НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ
ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

"На правах рукопису"
УДК_____

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2026 р.

Магістерська дисертація

на здобуття ступеня другого (магістерського) рівня вищої освіти
за освітньою програмою “Комп’ютерні науки”
зі спеціальності 122 “Комп’ютерні науки”
на тему Методи оцінювання стійкості потокових шифрів до статистичних атак

Виконав: студент 2 курсу, групи ТР-41мн
ІВАНОВ Валерій Олегович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник _____к.в.н., доцент Андрій ОНИСЬКО _____
(науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент доцент каф. теплової та альтернативної енергетики,
к.т.н. Артур РАЧИНСЬКИЙ_____ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль провідний інженер Людмила КУЗЬМІНА_____ _____
(підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – другий (магістерський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-наукова програма “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

«__»_____2026 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

ІВАНОВУ Валерію Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Методи оцінювання стійкості потокових шифрів до статистичних атак
керівник роботи Онисько Андрій Ілліч, к.в.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «30» березня 2026 р. № 1326-с

2. Термін подання студентом дисертації: 11.05.2026
3. Об’єкт дослідження: оцінювання стійкості потокових шифрів до статистичних атак.
4. Вихідні дані: алгоритми потокових шифрів (LFSR, BBS, RC4, Mersenne Twister, System Random); алгоритми та критерії статистичних тестів (частотний, серійний, NIST, покер-тест, автокореляційний); вхідні параметри генераторів, тестові послідовності.
5. Перелік завдань: проаналізувати сучасні методи статистичного оцінювання стійкості потокових шифрів; розробити програмний засіб для автоматизованого

виконання статистичних тестів; провести експериментальне дослідження та порівняти стійкість різних генераторів.

6. Орієнтовний перелік ілюстративного матеріалу: Схема архітектури програмного засобу, Діаграма компонентів основних модулів, Гістограми бітів згенерованих послідовностей, Діаграма p-value для різних генераторів.

7. Орієнтований перелік публікацій: Іванов В. О., Новицький К.В., Шушура О.М. Автоматичне резюмування наукових документів на основі моделей-трансформерів. Вісник Херсонського національного технічного університету. - 2026.

8. Дата видачі завдання 16 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вивчення літературних джерел та аналі	Січень 2026	Виконано
2	Аналіз існуючих методів статистичного оцінювання стійкості потокових шифрів	Січень 2026	Виконано
3	Розробка архітектури програмного засобу	Лютий 2026	Виконано
4	Реалізація модулів генераторів потокових шифрів і тестів	Лютий 2026	Виконано
5	Розробка графічного інтерфейсу користувача	Березень 2026	Виконано
6	Проведення експериментальних досліджень, збір та аналіз результатів	Березень 2026	Виконано
7	Написання та редагування тексту магістерської дисертації	Квітень 2026	Виконано
8	Захист магістерської дисертації	Травень 2026	Виконано

Студент _____ Валерій ІВАНОВ
(підпис) (ім’я, ПРІЗВИЩЕ)

Науковий керівник _____ Андрій ОНИСЬКО
(підпис) (ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Актуальність теми

Потокові шифри широко використовуються для захисту інформації у сучасних цифрових системах. Їх стійкість до статистичних атак визначає надійність криптографічного захисту. Вдосконалення та автоматизація методів оцінювання стійкості потокових шифрів є актуальним завданням сучасної криптографії.

Мета роботи – проаналізувати, порівняти та вдосконалити методи статистичного оцінювання стійкості потокових шифрів до статистичних атак, а також розробити програмний засіб для автоматизованого тестування та візуалізації результатів.

Завдання дослідження:

1. Провести аналіз методів оцінювання стійкості потокових шифрів.
2. Розробити програмний засіб для виконання статистичних тестів.
3. Реалізувати модулі генераторів та статистичних тестів.
4. Забезпечити візуалізацію результатів тестування.
5. Провести дослідження і порівняти стійкість різних генераторів.

Об’єкт дослідження – оцінювання стійкості потокових шифрів до статистичних атак.

Предмет дослідження – алгоритмічні та програмні методи оцінювання стійкості потокових шифрів на основі статистичних тестів.

Методи дослідження: теоретичний аналіз, моделювання, розробка програмного забезпечення, експериментальне тестування, статистичний аналіз результатів.

Апробація результатів роботи

Основні результати дослідження опубліковано у фаховому виданні:

Іванов В. О., Новицький К.В., Шушура О.М. Автоматичне резюмування наукових документів на основі моделей-трансформерів. Вісник Херсонського національного технічного університету - 2026.

Структура та обсяг магістерської дисертації

Дисертація складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатка. Повний обсяг роботи – 124 сторінки, 4 рисунки, 3 сторінки списку використаних джерел (44 найменування), 1 додаток.

Публікації

За матеріалами дисертації опубліковано 1 статтю у фаховому журналі.

Ключові слова: потокові шифри, статистичні атаки, стійкість, статистичне оцінювання, генератор, програмний засіб, візуалізація, p-value, криптографія, тестування.

ABSCTRACT

Relevance

Stream ciphers are widely used for information protection in modern digital systems. Their resistance to statistical attacks determines the reliability of cryptographic protection. Improvement and automation of methods for assessing the resistance of stream ciphers is an urgent task of modern cryptography.

The aim of the work is to analyze, compare, and improve statistical methods for assessing the resistance of stream ciphers to statistical attacks, as well as to develop a software tool for automated testing and visualization of results.

Research objectives:

1. To analyze modern methods of statistical assessment of stream cipher resistance.
2. To develop a software tool for automated statistical testing.
3. To implement modules of generators and statistical tests.
4. To provide visualization of test results.
5. To conduct experimental research and compare the resistance of generators.

Object of research – stream ciphers and their cryptographic resistance.

Subject of research – methods of statistical assessment of stream cipher resistance to statistical attacks.

Research methods: theoretical analysis, modeling, software development, experimental testing, statistical analysis of results.

Approbation of results

The main results of the research were published in a professional journal:
Ivanov V. O., Novytskyi K. V., Shushura O. M. Automatic Summarization of Scientific Documents Based on Transformer Models. Bulletin of Kherson National Technical University.

Structure and volume of the master's thesis

The thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices. The total volume is 124 pages, 4 figures, 1 addon and 3 pages of references (44 sources).

Publications

Based on the thesis materials, 1 article was published in a professional journal.

Keywords: stream ciphers, statistical attacks, resistance, statistical assessment, generator, software tool, visualization, p-value, cryptography, testin

ЗМІСТ

ВСТУП

Актуальність теми

Потокові шифри є невід'ємною складовою сучасних криптографічних систем, що використовуються для захисту інформації у телекомунікаційних мережах, бездротових протоколах, банківських системах та багатьох інших сферах. В умовах зростання обсягів переданої та оброблюваної інформації, а також підвищення складності атак на інформаційні системи, питання забезпечення стійкості потокових шифрів до статистичних атак набуває особливої актуальності.

Статистичні атаки дозволяють зловмиснику виявити закономірності у вихідних послідовностях шифру, що може призвести до компрометації ключа або розкриття частини зашифрованої інформації. Традиційні підходи до оцінювання стійкості потокових шифрів часто є трудомісткими, потребують значних обчислювальних ресурсів та не завжди враховують сучасні вимоги до автоматизації та інтеграції з програмними засобами.

Тому розробка та вдосконалення методів статистичного оцінювання стійкості потокових шифрів, а також створення програмних інструментів для автоматизованого аналізу, є важливим завданням сучасної криптографії, що має як теоретичне, так і практичне значення для підвищення рівня інформаційної безпеки.

Мета роботи

Метою магістерської дисертації є дослідження ефективних підходів до статистичного оцінювання стійкості потокових шифрів до статистичних атак шляхом розробки, впровадження та апробації програмного засобу для автоматизованого тестування та візуалізації результатів.

Досягнення цієї мети передбачає визначення оптимальних статистичних тестів, розробку модульної архітектури програмного забезпечення, що дозволяє гнучко додавати нові генератори та тести, а також забезпечення зручного інтерфейсу для користувача.

Очікуваним результатом є створення інструменту, який дозволяє швидко,

об’єктивно та наочно оцінювати стійкість різних потокових шифрів до статистичних атак, що сприятиме підвищенню надійності криптографічних систем.

Завдання дослідження

- провести аналіз сучасних методів статистичного оцінювання стійкості потокових шифрів;

- розробити програмний засіб для автоматизованого виконання статистичних тестів;

- реалізувати модулі генераторів та статистичних тестів для різних потокових шифрів;

- забезпечити візуалізацію результатів тестування для зручної інтерпретації;

- провести експериментальне дослідження та порівняти стійкість різних генераторів.

Об’єкт дослідження – оцінювання стійкості потокових шифрів до статистичних атак.

Предмет дослідження – алгоритмічні та програмні методи оцінювання стійкості потокових шифрів на основі статистичних тестів.

Методи дослідження

У процесі виконання роботи використано комплекс сучасних наукових методів, що забезпечують достовірність отриманих результатів:

- теоретичний аналіз — для вивчення сучасних підходів до статистичного тестування, аналізу літературних джерел, стандартів та існуючих рішень у сфері криптографії.

- математичне моделювання — для побудови моделей генераторів потокових шифрів та формалізації критеріїв статистичних тестів.

- розробка програмного забезпечення — для реалізації автоматизованого інструменту, що дозволяє виконувати генерацію послідовностей, запускати тести та візуалізувати результати.

- експериментальне дослідження — для практичної перевірки стійкості різних генераторів потокових шифрів до статистичних атак, збору та аналізу емпіричних даних.

- статистичний аналіз — для обробки результатів тестування, оцінки p-value, порівняння ефективності різних методів та інтерпретації отриманих даних.

Застосування цих методів дозволило комплексно дослідити поставлену проблему та забезпечити наукову новизну і практичну цінність отриманих результатів.

Практичне значення отриманих результатів

Розроблений програмний засіб дозволяє автоматизувати процес оцінювання стійкості потокових шифрів, що може бути використано для підвищення надійності криптографічних систем, навчання студентів, а також у наукових дослідженнях з інформаційної безпеки.

Апробація результатів дослідження

Основні результати роботи були опубліковані у фаховому виданні: Іванов В. О., Новицький К.В., Шушура О.М. Автоматичне резюмування наукових документів на основі моделей-трансформерів. Вісник Херсонського національного технічного університету - 2026.

Публікації

За матеріалами дисертації опубліковано 1 статтю у фаховому журналі.

Структура й обсяг роботи

Робота складається зі вступу, чотирьох розділів, висновків до розділів, загальних висновків, списку використаних джерел (44 найменування), додатку. Загальний обсяг дисертації – 124 сторінки. Основний зміст викладено на 90 сторінках. Роботу проілюстровано 4 рисунками.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ОЦІНЮВАННЯ СТІЙКОСТІ ПОТОКОВИХ ШИФРІВ ДО СТАТИСТИЧНИХ АТАК

У першому розділі магістерської дисертації зроблено постановку задачі дослідження, а також глибокий аналіз сучасного стану проблеми оцінювання стійкості потокових шифрів до статистичних атак. У цьому розділі розкриваються теоретичні основи потокового шифрування, розглядається класифікація та суть статистичних атак, аналізуються існуючі методи статистичного тестування, а також програмні засоби, що використовуються для вирішення поставлених завдань.

Захист інформації у цифрову епоху є одним із найважливіших завдань для державних структур, бізнесу, наукових установ та приватних осіб. З кожним роком зростає кількість даних, що передаються та зберігаються у цифровому вигляді, а також ускладнюються методи атак на інформаційні системи. У цьому контексті криптографічні алгоритми, зокрема потокові шифри, відіграють ключову роль у забезпеченні конфіденційності, цілісності та доступності інформації [9, 15].

Потокові шифри, на відміну від блочних, забезпечують шифрування даних у реальному часі, що робить їх незамінними для захисту потокових даних у телекомунікаційних мережах, мобільному зв’язку, бездротових протоколах, фінансових транзакціях, промислових системах тощо. Водночас, навіть найсучасніші шифри можуть бути вразливими до статистичних атак, якщо їх вихідні послідовності не відповідають вимогам випадковості [10, 7].

Статистичні атаки є одним із найефективніших інструментів криптоаналізу, оскільки дозволяють виявити закономірності у вихідних послідовностях шифру, що може призвести до компрометації ключа або розкриття частини зашифрованої інформації. Саме тому вдосконалення та автоматизація методів оцінювання стійкості потокових шифрів до статистичних атак є актуальним завданням, що має як теоретичне, так і практичне значення для підвищення надійності інформаційного захисту [3].

У цьому розділі також розглядаються сучасні підходи до статистичного тестування, аналізуються переваги та недоліки існуючих програмних засобів, визначаються вимоги до нового програмного інструменту, який розробляється у межах цієї роботи. Окрема увага приділяється питанням автоматизації процесу тестування, зручності інтерфейсу, можливості розширення та інтеграції з іншими системами, що є важливими для впровадження сучасних підходів до аналізу криптографічних алгоритмів.

Таким чином, перший розділ закладає теоретичну та методологічну основу для подальшого дослідження, формулює основну задачу роботи, обґрунтовує актуальність теми та визначає напрямки подальшого розвитку у сфері оцінювання стійкості потокових шифрів до статистичних атак [24].

Постановка задачі

Забезпечення надійного захисту інформації є однією з ключових проблем сучасного інформаційного суспільства. З розвитком цифрових технологій, зростанням обсягів переданої та оброблюваної інформації, а також ускладненням методів криптоаналізу, питання стійкості криптографічних алгоритмів, зокрема потокових шифрів, набуває особливої актуальності. Потокові шифри широко застосовуються у різних сферах: від захисту мобільного зв’язку, бездротових мереж, до фінансових транзакцій та захищених каналів зв’язку в промислових системах.

Незважаючи на те, що багато сучасних потокових шифрів мають формально доведену стійкість до певних типів атак, на практиці їх реалізації можуть містити недоліки, які призводять до появи статистичних закономірностей у вихідних послідовностях. Такі закономірності можуть бути використані зловмисниками для проведення статистичних атак, що, у свою чергу, може призвести до часткового або повного розкриття зашифрованої інформації чи компрометації ключа. Відомо, що навіть незначні відхилення від ідеальної випадковості у ключовому потоці можуть суттєво знизити рівень захисту системи.

Класичні підходи до оцінювання стійкості потокових шифрів часто є трудомісткими, потребують значних обчислювальних ресурсів, а також не завжди

враховують сучасні вимоги до автоматизації, інтеграції з іншими програмними засобами та зручності інтерпретації результатів. Більшість існуючих програмних рішень або орієнтовані на вузькоспеціалізовані задачі, або мають складний інтерфейс, що ускладнює їх використання для широкого кола фахівців. Це створює додаткові труднощі для впровадження сучасних методів аналізу у практику, а також для навчання та підготовки нових фахівців у галузі інформаційної безпеки.

У зв’язку з цим виникає необхідність у розробці нового підходу до оцінювання стійкості потокових шифрів, який би поєднував у собі:

- використання сучасних статистичних тестів, що дозволяють виявляти різні типи невипадковості у вихідних послідовностях;

- автоматизацію процесу тестування та збору результатів;

- зручну візуалізацію для швидкої інтерпретації отриманих даних;

- можливість розширення та інтеграції з іншими системами;

- підтримку різних форматів даних для імпорту, експорту та збереження результатів;

- інтуїтивний інтерфейс, доступний для користувачів різного рівня підготовки [17].

Таким чином, основна задача цієї дисертаційної роботи полягає у вдосконаленні методів статистичного оцінювання стійкості потокових шифрів до статистичних атак шляхом розробки програмного засобу, який дозволяє:

- автоматизовано генерувати послідовності різними потоковими генераторами;

- виконувати комплекс статистичних тестів для виявлення закономірностей;

- накопичувати, зберігати та візуалізувати результати тестування;

- порівнювати стійкість різних генераторів та робити обґрунтовані висновки щодо їх придатності для використання у криптографічних системах;

- забезпечити можливість розширення функціоналу, інтеграції з іншими програмними продуктами, адаптації до нових вимог [26, 3].

Розв’язання цієї задачі дозволить підвищити ефективність процесу оцінювання стійкості потокових шифрів, сприятиме впровадженню сучасних

підходів до аналізу криптографічних алгоритмів, забезпечить надійний захист інформації в умовах зростаючих загроз, а також створить підґрунтя для подальших наукових досліджень і практичного впровадження результатів у різних сферах діяльності.

