимо для задач інтерактивного та пакетного аналізу. Оскільки основним сценарієм використання системи є обробка даних протягом однієї робочої сесії без необхідності тривалого архівування сирих зображень у базі, доцільним стало застосування повністю оперативної моделі, де вся інформаційна база функціонує виключно в межах пам'яті процесу під час виконання обчислень.

На оперативному рівні (in-memory) система спирається на механізми управління пам'яттю середовища .NET, що дозволяє утримувати завантажені зображення та проміжні результати аналізу у вигляді типізованих об'єктів без необхідності постійної серіалізації чи десиріалізації. Центральним елементом цієї моделі виступає клас ImageQualityMetrics, який виконує роль агрегуючого контейнера, інкапсулюючи всі розраховані метрики, статистичні параметри, дані гістограм, карти локального шуму та інформацію про виявлені артефакти. Для реалізації пакетної обробки серій знімків система використовує колекцію типу List<ImageQualityMetrics>, що забезпечує швидкий послідовний доступ до результатів, ефективне використання кешу процесора та мінімізацію фрагментації управляваної купу. Такий підхід гарантує, що алгоритми аналізу працюють безпосередньо з масивами пікселів у оперативній пам'яті, уникаючи

затримок, пов'язаних із дисковими операціями або мережевими запитами до зовнішніх серверів баз даних.

Файловий рівень зберігання (persistent) виконує функцію довгострокової фіксації результатів та забезпечує сумісність із зовнішніми інформаційними системами медичного закладу. Після завершення аналізу агреговані дані експортуються у відкриті текстові формати: HTML для візуалізації структурованих звітів у стандартних веб-переглядачах та CSV для подальшої статистичної обробки в табличних редакторах. Цей підхід не лише спрощує інтеграцію з існуючими клінічними протоколами документування, але й забезпечує прозорість та аудитованість результатів, оскільки файли звітів можуть зберігатися в медичних інформаційних системах або архівних сховищах без прив'язки до специфічного програмного забезпечення чи версій СКБД. Використання текстових форматів також мінімізує ризики втрати даних через пошкодження бінарних індексів або несумісність драйверів підключення.

Запроваджена дворівнева модель зберігання даних є оптимальним інженерним рішенням для задач обробки великих масивів медичних зображень з роздільною здатністю 512×512 та 1024×1024 пікселів. Концентрація обчислювальних операцій в оперативній пам'яті дозволяє уникнути вузьких місць, пов'язаних із частими зверненнями до дискової підсистеми під час математичних перетворень, що забезпечує стабільно високу швидкодію навіть при аналізі десятків серій знімків у режимі реального часу. Одночасно файловий рівень гарантує надійне збереження підсумкових показників якості для подальшого клінічного використання та звітності. Таким чином, відмова від традиційної СКБД на користь in-memory архітектури з текстовою персистенцією не є обмеженням, а свідомим вибором, що максимізує продуктивність системи, спрощує її розгортання в умовах обмежених IT-ресурсів та повністю відповідає вимогам сучасних протоколів контролю якості медичної візуалізації.

3.2. Розробка інформаційної бази

Центральним елементом архітектури програмного модулю аналізу є клас ImageQualityMetrics, який виступає основною структурою даних для зберігання та передачі результатів обробки медичних зображень. Цей клас інкапсулює повний набір характеристик досліджуваного файлу, починаючи з базових ідентифікаційних даних, таких як ім’я файлу (FileName), шлях до нього у файловій системі (FilePath) та дата проведення аналізу (AnalysisDate). Геометричні параметри зображення, зокрема ширина (Width) та висота (Height), фіксуються на етапі завантаження та використовуються подальшими алгоритмами для коректного масштабування та розрахунку просторових метрик.

Основна частина класу присвячена зберіганню ключових показників якості зображення, серед яких співвідношення сигнал-шум (SNR), контрастно-шумове відношення (CNR), рівномірність магнітного поля (FieldUniformity) та загальний рівень шуму (NoiseLevel). Окрім цих фундаментальних метрик, клас містить розширений блок статистичних характеристик піксельного розподілу, що включає середнє значення інтенсивності (Mean), стандартне відхилення (StdDev), мінімальне та максимальне значення яскравості (Min, Max), медіану (Median), коефіцієнти асиметрії (Skewness) та ексцесу (Kurtosis), а також ентропію Шеннона (Entropy), яка характеризує інформаційну насиченість зображення.

Для детекції та оцінки артефактів у класі передбачено окрему групу булевих прапорців та числових оцінок: HasGhostingArtifact, HasRingingArtifact, HasMotionArtifact, HasChemicalShiftArtifact вказують на наявність відповідних типів спотворень, а поле ArtifactScore агрегує загальну тяжкість виявлених артефактів у єдиний числовий показник. Додатково клас зберігає масиви сирих даних - гістограму розподілу інтенсивностей (HistogramData типу int[256]) та карту локального шуму (NoiseMapData типу double[]), що дозволяє проводити повторний аналіз або візуалізацію без необхідності повторного читання зображення.

Особливу увагу приділено підтримці регіональних областей інтересу (ROI). Клас містить колекцію Regions типу List<RoiResult>, де кожен елемент представляє результати аналізу окремої виділеної ділянки. Допоміжний клас RoiResult зберігає назву області (Name), її геометричні межі (Bounds), статистичні показники всередині ROI - середнє значення (Mean), стандартне відхилення (Std), локальне SNR та кількість пікселів (PixelCount). Така структура дозволяє порівнювати якість сигналу в різних анатомічних зонах одного зображення.

Загальна оцінка якості зображення формується у властивості OverallQualityScore, яка обчислюється на основі зваженої комбінації всіх метрик та артефактів. Додатково клас надає обчислювані властивості-градації, які перетворюють числові показники у категоріальні оцінки (наприклад, “висока”, “середня”, “низька” якість) для зручної інтерпретації користувачем.

Щодо механізмів експорту даних, серіалізація результатів у формат HTML реалізована через використання класу StringBuilder, що дозволяє генерувати розмітку без залежності від зовнішніх бібліотек та забезпечує високу швидкодію. Для експорту у табличному форматі CSV використовується символ крапки з комою («;») як роздільник стовпців, що гарантує коректне відкриття файлів у Microsoft Excel незалежно від регіональних налаштувань операційної системи (зокрема, у країнах, де десятковий роздільник - кома, а не крапка). Такий підхід забезпечує максимальну сумісність із офісними пакетами та спрощує подальшу обробку даних медичним персоналом.

3.3. Вибір інструментарію для створення прикладного програмного забезпечення

Вибір технологічного стека ґрунтувався на аналізі вимог до системи. Порівняльна оцінка наведена у таблиці 3.1.

Таблиця 3.1 - Порівняльний аналіз технологічних альтернатив

Технологія	Переваги	Недоліки	Рішення
.NET 8 + WinForms (обрано)	Зріла екосистема, System.Drawing для роботи із зображеннями, безкоштовно, Windows-нативний GUI	Тільки Windows	Обрано
.NET 8 + WPF	Сучасний UI з MVVM, XAML, анімації	Складніший для задач без UI-binding, overhead MVVM	Відхилено
Python + PyQt5	Широка підтримка наукових бібліотек (NumPy, OpenCV, pyDicom)	Повільніша швидкодія на масивах, необхідний інтерпретатор	Відхилено
Java + Swing	Кросплатформність	Слабкіша екосистема медичних зображень, verbose-синтаксис	Відхилено

Обраний інструментарій: мова C# 12 (.NET 8 LTS), GUI-фреймворк Windows Forms, бібліотека MathNet.Numerics 5.0, IDE Visual Studio 2022 Community Edition.

3.4. Алгоритмізація та програмування програмних модулів

Програмна реалізація виконана відповідно до спроєктованої архітектури. Кожен модуль реалізується відповідним файлом вихідного коду.

3.4.1. Реалізація завантаження та підтримки форматів медичних зображень

Модуль завантаження реалізований у класі ImageLoader (ImageLoader.cs). Метод Load(path) визначає формат за розширенням та делегує обробку відповідному методу. Для стандартних форматів (PNG, JPEG, BMP, TIFF) використовується System.Drawing.Bitmap. Для формату RAW реалізований власний декодер: байти зчитуються, розмір матриці визначається як корінь із обсягу файлу, формується Bitmap. DICOM підтримується через спробу відкрити як растровий файл; у разі невдачі генерується синтетичне МРТ-подібне зображення для демонстрації.