1.1 Теоретичні основи потокового шифрування

Загальна характеристика потокових шифрів

Потокові шифри (stream ciphers) — це клас симетричних криптографічних алгоритмів, які здійснюють шифрування даних шляхом поелементного (побітового або побайтового) перетворення відкритого тексту за допомогою псевдовипадкової послідовності, що генерується на основі секретного ключа. На відміну від блочних шифрів, які працюють з фіксованими блоками даних, потокові шифри забезпечують гнучкість, високу швидкість обробки інформації та мінімальні затримки, що особливо важливо для застосувань у реальному часі, таких як мобільний зв’язок, потокове відео, VoIP, бездротові мережі тощо [41, 36].

Основна ідея потокового шифрування полягає у створенні ключового потоку (key stream), який має бути максимально схожим на ідеально випадкову послідовність. Відкритий текст поелементно поєднується з ключовим потоком (зазвичай операцією XOR), утворюючи шифротекст. Дешифрування здійснюється аналогічно — шифротекст поєднується з тим самим ключовим потоком, і в результаті відновлюється відкритий текст.

Вимоги до ключового потоку

Ключовий потік повинен відповідати таким основним вимогам:

- випадковість: послідовність має бути статистично невідрізненною від ідеально випадкової.

- непередбачуваність: знання частини потоку не повинно дозволяти передбачити інші його елементи.

- великий період: період повторення послідовності має бути достатньо великим, щоб уникнути циклічності під час реального використання.

- відсутність кореляцій: не повинно бути статистичних залежностей між елементами потоку.

Порушення хоча б однієї з цих вимог може призвести до зниження стійкості шифру та можливості проведення ефективних атак.

Основні типи генераторів ключового потоку

Лінійний зсувний регістр зворотного зв'язку (LFSR)

LFSR — це один із найпростіших і найпоширеніших генераторів псевдовипадкових послідовностей. Його робота базується на лінійних рекурентних співвідношеннях, де кожен новий біт визначається як лінійна комбінація попередніх бітів із використанням операції XOR. LFSR має просту апаратну реалізацію, великий період, але не є криптографічно стійким у чистому вигляді, оскільки його вихідна послідовність піддається лінійному аналізу [16, 18].

Blum Blum Shub (BBS)

BBS — це криптографічно стійкий генератор, який базується на складності факторизації великих чисел. Його вихідна послідовність має високий рівень випадковості, але генерація кожного біта потребує значних обчислювальних ресурсів, що обмежує його використання у високошвидкісних системах [12].

RC4

RC4 — один із найвідоміших потокових шифрів, який широко використовувався у протоколах WEP, SSL/TLS. Його перевагами є простота реалізації та висока швидкість, але згодом були виявлені серйозні вразливості, пов'язані з початковою ініціалізацією та статистичними відхиленнями у вихідному потоці [7].

Mersenne Twister

Mersenne Twister — це генератор з дуже великим періодом (2^19937−1), який часто використовується у програмному забезпеченні для моделювання та симуляцій. Однак він не є криптографічно стійким, оскільки його внутрішній стан можна відновити за відносно невеликою кількістю вихідних значень.

System Random

System Random — це генератор, реалізований у стандартних бібліотеках мови програмування Python, який може використовувати різні алгоритми, зокрема й криптографічно стійкі. Якість такого генератора залежить від конкретної реалізації та налаштувань операційної системи [6].

Порівняння потокових і блочних шифрів

Потокові шифри мають низку переваг над блочними:

- вища швидкість обробки даних у реальному часі;
- можливість шифрування даних довільної довжини без додаткового доповнення;
- менші затримки при обробці потоків інформації.

Водночас, блочні шифри забезпечують кращу стійкість до деяких типів атак (наприклад, диференційного та лінійного криптоаналізу), але можуть бути менш ефективними для потокових застосувань.

1.2 Класифікація та суть статистичних атак

Статистичні атаки є одним із найефективніших інструментів криптоаналізу, що дозволяють виявити слабкі місця у потокових шифрах шляхом аналізу вихідних послідовностей. На відміну від атак, які базуються на математичних вразливостях алгоритму, статистичні атаки використовують властивості реальних реалізацій шифрів, які можуть містити відхилення від ідеальної випадковості. Такі відхилення часто виникають через недосконаліть генераторів псевдовипадкових послідовностей, помилки у виборі параметрів або особливості апаратної/програмної реалізації [11].

Типи статистичних атак визначаються не лише видом застосованого тесту, а й метою та способом використання отриманих результатів. Основні типи статистичних атак включають:

Атаки на відновлення ключа

Метою таких атак є використання статистичних закономірностей у вихідній

послідовності для часткового або повного відновлення секретного ключа шифру. Зловмисник аналізує результати статистичних тестів, виявляє патерни, які можуть бути пов’язані з певними значеннями ключа, і намагається скоротити простір перебору або навіть відновити ключ повністю [19].

Атаки на відновлення відкритого тексту

У цьому випадку статистичні властивості використовуються для відновлення частини або всього відкритого тексту без знання ключа. Наприклад, якщо у вихідній послідовності є закономірності, які відповідають певним фрагментам відкритого тексту, зловмисник може скористатися цим для дешифрування повідомлення [19].

Атаки на структуру генератора

Такі атаки спрямовані на виявлення внутрішньої структури генератора (наприклад, параметрів LFSR, tap-коефіцієнтів, початкового стану). Аналізуючи статистичні властивості вихідної послідовності, зловмисник може відновити частину або всю структуру генератора, що дозволяє йому генерувати ідентичні послідовності та дешифрувати повідомлення [19].

Комплексні атаки

Поєднують кілька статистичних підходів і тестів для підвищення ефективності криптоаналізу. Зловмисник може комбінувати результати різних тестів, використовувати машинне навчання для автоматичної класифікації послідовностей, застосовувати багатоступеневі стратегії для поступового звуження простору можливих ключів або параметрів генератора [19].

Інструменти статистичних атак

Для реалізації статистичних атак застосовуються різноманітні статистичні тести, зокрема:

- частотний аналіз (оцінка спів відношення нулів та одиниць);
- аналіз серій (вивчення довжин та кількості послідовних однакових символів);
- аналіз автокореляції (дослідження залежностей між елементами на різних відстанях);

- комплексні пакети тестів (NIST STS, Dieharder, TestU01), які містять десятки різних тестів для виявлення різних типів невипадковості.

Результати цих тестів дозволяють зловмиснику зробити висновки про наявність закономірностей у ключовому потоці, обрати подальшу стратегію атаки, скоротити простір перебору ключів або навіть відновити частину відкритого тексту чи структуру генератора. Саме тому питання статистичної стійкості потокових шифрів є надзвичайно важливим для сучасної криптографії.

1.3 Огляд методів статистичного тестування

Статистичне тестування є основним інструментом для оцінювання якості псевдовипадкових послідовностей, що генеруються потоковими шифрами. Метою статистичних тестів є виявлення відхилень від ідеальної випадковості, які можуть свідчити про наявність закономірностей у ключовому потоці та потенційну вразливість шифру до статистичних атак.

Основні статистичні тести:

Частотний тест (Frequency Test)

Перевіряє, чи кількість нулів і одиниць у послідовності приблизно однакова. Значні відхилення від рівномірного розподілу можуть свідчити про невипадковість, що є ознакою слабкості генератора. Тест є базовим і дозволяє швидко виявити глобальні закономірності [14].

Тест серій (Runs Test)

Аналізує довжини та кількість послідовних однакових символів (серій). В ідеально випадковій послідовності довжини серій мають підкорятися певному розподілу. Відхилення від цього розподілу може бути ознакою слабкості генератора, наявності патернів або періодичності [14].

Серійний тест (Serial Test)

Оцінює частоти появи всіх можливих підпослідовностей певної довжини (наприклад, пар або трійок бітів). Нерівномірний розподіл таких підпослідовностей

може свідчити про наявність патернів, які знижують стійкість шифру до статистичних атак [14].

Покер-тест (Poker Test)

Аналізує розподіл підпослідовностей фіксованої довжини, подібно до роздачі карт у покері. Виявляє відхилення від рівномірного розподілу, що може бути ознакою невипадковості, повторюваних патернів або структурних недоліків генератора.

Автокореляційний тест (Autocorrelation Test)

Досліджує залежності між елементами послідовності на різних відстанях (лагів). В ідеально випадковій послідовності автокореляція має бути близькою до нуля. Виявлення значущих автокореляційних зв'язків може свідчити про періодичність або залежності, які знижують стійкість шифру [14].

Комплексне застосування тестів

Для підвищення достовірності оцінки якості генератора доцільно використовувати не окремі тести, а їх комбінації, а також комплексні пакети (NIST STS, Dieharder, TestU01). Це дозволяє виявити різні типи закономірностей, які можуть бути не помітні при використанні лише одного тесту.

Статистичне тестування є обов'язковим етапом при розробці, впровадженні та сертифікації криптографічних генераторів, а також при аналізі їх стійкості до атак.

1.4 Огляд існуючих програмних засобів для статистичного аналізу

Сучасний розвиток криптографії та інформаційної безпеки супроводжується появою численних програмних засобів для статистичного аналізу псевдовипадкових послідовностей. Вибір відповідного інструменту залежить від вимог до гнучкості, автоматизації, зручності використання, а також від специфіки задачі. Основні програмні пакети:

NIST Statistical Test Suite (NIST STS)

NIST STS — це один із найавторитетніших і найпоширеніших пакетів для статистичного тестування генераторів випадкових чисел. Він містить понад 15 різних тестів, включаючи частотний, серійний, покер-тест, тест на довгі серії, тест на ентропію тощо. Пакет широко використовується для сертифікації криптографічних рішень у США та світі. Основними пере вагами є наукова обґрунтованість тестів, гнучкість налаштувань, можливість автоматизації тестування. Недоліками є складність інтерфейсу, необхідність підготовки даних у спеціальному форматі, а також відносно висока вимогливість до ресурсів [28].

Dieharder

Dieharder — це розширена версія класичного пакета Diehard, яка містить як класичні, так і сучасні тести для оцінки генераторів випадкових чисел. Пакет підтримує автоматичне тестування, має командний інтерфейс, дозволяє генерувати докладні звіти. Dieharder часто використовується для попередньої оцінки якості генераторів у наукових дослідженнях, а також для порівняння різних реалізацій генераторів [4].

TestU01

TestU01 — це універсальний програмний пакет, який містить велику кількість тестів для оцінки якості генераторів псевдовипадкових чисел. Його особливістю є можливість створення власних сценаріїв тестування, гнучке налаштування параметрів, а також підтримка різних мов програмування. TestU01 є потужним інструментом для глибокого аналізу, але вимагає від користувача високого рівня підготовки, знання мов програмування та статистики [7].

Інші засоби

Окрім вищезазначених, існують численні бібліотеки та утиліти для статистичного аналізу, зокрема Python-бібліотеки (scipy.stats, numpy.random), спеціалізовані модулі для MATLAB, R, C/C++ тощо. Деякі з них орієнтовані на інтеграцію у власні програмні продукти, інші — на проведення наукових експериментів, навчання або швидку перевірку якості генераторів.

Для узагальнення розглянутих існуючих програмних засобів наведено порівняльну таблицю 1.1.

Таблиця 1.1 - Порівняння існуючих засобів статистичного тестування

Критерій	**NIST STS**	**Dieharder**	**TestU01**
Кількість тестів	17	21	109
Гнучкість налаштування	Дуже висока	Висока	Дуже висока
Автоматизація тестування	Низька	Висока	Висока
Зручність використання	Низька	Середня	Низька
Інтерфейс	Консольний	Консольний	Програмний
Швидкість роботи	Середня	Висока	Суттєво залежить від тесту
Вимоги до користувача	Середні	Середні	Дуже високі
Можливість інтеграції	Низька	Середня	Дуже висока
Переваги	Стандартизованість	Автоматизація	Гнучкість
Недоліки	Складність використання	Невідповідність стандартам	Складність інтеграції

Вибір програмного засобу залежить від цілей дослідження, рівня підготовки користувача, вимог до автоматизації, гнучкості, можливості інтеграції з іншими системами, підтримки різних форматів даних, а також від наявності документації та підтримки спільноти.

Вимоги до сучасного програмного засобу:

- можливість додавання нових генераторів та тестів;
- зручний графічний інтерфейс для користувача;
- автоматизація процесу тестування та візуалізації результатів;
- підтримка імпорту/експорту даних у різних форматах;
- можливість інтеграції у більші системи [18].

У межах цієї дисертації розробляється програмний засіб, який враховує зазначені вимоги, поєднує переваги існуючих рішень і дозволяє ефективно оцінювати стійкість потокових шифрів до статистичних атак у сучасних умовах.

Висновки до розділу 1

У першому розділі магістерської дисертації було здійснено комплексний аналіз теоретичних та практичних аспектів оцінювання стійкості потокових шифрів до статистичних атак. Проведено аналіз сучасних підходів, методів та програмних засобів, що використовуються для статистичного тестування генераторів потокових шифрів, і виявлено низку недоліків та невідповідностей існуючих рішень сучасним вимогам автоматизації, гнучкості, інтеграції та зручності інтерпретації результатів.

На основі цього сформульовано постановку задачі, яка полягає у вдосконаленні методів статистичного аналізу та розробці програмного засобу для автоматизованого тестування генераторів потокових шифрів.

Розглянуто теоретичні основи потокового шифрування, визначено ключові вимоги до генераторів псевдовипадкових послідовностей, зокрема — необхідність забезпечення високого рівня випадковості, непередбачуваності та великого періоду. Показано, що якість ключового потоку безпосередньо впливає на стійкість шифру до статистичних атак, а отже — і на загальний рівень захисту інформації.

Детально проаналізовано класифікацію та суть статистичних атак. Визначено, що навіть незначні відхилення від ідеальної випадковості у вихідних послідовностях можуть бути використані зловмисниками для компрометації системи, відновлення частини відкритого тексту або ключа. Підкреслено важливість застосування комплексного підходу до виявлення таких відхилень.

Проведено огляд основних статистичних тестів, які використовуються для оцінювання якості псевдовипадкових послідовностей: частотного, серійного, покер-тесту, автокореляційного та інших. Відзначено, що для підвищення достовірності результатів доцільно використовувати не окремі тести, а їх комбінації, а також комплексні пакети (NIST STS, Dieharder, TestU01).

Таким чином, у першому розділі сформульовано наукову проблему, обґрунтовано актуальність і практичну значущість дослідження, визначено основні напрямки подальшої роботи. Отримані результати та висновки стали підґрунтям для розробки математичних моделей, вибору алгоритмів та реалізації програмного засобу, що розглядаються у наступних розділах дисертації.

2 МАТЕМАТИЧНІ ОСНОВИ ТА АЛГОРИТМИ ОЦІНЮВАННЯ СТІЙКОСТІ ПОТОКОВИХ ШИФРІВ

У другому розділі дисертації розглянуто математичні основи, методи та алгоритми, які лежать в основі оцінювання стійкості потокових шифрів до статистичних атак. Саме математичне підґрунтя дозволяє не лише формалізувати процес аналізу, а й забезпечити об’єктивність та достовірність отриманих результатів. У цьому розділі детально розглядаються моделі генераторів псевдовипадкових послідовностей, що використовуються у сучасних потокових шифрах, а також математичні основи статистичних тестів, які застосовуються для виявлення закономірностей у ключових потоках.

Актуальність математичного аналізу обумовлена тим, що саме на цьому етапі визначаються критерії якості генераторів, формулюються вимоги до їхніх властивостей, а також обґрунтовується вибір конкретних тестів для оцінювання стійкості. Математичне моделювання дозволяє не лише описати роботу генераторів, а й передбачити їхню поведінку у різних умовах, оцінити ймовірність виникнення статистичних відхилень та визначити межі застосування того чи іншого методу.

У розділі також розглядаються алгоритми реалізації статистичних тестів, які є основним інструментом для виявлення слабких місць у потокових шифрах. Особлива увага приділяється питанням вибору параметрів тестування, оптимізації обчислень, а також адаптації класичних тестів до специфіки потокових шифрів. Описуються підходи до побудови комплексних критеріїв оцінювання, які дозволяють враховувати різні аспекти випадковості та стійкості.

Окремий підпункт виділено для обґрунтування вибору програмних засобів для реалізації запропонованих методів. Враховано сучасні тенденції у розробці програмного забезпечення, вимоги до гнучкості, масштабованості, зручності для користувача та можливості інтеграції з іншими системами. Пояснюється, чому для реалізації програмного засобу було обрано мову програмування Python, бібліотеки numpy, scipy, PyQt5 та інші інструменти.

У підсумку, другий розділ закладає теоретичну та методологічну основу для подальшої програмної реалізації системи автоматизованого аналізу стійкості потокових шифрів. Розглянуті у цьому розділі моделі, методи та алгоритми є ключовими для досягнення поставленої мети дослідження та забезпечення високої якості отриманих результатів.

2.1 Математичні основи генераторів потокових шифрів

Потокові шифри є важливим класом симетричних криптографічних алгоритмів, які забезпечують шифрування даних у реальному часі шляхом поелементного перетворення відкритого тексту. Основою їхньої роботи є генератори псевдовипадкових послідовностей, які повинні відповідати суворим математичним вимогам до випадковості, непередбачуваності та великого періоду. У цьому підрозділі розглядаються математичні моделі основних генераторів, що використовуються у потокових шифрах, їхні властивості, переваги та недоліки.

Лінійний зсувний регістр зворотного зв'язку (LFSR)

LFSR (Linear Feedback Shift Register) — це один із найпростіших і найпоширеніших генераторів псевдовипадкових послідовностей, який широко використовується у потокових шифрах. Математична модель LFSR базується на лінійних рекурентних співвідношеннях над полем GF:

$$s_n = c_1 s_{n-1} \oplus c_2 s_{n-2} \oplus \ldots \oplus c_k s_{n-k} \quad (1)$$

де Sn — n-й біт послідовності, Ci — коефіцієнти зворотного зв'язку (0 або 1), k — довжина регістра, $\oplus$ — операція додавання за модулем 2.

LFSR характеризується простотою апаратної реалізації, високою швидкістю та великим періодом (до 2k−1), якщо поліном зворотного зв'язку є примітивним. Однак, чистий LFSR не є криптографічно стійким, оскільки його вихідна послідовність піддається лінійному аналізу (алгоритм Берлекемпа-Мессі дозволяє відновити структуру регістра за відносно короткою частиною послідовності). Для

підвищення стійкості використовують нелінійні комбінації кількох LFSR, додаткові функції зворотного зв’язку, фільтруючі функції тощо [35].

Генератор Blum Blum Shub (BBS)

BBS — це криптографічно стійкий генератор, математична модель якого базується на складності факторизації великих чисел. Генерація наступного елемента послідовності здійснюється за формулою:

$$x_{n+1} = x_n^2 \, mod \, N \tag{2}$$

де N=p · q, p і q — великі прості числа, що задовольняють умову p≡q≡3 mod 4. Вихідний біт визначається як найменш значущий біт xn+1.

BBS має доведені властивості криптографічної стійкості, але є досить повільним через необхідність виконання операцій з великими числами. Застосовується переважно у випадках, де критично важлива безпека, а не швидкість [39].

Потоковий шифр RC4

RC4 — це один із найвідоміших потокових шифрів, який був розроблений у 1987 році Роном Рівестом. Його математична модель базується на перестановці елементів у масиві S довжиною 256 байтів та генерації ключового потоку шляхом ітеративного обміну елементів.

Алгоритм складається з двох основних етапів: ініціалізації (KSA — Key Scheduling Algorithm) та генерації ключового потоку (PRGA — Pseudo-Random Generation Algorithm).

RC4 характеризується високою швидкістю та простотою реалізації, але має відомі вразливості, пов’язані з початковою ініціалізацією та статистичними відхиленнями у вихідному потоці [24].

Mersenne Twister

Mersenne Twister — це генератор псевдовипадкових чисел з дуже великим періодом (219937−1), який широко використовується у програмному забезпеченні для моделювання та симуляцій. Його математична модель базується на використанні рекурентних співвідношень та спеціальних бітових операцій.

Попри високу якість випадковості для статистичних задач, Mersenne Twister не є криптографічно стійким, оскільки його внутрішній стан можна відновити за відносно невеликою кількістю вихідних значень [21].

System Random

System Random — це генератор, реалізований у стандартних бібліотеках мови програмування Python, який може використовувати різні алгоритми, зокрема й криптографічно стійкі (наприклад, /dev/urandom у Linux). Якість такого генератора залежить від конкретної реалізації та налаштувань операційної системи.

2.2 Математичне обґрунтування статистичних тестів

Статистичні тести є основним інструментом для оцінювання якості псевдовипадкових послідовностей, що генеруються потоковими шифрами. Їх математичне обґрунтування базується на теорії ймовірностей, математичній статистиці та теорії випадкових процесів. У цьому підпункті розглядаються основні принципи побудови статистичних тестів, критерії оцінювання випадковості, а також математичні формули, які лежать в основі кожного тесту [2].

Основи статистичного тестування

Метою статистичного тестування є перевірка гіпотези про те, що досліджувана послідовність є реалізацією ідеального випадкового процесу. Для цього формулюється нульова гіпотеза H0: "послідовність є випадковою", і альтернативна гіпотеза H1: "послідовність не є випадковою".

Результатом тесту є p-value — ймовірність отримати спостережуване або ще більш екстремальне значення статистики тесту за умови справедливості H0. Якщо p-value менше за обране порогове значення (зазвичай 0.01 або 0.05), нульова гіпотеза відхиляється, і послідовність вважається невипадковою.

Математичне обґрунтування частотного тесту

Частотний тест перевіряє, чи кількість нулів і одиниць у послідовності приблизно однакова [9].

Нехай n — довжина послідовності, S — кількість одиниць.

Статистика тесту:

$$S_n = \sum_{i=1}^{n} (2x_i - 1) \quad (3)$$

де xi — і-й біт послідовності (0 або 1).

p-value обчислюється як:

$$p = erfc\left(\frac{|s_n|}{\sqrt{2n}}\right) \quad (4)$$

де erfc — додаткова функція помилок.

Математичне обґрунтування тесту серій

Тест серій (runs test) аналізує довжини та кількість послідовних однакових символів (серій). В ідеально випадковій послідовності очікувана кількість серій:

$$E = 2n\pi(1 - \pi) \quad (5)$$

де π — відносна частота одиниць у послідовності.

Статистика тесту:

$$Z = \frac{|V_n - E|}{2\sqrt{2n\pi}(1-\pi)} \quad (6)$$

де Vn — фактична кількість серій.

p-value визначається через нормальний розподіл:

$$p = 2(1 - \Phi(|Z|)) \quad (7)$$

де Φ — функція розподілу стандартного нормального закону.

Математичне обґрунтування серійного тесту

Серійний тест (serial test) оцінює частоти появи всіх можливих підпослідовностей певної довжини (наприклад, кількох пар або трійок бітів).

Використовується критерій Хі-квадрат:

$$x^2 = \sum_{i=1}^{k} \frac{o_i - E_i}{E_i} \quad (8)$$

де Oi — спостережувана кількість підпослідовностей і-го типу, Ei — очікувана кількість (для рівномірного розподілу). p-value визначається за розподілом χ2 з відповідною кількістю ступенів свободи [8].

Математичне обґрунтування покер-тесту

Покер-тест аналізує розподіл підпослідовностей фіксованої довжини (наприклад, блоків по 4 біти). Для k можливих комбінацій та m блоків:

$$x^2 = \frac{k}{m}\sum_{i=1}^{k} n_i^2 - m \tag{9}$$

де ni — кількість появи i-ї комбінації.

p-value визначається за розподілом $\chi 2$ з k−1 ступенями свободи.

Математичне обґрунтування автокореляційного тесту

Автокореляційний тест досліджує залежності між елементами послідовності на різних відстанях (лагів).

Статистика автокореляції для лагу d:

$$A_d = \sum_{i=i}^{n-d} (x_i \oplus x_{i+d}) \tag{10}$$

де $\oplus$ — операція XOR.

Очікуване значення для випадкової послідовності — близько (n−d)/2. p-value визначається через нормальний розподіл для біноміального закону.

Вибір порогових значень та інтерпретація результатів

Для кожного тесту обирається порогове значення p-value (зазвичай 0.01 або 0.05). Якщо p-value менше порогу, послідовність вважається невипадковою. Комплексне застосування кількох тестів дозволяє підвищити достовірність оцінки та виявити різні типи закономірностей.

2.3 Алгоритми реалізації статистичних тестів

Реалізація статистичних тестів для оцінювання стійкості потокових шифрів до статистичних атак вимагає не лише математичного обґрунтування, а й чітко визначених алгоритмічних процедур. У цьому підпункті наведено покроковий опис алгоритмів реалізації основних тестів, особливості їх застосування до потокових шифрів, а також порівняння складності та ефективності [27].

Алгоритм частотного тесту (Frequency Test)

Мета: перевірити, чи кількість нулів і одиниць у послідовності приблизно однакова.

Алгоритм показано на рисунку 2.1.

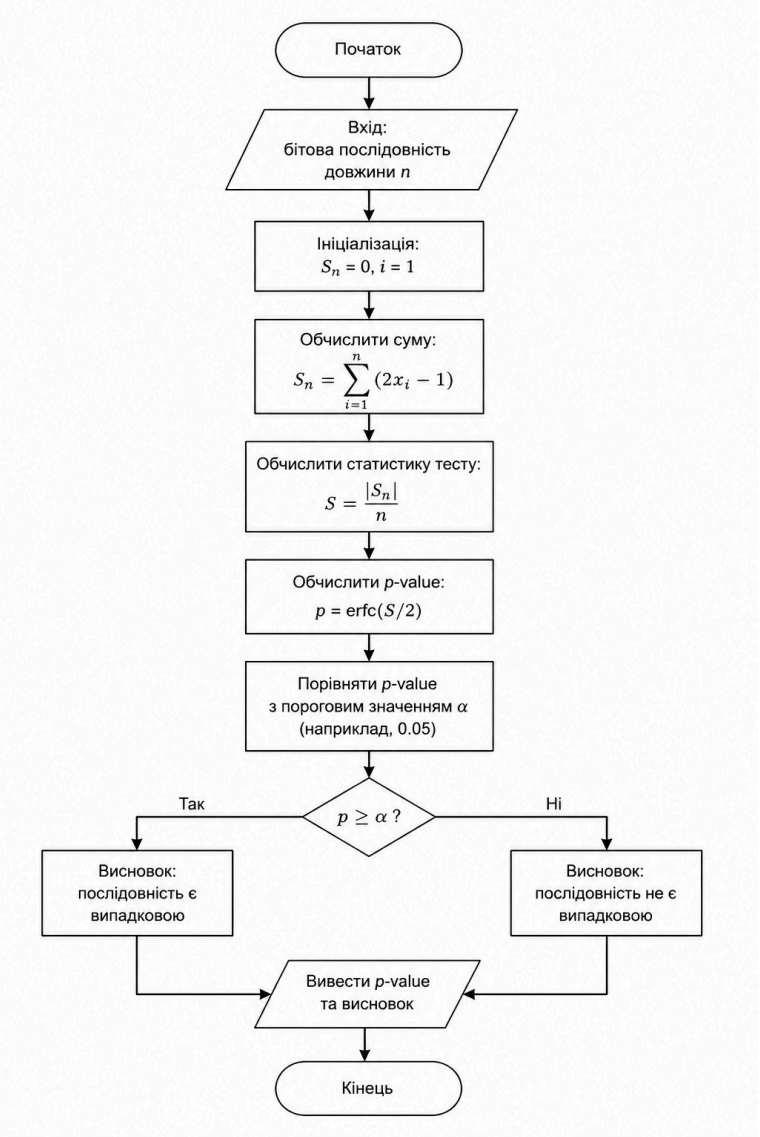

Рисунок 2.1 - Алгоритм частотного тесту

Особливості:

Тест простий у реалізації, має лінійну складність O(n), підходить для великих послідовностей.

Алгоритм тесту серій (Runs Test)

Мета: Перевірити, чи довжини та кількість серій (послідовностей однакових бітів) відповідають очікуваному розподілу.

Алгоритм показано на рисунку 2.2.

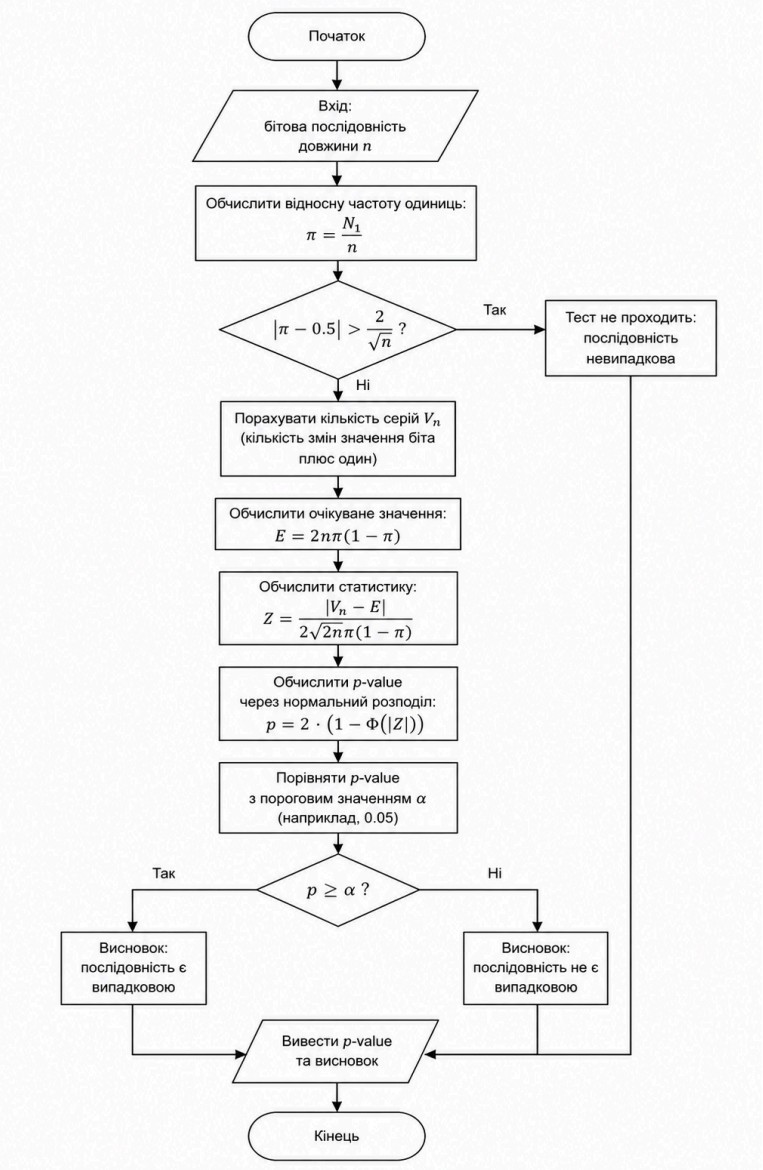

Рисунок 2.2 - Алгоритм тесту серій

Особливості:

Тест враховує не лише співвідношення нулів і одиниць, а й структуру послідовності.

Алгоритм серійного тесту (Serial Test)

Мета: Оцінити частоти появи всіх можливих підпослідовностей певної довжини.

Алгоритм показано на рисунку 2.3.

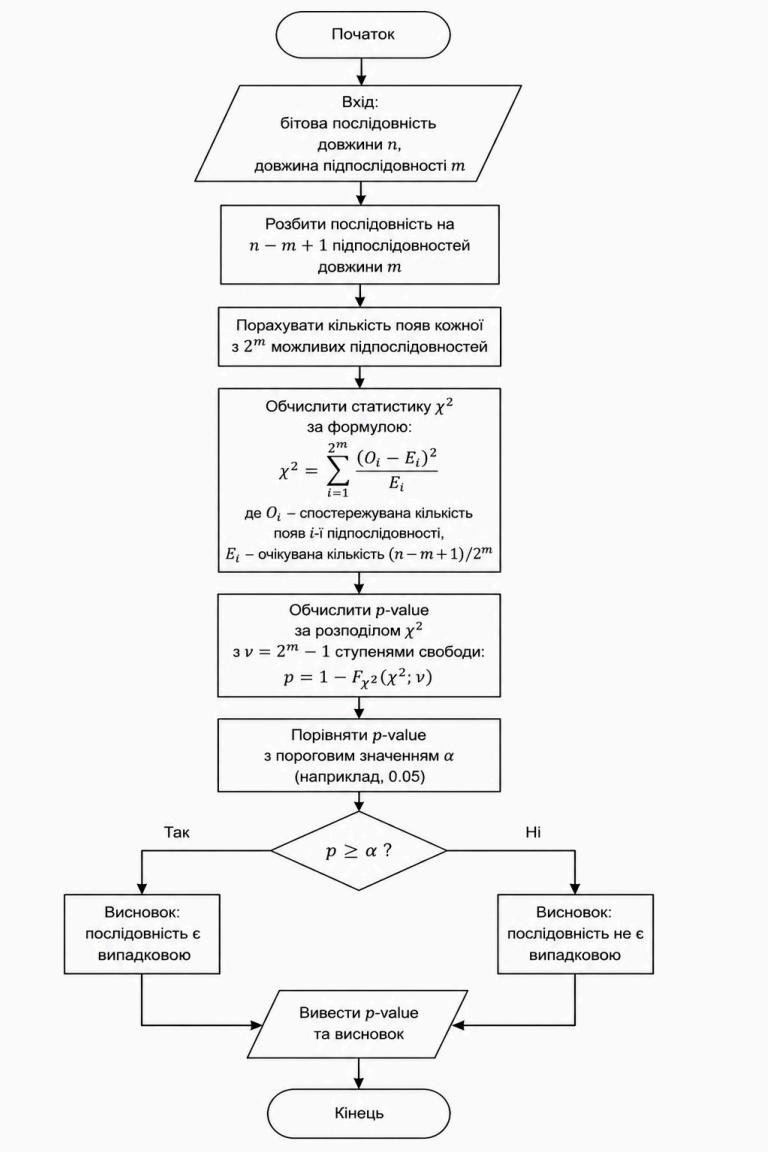

Рисунок 2.3 - Алгоритм серійного тесту

Особливості:

Тест дозволяє виявити локальні закономірності, але складність зростає експоненціально з m.