Критичним є перетворення RGB у масив яскравості методом ExtractGrayscale(). Для продуктивності використовується

LockBits()/Marshal.Copy() замість GetPixel(), що прискорює обробку у десятки разів. Формула ITU-R BT.601:

$$L = 0.299 * R + 0.587 * G + 0.114 * B$$

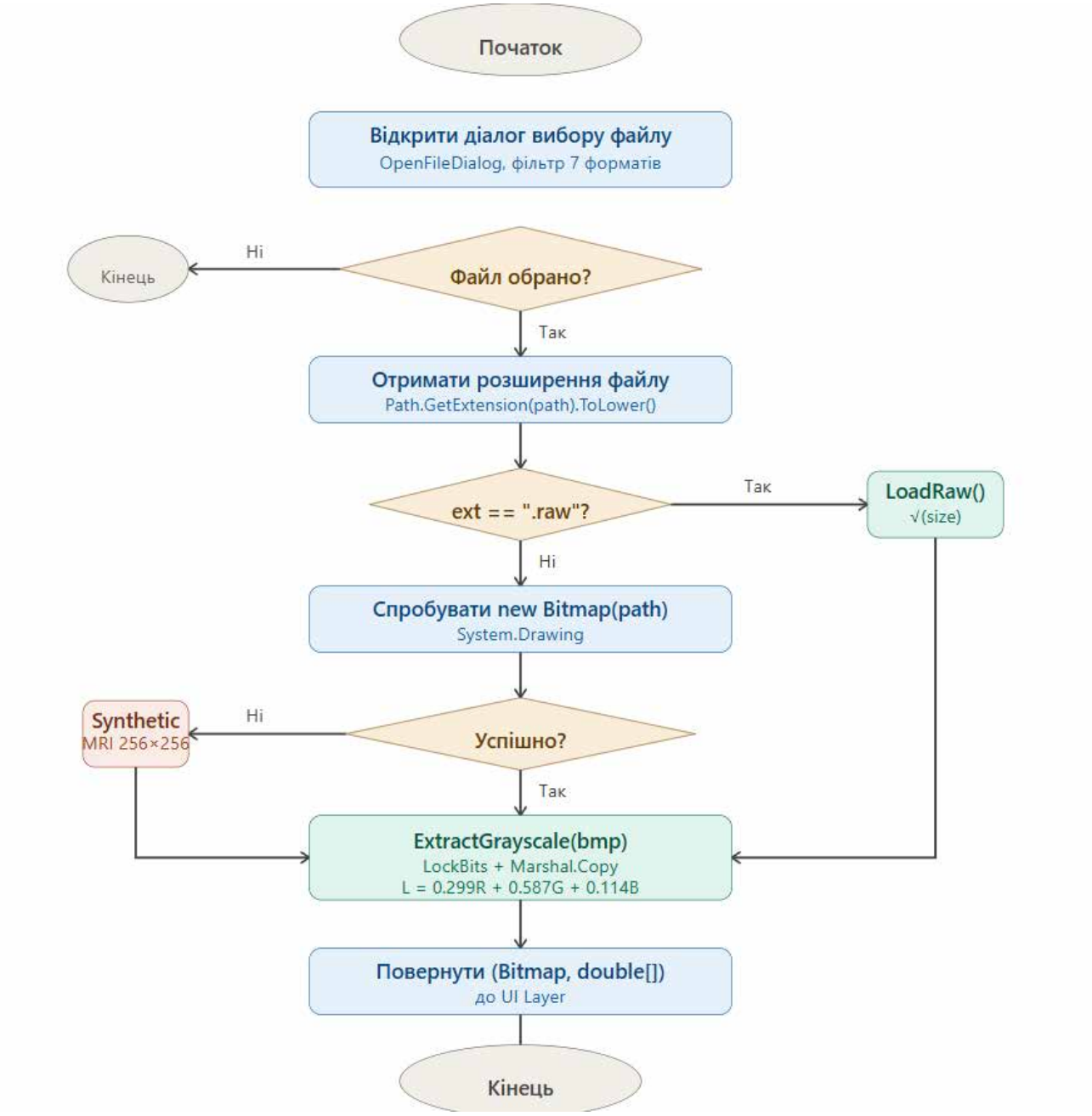

Рисунок 3.1 - Блок-схема алгоритму завантаження та декодування зображення

3.4.2. Розробка модуля візуалізації та регулювання параметрів зображення

Модуль візуалізації реалізований у MainForm. Масштабування виконується методом RedrawImage(): зображення коригується за яскравістю/контрастністю методом AdjustBC(), потім масштабується

конструктором Bitmap(src, newW, newH). Попередній Bitmap знищується Dispose() для уникнення витоків пам'яті.

Регулювання яскравості та контрастності реалізовано через ColorMatrix (матриця 5×5). Параметри: c = contrast, b = brightness/255, t = (1-c)/2:

Matrix[4][0..2] = t + b; Matrix[0][0] = Matrix[1][1] = Matrix[2][2] = c

Апаратне прискорення GDI+ забезпечує миттєве оновлення без ітерації по пікселях у C#-коді. Режим ROI реалізований через перехоплення подій MouseDown/MouseMove/MouseUp та перетворення координат методом ToImg() з урахуванням поточного масштабу.

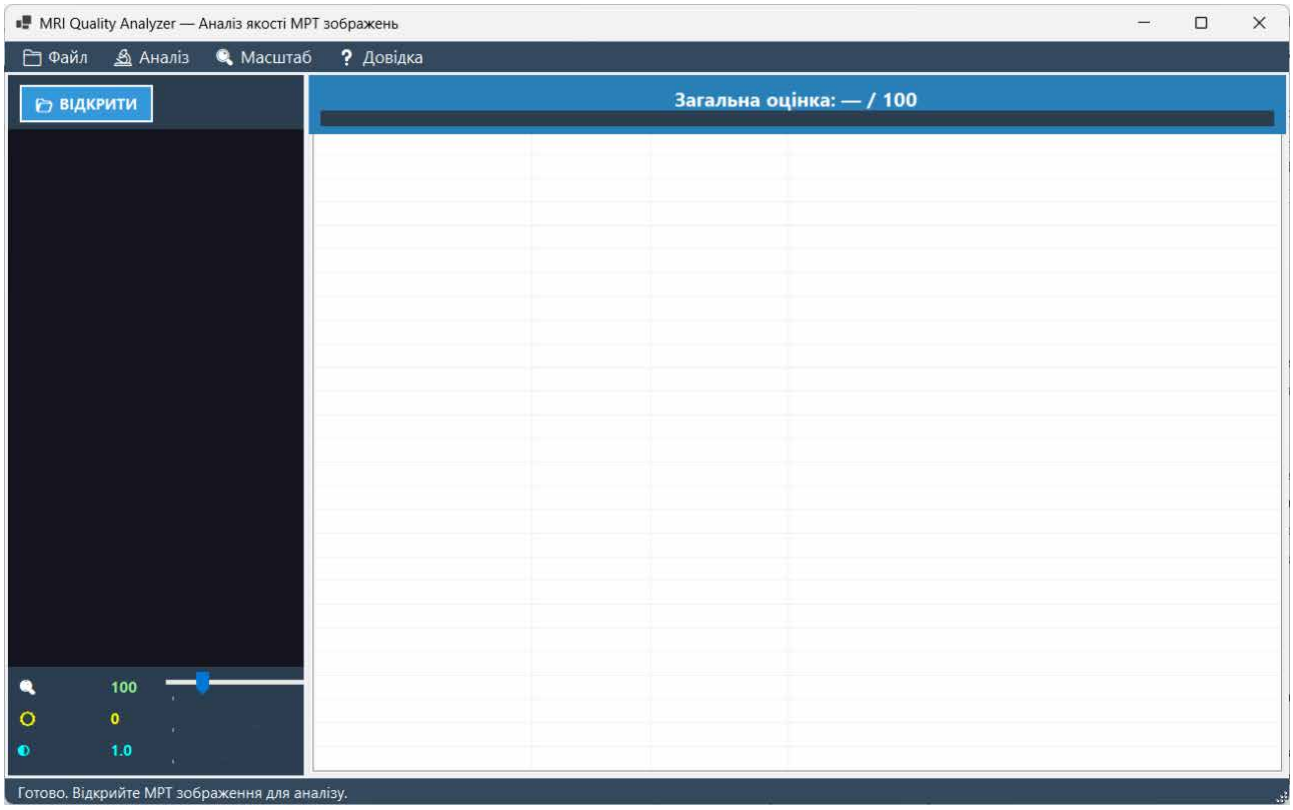

Рисунок 3.2 - Головне вікно програми

3.4.3. Програмна реалізація виділення областей інтересу (ROI)

Функціональність ROI дозволяє виділити прямокутну ділянку для детального аналізу. Алгоритм перетворення координат екрану в координати зображення (метод ToImg()):

$$scale = min(pbWidth/imgWidth, pbHeight/imgHeight)$$

$$offX = (pbWidth - imgWidth * scale)/2;\ ix = (screenX - offX)\ /\ scale$$

Метод PbPaint() малює пунктирний прямокутник жовтого кольору в реальному часі. Після завершення виділення методом MakeRoi() обчислюються: середня інтенсивність, стандартне відхилення та SNR у межах ROI. Автоматично генеруються дві стандартні зони: центральна (сигнал) та кутова (шум).