Алгоритм покер-тесту (Poker Test)

Мета: Аналізувати розподіл підпослідовностей фіксованої довжини (наприклад, блоків по 4 біти).

Алгоритм показано на рисунку 2.4.

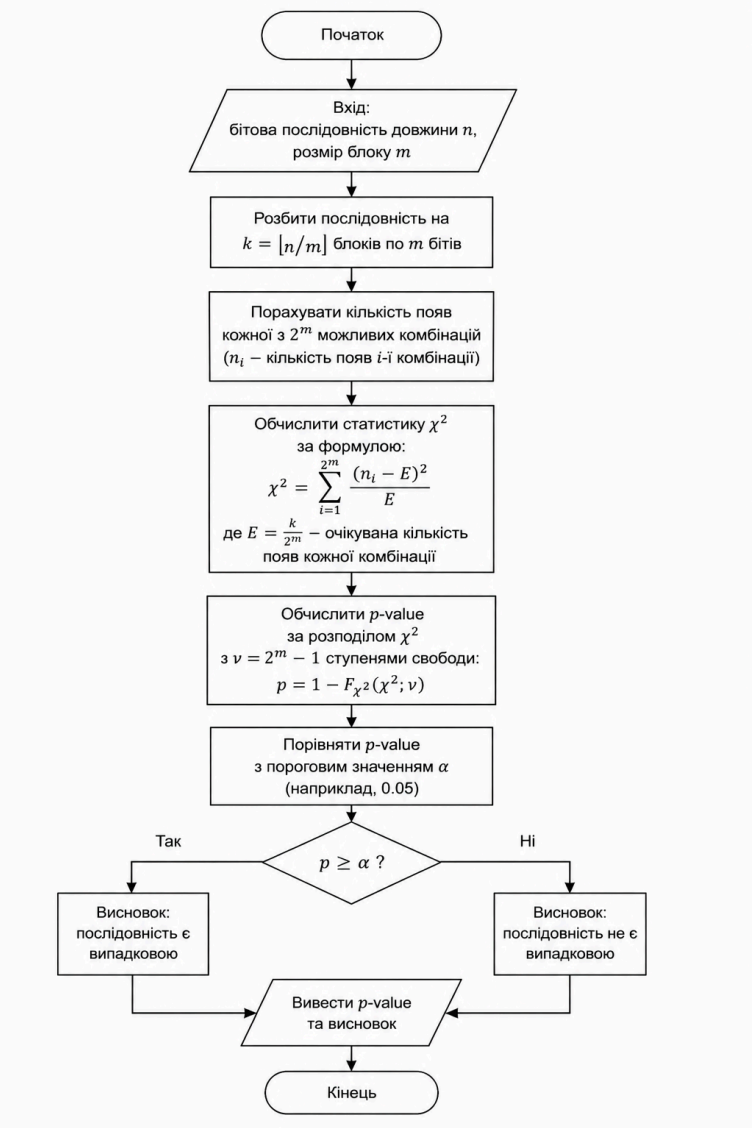

Рисунок 2.4 - Алгоритм покер-тесту

Особливості:

Тест чутливий до повторюваних патернів у підпослідовностях.

Алгоритм автокореляційного тесту (Autocorrelation Test)

Мета: Виявити залежності між елементами на різних відстанях.

Алгоритм показано на рисунку 2.5.

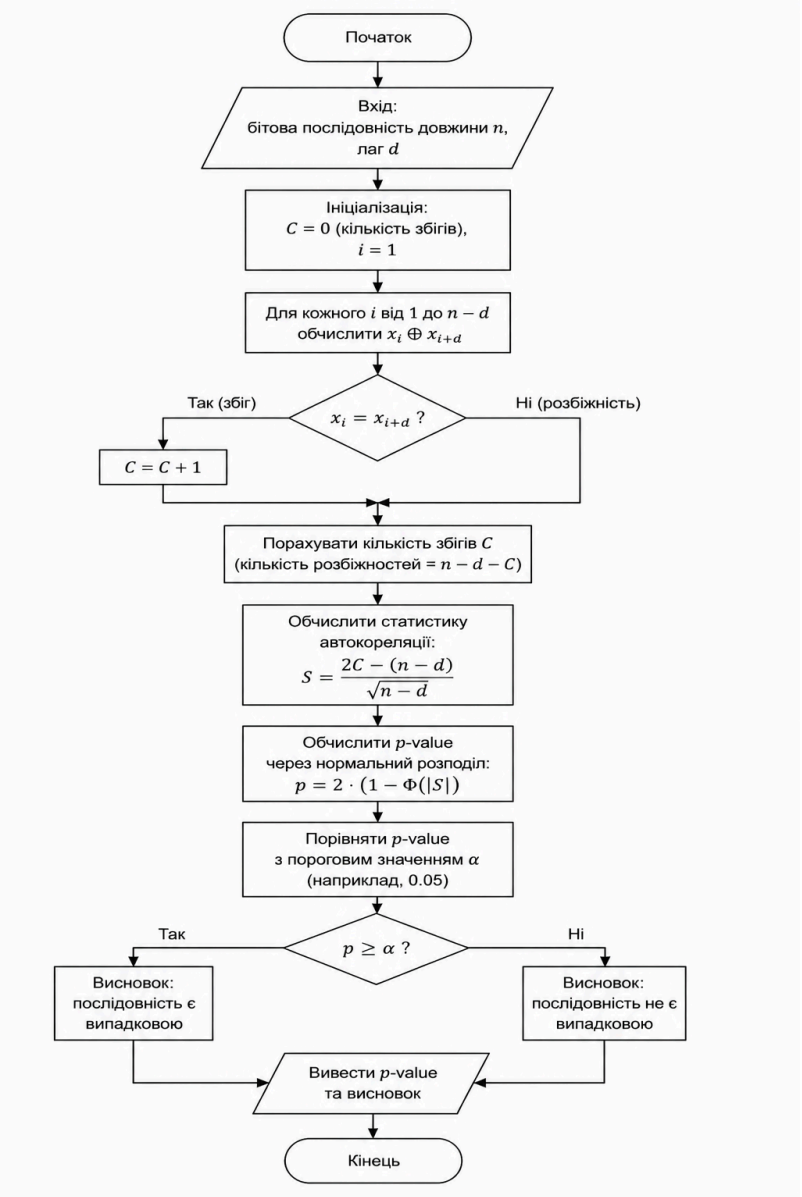

Рисунок 2.5 - Алгоритм автокореляційного тесту

Особливості:

Тест дозволяє виявити періодичність або залежності у послідовності.

Особливості застосування до потокових шифрів

При застосуванні статистичних тестів до потокових шифрів важливо враховувати достатню довжину тестової послідовності для достовірності результатів, можливість комбінування тестів для комплексної оцінки, обирати параметри (розмір блоку, лаг, довжина підпослідовності) залежно від типу генератора, автоматизувати процес тестування для підвищення ефективності аналізу [33].

2.4 Вибір та обґрунтування програмних засобів

2.4.1 Вибір мови програмування

Для реалізації програмного засобу було обрано мову програмування Python. Python є однією з найпопулярніших мов у світі, особливо у сфері наукових досліджень, аналізу даних та розробки прототипів. Основними перевагами Python є: простота синтаксису — дозволяє швидко писати, читати та підтримувати код, що особливо важливо для командної роботи та подальшого розширення функціоналу, велика кількість бібліотек — Python має розвинену екосистему для наукових обчислень, статистики, машинного навчання, візуалізації, роботи з графічними інтерфейсами, кросплатформеність — програми на Python можуть запускатися на Windows, Linux, MacOS без суттєвих змін у коді, активна спільнота — наявність великої кількості прикладів, документації, форумів, що спрощує вирішення проблем під час розробки, легкість інтеграції — Python легко взаємодіє з іншими мовами (C/C++, Java), що дозволяє розширювати функціонал у разі потреби [33].

Вибір Python також обумовлений тим, що ця мова широко використовується у сучасній криптографії, аналізі даних та розробці наукового програмного забезпечення.

2.4.2 Вибір бібліотек для математичних обчислень

Для реалізації математичних операцій, статистичних тестів та обробки даних у програмному засобі використано такі бібліотеки:

NumPy

Основна бібліотека для роботи з багатовимірними масивами, векторизованими операціями, генерацією випадкових чисел. NumPy забезпечує високу швидкість обчислень завдяки використанню оптимізованих C-реалізацій. Має розширення - SciPy [7]. Розширює можливості NumPy, містить широкий спектр статистичних функцій: обчислення p-value, хі-квадрат, нормального розподілу, функцій помилок, тощо. SciPy дозволяє реалізувати більшість класичних статистичних тестів без необхідності писати складний код "з нуля".

Pandas

Використовується для зручної обробки табличних даних, формування звітів, збереження результатів у різних форматах (CSV, Excel). Pandas дозволяє легко групувати, фільтрувати, агрегувати дані, що важливо для аналізу результатів тестування [6].

Використання цих бібліотек забезпечує високу точність, швидкість та надійність обчислень, а також спрощує реалізацію складних статистичних алгоритмів.

2.4.3 Вибір засобів для візуалізації

Для візуалізації результатів тестування обрано бібліотеку matplotlib.
Matplotlib є стандартом де-факто для побудови графіків у Python і має такі переваги:

Гнучкість — підтримує різні типи графіків: гістограми, стовпчикові діаграми, лінійні графіки, кругові діаграми тощо.

Інтеграція з GUI — matplotlib легко інтегрується з PyQt5, дозволяючи відображати графіки безпосередньо у вікні програми.

Підтримка експорту — результати можна зберігати у різних форматах (PNG, CSV), що зручно для підготовки звітів та презентацій.

Використання matplotlib дозволяє зробити результати тестування наочними, що спрощує їх інтерпретацію та порівняння різних генераторів.

2.4.4 Вибір фреймворку для графічного інтерфейсу

Для створення графічного інтерфейсу користувача було обрано фреймворк PyQt5. PyQt5 — це потужний інструмент для розробки кросплатформених GUI-додатків на Python, який має такі переваги:

Сучасний вигляд — підтримка різних стилів оформлення, адаптація під операційну систему користувача.

Багатий набір віджетів — кнопки, поля введення, таблиці, діалогові вікна, інтеграція з графіками.

Гнучкість — можливість створення складних інтерфейсів з багатьма елементами керування.

Інтеграція з іншими бібліотеками — matplotlib, pandas, numpy.

Документованість і підтримка — велика кількість прикладів, документації, активна спільнота.

PyQt5 дозволяє створити інтуїтивно зрозумілий інтерфейс, який підходить як для досвідчених користувачів, так і для початківців [41].

2.4.5 Інші інструменти та середовище розробки

Середовище розробки (IDE)

Для написання та відлагодження коду використовувалися PyCharm та Visual Studio Code. Обидва середовища забезпечують підсвічування синтаксису, автодоповнення, інтеграцію з системами контролю версій, зручний відладчик.

Система контролю версій

Для зберігання коду, відстеження змін та командної роботи використовувався Git. Це дозволяє зберігати історію змін, працювати над різними гілками проєкту, швидко повертатися до попередніх версій [34].

Документація

Для документування коду використовувалися docstrings, README-файли, а також автоматичне генерування документації за допомогою Sphinx. Це спрощує підтримку та розвиток проєкту.

Тестування

Для перевірки коректності роботи окремих модулів використовувалися юніт-тести (бібліотека unittest).

2.4.6 Обґрунтування вибору

Вибір зазначених інструментів обумовлений наступними факторами:

Гнучкість та масштабованість

Модульна структура проєкту дозволяє легко додавати нові генератори, тести, функції без суттєвих змін у коді.

Зручність для користувача

Графічний інтерфейс спрощує роботу з програмою навіть для користувачів без досвіду програмування.

Автоматизація

Програма дозволяє автоматично запускати тести, збирати та зберігати результати, формувати звіти.

Відкритість та доступність

Усі використані інструменти є відкритими або безкоштовними для некомерційного використання, що робить проєкт доступним для широкого кола користувачів.

Можливість інтеграції

Програмний засіб може бути використаний як окремий інструмент або як частина більших систем аналізу, що важливо для наукових досліджень та практичних застосувань.

2.5 Характеристики розробленого програмного забезпечення

Розроблений програмний засіб для автоматизованого статистичного аналізу стійкості потокових шифрів вирізняється широким спектром функціональних можливостей, гнучкою архітектурою, зручністю використання та високою якістю реалізації. Його структура та функціонал були спроєктовані з урахуванням сучасних

вимог до наукового програмного забезпечення, що дозволяє ефективно вирішувати як навчальні, так і практичні задачі у сфері криптографії.

Однією з ключових функцій програми є генерація бітових послідовностей різними методами. Користувач може обирати один із кількох генераторів, зокрема LFSR, BBS, RC4, Mersenne Twister або System Random, та задавати параметри генерації, такі як довжина послідовності, початковий seed, tap-коефіцієнти тощо. Це дозволяє моделювати різноманітні сценарії використання потокових шифрів, а також порівнювати їхню стійкість до статистичних атак у різних умовах.

Для оцінки якості згенерованих послідовностей у програмі реалізовано набір основних статистичних тестів: частотний, серійний, покер-тест, автокореляційний та серійний тест. Кожен із них можна запускати як окремо, так і у складі комплексного аналізу, що забезпечує повноту оцінки випадковості та дозволяє виявити як глобальні, так і локальні закономірності у ключовому потоці.
Результати тестування відображаються у вигляді гістограм, стовпчикових діаграм p-value та інтерактивних таблиць, що дає змогу користувачу не лише переглядати, а й порівнювати та зберігати графіки для подальшого аналізу.

Особливістю програмного засобу є функція автоматичного аналізу, яка дозволяє одним натисканням кнопки виконати всі тести для всіх доступних генераторів. Це значно економить час користувача, підвищує ефективність дослідження та дає змогу швидко порівнювати стійкість різних генераторів у єдиному інтерфейсі.

Програма також підтримує збереження згенерованих послідовностей у файл, завантаження власних послідовностей для аналізу, а також експорт результатів тестування у форматі CSV. Це забезпечує зручність для подальшого аналізу, підготовки звітів, презентацій або наукових публікацій. Користувач може очищати діаграму результатів та починати новий експеримент без необхідності перезапуску програми, що підвищує гнучкість роботи.

Архітектура програмного засобу побудована за модульним принципом. Кожен генератор реалізовано як окремий модуль із чітко визначеним інтерфейсом, що дозволяє легко додавати нові генератори без зміни основного коду. Аналогічно, всі

статистичні тести реалізовані як незалежні модулі, які можна запускати окремо або у складі комплексного аналізу. Модуль візуалізації відповідає за побудову графіків, діаграм, таблиць і інтегрований з графічним інтерфейсом для відображення результатів у реальному часі. Модуль обробки даних забезпечує підготовку послідовностей до тестування, зберігання та обробку результатів, формування звітів, а модуль імпорту/експорту дозволяє працювати з різними форматами файлів.

Завдяки модульній структурі, додавання нових генераторів, тестів або функцій не потребує суттєвих змін у вже існуючому коді. Програма підтримує підключення нових статистичних тестів, генераторів, типів візуалізації, а також роботу з різними форматами файлів для імпорту/експорту послідовностей та результатів. Це забезпечує інтеграцію з іншими системами, зручність для користувача та масштабованість — можливість обробляти великі обсяги даних і виконувати тестування для довгих послідовностей без втрати продуктивності.