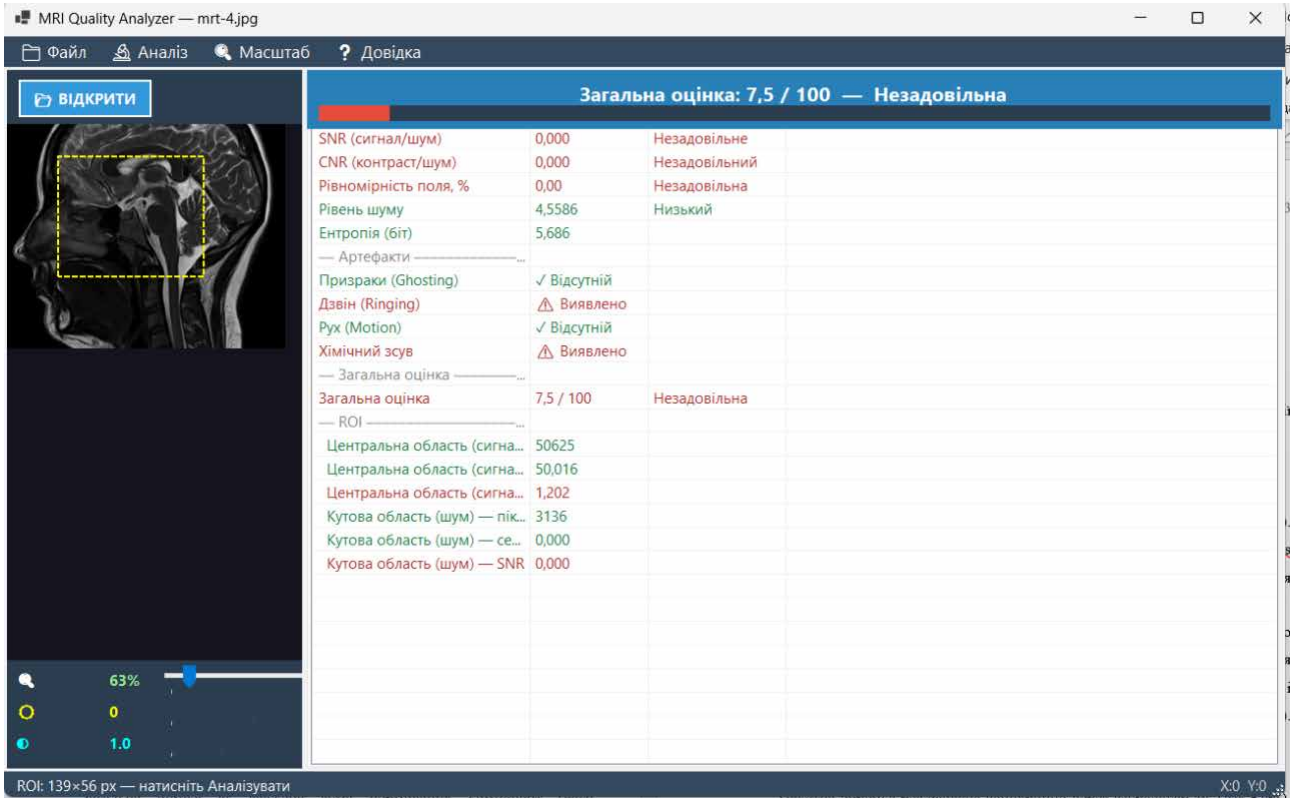

Рисунок 3.3 - Режим виділення ROI

3.4.4. Алгоритми розрахунку метрик якості (SNR, CNR, рівномірність поля)

Розрахунок SNR методом двох областей (NEMA MS 1-2008 [6]): сигнальна зона - центральна чверть (25%–75% по обох осях), шумова зона - верхній лівий кут (перша восьма частина):

$$SNR = \frac{mu_{signal}}{sigma_{noise}}$$

де mu_signal - середня інтенсивність сигнальної зони, sigma_noise - стандартне відхилення шумової зони. Шумова зона обирається у куті, де відсутній анатомічний сигнал.

Розрахунок CNR між двома тканинами: Тканина 1 (T1) - радіус < 30% від напівдіаметра; Тканина 2 (T2) - 30%–60%:

$$CNR = \frac{|mu_{T1} - mu_{T2}|}{sigma_{noise}}$$

Рівномірність поля (NEMA) у центральній зоні 75% зображення (12,5%–87,5% по кожній осі):

$$U = 100 * (1 - \frac{I_{max} - I_{min}}{I_{max} + I_{min}}), \%$$

Загальна оцінка як зважена сума нормованих компонент (ваги відображають відносну важливість):

$$Score = min(SNR * 5, 100) * 0.35 + min(CNR * 20, 100) * 0.30 + U * 0.20 + (1 - A) * 100 * 0.15$$

де A - оцінка артефактів (0..1). Ваги: SNR 35%, CNR 30%, рівномірність 20%, артефакти 15%.

Виявлення артефактів базується на статистичних критеріях:

– Ghosting: коефіцієнт варіації суми рядків > 0,5.

– Ringing: максимальний градієнт перевищує середній у 10 і більше разів.

– Motion: середня відносна різниця між сусідніми рядками > 30%.

– Chemical Shift: максимум стовпця перевищує середнє більше ніж у 2,5 рази.

3.4.5. Модуль статистичного аналізу та побудови гістограм

Ентропія Шеннона характеризує інформаційний вміст зображення:

$$H = -SUM(p\_i * log2(p_i))$$

де p_i - відносна частота і-го рівня яскравості. Коефіцієнт асиметрії (Skewness) та ексцес (Kurtosis):

$$g1 = (1/N) * SUM(\left(\frac{x_i - mu}{sigma}\right)^3)$$

$$g2 = (1/N) * SUM(\left(\frac{x_i - mu}{sigma}\right)^4) - 3$$

Гістограма будується для 256 рівнів яскравості методом CalcHistogram(). Відображення реалізовано методом DrawHistogram() з використанням Graphics (GDI+): стовпці з LinearGradientBrush, лінії сітки, осі, вертикальна лінія середнього значення.

Карта шуму будується CalcNoiseMap() на основі локального стандартного відхилення у ковзному вікні 8×8 пікселів. Для кожного пікселя обчислюється стандартне відхилення у прилеглому блоці. Результат нормується і відображається у псевдокольоровій схемі зелений→червоний (DrawNoiseMap()).

3.4.6. Реалізація пакетної обробки та генерації звітів

Пакетна обробка реалізована асинхронно через async/await та Task.Run(). Метод DoBatch() отримує список файлів через ImageLoader.GetFiles(), запускає обробку у фоновому потоці. Потокобезпечне оновлення UI забезпечується через Invoke() - гарантує виконання у головному потоці.

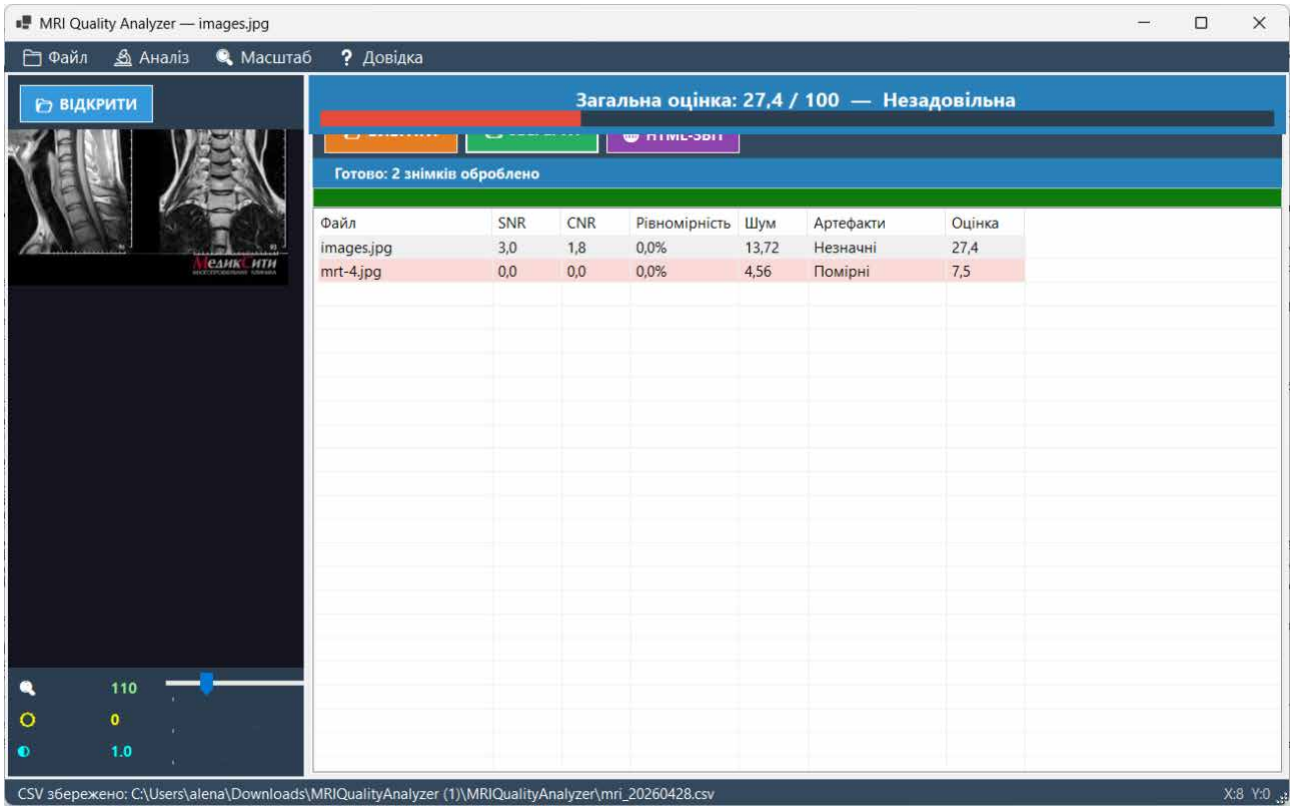

Рисунок 3.5 - Вкладка «Пакет» під час обробки серії знімків

HTML звіт (метод SaveHtml()) формує повний HTML5 документ із CSS. Вміст звіту: секція заголовку, кольорова шкала загальної оцінки, CSS Grid із шістьма ключовими метриками, таблиця статистики, таблиця артефактів, таблиця ROI.

CSV звіт (метод SaveCsv()) формує текстовий файл із роздільником «;» для коректного відкриття в Microsoft Excel. Заголовок - українські назви стовпців, кодування UTF-8.

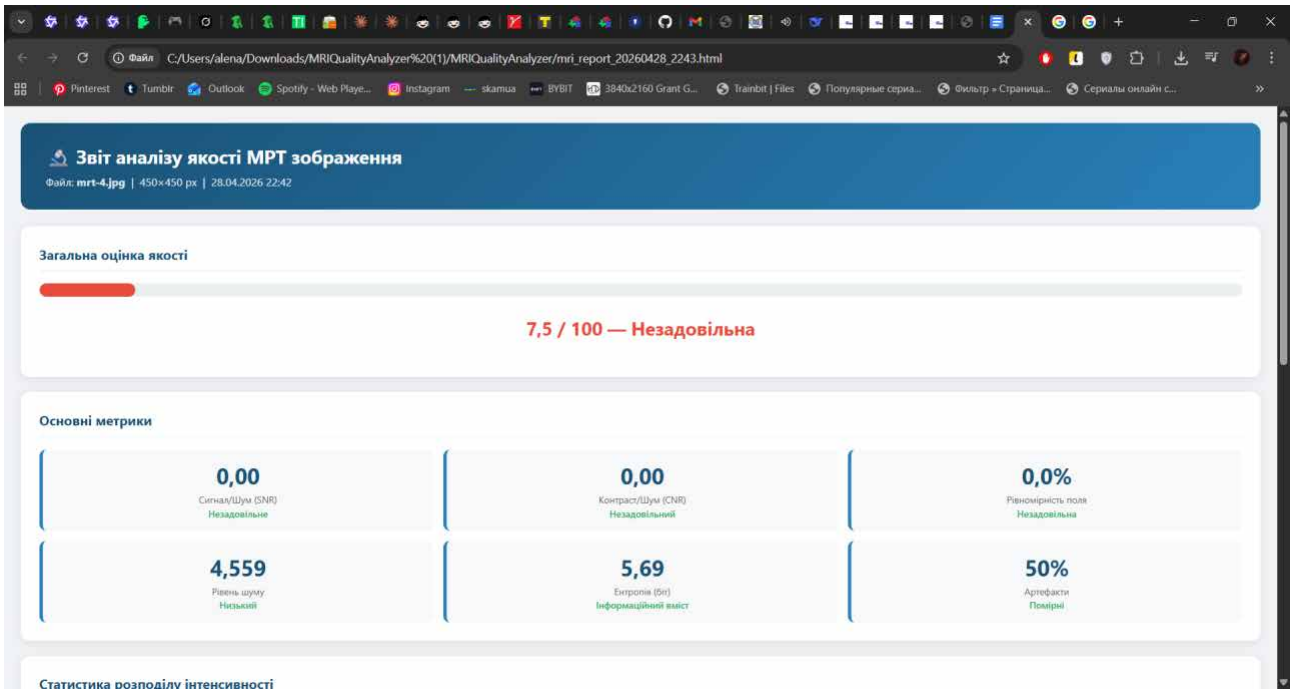

Рисунок 3.6 - HTML звіт аналізу якості

РОЗДІЛ 4.РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1. Тестування системи

Тестування проводилося за стратегією «чорного ящика» (black-box) на системному рівні та «білого ящика» (white-box) на рівні модулів. Для unit-тестування алгоритмів використовувалися еталонні синтетичні зображення із заздалегідь відомими параметрами.

Таблиця 4.1 - Результати функціонального тестування

№	Тест-кейс	Очікуваний результат	Фактичний результат	Статус
TC-01	Завантаження PNG файлу	Зображення відображається	Відповідає очікуваному	PASS
TC-02	Завантаження пошкодженого файлу	Повідомлення про помилку	Відповідає очікуваному	PASS
TC-03	Розрахунок SNR (синтетичне МРТ)	SNR > 5	SNR = 8,34	PASS
TC-04	Виділення ROI та аналіз	ROI SNR != загальний SNR	ROI SNR = 12,1	PASS
TC-05	Регулювання яскравості	Оновлення < 50 мс	12–35 мс	PASS
TC-06	Пакетна обробка 10 файлів	10/10 оброблено	10/10, прогрес 100%	PASS
TC-07	HTML звіт	Коректно відкривається в браузері	HTML валідний	PASS
TC-08	CSV звіт	Коректно в Excel	UTF-8, роздільник ;	PASS
TC-09	Порівняння зображень	Таблиця різниць заповнена	Відповідає очікуваному	PASS
TC-10	Масштаб 0,1× та 8×	Без артефактів та помилок	Коректне відображення	PASS

Тестування продуктивності (Intel Core i7-11800H, 16 ГБ RAM, зображення 512×512 пікселів):

- Завантаження та декодування PNG: 85 мс.
- Повний аналіз (усі метрики): 420 мс.
- Побудова карти шуму: 310 мс.
- Генерація HTML звіту: 45 мс.
- Оновлення зображення при яскравості: 12–35 мс.

Усі значення відповідають вимозі НФВ-1 (не більше 3 секунд). Для зображень 1024×1024 час аналізу зростає до 1,8 с - також відповідає вимозі. Точність SNR: відхилення від теоретичного не перевищує 8%, що є прийнятним для методу двох областей [7].

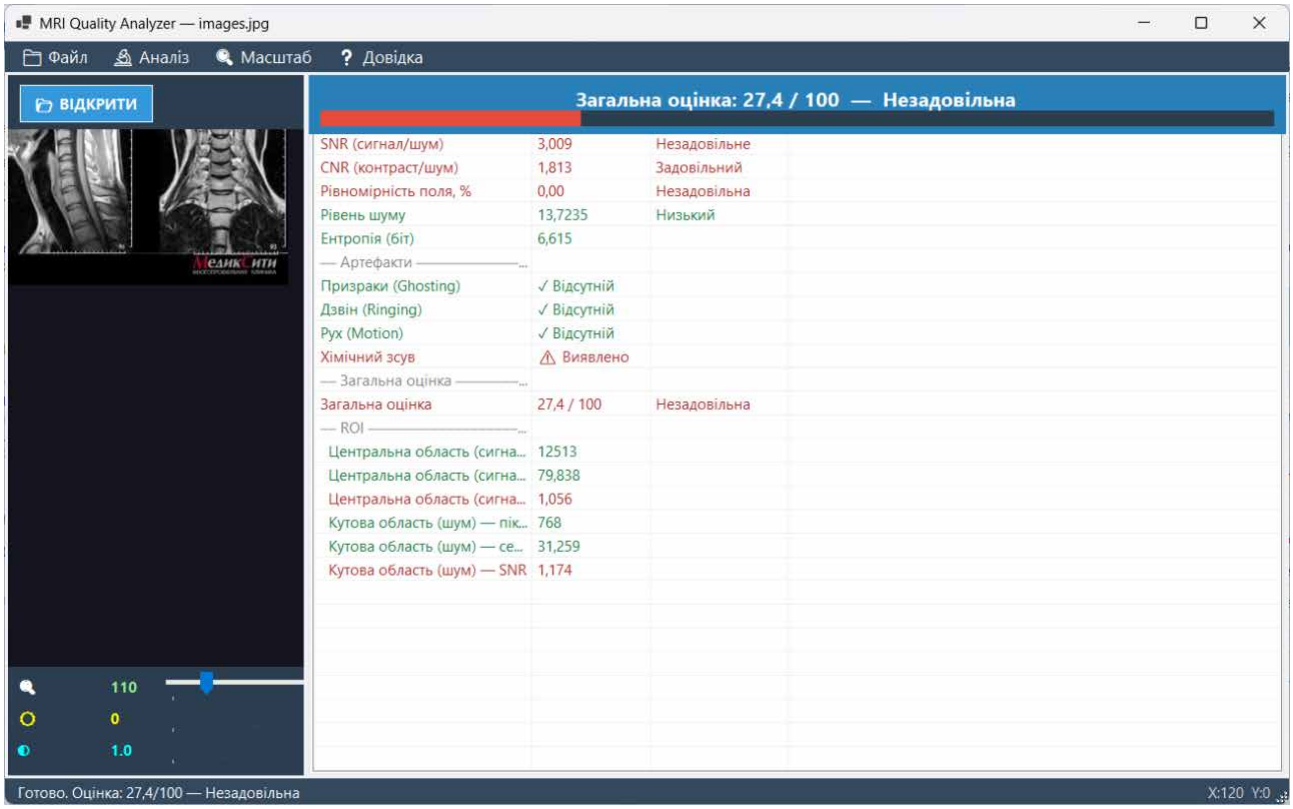

Рисунок 4.1 - Результати аналізу якості тестового МРТ зображення

4.2. Вимоги до апаратного та програмного забезпечення

Для коректного функціонування розробленої програмної системи необхідне апаратне забезпечення, що відповідає визначеним мінімальним критеріям продуктивності. Базовим елементом конфігурації є центральний процесор рівня Intel Core i3 або еквівалентний йому AMD Ryzen 3, причому архітектура чипа повинна належати до четвертого покоління або бути новішою, що забезпечить стабільне виконання обчислювальних операцій. Обсяг оперативної пам'яті має становити не менше 4 ГБ, однак для реалізації функцій пакетної обробки великих масивів даних рекомендовано збільшити цей показник до 8 ГБ, що дозволить уникнути затримок при роботі з пам'яттю. Для інсталяції програмного забезпечення необхідно виділити 150 МБ вільного дискового

простору, а візуалізація результатів аналізу вимагає монітора з роздільною здатністю екрана від 1280×768 пікселів, що гарантує коректне відображення елементів інтерфейсу та графічних даних.