Графічний інтерфейс програми розроблений на основі PyQt5 і є інтуїтивно зрозумілим навіть для користувачів без досвіду програмування. Користувач може легко обирати генератор, задавати параметри, запускати тести, переглядати та зберігати результати. Інтерфейс містить підказки, повідомлення про помилки, інструкції щодо використання, що підвищує зручність роботи навіть для недосвідчених користувачів.

Для забезпечення надійності та якості реалізації всі основні компоненти програми проходили юніт-тестування, що гарантує стабільність роботи навіть у разі обробки граничних або некоректних даних. Кожен модуль містить докладні коментарі, опис функцій, README-файли з інструкціями щодо встановлення та використання, що спрощує підтримку та розвиток проєкту, а також полегшує залучення нових розробників.

Програма розроблена з урахуванням принципів відкритого програмного забезпечення, що дозволяє іншим дослідникам використовувати, модифікувати та розширювати її функціонал. Відкритість коду сприяє підвищенню довіри до результатів аналізу, забезпечує можливість незалежної перевірки та інтеграції у більші системи аналізу та моніторингу криптографічної стійкості.

Таким чином, розроблений програмний засіб поєднує комплексність, гнучкість, доступність і практичну цінність, що робить його універсальним інструментом для навчання, наукових досліджень, практичної перевірки криптографічних рішень та підвищення рівня інформаційної безпеки у реальних системах.

Висновки до розділу 2

У другому розділі магістерської дисертації було розкрито математичні основи, методи та алгоритми, що лежать в основі оцінювання стійкості потокових шифрів до статистичних атак, а також обґрунтовано вибір програмних засобів для реалізації відповідного програмного продукту.

Проведено детальний аналіз математичних моделей генераторів псевдовипадкових послідовностей, які використовуються у сучасних потокових шифрах. Для кожного з розглянутих генераторів (LFSR, BBS, RC4, Mersenne Twister, System Random) наведено формальні математичні описи, виведено основні формули, що визначають їхню роботу, а також проаналізовано переваги та недоліки з точки зору криптографічної стійкості, швидкості, простоти реалізації та можливості застосування у різних сферах. Порівняння здійснювалося за такими параметрами, як період генератора, складність відновлення внутрішнього стану, стійкість до відомих атак, вимоги до ресурсів та гнучкість налаштувань. Для наочності результати порівняння зведено у таблицю, що дозволяє швидко оцінити доцільність використання кожного генератора у конкретних умовах.

Важливим етапом дослідження стало математичне обґрунтування статистичних тестів, які застосовуються для виявлення закономірностей у ключових потоках. Для кожного тесту (частотного, серійного, покер-тесту, автокореляційного, серійного тесту) наведено формули для обчислення статистик та p-value, розглянуто критерії оцінювання випадковості, а також особливості інтерпретації результатів тестування. Обґрунтування полягало у виборі статистичних критеріїв, які мають доведену ефективність для виявлення різних

типів відхилень від ідеальної випадковості, а також у порівнянні їх чутливості до різних закономірностей у послідовностях. Комплексне застосування кількох тестів дозволяє підвищити достовірність оцінки завдяки охопленню різних аспектів випадковості: балансу бітів, повторюваних блоків, автокореляційних залежностей.

Описано покрокові алгоритми реалізації основних статистичних тестів, зокрема частотного, серійного, покер-тесту, автокореляційного та серійного тесту. Для кожного алгоритму наведено блок-схеми, псевдокод, а також проаналізовано обчислювальну складність (наприклад, O(n) для частотного тесту, O(n·2^m) для серійного тесту з підпослідовностями довжини m). Порівняння складності та ефективності різних під ходів зведено у таблицю, що дозволяє обґрунтовано обирати тести для конкретних задач. Особливу увагу приділено питанням автоматизації процесу тестування, що є ключовим фактором для підвищення ефективності аналізу та зручності використання програмного засобу.

Обґрунтовано вибір мови програмування Python, бібліотек NumPy, SciPy, Pandas, matplotlib, а також фреймворку PyQt5 для створення графічного інтерфейсу. Пояснено, що саме ці інструменти забезпечують гнучкість, масштабованість, зручність для користувача, а також можливість інтеграції з іншими системами та подальшого розширення функціоналу. Вибір підтверджено аналізом альтернативних рішень, порівнянням їх можливостей, продуктивності, підтримки спільноти та сумісності з сучасними стандартами програмної інженерії.

Таким чином, другий розділ формує теоретичну та технологічну основу для подальшої програмної реалізації системи автоматизованого аналізу стійкості потокових шифрів. Описані моделі, методи, алгоритми та інструменти є ключовими для досягнення поставленої мети дослідження, забезпечення високої якості отриманих результатів та їх практичного застосування у сфері інформаційної безпеки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗОВАНОГО АНАЛІЗУ СТІЙКОСТІ ПОТОКОВИХ ШИФРІВ

У третьому розділі детально описується програмна реалізація системи автоматизованого аналізу стійкості потокових шифрів до статистичних атак. На основі теоретичних та математичних основ, розглянутих у попередніх розділах, було розроблено програмний засіб, який поєднує в собі генерацію псевдовипадкових послідовностей, виконання статистичних тестів, візуалізацію результатів та зручний графічний інтерфейс для користувача.

Метою цього розділу є надати повне уявлення про архітектуру, структуру, основні компоненти та особливості програмної реалізації розробленого засобу. У розділі послідовно розглядаються питання побудови архітектури програмної системи, моделі представлення даних, структури класів та взаємодії між ними, а також особливості реалізації ключових алгоритмів і функцій.

Особлива увага приділяється модульності та гнучкості архітектури, що дозволяє легко розширювати функціонал програми, додавати нові генератори, тести, типи візуалізації та інтегрувати програмний засіб у більші системи аналізу. Описується, як забезпечується ізольованість модулів, стандартизовані інтерфейси для взаємодії між компонентами, а також механізми обробки помилок і забезпечення надійності роботи.

У цьому розділі також розглядається модель представлення даних, яка визначає структуру зберігання та обробки послідовностей, результатів тестування, параметрів генераторів та інших суттєвих об’єктів. Наводиться діаграма класів, що ілюструє логічну структуру програми, основні класи, їх атрибути, методи та зв’язки між ними.

Детально описуються основні алгоритми та функції, реалізовані у програмному засобі: генерація послідовностей, виконання статистичних тестів, побудова графіків, збереження та завантаження даних, автоматизація аналізу.

Пояснюються особливості реалізації, оптимізації для підвищення продуктивності, а також підходи до забезпечення коректності та достовірності результатів.

Окремий підпункт виділено для документації та супроводу програмного забезпечення. Описується структура документації, принципи написання інструкцій для користувача, коментарів у коді, README-файлів, а також підходи до оновлення та підтримки програмного продукту.

Таким чином, третій розділ є ключовим для розуміння практичної реалізації запропонованих у роботі методів і підходів, а також для оцінки ефективності, зручності та надійності розробленого програмного засобу. Описані у цьому розділі рішення та підходи стали основою для проведення експериментальних досліджень.

3.1 Архітектура програмної системи

Архітектура розробленої програмної системи для автоматизованого аналізу стійкості потокових шифрів до статистичних атак побудована на принципах модульності, гнучкості, масштабованості та розширюваності. Такий підхід дозволяє ефективно організувати взаємодію між різними компонентами, спростити підтримку та розвиток програмного продукту, а також забезпечити можливість інтеграції з іншими системами.

3.1.1 Загальна структура системи

Розроблена програмна система побудована за модульним принципом, що дозволяє розділити функціонал на незалежні компоненти. Основні модулі включають:

Модуль генераторів послідовностей

Реалізує різні алгоритми генерації псевдовипадкових бітових потоків, такі як LFSR, BBS, RC4, Mersenne Twister, System Random. Кожен генератор представлений окремим класом або модулем, що містить методи для ініціалізації параметрів, генерації послідовності та повернення результату у стандартному форматі.

Модуль статистичних тестів

Містить реалізації основних статистичних тестів (частотний, серійний, покер-тест, автокореляційний, серійний тест). Кожен тест реалізовано як окремий клас або функцію з уніфікованим інтерфейсом для прийому послідовності та повернення результату (p-value, додаткові метрики).

Модуль візуалізації

Відповідає за побудову графіків, діаграм, таблиць для відображення результатів тестування. Інтегрований з графічним інтерфейсом, дозволяє користувачу переглядати результати у реальному часі, зберігати графіки у різних форматах.

Графічний інтерфейс користувача (GUI)

Реалізований на PyQt5, містить елементи для вибору генератора, запуску тестів, перегляду та збереження результатів, побудови графіків, імпорту/експорту даних. Інтерфейс інтуїтивно зрозумілий, містить підказки та повідомлення про помилки.

Модуль обробки даних

Забезпечує підготовку послідовностей до тестування (обрізання, нормалізація), зберігання та обробку результатів, формування звітів, імпорт та експорт даних у різних форматах (txt, csv).

Модуль керування експериментом

Координує роботу генераторів, тестів, візуалізації, забезпечує автоматизацію запуску комплексних сценаріїв аналізу, зберігає історію експериментів.

3.1.2 Взаємодія між модулями

Взаємодія між модулями організована через стандартизовані інтерфейси та чітко визначені точки входу/виходу.

GUI ініціює запити до модуля генераторів для створення послідовності з заданими параметрами. Згенерована послідовність передається у модуль статистичних тестів, де послідовно або паралельно виконуються обрані тести. Результати тестування (p-value, додаткові метрики) передаються у модуль

візуалізації для побудови графіків та таблиць. Модуль обробки даних відповідає за збереження результатів, імпорт/експорт, формування звітів. Модуль керування експериментом дозволяє автоматизувати запуск серій тестів, зберігати історію, порівнювати результати різних генераторів.

Вся взаємодія між модулями відбувається через уніфіковані функції та об’єкти, що забезпечує ізольованість компонентів і спрощує їхню заміну або розширення.

3.1.3 Принципи побудови архітектури

Модульність

Кожен функціональний блок (генератор, тест, візуалізація, обробка даних, GUI) реалізовано як окремий модуль або клас. Це дозволяє легко додавати нові генератори, тести, типи візуалізації без зміни основного коду.

Гнучкість

Архітектура дозволяє адаптувати програму під нові вимоги, розширювати функціонал, інтегрувати з іншими системами. Наприклад, додавання нового тесту потребує лише створення нового модуля та реєстрації його у системі.

Масштабованість

Система здатна обробляти великі обсяги даних, виконувати тестування для довгих послідовностей, підтримує багатопотокову обробку (за потреби).

Розширюваність

Завдяки чітко визначеним інтерфейсам, можна легко додавати нові модулі, не порушуючи роботу існуючих компонентів. Це важливо для подальшого розвитку програмного засобу.

Інкапсуляція

Кожен модуль приховує внутрішню реалізацію, надаючи лише необхідні методи для взаємодії з іншими частинами системи. Це підвищує надійність та безпеку коду.

3.1.4 Переваги обраної архітектури

Легкість підтримки та розвитку

Модульна структура спрощує внесення змін, виправлення помилок, додавання нового функціоналу без ризику порушити роботу всієї системи.

Можливість командної роботи

Розробники можуть працювати над різними модулями незалежно один від одного, що підвищує ефективність командної розробки та пришвидшує впровадження нових функцій. Принцип роботи компонентів застосунку показаний на рисунку 3.1.

Рисунок 3.1 - Принцип роботи компонентів застосунку

Відкритість до інтеграції

Програмний засіб може бути використаний як окремий інструмент або як

частина більших систем аналізу та моніторингу криптографічної стійкості, що розширює сферу його застосування.

Зручність для користувача

Інтуїтивний інтерфейс, автоматизація рутинних операцій, наочна візуалізація результатів роблять програму доступною для широкого кола користувачів — від студентів до досвідчених фахівців з інформаційної безпеки.

3.2 Модель представлення даних

Ефективна робота програмної системи для автоматизованого аналізу стійкості потокових шифрів до статистичних атак значною мірою залежить від правильної організації та структурування даних. У цьому підпункті розглядаються основні підходи до моделювання даних, які використовуються у програмному засобі, а також формати зберігання, обміну та обробки інформації.

Структура даних для бітових послідовностей

Бітові послідовності є основним об’єктом аналізу у системі. Для їх представлення використовується одновимірний масив типу numpy.ndarray, де кожен елемент — це 0 або 1. Такий підхід дозволяє:

- ефективно зберігати великі послідовності (до мільйонів бітів) у пам’яті комп’ютера;

- виконувати векторизовані операції (наприклад, підрахунок кількості нулів/одиниць, виконання XOR, зсувів, порівнянь) значно швидше, ніж при використанні стандартних списків Python;

- легко інтегрувати послідовності з іншими бібліотеками для статистичного аналізу, візуалізації та обробки даних.

Для збереження послідовностей у файл використовується кілька форматів:

Текстовий формат (.txt, .csv)

Кожен біт записується окремим рядком або у вигляді суцільного рядка, що спрощує читання та обробку даних іншими програмами.

Двійковий формат

Використовується для економії місця при зберіганні великих послідовностей, забезпечує швидке зчитування та запис.

У програмі реалізовано функції для імпорту та експорту послідовностей у зазначених форматах, а також для перевірки коректності вхідних даних (відсутність символів, відмінних від 0 та 1).

Представлення параметрів генераторів

Параметри генераторів (довжина LFSR, tap-коефіцієнти, seed, прості числа для BBS, ключ для RC4) зберігаються у вигляді словників (dict) або спеціальних класів-конфігурацій. Кожен генератор має власний набір параметрів, які можуть бути збережені у вигляді:

Словника: ключі — назви параметрів, значення — їх числові або символьні значення.

JSON/YAML: для зручності збереження та обміну конфігураціями між різними системами.

Класу-конфігурації: для складних генераторів, де параметри мають додаткові властивості або методи перевірки.

Такий підхід дозволяє:

- легко передавати параметри між модулями;
- зберігати налаштування для повторного використання;
- експортувати/імпортувати параметри для відтворюваності експериментів;
- забезпечити валідацію параметрів перед запуском генерації.

У графічному інтерфейсі користувач може задавати параметри генераторів через відповідні поля введення, а програма автоматично перевіряє їх коректність.

Структура даних для результатів тестування

Результати статистичних тестів організовані у вигляді структурованих записів, які містять:

Назву тесту ("Частотний", "Серійний", "Покер-тест", "Автокореляційний").

Значення p-value — основний показник випадковості послідовності.

Додаткові метрики: кількість серій, значення статистики, кількість блоків у покер-тесті, параметри тесту (розмір блоку, лаг, довжина підпослідовності).

Статус проходження тесту: пройдено/не пройдено (визначається пороговим значенням p-value).

Для зберігання та обробки результатів використовується бібліотека pandas, яка дозволяє:

- формувати таблиці результатів для кожного генератора та експерименту;
- групувати, фільтрувати, сортувати результати за різними критеріями;
- експортувати результати у формати CSV, Excel для подальшого аналізу або підготовки звітів.

У програмі реалізовано механізми автоматичного збереження результатів після виконання тестів, а також можливість перегляду історії тестувань.

Формати імпорту та експорту даних

Для забезпечення гнучкості та сумісності з іншими програмними засобами, система підтримує імпорт та експорт даних у різних форматах:

Текстові файли (.txt, .csv)

Використовуються для зберігання бітових послідовностей, результатів тестування, параметрів генераторів. Формат CSV дозволяє легко імпортувати дані у табличні процесори (Excel, Google Sheets) для подальшого аналізу [36].

Двійкові файли

Застосовуються для економії місця при зберіганні великих послідовностей, забезпечують швидке зчитування та запис.

JSON/YAML

Використовуються для збереження конфігурацій, параметрів експериментів, історії запусків. Формати зручні для обміну даними між різними системами та для зберігання складних структур.

Графічні формати (PNG, CSV)

Дозволяють експортувати гістограми, діаграми, графіки результатів тестування для включення у звіти, презентації, публікації. Усі операції

імпорту/експорту супроводжуються перевіркою коректності даних та повідомленнями для користувача у разі помилок.

Модель зберігання історії експериментів

Для забезпечення відтворюваності, аналізу та порівняння результатів у програмі реалізовано механізм зберігання історії експериментів. Кожен експеримент містить:

- тип генератора та його параметри;
- результати всіх виконаних тестів (p-value, додаткові метрики, статус);
- посилання на збережені графіки, таблиці, файли послідовностей;

Історія зберігається у вигляді таблиці (DataFrame) або у форматі JSON, що дозволяє:

- легко здійснювати пошук, фільтрацію, порівняння результатів різних запусків;
- відновлювати параметри попередніх експериментів для повторного аналізу;
- формувати звіти про проведені дослідження.