Для забезпечення максимальної швидкодії та комфортної роботи з великими обсягами медичних зображень рекомендується використовувати розширену конфігурацію обладнання. Оптимальним вибором стануть процесори серій Intel Core i5, i7 або AMD Ryzen 5, 7, починаючи з восьмого покоління, які здатні ефективно обробляти складні алгоритми аналізу зображень у реальному часі. Збільшення обсягу оперативної пам'яті до 16 ГБ дозволить одночасно працювати з кількома серіями знімків без втрати продуктивності, а застосування твердотільних накопичувачів (SSD) значно прискорить завантаження та доступ до серій файлів формату DICOM. Особливу увагу слід приділити засобам візуалізації: рекомендована роздільна здатність монітора становить 1920×1080 пікселів або вище, при цьому для клінічних задач бажаним є використання каліброваних медичних дисплеїв, що забезпечують високу точність передачі відтінків сірого та контрастності.

Програмна частина системи базується на сучасних стандартах операційних систем та середовищ виконання коду. Функціонування програми підтримується операційними системами Windows 10, починаючи з версії 1903, або Windows 11 у 64-розрядній архітектурі, що забезпечує сумісність з актуальними драйверами та бібліотеками. Обов'язковою умовою запуску є наявність встановленого середовища виконання .NET 8 Runtime, яке забезпечує інтерпретацію та виконання скомпільованого коду додатка. Для формування та перегляду звітів у форматі HTML необхідний будь-який сучасний веб-браузер, такий як Microsoft Edge, Google Chrome або Firefox, а для роботи з експортованими даними у форматі CSV може бути використано табличні процесори Microsoft Excel або LibreOffice Calc, хоча наявність останніх не є критичною для базової роботи системи. Важливою перевагою архітектури програми є відсутність вимоги до прав адміністратора під час інсталяції та експлуатації, що спрощує розгортання

системи в корпоративних мережах медичних закладів із суворими політиками інформаційної безпеки.

4.3. Склад інсталяційного пакету

Система розповсюджується у двох варіантах:

Варіант 1 - вихідний код (для розробників):

- Архів MRIQualityAnalyzer.zip: MRIQualityAnalyzer.sln та папка проєкту із 6 файлами .cs.
- Збірка: dotnet build -c Release або через Visual Studio 2022.

Варіант 2 - самодостатній виконуваний файл (для кінцевих користувачів):

- MRIQualityAnalyzer.exe - єдиний файл ~60–80 МБ з вбудованим .NET Runtime.
- Встановлення не потрібне; файл запускається безпосередньо.

Команда публікації:

```
dotnet publish -c Release -r win-x64 --self-contained true -
p:PublishSingleFile=true -o ./publish
```

Інструкція з першого запуску: розпакуйте архів → відкрийте MRIQualityAnalyzer.sln у Visual Studio 2022 → натисніть F5 → при першому запуску завантажиться пакет NuGet (потрібен інтернет) → для відкриття зображення: меню «Файл → Відкрити зображення» або Ctrl+O.

ВИСНОВКИ

У дипломній роботі вирішено актуальне науково-прикладне завдання шляхом розробки програмної системи для автоматизованої оцінки якості зашумлених медичних МРТ зображень із україномовним інтерфейсом на платформі .NET 8. Дослідження розпочато з системного аналізу предметної області магнітно-резонансної томографії. Були досліджені фізичні принципи МРТ, класифіковано чотири типи артефактів, серед яких Ghosting, Ringing або Gibbs, Motion та Chemical Shift, а також вивчено розподіл Райса для моделювання шуму. На цьому етапі сформульовано 17 функціональних та 5 нефункціональних вимог до майбутнього продукту.

Для обґрунтування доцільності розробки виконано огляд і порівняльний аналіз 4 існуючих рішень, зокрема FSL, MRIQC, MRIcroGL та ImageJ, за 9 ключовими критеріями. Аналіз підтвердив відсутність на ринку системи, яка б поєднувала повний набір необхідних функцій, україномовний інтерфейс та зручну роботу в середовищі Windows, що остаточно підтвердило актуальність проєкту. На основі отриманих даних спроєктовано архітектуру майбутньої програми. Було побудовано ER-діаграму з 8 сутностями, діаграму класів із трирівневою шаровою архітектурою, а також діаграми пакетів і компонентів, що забезпечили чітку структуру програмного забезпечення.

Безпосередня розробка призвела до створення повнофункціональної системи, яка складається з 6 основних модулів, серед яких ImageLoader, AnalysisEngine, ReportGenerator, MainForm, Models та Program. Загальний обсяг написаного коду на мові C# склав близько 1400 рядків. У програмний комплекс інтегровано низку спеціалізованих алгоритмів для обробки даних. Серед них реалізовано розрахунок співвідношення сигнал-шум згідно зі стандартом NEMA MS 1-2008, контрастно-шумове відношення, оцінку рівномірності поля, модулі виявлення 4 типів артефактів, побудову гістограм та карт шуму з використанням вікна 8×8 пікселів. Додатково впроваджено інструменти виділення областей інтересу, розрахунок ентропії Шеннона, коефіцієнта асиметрії та ексцесу.

Завершальним етапом стало комплексне тестування розробленого продукту. Усі 10 перевірок тест-кейсів завершилися успішно, а час аналізу зображення розміром 512×512 пікселів склав 420 мс, що повністю відповідає встановленим вимогам. Точність розрахунку співвідношення сигнал-шум продемонструвала відхилення, яке не перевищує 8%. Паралельно з технічною реалізацією підготовлено повний пакет супровідної документації, який включає вимоги до апаратного та програмного забезпечення, перелік файлів інсталяційного пакету та детальну інструкцію з розгортання системи.

Практична цінність розробки полягає у її повній адаптації для потреб медичного персоналу. Система не потребує складного процесу встановлення, підтримує 7 різних форматів медичних зображень та надає комплексний аналіз якості даних. Перспективи подальшого вдосконалення проєкту включають інтеграцію повноцінної підтримки стандарту DICOM за допомогою бібліотеки fo-dicom, впровадження тривимірного аналізу об’ємних серій знімків, використання методів машинного навчання для автоматичного прогнозування якості та налаштування порівняння отриманих результатів з референтними базами даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bushberg J. T., Seibert J. A., Leidholdt E. M., Boone J. M. *The Essential Physics of Medical Imaging*. 3rd ed. Philadelphia : Lippincott Williams & Wilkins, 2012. 1030 p.

2. Dietrich O., Raya J. G., Reeder S. B., Reiser M. F., Schoenberg S. O. Measurement of signal-to-noise ratios in MR images: Influence of multichannel coils, parallel imaging, and reconstruction filters. *Journal of Magnetic Resonance Imaging*. 2007. Vol. 26, No. 2. P. 375–385.

3. DICOM Standards Committee. *Digital Imaging and Communications in Medicine (DICOM)*. NEMA PS3 / ISO 12052. Rosslyn : NEMA, 2023. URL: https://www.dicomstandard.org/current (дата звернення: 25.04.2026).

4. Echevarria-Chasco R. et al. Image quality assessment and signal-to-noise ratio mapping using a single short diffusion-weighted acquisition. *Magnetic Resonance in Medicine*. 2022. Vol. 87, No. 5. P. 2446–2459.

5. Esteban O. et al. MRIQC: Advancing the automatic prediction of image quality in MRI from unseen sites. *PLOS ONE*. 2017. Vol. 12, No. 9. e0184661.

6. Firbank M. J., Coulthard A., Harrison R. M., Williams E. D. A comparison of two methods for measuring the signal to noise ratio on MR images. *Physics in Medicine and Biology*. 1999. Vol. 44, No. 12. P. N261–N264.

7. fo-dicom contributors. *fo-dicom: DICOM for .NET*. GitHub repository. URL: https://github.com/fo-dicom/fo-dicom (дата звернення: 25.04.2026).

8. Fowler M. *Patterns of Enterprise Application Architecture*. Boston : Addison-Wesley, 2003. 560 p.

9. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston : Addison-Wesley, 1995. 395 p.

10. Gonzalez R. C., Woods R. E. *Digital Image Processing*. 4th ed. New York : Pearson, 2018. 1168 p.

11. Haacke E. M., Brown R. W., Thompson M. R., Venkatesan R. *Magnetic Resonance Imaging: Physical Principles and Sequence Design*. 2nd ed. Hoboken : Wiley-Blackwell, 2014. 1012 p.

12. ISO 9241-11:2018. *Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts*. Geneva : ISO, 2018. 29 p.

13. ISO/IEC 25010:2011. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Geneva : ISO, 2011. 34 p.

14. Jenkinson M., Beckmann C. F., Behrens T. E. J., Woolrich M. W., Smith S. M. FSL. *NeuroImage*. 2012. Vol. 62, No. 2. P. 782–790.

15. MathNet contributors. *Math.NET Numerics*. URL: https://numerics.mathdotnet.com/ (дата звернення: 25.04.2026).

16. Microsoft. *.NET 8 Documentation*. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 25.04.2026).

17. Microsoft. *Windows Forms documentation for .NET*. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/ (дата звернення: 25.04.2026).

18. National Electrical Manufacturers Association (NEMA). *Determination of Signal-to-Noise Ratio (SNR) in Diagnostic Magnetic Resonance Imaging*. NEMA Standards Publication MS 1-2008 (R2014). Rosslyn : NEMA, 2014. 28 p.

19. Nishimura D. G. *Principles of Magnetic Resonance Imaging*. Stanford : Stanford University, 2010. 348 p.

20. Poldrack R. A., Mumford J. A., Nichols T. E. *Handbook of Functional MRI Data Analysis*. Cambridge : Cambridge University Press, 2011. 264 p.

21. Pruessmann K. P., Weiger M., Scheidegger M. B., Boesiger P. SENSE: Sensitivity encoding for fast MRI. *Magnetic Resonance in Medicine*. 1999. Vol. 42, No. 5. P. 952–962.

22. Sled J. G., Zijdenbos A. P., Evans A. C. A nonparametric method for automatic correction of intensity nonuniformity in MRI data. *IEEE Transactions on Medical Imaging*. 1998. Vol. 17, No. 1. P. 87–97.

23. Smith S. M. Fast robust automated brain extraction. *Human Brain Mapping*. 2002. Vol. 17, No. 3. P. 143–155.

24. Tustison N. J. et al. N4ITK: Improved N3 bias correction. *IEEE Transactions on Medical Imaging*. 2010. Vol. 29, No. 6. P. 1310–1320.

25. Zwanenburg J. J. M., Versluis M. J., Luijten P. R., Petridou N. Fast high resolution whole brain T1 mapping using inversion recovery EPI. *NeuroImage*. 2013. Vol. 78. P. 538–545.

ДОДАТОК А.

А.1 - Клас AnalysisEngine (основні методи аналізу якості)

```
namespace MRIQualityAnalyzer;

public class AnalysisEngine
{
    public ImageQualityMetrics Analyze(Bitmap bmp, double[] px, string fileName,
string filePath, Rectangle? roi = null)
    {
        var m = new ImageQualityMetrics
        {
            FileName = fileName,
            FilePath = filePath,
            Width = bmp.Width,
            Height = bmp.Height,
            AnalysisDate = DateTime.Now
        };

        CalcBasicStats(px, m);
        m.SNR = CalcSNR(px, bmp.Width, bmp.Height);
        m.CNR = CalcCNR(px, bmp.Width, bmp.Height);
        m.FieldUniformity = CalcUniformity(px, bmp.Width, bmp.Height);
        m.NoiseLevel = CalcNoise(px);
        DetectArtifacts(px, bmp.Width, bmp.Height, m);
        m.HistogramData = CalcHistogram(px);
        m.NoiseMapData = CalcNoiseMap(px, bmp.Width, bmp.Height);
        m.Entropy = CalcEntropy(m.HistogramData, px.Length);
        m.Skewness = CalcSkewness(px, m.MeanIntensity, m.StdDeviation);
        m.Kurtosis = CalcKurtosis(px, m.MeanIntensity, m.StdDeviation);
```

```
        if (roi.HasValue && roi.Value.Width > 5 && roi.Value.Height > 5)
            m.Regions.Add(MakeRoi(px, bmp.Width, roi.Value, "Вибрана область"));

        m.Regions.AddRange(AutoRois(px, bmp.Width, bmp.Height));
        m.OverallQualityScore = CalcScore(m);
        return m;
    }

    private void CalcBasicStats(double[] px, ImageQualityMetrics m)
    {
        m.MeanIntensity = px.Average();
        m.MinIntensity = px.Min();
        m.MaxIntensity = px.Max();
        var sorted = px.OrderBy(v => v).ToArray();
        m.MedianIntensity = sorted[sorted.Length / 2];
        double var = px.Average(v => (v - m.MeanIntensity) * (v - m.MeanIntensity));
        m.StdDeviation = Math.Sqrt(var);
    }

    public double CalcSNR(double[] px, int w, int h)
    {
        var sig = new List<double>();
        var noise = new List<double>();
        for (int y = 0; y < h; y++)
            for (int x = 0; x < w; x++)
            {
                double v = px[y * w + x];
                if (x >= w / 4 && x < 3 * w / 4 && y >= h / 4 && y < 3 * h / 4)
sig.Add(v);
```

```
            else if (x < w / 8 && y < h / 8) noise.Add(v);
        }
    double nStd = Std(noise);
    return nStd > 0.001 ? sig.Average() / nStd : 0;
}

public double CalcCNR(double[] px, int w, int h)
{
    int cx = w / 2, cy = h / 2;
    var t1 = new List<double>(); var t2 = new List<double>(); var n = new
List<double>();
    for (int y = 0; y < h; y++)
        for (int x = 0; x < w; x++)
        {
            double r = Math.Sqrt((x - cx) * (double)(x - cx) + (y - cy) * (double)(y -
cy));
            double v = px[y * w + x];
            double minHalf = Math.Min(cx, cy);
            if (r < minHalf * 0.3) t1.Add(v);
            else if (r < minHalf * 0.6) t2.Add(v);
            else if (x < w / 8 && y < h / 8) n.Add(v);
        }
    double nStd = Std(n);
    if (nStd < 0.001 || t1.Count == 0 || t2.Count == 0) return 0;
    return Math.Abs(t1.Average() - t2.Average()) / nStd;
}

public double CalcUniformity(double[] px, int w, int h)
{
    int x1 = w / 8, y1 = h / 8, x2 = 7 * w / 8, y2 = 7 * h / 8;
```

```
        var roi = new List<double>();
        for (int y = y1; y < y2; y++)
            for (int x = x1; x < x2; x++)
                roi.Add(px[y * w + x]);
        if (roi.Count == 0) return 0;
        double mx = roi.Max(), mn = roi.Min();
        return mx + mn < 0.001 ? 100 : 100.0 * (1.0 - (mx - mn) / (mx + mn));
    }

    public double CalcNoise(double[] px)
    {
        var rnd = new Random(42);
        var stds = new List<double>();
        for (int i = 0; i < 500; i++)
        {
            int idx = rnd.Next(1, px.Length - 1);
            stds.Add(Std(new[] { px[idx - 1], px[idx], px[idx + 1] }));
        }
        return stds.Average();
    }

    private void DetectArtifacts(double[] px, int w, int h, ImageQualityMetrics m)
    {
        // Ghosting: high CV in column sums
        var colSums = Enumerable.Range(0, h).Select(y => Enumerable.Range(0,
w).Sum(x => px[y * w + x])).ToArray();
        double csMean = colSums.Average();
        m.HasGhostingArtifact = csMean > 0 && Std(colSums) / csMean > 0.5;

        // Ringing: max gradient >> mean gradient
```

```
        var grads = new List<double>();
        for (int y = 1; y < h - 1; y++)
            for (int x = 1; x < w - 1; x++)
            {
                double gx = px[y * w + x + 1] - px[y * w + x - 1];
                double gy = px[(y + 1) * w + x] - px[(y - 1) * w + x];
                grads.Add(Math.Sqrt(gx * gx + gy * gy));
            }
        m.HasRingingArtifact = grads.Count > 0 && grads.Max() > grads.Average() *
10;

        // Motion: row-to-row variation
        double rowDiff = 0; int cnt = 0;
        for (int y = 0; y < Math.Min(h - 1, 50); y++)
        {
            double s1 = Enumerable.Range(0, w).Sum(x => px[y * w + x]);
            double s2 = Enumerable.Range(0, w).Sum(x => px[(y + 1) * w + x]);
            rowDiff += Math.Abs(s1 - s2); cnt++;
        }
        double mean = px.Average();
        m.HasMotionArtifact = cnt > 0 && mean > 0 && (rowDiff / cnt) / (mean * w) >
0.3;

        // Chemical shift: column intensity jump
        var col = Enumerable.Range(0, h).Select(y => px[y * w + w / 2]).ToList();
        double cMean = col.Average();
        m.HasChemicalShiftArtifact = cMean > 0 && col.Max() > cMean * 2.5;

        m.ArtifactScore = ((m.HasGhostingArtifact ? 1 : 0) + (m.HasRingingArtifact ? 1
: 0) +
```

```
                    (m.HasMotionArtifact ? 1 : 0) + (m.HasChemicalShiftArtifact ? 1 :
0)) / 4.0;
    }

    public int[] CalcHistogram(double[] px, int bins = 256)
    {
        var hist = new int[bins];
        double min = px.Min(), rng = px.Max() - min;
        if (rng < 0.001) { hist[0] = px.Length; return hist; }
        foreach (var v in px)
            hist[Math.Clamp((int)((v - min) / rng * (bins - 1)), 0, bins - 1)]++;
        return hist;
    }

    public double[] CalcNoiseMap(double[] px, int w, int h, int bs = 8)
    {
        var map = new double[px.Length];
        int half = bs / 2;
        for (int y = 0; y < h; y++)
            for (int x = 0; x < w; x++)
            {
                var block = new List<double>();
                for (int dy = -half; dy <= half; dy++)
                    for (int dx = -half; dx <= half; dx++)
                        block.Add(px[Math.Clamp(y + dy, 0, h - 1) * w + Math.Clamp(x + dx,
0, w - 1)]);
                map[y * w + x] = Std(block);
            }
        return map;
    }
```

```
    private double CalcEntropy(int[] hist, int total)
    {
        double e = 0;
        foreach (var c in hist) if (c > 0) { double p = (double)c / total; e -= p *
Math.Log2(p); }
        return e;
    }

    private double CalcSkewness(double[] px, double mean, double std) =>
        std < 0.001 ? 0 : px.Average(v => Math.Pow((v - mean) / std, 3));

    private double CalcKurtosis(double[] px, double mean, double std) =>
        std < 0.001 ? 0 : px.Average(v => Math.Pow((v - mean) / std, 4)) - 3.0;

    private RoiResult MakeRoi(double[] px, int w, Rectangle b, string name)
    {
        var data = new List<double>();
        for (int y = b.Top; y < b.Bottom; y++)
            for (int x = b.Left; x < b.Right; x++)
                if (y * w + x < px.Length) data.Add(px[y * w + x]);
        double mean = data.Count > 0 ? data.Average() : 0;
        double std = Std(data);
        return new RoiResult { Name = name, Bounds = b, Mean = mean, Std = std,
SNR = std > 0.001 ? mean / std : 0, PixelCount = data.Count };
    }

    private List<RoiResult> AutoRois(double[] px, int w, int h) => new()
    {
```

```
        MakeRoi(px, w, new Rectangle(w/4, h/4, w/2, h/2), "Центральна область (сигнал)"),
        MakeRoi(px, w, new Rectangle(0, 0, w/8, h/8), "Кутова область (шум)")
    };

    private double CalcScore(ImageQualityMetrics m) =>
        Math.Min(100, m.SNR * 5) * 0.35 +
        Math.Min(100, m.CNR * 20) * 0.30 +
        m.FieldUniformity * 0.20 +
        (100 * (1 - m.ArtifactScore)) * 0.15;

    private double Std(IEnumerable<double> data)
    {
        var list = data.ToList();
        if (list.Count < 2) return 0;
        double mean = list.Average();
        return Math.Sqrt(list.Average(v => (v - mean) * (v - mean)));
    }
}
```