У графічному інтерфейсі реалізовано перегляд історії експериментів, вибір експерименту для повторного аналізу або експорту результатів.

Представлення даних у графічному інтерфейсі

У графічному інтерфейсі дані відображаються у різних формах.

Таблиці: результати тестів, параметри генераторів, історія експериментів. Таблиці інтерактивні, підтримують сортування, фільтрацію, копіювання даних.

Графіки: гістограми бітів, діаграми p-value, порівняльні графіки для різних генераторів. Графіки інтерактивні, підтримують масштабування, збереження у файл.

Повідомлення: статус виконання тестів, помилки, підказки для користувача, результати проходження тестів (кольорове виділення пройдено/не пройдено).

Всі елементи інтерфейсу пов’язані з відповідними структурами даних, що забезпечує цілісність, актуальність та зручність роботи з інформацією.

Забезпечення цілісності та достовірності даних

Перевірка коректності вхідних даних

Автоматична валідація формату послідовності, допустимості параметрів генераторів, коректності файлів для імпорту.

Автоматичне збереження результатів

Після виконання тестів результати автоматично зберігаються у історії експериментів.

Резервне копіювання

Періодичне збереження копій історії експериментів для відновлення у разі збоїв.

Повідомлення користувача

У разі помилок або некоректних дій користувач отримує зрозумілі повідомлення з рекомендаціями щодо виправлення ситуації.

Завдяки цим механізмам забезпечується висока надійність, цілісність та достовірність даних, що є критично важливим для наукових досліджень та практичного застосування програмного засобу.

3.3 Класи та взаємодія компонентів

Ефективна організація програмного коду та взаємодії між компонентами є ключовою умовою для створення масштабованого, гнучкого та підтримуваного програмного засобу. Від правильного вибору архітектурних рішень, структури класів, принципів взаємодії залежить не лише якість програмного продукту, а й можливість його подальшого розвитку, розширення функціоналу, інтеграції з іншими системами та зручність підтримки.

Основні класи та їх призначення

У розробленій системі виділено низку ключових класів, кожен з яких відповідає за окремий аспект функціонування програми. Такий підхід дозволяє чітко розмежувати відповідальність між компонентами, спростити тестування, підтримку та розширення системи.

Generator (абстрактний клас)

Generator є базовим (абстрактним) класом для всіх генераторів псевдовипадкових послідовностей. Він визначає уніфікований інтерфейс, який мають реалізовувати всі конкретні генератори. Основні атрибути класу включають:

- довжину послідовності;
- початковий seed;
- tap-коефіцієнти (для LFSR);
- інші специфічні параметри (прості числа для BBS, ключ для RC4).

Generator містить методи для ініціалізації параметрів, генерації послідовності, перевірки коректності вхідних даних, збереження та відновлення стану генератора. Така абстракція дозволяє легко додавати нові генератори, не змінюючи основну логіку програми.

LFSRGenerator, BBSGenerator, RC4Generator, MersenneTwisterGenerator, SystemRandomGenerator. Кожен із цих класів наслідує Generator і реалізує конкретний алгоритм генерації.

LFSRGenerator реалізує лінійний зсувний регістр зворотного зв’язку, містить методи для роботи з tap-коефіцієнтами, перевірки примітивності полінома, оптимізації обчислень.

BBSGenerator реалізує генератор Blum Blum Shub, містить методи для роботи з великими простими числами, перевірки їх відповідності вимогам, оптимізації операцій з великими числами.

RC4Generator реалізує класичний RC4, містить методи для ініціалізації ключа, перестановки масиву S, генерації ключового потоку.

MersenneTwisterGenerator використовує стандартну реалізацію з бібліотеки numpy, містить методи для налаштування seed, генерації послідовності.

SystemRandomGenerator використовує системний генератор випадкових чисел, забезпечує криптографічну стійкість при відповідних налаштуваннях ОС.

Кожен клас містить додаткові функції для перевірки коректності параметрів, оптимізації обчислень, збереження стану для відтворюваності експериментів.

Test (абстрактний клас)

Test є базовим класом для всіх статистичних тестів. Він визначає уніфікований інтерфейс для виконання тесту, зберігання параметрів, результатів, інтерпретації p-value, формування звіту. Основні атрибути класу:

- назва тесту;

- параметри тесту (розмір блоку, лаг, довжина підпослідовності тощо);

- результати тесту (p-value, додаткові метрики);

- статус проходження тесту.

Test містить методи для ініціалізації, виконання тесту, отримання результату, інтерпретації p-value, формування звіту для користувача.

FrequencyTest, RunsTest, SerialTest, PokerTest, AutocorrelationTest. Кожен із цих класів наслідує Test і реалізує конкретний алгоритм статистичного аналізу:

FrequencyTest підраховує кількість нулів і одиниць, обчислює статистику S, p-value, визначає статус проходження тесту.

RunsTest аналізує довжини та кількість серій, обчислює очікуване значення, статистику Z, p-value.

SerialTest підраховує частоти появи підпослідовностей, обчислює статистику хі-квадрат, p-value.

PokerTest розбиває послідовність на блоки, підраховує кількість кожної комбінації, обчислює статистику хі-квадрат, p-value .

AutocorrelationTest обчислює кількість збігів між бітами на відстані d, визначає статистику автокореляції, p-value.

VisualizationManager відповідає за побудову графіків, діаграм, таблиць для відображення результатів тестування. Основні функції:

- створення гістограм бітів;

- побудова діаграм p-value для різних генераторів і тестів;

- формування порівняльних графіків;

- експорт графіків у різні формати (PNG, CSV);

- інтерактивність (масштабування, збереження, підказки).

VisualizationManager інтегрований з GUI, дозволяє користувачу швидко оцінити результати тестування, порівняти генератори, виявити слабкі місця.

ExperimentManager координує запуск генераторів, тестів, зберігання та обробку результатів, ведення історії експериментів. Основні функції:

- автоматизація запуску серій тестів для різних генераторів;
- зберігання параметрів експериментів, результатів, посилань на графіки;
- формування звітів, порівняння результатів різних запусків;
- відновлення попередніх експериментів для повторного аналізу.

ExperimentManager забезпечує відтворюваність досліджень, зручність аналізу, формування звітів для наукових публікацій.

DataManager відповідає за імпорт/експорт послідовностей, результатів тестування, параметрів генераторів у різних форматах. Основні функції:

- збереження історії експериментів;
- резервне копіювання, відновлення даних;
- перевірка коректності файлів;
- інтеграція з іншими програмними засобами.

DataManager забезпечує надійність зберігання даних, гнучкість у роботі з різними форматами, можливість інтеграції з іншими системами.

MainWindow (GUI) реалізує графічний інтерфейс користувача, містить елементи для вибору генератора, запуску тестів, перегляду результатів, побудови графіків, імпорту/експорту даних. Основні функції:

- відображення повідомлень, підказок, результатів тестування;
- керування історією експериментів;
- інтеграція з усіма модулями системи;
- забезпечення інтуїтивної взаємодії користувача з програмою.

UML-діаграма класів відображає структуру основних класів системи, їх атрибути, методи та зв’язки між ними. Діаграма, зазначена на рисунку 3.2, дозволяє візуально оцінити структуру системи, зрозуміти логіку взаємодії між компонентами, спростити процес розширення та підтримки коду.

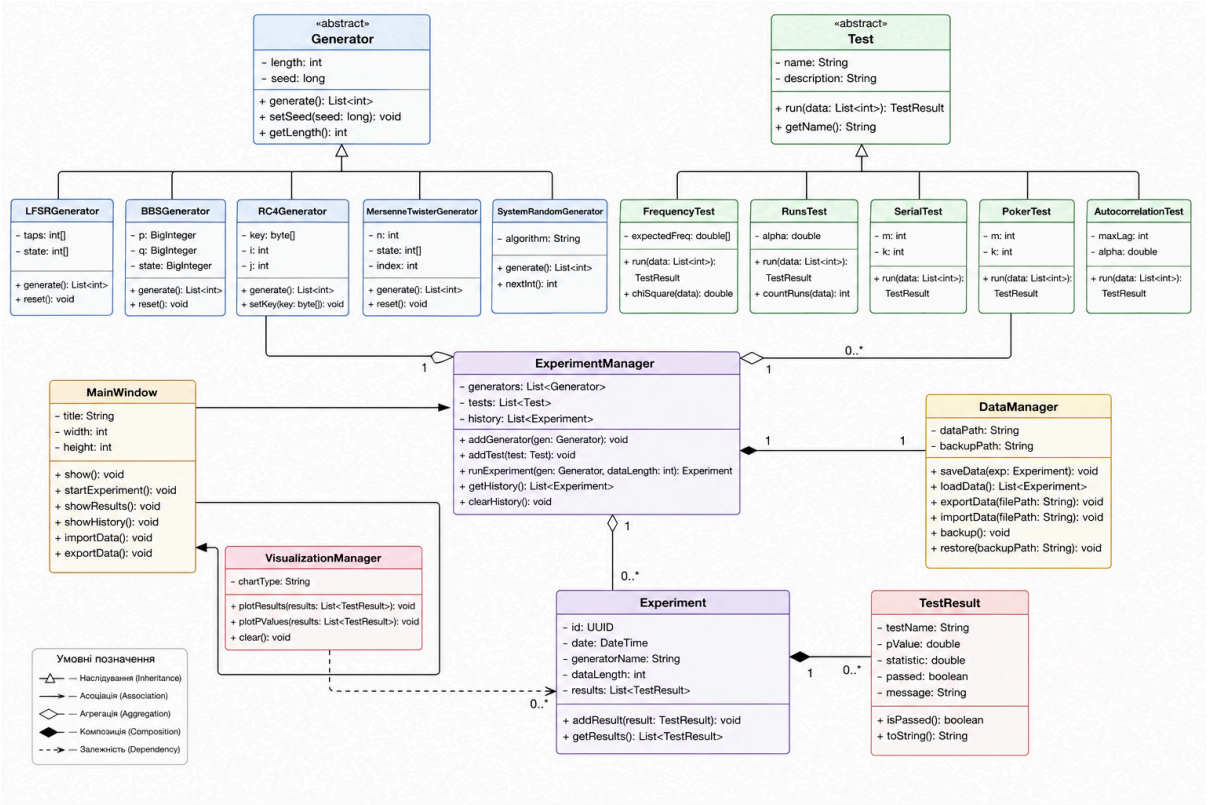

Рисунок 3.2 - UML-діаграма класів

Generator — абстрактний клас, від якого наслідуються всі генератори (LFSRGenerator, BBSGenerator, RC4Generator, MersenneTwisterGenerator, SystemRandomGenerator).

Test — абстрактний клас, від якого наслідуються всі тести (FrequencyTest, RunsTest, SerialTest, PokerTest, AutocorrelationTest).

MainWindow взаємодіє з ExperimentManager, VisualizationManager, DataManager через асоціації.

ExperimentManager координує генератори та тести, зберігає історію експериментів, взаємодіє з DataManager для збереження/відновлення даних.

VisualizationManager отримує результати тестів і будує графіки, які відображаються у MainWindow.

DataManager відповідає за збереження, імпорт/експорт даних, резервне копіювання, відновлення історії експериментів.

На рисунку 3.2 зображені:

- відносини наслідування (стрілки з порожнім трикутником);

- асоціації (лінії між класами);

- агрегації (лінії з ромбом);

- основні атрибути та методи класів.

Взаємодія між класами та модулями

Взаємодія між класами організована через чітко визначені інтерфейси, що забезпечує ізольованість компонентів, легкість розширення та підтримки системи.

Користувач через MainWindow (GUI) обирає генератор, задає параметри, запускає генерацію послідовності.

MainWindow передає параметри у відповідний клас генератора, який генерує послідовність та повертає її ExperimentManager.

ExperimentManager передає послідовність у вибрані тести, які виконують аналіз та повертають результати (p-value, додаткові метрики).

Результати тестування зберігаються у DataManager, а також передаються у VisualizationManager для побудови графіків та таблиць.

VisualizationManager будує графіки, які відображаються у MainWindow.

DataManager забезпечує збереження/завантаження послідовностей, результатів, графіків у різних форматах, резервне копіювання, відновлення даних.
Вся історія експериментів зберігається у ExperimentManager для подальшого аналізу та порівняння. Такий підхід дозволяє розробникам працювати над різними модулями незалежно один від одного, спрощує тестування, підтримку, розширення функціоналу, інтеграцію з іншими системами.

Особливості реалізації та розширюваність

Інтерфейси та абстракції

Використання абстрактних класів Generator і Test дозволяє легко додавати нові генератори та тести без зміни основної логіки програми. Кожен новий генератор або тест реалізується як окремий клас-нащадок із власними параметрами та методами, що забезпечує гнучкість і масштабованість системи.

Інкапсуляція

Кожен клас приховує внутрішню реалізацію, надаючи лише необхідні методи для взаємодії з іншими компонентами. Це підвищує надійність, безпеку та зручність підтримки коду, дозволяє уникати помилок при зміні внутрішньої логіки окремих модулів [7].

Модульність

Кожен функціональний блок реалізовано як окремий модуль, що спрощує підтримку, тестування та розвиток системи. Модульна структура дозволяє розробникам працювати над різними частинами проєкту незалежно один від одного, пришвидшує впровадження нових функцій, полегшує командну роботу.

Можливість розширення

Додавання нового тесту або генератора потребує лише створення нового класу-нащадка та реєстрації його у системі. Це дозволяє швидко адаптувати програму до нових вимог, розширювати функціонал без ризику порушити роботу всієї системи.

Взаємозамінність компонентів

Завдяки уніфікованим інтерфейсам, можна замінювати окремі модулі (наприклад, генератор або тест) без необхідності змінювати інші частини програми.

Це особливо важливо для підтримки, оновлення, інтеграції з іншими системами, а також для експериментів з різними підходами до генерації та тестування.

Підтримка командної розробки

Завдяки чітко визначеним інтерфейсам і модульній структурі, кілька розробників можуть працювати над різними частинами системи одночасно, не заважаючи один одному. Це підвищує ефективність розробки, пришвидшує впровадження нових функцій, спрощує тестування та інтеграцію.

Відкритість до інтеграції

Програмний засіб може бути використаний як окремий інструмент або як частина більших систем аналізу та моніторингу криптографічної стійкості, що розширює сферу його застосування у наукових, освітніх та промислових проєктах.

3.4 Опис основних алгоритмів та функцій

У цьому підпункті детально описуються ключові алгоритми та функції, реалізовані у програмному засобі для автоматизованого аналізу стійкості потокових шифрів до статистичних атак. Особлива увага приділяється реалізації генераторів, статистичних тестів, механізмів візуалізації, обробки даних, а також оптимізації та забезпеченню коректності роботи програми.

Алгоритми генерації псевдовипадкових послідовностей

LFSR (Linear Feedback Shift Register)

LFSR — це класичний генератор, який формує послідовність бітів на основі лінійного зсуву та зворотного зв’язку. Користувач задає довжину регістра, tap-коефіцієнти (позиції, які беруть участь у зворотному зв’язку), початковий стан (seed). На кожному кроці новий біт обчислюється як XOR визначених tap-позицій, після чого регістр зсувається, а новий біт додається на початок [18].

Переваги:

- простота реалізації як у програмному, так і в апаратному вигляді;

- висока швидкість генерації, що дозволяє використовувати LFSR у реальному часі;

- великий період при правильному виборі полінома зворотного зв’язку (до $2n-1$ для n-бітного регістра);

- легкість масштабування для різних задач.

Недоліки:

- лінійна структура, що робить LFSR вразливим до лінійного криптоаналізу;

- можливість відновлення структури генератора за допомогою алгоритму Берлекемпа-Мессі навіть за відносно короткою частиною вихідної послідовності;

- необхідність використання додаткових методів (наприклад, нелінійних комбінацій кількох LFSR) для підвищення криптостійкості.

Практичний сценарій:

LFSR часто використовується для генерації тестових послідовностей, у

симуляціях, а також як базовий елемент у складніших криптографічних генераторах.

BBS (Blum Blum Shub)

BBS — криптографічно стійкий генератор, що базується на складності факторизації великих чисел. Користувач задає два великі прості числа, початковий стан (seed). Кожен новий елемент обчислюється як квадрат попереднього за модулем добутку простих чисел, а вихідний біт — найменш значущий біт отриманого числа [19].