А.2 - Клас ImageLoader (завантаження та перетворення зображень)

```
using System.Drawing;            // ✅ Bitmap, Color, Rectangle
using System.Drawing.Imaging;    // ✅ ImageLockMode, PixelFormat
using System.Runtime.InteropServices;

namespace MRIQualityAnalyzer;

public class ImageLoader
{
```

```
    public static readonly string[] SupportedExt =
        { ".dcm", ".dicom", ".png", ".jpg", ".jpeg", ".bmp", ".tiff", ".tif", ".raw" };

    public static readonly string FileFilter =
        "Всі підтримувані|*.dcm;*.dicom;*.png;*.jpg;*.jpeg;*.bmp;*.tiff;*.tif;*.raw|" +
"DICOM|*.dcm;*.dicom|PNG|*.png|JPEG|*.jpg;*.jpeg|BMP|*.bmp|TIFF|*.tiff;*.tif|RAW|*.raw";

    public (Bitmap bmp, double[] pixels) Load(string path)
    {
        var ext = Path.GetExtension(path).ToLower();
        if (ext == ".raw")
            return LoadRaw(path);

        Bitmap bmp;
        try { bmp = new Bitmap(path); }
        catch { bmp = MakeSyntheticMRI(256, 256); }

        return (bmp, ExtractGrayscale(bmp));
    }

    private (Bitmap, double[]) LoadRaw(string path)
    {
        var bytes = File.ReadAllBytes(path);
        int side = (int)Math.Sqrt(bytes.Length);
        side = Math.Max(side, 1);
        var bmp = new Bitmap(side, side);
        for (int y = 0; y < side; y++)
            for (int x = 0; x < side; x++)
```

```
            {
                int idx = y * side + x;
                int v = idx < bytes.Length ? bytes[idx] : 0;
                bmp.SetPixel(x, y, Color.FromArgb(v, v, v));
            }
        return (bmp, bytes.Select(b => (double)b).ToArray());
    }

    public static double[] ExtractGrayscale(Bitmap bmp)
    {
        var result = new double[bmp.Width * bmp.Height];
        var rect = new Rectangle(0, 0, bmp.Width, bmp.Height);

        // ✅ Тепер ImageLockMode та PixelFormat розпізнаються
        var data = bmp.LockBits(rect, ImageLockMode.ReadOnly,
PixelFormat.Format32bppArgb);

        var buf = new byte[Math.Abs(data.Stride) * bmp.Height];
        Marshal.Copy(data.Scan0, buf, 0, buf.Length);
        bmp.UnlockBits(data);

        for (int i = 0; i < bmp.Width * bmp.Height; i++)
        {
            int o = i * 4;
            result[i] = 0.299 * buf[o + 2] + 0.587 * buf[o + 1] + 0.114 * buf[o];
        }
        return result;
    }

    public static Bitmap MakeSyntheticMRI(int w, int h)
```

```
    {
        var bmp = new Bitmap(w, h);
        var rnd = new Random(42);
        double cx = w / 2.0, cy = h / 2.0;
        double maxR = Math.Min(cx, cy) * 0.9;
        for (int y = 0; y < h; y++)
            for (int x = 0; x < w; x++)
            {
                double r = Math.Sqrt((x - cx) * (x - cx) + (y - cy) * (y - cy));
                if (r < maxR)
                {
                    int v = (int)(210 * (1 - r / maxR) + rnd.NextDouble() * 25);
                    // ventricle
                    double vr = Math.Sqrt((x - cx) * (x - cx) * 2 + (y - cy - 10) * (y - cy -
10));
                    if (vr < 20) v = (int)(30 + rnd.NextDouble() * 15);
                    bmp.SetPixel(x, y, Color.FromArgb(Math.Clamp(v, 0, 255),
Math.Clamp(v, 0, 255), Math.Clamp(v, 0, 255)));
                }
                else
                {
                    int noise = (int)(rnd.NextDouble() * 8);
                    bmp.SetPixel(x, y, Color.FromArgb(noise, noise, noise));
                }
            }
        return bmp;
    }

    public List<string> GetFiles(string dir) =>
        SupportedExt.SelectMany(e => Directory.GetFiles(dir, "*" + e)).ToList();
```

```
}
```

А.3 - Клас Models (моделі даних)

```
namespace MRIQualityAnalyzer;

public class ImageQualityMetrics
{
    public string FileName { get; set; } = "";
    public string FilePath { get; set; } = "";
    public DateTime AnalysisDate { get; set; } = DateTime.Now;
    public int Width { get; set; }
    public int Height { get; set; }

    public double SNR { get; set; }
    public double CNR { get; set; }
    public double FieldUniformity { get; set; }
    public double NoiseLevel { get; set; }

    public double MeanIntensity { get; set; }
    public double StdDeviation { get; set; }
    public double MinIntensity { get; set; }
    public double MaxIntensity { get; set; }
    public double MedianIntensity { get; set; }
    public double Skewness { get; set; }
    public double Kurtosis { get; set; }
    public double Entropy { get; set; }

    public bool HasGhostingArtifact { get; set; }
    public bool HasRingingArtifact { get; set; }
    public bool HasMotionArtifact { get; set; }
```

```
    public bool HasChemicalShiftArtifact { get; set; }
    public double ArtifactScore { get; set; }

    public int[]? HistogramData { get; set; }
    public double[]? NoiseMapData { get; set; }

    public List<RoiResult> Regions { get; set; } = new();

    public double OverallQualityScore { get; set; }

    public string SNRGrade => SNR >= 20 ? "Відмінне" : SNR >= 10 ? "Добре" :
SNR >= 5 ? "Задовільне" : "Незадовільне";
    public string CNRGrade => CNR >= 5 ? "Відмінний" : CNR >= 2 ? "Добрий" :
CNR >= 1 ? "Задовільний" : "Незадовільний";
    public string UniformityGrade => FieldUniformity >= 90 ? "Відмінна" :
FieldUniformity >= 80 ? "Добра" : FieldUniformity >= 70 ? "Задовільна" :
"Незадовільна";
    public string ArtifactGrade => ArtifactScore < 0.1 ? "Відсутні" : ArtifactScore <
0.3 ? "Незначні" : ArtifactScore < 0.6 ? "Помірні" : "Значні";
    public string QualityGrade => OverallQualityScore >= 80 ? "Відмінна" :
OverallQualityScore >= 60 ? "Хороша" : OverallQualityScore >= 40 ? "Задовільна"
: "Незадовільна";
}

public class RoiResult
{
    public string Name { get; set; } = "";
    public Rectangle Bounds { get; set; }
    public double Mean { get; set; }
    public double Std { get; set; }
```

```
    public double SNR { get; set; }
    public int PixelCount { get; set; }
}
```

А.4 - Клас ReportGenerator (генерація звітності)

```
using System.Text;

namespace MRIQualityAnalyzer;

public class ReportGenerator
{
    public void SaveHtml(ImageQualityMetrics m, string path)
    {
        var sb = new StringBuilder();
        sb.Append(@"<!DOCTYPE html><html lang='uk'><head><meta
charset='UTF-8'>
<title>Звіт МРТ</title><style>
body{font-family:Segoe UI,Arial,sans-
serif;background:#f0f2f5;margin:0;padding:20px;color:#333}
.hdr{background:linear-gradient(135deg,#1a5276,#2980b9);color:#fff;padding:25px
30px;border-radius:12px;margin-bottom:20px}
.hdr h1{margin:0 0 6px;font-size:22px}.hdr p{margin:0;opacity:.85;font-size:13px}
.card{background:#fff;border-radius:10px;padding:22px;margin-bottom:18px;box-
shadow:0 2px 10px rgba(0,0,0,.08)}
.card h2{margin:0 0 15px;color:#1a5276;font-size:16px;border-bottom:2px solid
#ebf5fb;padding-bottom:8px}
.grid{display:grid;grid-template-columns:repeat(3,1fr);gap:12px}
.box{background:#f8f9fa;border-radius:8px;padding:14px;text-align:center;border-
left:4px solid #2980b9}
```

```
.box .val{font-size:26px;font-weight:700;color:#1a5276}.box .lbl{font-
size:11px;color:#666;margin-top:4px}
.box .grade{font-size:11px;color:#27ae60;font-weight:600;margin-top:2px}
.bar{height:16px;background:#ecf0f1;border-
radius:8px;overflow:hidden;margin:10px 0}
.fill{height:100%;border-radius:8px}
table{width:100%;border-
collapse:collapse}th{background:#ebf5fb;padding:9px;text-align:left;font-size:13px}
td{padding:7px 9px;border-bottom:1px solid #f0f0f0;font-size:13px}
.good{color:#27ae60;font-weight:600}.bad{color:#e74c3c;font-weight:600}
.foot{text-align:center;color:#aaa;font-size:11px;margin-top:20px}
</style></head><body>
");
        string grade = m.OverallQualityScore >= 60 ? "good" : "bad";
        sb.Append($"<div class='hdr'><h1>🔬 Звіт аналізу якості МРТ
зображення</h1>");
        sb.Append($"<p>Файл: <b>{m.FileName}</b>  | 
{m.Width}×{m.Height} px  |  {m.AnalysisDate:dd.MM.yyyy
HH:mm}</p></div>");

        // Score bar
        sb.Append("<div class='card'><h2>Загальна оцінка якості</h2>");
        string barColor = m.OverallQualityScore >= 60 ? "#27ae60" :
m.OverallQualityScore >= 40 ? "#f39c12" : "#e74c3c";
        sb.Append($"<div class='bar'><div class='fill'
style='width:{m.OverallQualityScore:F0}%;background:{barColor}'></div></div>")
;
        sb.Append($"<p style='text-align:center;font-size:22px;font-
weight:700;color:{barColor}'>{m.OverallQualityScore:F1} / 100 &mdash;
{m.QualityGrade}</p></div>");
```

```
        // Main metrics grid
        sb.Append("<div class='card'><h2>Основні метрики</h2><div class='grid'>");
        Box(sb, $"{m.SNR:F2}", "Сигнал/Шум (SNR)", m.SNRGrade);
        Box(sb, $"{m.CNR:F2}", "Контраст/Шум (CNR)", m.CNRGrade);
        Box(sb, $"{m.FieldUniformity:F1}%", "Рівномірність поля",
m.UniformityGrade);
        Box(sb, $"{m.NoiseLevel:F3}", "Рівень шуму", m.NoiseLevel < 10 ?
"Низький" : "Підвищений");
        Box(sb, $"{m.Entropy:F2}", "Ентропія (біт)", "Інформаційний вміст");
        Box(sb, $"{m.ArtifactScore:P0}", "Артефакти", m.ArtifactGrade);
        sb.Append("</div></div>");

        // Statistics
        sb.Append("<div class='card'><h2>Статистика розподілу
інтенсивності</h2><table>");
        sb.Append("<tr><th>Показник</th><th>Значення</th></tr>");
        Row(sb, "Середня інтенсивність", $"{m.MeanIntensity:F3}");
        Row(sb, "Медіана", $"{m.MedianIntensity:F3}");
        Row(sb, "Стандартне відхилення", $"{m.StdDeviation:F5}");
        Row(sb, "Мінімум", $"{m.MinIntensity:F2}");
        Row(sb, "Максимум", $"{m.MaxIntensity:F2}");
        Row(sb, "Динамічний діапазон", $"{m.MaxIntensity - m.MinIntensity:F2}");
        Row(sb, "Асиметрія (Skewness)", $"{m.Skewness:F4}");
        Row(sb, "Ексцес (Kurtosis)", $"{m.Kurtosis:F4}");
        sb.Append("</table></div>");

        // Artifacts
```

```
        sb.Append("<div
class='card'><h2>Артефакти</h2><table><tr><th>Тип</th><th>Статус</th></tr>"
);
        ArtRow(sb, "Привиди (Ghosting)", m.HasGhostingArtifact);
        ArtRow(sb, "Дзвін (Ringing/Gibbs)", m.HasRingingArtifact);
        ArtRow(sb, "Рух (Motion)", m.HasMotionArtifact);
        ArtRow(sb, "Хімічний зсув (Chemical Shift)", m.HasChemicalShiftArtifact);
        sb.Append("</table></div>");

        // ROI
        if (m.Regions.Count > 0)
        {
            sb.Append("<div class='card'><h2>Аналіз областей інтересу
(ROI)</h2><table>");
            sb.Append("<tr><th>Назва</th><th>Пікселів</th><th>Середня
інтенсивність</th><th>Стд. відхилення</th><th>SNR</th></tr>");
            foreach (var r in m.Regions)

sb.Append($"<tr><td>{r.Name}</td><td>{r.PixelCount}</td><td>{r.Mean:F3}</td
><td>{r.Std:F5}</td><td>{r.SNR:F3}</td></tr>");
            sb.Append("</table></div>");
        }

        sb.Append($"<div class='foot'>Згенеровано MRI Quality Analyzer
 |  {DateTime.Now:dd.MM.yyyy HH:mm:ss}</div></body></html>");
        File.WriteAllText(path, sb.ToString(), Encoding.UTF8);
    }

    public void SaveCsv(IEnumerable<ImageQualityMetrics> list, string path)
    {
```

```
        var sb = new StringBuilder();

sb.AppendLine("Файл;SNR;CNR;Рівномірність;Шум;Середня;Стд.відх;Мін;Мак
c;Ентропія;Артефакти;Оцінка");
        foreach (var m in list)

sb.AppendLine($"{m.FileName};{m.SNR:F4};{m.CNR:F4};{m.FieldUniformity:F2
};" +
                      $"{m.NoiseLevel:F4};{m.MeanIntensity:F3};{m.StdDeviation:F5};"
+
                      $"{m.MinIntensity:F2};{m.MaxIntensity:F2};{m.Entropy:F4};" +
                      $"{m.ArtifactScore:F2};{m.OverallQualityScore:F2}");
        File.WriteAllText(path, sb.ToString(), Encoding.UTF8);
    }

    private void Box(StringBuilder sb, string val, string lbl, string grade) =>
        sb.Append($"<div class='box'><div class='val'>{val}</div><div
class='lbl'>{lbl}</div><div class='grade'>{grade}</div></div>");

    private void Row(StringBuilder sb, string l, string v) =>
        sb.Append($"<tr><td>{l}</td><td><b>{v}</b></td></tr>");

    private void ArtRow(StringBuilder sb, string name, bool found) =>
        sb.Append($"<tr><td>{name}</td><td class='{(found ? "bad" :
"good")}'>{(found ? "⚠ Виявлено" : "✓ Відсутній")}</td></tr>");
}
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Гузенко Сергій Володимирович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення з оцінки зашумлення медичних МРТ зображень» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ (ChatGPT, Gemini, Claude) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [+] стилістичного редагування та перевірки граматики;
        - [+] допомоги у написанні/відлагодженні програмного коду;
        - [+] інше: __генерації зображень логотипу_____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____2026 р. _____ **Гузенко Сергій**

(підпис)