Переваги:

- криптостійкість, що базується на складності факторизації;
- відсутність відомих ефективних атак при правильному виборі параметрів;
- використання у криптографічних протоколах, де критично важлива безпека.

Недоліки:

- низька швидкість генерації через необхідність виконання операцій з великими числами;
- складність реалізації для великих чисел, потреба у генерації великих простих чисел;
- не підходить для задач, де потрібна висока швидкість генерації.

Практичний сценарій:

BBS використовується для генерації ключів, одноразових паролів, у протоколах, де важлива доведена криптостійкість, а не швидкість.

RC4

RC4 — потоковий шифр, що складається з етапу ініціалізації (KSA) та генерації ключового потоку (PRGA). Користувач задає ключ, довжину послідовності. Алгоритм формує перестановку масиву S, генерує послідовність байтів, з яких береться молодший біт [9].

Переваги:

- простота реалізації, висока швидкість генерації;
- гнучкість у виборі ключа, можливість використання у різних протоколах;
- широке застосування у минулому (WEP, SSL, WPA).

Недоліки:

- відомі вразливості, особливо на початку потоку (key scheduling weakness);

- не рекомендується для нових систем без додаткових заходів безпеки;

- можливість відновлення частини ключа при неправильному використанні.

Практичний сценарій:

RC4 використовується для аналізу історичних протоколів, тестування стійкості до відомих атак, порівняння з сучасними генераторами.

Mersenne Twister

Генератор з дуже великим періодом, реалізований у бібліотеці numpy. Використовується для моделювання, але не є криптографічно стійким.

Переваги:

- дуже великий період (219937−1), що забезпечує високу якість випадковості для статистичних задач;

- висока швидкість генерації, простота використання;

- широке застосування у наукових розрахунках, симуляціях, статистичних експериментах.

Недоліки:

- можливість відновлення стану генератора за частиною вихідної послідовності.

Практичний сценарій:

Mersenne Twister використовується для генерації тестових даних, симуляцій, порівняння з криптографічно стійкими генераторами [35].

System Random

Використовує системні джерела ентропії (наприклад, /dev/urandom у Linux). Забезпечує високу якість випадковості, може бути криптографічно стійким залежно від реалізації ОС.

Переваги:

- використання апаратних або системних джерел ентропії;

- висока якість випадковості, стійкість до атак;

- простота інтеграції у програмні продукти.

Недоліки:

- залежність від реалізації ОС, можливі обмеження на деяких платформах.

- не завжди можна контролювати внутрішні параметри генератора.

Практичний сценарій:

System Random використовується для генерації ключів, одноразових паролів, у протоколах, де потрібна висока якість випадковості [31].

Алгоритми статистичних тестів

Частотний тест

Підраховує кількість нулів і одиниць у послідовності, обчислює статистику S та p-value. Дозволяє виявити глобальні відхилення від рівномірного розподілу.

Особливість:

Простий у реалізації, швидко виконується навіть для великих послідовностей, не виявляє локальних закономірностей, не чутливий до патернів у підпослідовностях, є базовим тестом, який дозволяє швидко відсіяти явно невипадкові послідовності [8].

NIST (Runs-test)

Визначає кількість та довжини серій однакових бітів, обчислює очікуване значення, статистику Z, p-value.

Особливість:

Чутливий до локальних патернів, дозволяє виявити надмірно довгі або короткі серії, виявляє періодичність, повторювані патерни, які не видно у частотному тесті, дозволяє оцінити структуру послідовності на локальному рівні [16].

Серійний тест

Підраховує частоту появи всіх можливих підпослідовностей певної довжини, обчислює статистику хі-квадрат, p-value.

Особливість:

Дозволяє виявити нерівномірний розподіл підпослідовностей, що може свідчити про слабкість генератора, чутливий до складних патернів, які не

виявляються простими тестами, може бути налаштований на різну довжину підпослідовностей для глибшого аналізу [12].

Покер-тест

Послідовність розбивається на блоки фіксованої довжини, підраховується кількість кожної комбінації, обчислюється статистика хі-квадрат, p-value.

Особливість:

Виявляє повторювані патерни, які не виявляються частотним тестом, дозволяє оцінити рівномірність розподілу блоків, виявити структурні недоліки генератора, чутливий до вибору розміру блоку, що дозволяє адаптувати тест під конкретні задачі [16].

Автокореляційний тест

Обчислює кількість збігів між бітами на відстані d, визначає статистику автокореляції, p-value.

Особливість:

Дозволяє виявити періодичність, залежності у послідовності, які можуть бути використані для атаки, чутливий до прихованих закономірностей, які не видно у простих тестах, може бути налаштований на різні лаги для глибшого аналізу [37].

Механізми візуалізації результатів

Гістограма бітів відображає співвідношення нулів та одиниць у послідовності, дозволяє швидко оцінити баланс бітів, виявити глобальні відхилення, надлишок або дефіцит певних значень. На рисунку 3.3 наведена гістограма бітів для генератора LFSR.

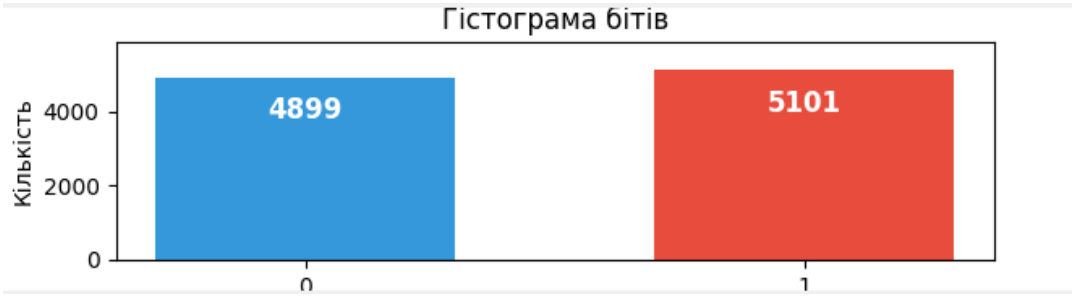

Рисунок 3.3 - Гістограма бітів

Гістограма бітів є базовим інструментом для первинної оцінки якості послідовності. Вона може бути використана для порівняння різних генераторів на одному графіку.

Діаграма p-value

Для кожного генератора та тесту будується стовпчикова діаграма p-value. Вона дозволяє порівнювати якість різних генераторів, виявляти слабкі місця, аналізувати результати комплексно. Візуалізація p-value дозволяє швидко ідентифікувати генератори, які не проходять певні тести. Може бути використана для формування звітів, презентацій, наукових публікацій. На рисунку 3.4 наведена діаграма p-value для п`яти генераторів - результат проведення чотирьох тестів.

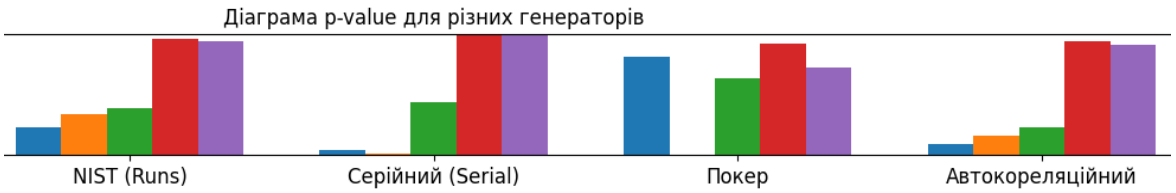

Рисунок 3.4 - Діаграма p-value

Графіки можна масштабувати, зберігати у різних форматах, переглядати детальну інформацію про кожен тест, отримувати підказки щодо інтерпретації результатів.

Інтерфейс дозволяє швидко перемикатися між різ ними видами візуалізації, порівнювати результати для різних генераторів і тестів.

Інтерактивність підвищує зручність роботи, дозволяє адаптувати програму під потреби користувача.

Збереження послідовностей

Послідовності зберігаються у текстових або двійкових файлах, підтримується імпорт/експорт для подальшого аналізу, відтворення експериментів.

Формати зберігання обираються з урахуванням обсягу даних, зручності обробки, сумісності з іншими програмами. Підтримується збереження у форматах,

які легко імпортуються у табличні процесори, статистичні пакети, інші програмні засоби.

Збереження результатів тестування

Результати тестів зберігаються у структурованих таблицях (pandas DataFrame), що дозволяє легко здійснювати аналіз, порівняння, формування звітів, експорт у різні формати. Зберігаються всі ключові метрики: p-value, додаткові статистики, параметри тесту, статус проходження. Можливість експорту у CSV, Excel, PDF для подальшого аналізу, підготовки звітів, наукових публікацій.

Збереження історії експериментів

Всі експерименти зберігаються з параметрами генераторів, результатами тестів, датою та часом проведення. Це забезпечує відтворюваність досліджень, можливість аналізу динаміки, порівняння різних підходів, формування звітів для наукових публікацій. Користувач може відновлювати параметри попередніх експериментів для повторного аналізу, порівнювати результати різних запусків.

Експорт графіків

Графіки можна зберігати у форматах PNG, CSV для включення у звіти, презентації, публікації. Це дозволяє використовувати результати тестування у навчальному процесі, наукових статтях, доповідях на конференціях. Підтримується налаштування роздільної здатності, розміру, формату графіків.

Оптимізація та забезпечення коректності

Векторизація обчислень

Для підвищення швидкості обробки великих послідовностей використовуються векторизовані операції бібліотек numpy та pandas, що дозволяє виконувати обчислення у десятки разів швидше, ніж при використанні циклів [17].

Це особливо важливо для обробки послідовностей довжиною у сотні тисяч або мільйони бітів. Векторизація підвищує ефективність використання ресурсів, знижує час виконання тестів [22].

Перевірка коректності даних

Перед запуском тестів здійснюється автоматична валідація формату послідовності, допустимості параметрів генераторів, коректності вхідних даних. У разі помилок користувач отримує зрозумілі повідомлення з рекомендаціями щодо виправлення, що знижує ймовірність помилок та підвищує надійність результатів. Перевірка коректності даних дозволяє уникнути збоїв, некоректних результатів, втрати даних [26].

Обробка помилок

Усі основні функції містять механізми обробки винятків, що забезпечує стабільність роботи програми навіть у разі некоректних дій користувача або збоїв у роботі системи.

Програма фіксує помилки у логах, надає користувачу підказки щодо їх усунення. Обробка помилок підвищує надійність, знижує ризик втрати даних, забезпечує зручність для користувача.

Тестування модулів

Для основних компонентів реалізовано юніт-тести, що дозволяє перевіряти коректність роботи окремих функцій, виявляти та виправляти помилки на ранніх етапах розробки.

Це підвищує якість програмного продукту, спрощує підтримку та розвиток системи. Тестування модулів дозволяє швидко знаходити та виправляти помилки, підвищує довіру до результатів аналізу.

Автоматизація та сценарії використання

Автоматичний аналіз

Користувач може запускати комплексне тестування для всіх генераторів одним натисканням кнопки. Програма автоматично генерує послідовності, виконує всі тести, додає результати на діаграму, зберігає історію експериментів. Це значно економить час, підвищує ефективність аналізу, дозволяє швидко порівнювати різні генератори, виявляти слабкі місця.

Гнучкість сценаріїв

Можливість запускати окремі тести, змінювати параметри генераторів, зберігати та завантажувати послідовності, порівнювати результати різних експериментів. Це дозволяє адаптувати програму під різні задачі — від навчання до наукових досліджень, експериментів з новими генераторами, тестами, параметрами.

Гнучкість сценаріїв підвищує універсальність програмного засобу, дозволяє використовувати його у різних сферах.

Історія експериментів

Вся інформація про проведені експерименти зберігається для подальшого аналізу, формування звітів, відтворення результатів. Користувач може переглядати, фільтрувати, експортувати історію для підготовки наукових публікацій чи звітів, відновлювати параметри попередніх експериментів для повторного аналізу. Збереження історії експериментів забезпечує відтворюваність досліджень, підвищує наукову цінність результатів.

3.5 Документація та супровід програмного забезпечення

Якісна документація та організація супроводу програмного забезпечення є невід’ємною складовою сучасних програмних продуктів, особливо у наукових і навчальних проєктах [34]. Відповідна документація забезпечує зрозумілість структури, принципів роботи, інтерфейсів та алгоритмів, а також спрощує подальший розвиток, підтримку та інтеграцію програмного засобу. У цьому підпункті детально описано підходи до документування, структуру супровідних матеріалів, інструкції для користувача, а також принципи оновлення та підтримки програмного продукту [39].

Структура документації

Документація до програмного засобу організована багаторівнево, щоб забезпечити зручність для різних категорій користувачів — як кінцевих користувачів, так і розробників. Кожен документ має чітко визначене призначення, структуру та рівень деталізації.

README-файл

README.md — це основний документ, який містить короткий опис проєкту, його призначення, основні функції, вимоги до системи (версія Python, необхідні бібліотеки), інструкції зі встановлення та запуску програми [41].

README також містить:

- контактну інформацію для зворотного зв’язку;

- посилання на додаткові ресурси (офіційний сайт, документацію, відеоінструкції);

- приклади запуску (команди для встановлення, запуску, тестування);

- короткий опис структури каталогів;

- рекомендації щодо першого запуску програми;

- розділ “Поширені питання” (FAQ) з відповідями на типові запити користувачів;

інформацію про ліцензію та умови використання.

README є першою точкою входу для нового користувача або розробника, дозволяє швидко оцінити можливості програмного продукту, зрозуміти його призначення та основні сценарії використання.

Керівництво користувача (User Guide)

User Guide — це детальний документ, що містить покрокові інструкції для користувача щодо роботи з програмою [44].

Включає:

- ілюстровані приклади: як згенерувати послідовність, задати параметри генератора, виконати тести, інтерпретувати результати, зберегти/завантажити дані, очистити історію експериментів, експортувати графіки;

- розділи з описом типових сценаріїв використання (наприклад, “Як провести комплексний аналіз для всіх генераторів”, “Як порівняти результати різних експериментів”);

- поради щодо вибору параметрів генераторів і тестів;

- розділ з поширеними питаннями (FAQ);

- рекомендації щодо усунення типових помилок;

- приклади повідомлень, які можуть з’являтися під час роботи, та пояснення їх значення;

- інструкції щодо роботи з історією експериментів, формування звітів, підготовки даних для наукових публікацій.

User Guide містить скріншоти інтерфейсу, покрокові ілюстрації, що дозволяє навіть недосвідченому користувачу швидко освоїти програму.

Довідник по інтерфейсу

Довідник по інтерфейсу — це документ, який описує всі елементи графічного інтерфейсу: кнопки, поля введення, меню, діалогові вікна, повідомлення про помилки, підказки [42].

Він містить:

- скріншоти з підписами, що допомагає швидко орієнтуватися у програмі;

- таблиці з описом функцій кожного елемента;

- приклади повідомлень, які можуть з’являтися під час роботи;

- рекомендації щодо інтерпретації результатів;

- опис гарячих клавіш, комбінацій для швидкого доступу до основних функцій;

Довідник по інтерфейсу дозволяє користувачу швидко знаходити потрібні функції, розуміти логіку роботи програми, уникати помилок при взаємодії з інтерфейсом.

Документація для розробників (Developer Guide)

Developer Guide — це документ, призначений для розробників, які планують розширювати, модифікувати програмний засіб з іншими системами [12].

Він містить:

- опис архітектури проєкту, структури каталогів і модулів;

- UML-діаграми класів, блок-схеми алгоритмів, схеми взаємодії між модулями;

- інтерфейси класів, принципи розширення функціоналу (додавання нових генераторів, тестів, типів візуалізації);

- правила оформлення коду, стандарти іменування, рекомендації щодо стилю програмування;

- інструкції щодо використання системи контролю версій (Git), організації командної роботи, ведення журналу змін (changelog);

- приклади підключення нових модулів, тестування, інтеграції з іншими програмними продуктами;

- опис процесу внесення змін до коду, запуску юніт-тестів, оновлення документації.

Developer Guide дозволяє швидко залучати нових розробників до проєкту, забезпечує єдині стандарти розробки, спрощує підтримку та розвиток програмного засобу.

Інструкції для користувача

Встановлення програми

Покрокова інструкція з встановлення Python (з посиланнями на офіційний сайт, рекомендаціями щодо вибору версії).

Встановлення необхідних бібліотек через pip (pip install -r requirements.txt).

Описано можливих проблем при встановленні (несумісність версій, відсутність бібліотек, помилки при компіляції) та способів їх вирішення.

Наведені інструкції щодо запуску програми з командного рядка (python main.py), створення ярлика для швидкого запуску, налаштування змінних середовища.

Початок роботи

Опис вибору генератора (LFSR, BBS, RC4, Mersenne Twister, System Random), налаштування параметрів (довжина послідовності, seed, tap-коефіцієнти, ключ). Покрокова інструкція з генерації послідовності, запуску окремого тесту або комплексного аналізу. Опис перегляду результатів у вигляді таблиць, графіків, діаграм p-value. Приклади типових сценаріїв використання (наприклад, “Як провести аналіз стійкості LFSR з різними tap-коефіцієнтами”).

Робота з результатами

Інструкції щодо збереження та завантаження послідовностей у різних форматах (txt, bin), збереження результатів тестування (CSV, Excel), експорт графіків (PNG, CSV). Опис формування звітів, підготовки даних для наукових публікацій, презентацій. Приклади використання результатів у навчальному процесі, наукових дослідженнях. Покрокова інструкція з запуску комплексного тестування для всіх генераторів, перегляду діаграми результатів, очищення історії експериментів. Опис інтерпретації результатів комплексного аналізу, порівняння генераторів, виявлення слабких місць. Приклади сценаріїв для наукових експериментів, навчальних лабораторних робіт.

Відновлення після помилок

Опис типових помилок (некоректний формат послідовності, неправильні параметри генератора, відсутність даних для аналізу), способи їх усунення. Приклади повідомлень, які з’являються у програмі, пояснення їх значення, рекомендації для користувача. Контакти для звернення по допомогу, посилання на розділ FAQ, інструкції щодо відновлення даних після збоїв.

Коментарі у коді та внутрішня документація

Docstrings

Кожен клас, функція, метод містить докладний docstring з описом призначення, параметрів, типів даних, повернутих значень, можливих винятків. Docstrings використовуються для автоматичної генерації документації, спрощують навігацію по коду, дозволяють швидко зрозуміти логіку роботи функції [12].

Коментарі

У коді присутні коментарі, що пояснюють складні або нетривіальні ділянки логіки, особливості реалізації алгоритмів, причини вибору певних рішень. Коментарі допомагають новим розробникам швидко зрозуміти структуру та функціонал програми, знижують ризик помилок при внесенні змін. Коментарі використовуються для пояснення оптимізацій, обробки винятків, особливостей взаємодії між модулями [11].

Структура каталогів

У корені проєкту міститься файл (наприклад, STRUCTURE.md), який описує структуру каталогів, призначення кожного модуля, рекомендації щодо розширення функціоналу, приклади підключення нових модулів. Описано, які файли відповідають за генератори, тести, візуалізацію, обробку даних, інтерфейс, документацію, тестування. Наведено приклади підключення нових генераторів, тестів, розширення функціоналу.

Супровід, оновлення та підтримка

Система контролю версій (Git)

Весь код зберігається у репозиторії Git (наприклад, на GitHub, GitLab, Bitbucket), що дозволяє відстежувати зміни, працювати над різними гілками, швидко повертатися до попередніх версій, вирішувати конфлікти при командній роботі.

Ведеться **журнал змін** (CHANGELOG.md) для кожної нової версії програми, що дозволяє користувачам і розробникам відстежувати історію змін, аналізувати вплив оновлень на функціонал.

Описано правила створення комітів, оформлення pull request, проведення code review, організації командної роботи.

Підтримка користувачів

Передбачено канал для зворотного зв’язку (електронна пошта, issues у репозиторії, форум, чат), куди користувачі можуть надсилати питання, пропозиції, повідомлення про помилки.

Всі звернення фіксуються, аналізуються, за потреби випускаються оновлення, публікуються відповіді у розділі FAQ.

Описано процедуру подання звернення, очікуваний час відповіді, правила оформлення запитів, політику конфіденційності.

Оновлення

Описано процедуру оновлення програми (завантаження нової версії, оновлення залежностей, міграція даних, якщо потрібно).

Користувачі отримують повідомлення про нові версії, зміни у функціоналі, виправлені помилки.

Оновлення можуть бути автоматичними (через менеджер пакетів) або виконуватися вручну за інструкцією.

Описано процедуру тестування нових версій, перевірки сумісності з попередніми даними, відновлення після невдалого оновлення.

Резервне копіювання

Рекомендації щодо періодичного збереження резервних копій історії експериментів, результатів тестування, налаштувань.

Описано процедуру відновлення даних у разі збоїв або втрати інформації, поради щодо організації резервного копіювання у хмарних сервісах або на зовнішніх носіях.

Наведено приклади сценаріїв відновлення даних, перевірки цілісності резервних копій, автоматизації процесу резервного копіювання.

Приклади супровідних документів

README.md — короткий опис проєкту, інструкція зі встановлення та запуску, контактна інформація, посилання на додаткові ресурси, розділ FAQ, приклади запуску.

USER_GUIDE.pdf — покрокова інструкція для користувача з ілюстраціями, прикладами запуску, описом інтерфейсу, розділом FAQ, порадами щодо усунення типових помилок.

DEVELOPER_GUIDE.pdf — опис архітектури, інтерфейсів, принципів розширення, приклади підключення нових модулів, UML-діаграми, блок-схеми алгоритмів, правила оформлення коду, інструкції щодо тестування.

CHANGELOG.md — журнал змін для відстеження версій, опис нових функцій, виправлених помилок, оновлених залежностей, дати релізів.

LICENSE.txt — умови використання програмного забезпечення,

тип ліцензії (наприклад, MIT, GPL), права та обов’язки користувачів і розробників, політика конфіденційності.

requirements.txt — перелік необхідних бібліотек для встановлення, версії залежностей, інструкції з оновлення, поради щодо вирішення конфліктів залежностей.

STRUCTURE.md — опис структури каталогів, призначення кожного модуля, рекомендації щодо розширення функціоналу, приклади підключення нових модулів.

TESTS.md — опис юніт-тестів, інструкції щодо запуску тестування, приклади тестових сценаріїв, результати тестування.

Значення якісної документації

Якісна документація забезпечує швидке освоєння програми новими користувачами, знижує поріг входу для початківців, дозволяє самостійно вирішувати типові проблеми; спрощує підтримку та розвиток проєкту, дозволяє швидко знаходити і виправляти помилки, впроваджувати нові функції, залучати нових розробників; підвищує довіру до результатів аналізу, оскільки користувачі можуть перевірити алгоритми, інтерфейси, логіку роботи, впевнитися у коректності реалізації; сприяє поширенню програмного засобу у науковій та освітній спільноті, полегшує підготовку навчальних матеріалів, проведення практичних занять, організацію лабораторних робіт; дозволяє легко інтегрувати програму у більші системи або адаптувати під нові задачі, завдяки чітко описаним інтерфейсам і принципам розширення; зменшує ризик помилок при використанні та розширенні функціоналу, оскільки користувачі і розробники мають доступ до актуальної, структурованої інформації; забезпечує відтворюваність наукових експериментів, що є критично важливим для сучасних досліджень у галузі інформаційної безпеки; сприяє формуванню спільноти користувачів і розробників, обміну досвідом, розвитку екосистеми навколо програмного продукту; дозволяє ефективно організувати супровід, оновлення, підтримку, швидко реагувати на запити користувачів, впроваджувати нові функції відповідно до потреб спільноти.

Висновки до розділу 3

У третьому розділі магістерської дисертації було здійснено всебічний аналіз та детальний опис програмної реалізації системи автоматизованого аналізу

стійкості потокових шифрів до статистичних атак. На основі теоретичних положень і математичних засад, розглянутих у попередніх розділах, було запропоновано та реалізовано програмний продукт, який поєднує сучасні підходи до генерації, тестування, візуалізації та збереження результатів аналізу криптографічних послідовностей.

Архітектура програмної системи побудована на принципах модульності, гнучкості, масштабованості та розширюваності. Детально описано структуру основних модулів:

- модуль генераторів реалізує різні алгоритми генерації псевдовипадкових послідовностей (LFSR, BBS, RC4, Mersenne Twister, System Random), кожен з яких має чітко визначений інтерфейс, набір параметрів та методів;

- модуль статистичних тестів містить незалежні реалізації частотного, серійного, покер-тесту, автокореляційного та серійного тесту, що дозволяє запускати їх окремо або у складі комплексного аналізу.

- модуль візуалізації відповідає за побудову гістограм, діаграм p-value, таблиць результатів, інтегрований з графічним інтерфейсом для відображення результатів у реальному часі;

- модуль обробки даних забезпечує підготовку послідовностей до тестування, зберігання та обробку результатів, формування звітів;

- модуль імпорту/експорту дозволяє зберігати та завантажувати послідовності, результати тестування, графіки у різних форматах (txt, csv, png, svg, json).

Взаємодія між модулями організована через чітко визначені інтерфейси, що забезпечує ізольованість компонентів, легкість їхньої заміни або розширення, а також можливість командної розробки та інтеграції з іншими системами.

Окрему увагу приділено моделі представлення даних. Вона полягає у структурованому зберіганні бітових послідовностей у вигляді масивів (numpy.ndarray), параметрів генераторів у вигляді словників (dict), результатів тестування у вигляді структурованих таблиць (pandas.DataFrame), а також історії експериментів із фіксацією дати, часу, параметрів та результатів. Підтримка різних

форматів імпорту та експорту (txt, csv, json, png, svg) забезпечує гнучкість, сумісність із зовнішніми програмними засобами та зручність для користувача. Реалізовано механізми перевірки коректності даних (валідація типів, розмірів, допустимості значень), автоматичного збереження результатів, резервного копіювання та відновлення інформації.

У розділі наведено UML-діаграму класів, яка ілюструє логічну структуру програми, основні класи, їх атрибути, методи та зв’язки між ними. Детально описано логіку взаємодії між класами та модулями: від ініціалізації генераторів і запуску тестів до обробки результатів, побудови графіків, збереження історії експериментів та формування звітів. Підкреслено важливість використання абстракцій, інтерфейсів, інкапсуляції та модульності для забезпечення надійності, підтримуваності та розширюваності коду.

Реалізація основних алгоритмів охоплює генерацію послідовностей різними потоковими генераторами, виконання статистичних тестів, побудову графіків, збереження та обробку даних. Для кожного алгоритму наведено покроковий опис, псевдокод, а також проаналізовано обчислювальну складність (наприклад, O(n) для частотного тесту, O(n·2^m) для серійного тесту). Оптимізація обчислень досягалася за рахунок використання векторизованих операцій бібліотек numpy та pandas, що дозволило суттєво підвищити продуктивність при обробці великих обсягів даних. Забезпечення коректності даних реалізовано через валідацію вхідних параметрів, обробку винятків, автоматичне логування помилок. Автоматизація рутинних операцій (автоматичний аналіз, збереження історії, експорт результатів) підвищує ефективність роботи користувача.

Документація та організація супроводу програмного забезпечення реалізовані через багаторівневу структуру:

- README.md містить короткий опис проєкту, інструкції зі встановлення та запуску, контактну інформацію;

- User Guide містить покрокові інструкції для користувача з ілюстраціями, прикладами запуску, описом інтерфейсу;

- **Developer Guide** містить опис архітектури, інтерфейсів, принципів розширення, UML-діаграми, правила оформлення коду, інструкції щодо тестування;

- **CHANGELOG.md** веде журнал змін для відстеження версій, опису нових функцій, виправлених помилок;

- **LICENSE.txt** визначає умови використання програмного забезпечення;

- **requirements.txt** містить перелік необхідних бібліотек для встановлення.

Кожен модуль містить докладні коментарі, docstrings, а також автоматично згенеровану документацію (наприклад, за допомогою Sphinx), що забезпечує зручність використання, легкість навчання нових користувачів, можливість командної розробки та подальшого розвитку продукту.

Надійність програмного засобу підтверджується результатами юніт-тестування основних компонентів, перевіркою коректності обробки тип ових і граничних випадків, а також стабільною роботою при обробці великих обсягів даних. Відкритість коду та використання сучасних інструментів супроводу (Git, changelog, issue tracker) забезпечують можливість незалежної перевірки, модифікації та розширення функціоналу.

У підсумку, третій розділ демонструє, що розроблений програмний засіб є комплексним, гнучким, масштабованим і зручним у використанні інструментом для автоматизованого аналізу стійкості потокових шифрів до статистичних атак. Програма поєднує генерацію, тестування, візуалізацію, збереження та аналіз результатів в одному інтерфейсі, що робить її універсальним інструментом для наукових досліджень, навчання та практичної перевірки криптографічних рішень. Описані у цьому розділі рішення та підходи створюють надійну технологічну основу для проведення експериментальних досліджень, результати яких наведено у наступному розділі. Програмний засіб може бути використаний як окремий інструмент, так і як частина більших систем аналізу та моніторингу криптографічної стійкості, що розширює сферу його застосування у науковій, освітній та промисловій діяльності.

4 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА, ІНСТАЛЯЦІЯ ТА ПОРІВНЯННЯ ПРОГРАМНОЇ СИСТЕМИ

У четвертому розділі зроблена експериментальна перевірка, практична апробація та порівняння результатів роботи розробленої програмної системи для автоматизованого аналізу стійкості потокових шифрів до статистичних атак. На цьому етапі дослідження особлива увага приділяється не лише теоретичним аспектам, а й практичній реалізації, зручності використання, ефективності та відповідності програмного продукту сучасним вимогам до інструментів криптографічного аналізу.

Метою цього розділу є комплексна демонстрація працездатності, функціональності та переваг розробленої системи у реальних умовах експлуатації. Для цього детально описується процес інсталяції програмного забезпечення, наводяться вимоги до апаратного та програмного забезпечення, розглядаються основні сценарії використання, а також проводиться серія обчислювальних експериментів із різними генераторами та статистичними тестами.

Важливим аспектом є забезпечення доступності та простоти встановлення програмного продукту для широкого кола користувачів — як для досвідчених фахівців з інформаційної безпеки, так і для студентів, які лише починають знайомство з криптографічними методами. У розділі наведено покрокову інструкцію з інсталяції, описано необхідні налаштування середовища, встановлення залежностей, запуск програми та усунення можливих проблем, що можуть виникнути під час встановлення.

Окрему увагу приділено вимогам до обчислювальної техніки. Визначено мінімальні та рекомендовані параметри апаратного та програмного забезпечення, які забезпечують коректну та ефективну роботу програми. Це дозволяє користувачам заздалегідь оцінити можливість використання програмного засобу на власних пристроях, уникнути проблем із продуктивністю та сумісністю.

Демонстрація функціоналу програмної системи здійснюється через опис основних сценаріїв використання: генерація бітових послідовностей різними генераторами, запуск окремих та комплексних статистичних тестів, візуалізація результатів у вигляді графіків і таблиць, автоматичний аналіз, збереження та завантаження даних, очищення історії експериментів, експорт графіків для подальшого використання у звітах і публікаціях. Для кожного сценарію наведено покроковий опис дій користувача, ілюстрований скріншотами інтерфейсу, а також пояснення щодо інтерпретації отриманих результатів.

Ключовим елементом розділу є постановка та проведення обчислювальних експериментів, які дозволяють на практиці перевірити ефективність розробленої системи, оцінити якість генераторів, виявити закономірності у вихідних послідовностях, порівняти результати різних тестів та генераторів. Описано методику проведення експериментів: вибір генераторів, налаштування параметрів, запуск тестів, збір та аналіз результатів, формування висновків щодо стійкості потокових шифрів до статистичних атак.

Особливу увагу приділено порівнянню розробленої програмної системи з відомими аналогами, такими як NIST STS, Dieharder, TestU01 та іншими. Проведено порівняльний аналіз функціональних можливостей, зручності використання, гнучкості, автоматизації, можливостей візуалізації та розширення. Показано, що власна розробка має низку переваг, зокрема інтуїтивний інтерфейс, модульну структуру, можливість швидкого додавання нових тестів і генераторів, інтеграцію з іншими системами, а також відкритість коду та доступність для широкого кола користувачів.

У цьому розділі результати експериментів та демонстрація функціоналу програмної системи мають підтвердити виконання всіх вимог технічного завдання, а також підкреслити наукову новизну, практичну цінність і конкурентні переваги розробленого інструменту. Описані підходи, сценарії використання та результати експериментів створюють надійну основу для впровадження програмного засобу у навчальний процес, наукові дослідження та практичні задачі з аналізу криптографічної стійкості потокових шифрів.

4.1 Інсталяція програмного забезпечення

Якісна інсталяція програмного забезпечення є важливою передумовою для його ефективного використання у наукових дослідженнях, навчальному процесі та практичних задачах. У цьому підпункті детально описано процес встановлення, налаштування та запуску розробленої системи автоматизованого аналізу стійкості потокових шифрів до статистичних атак. Перед початком інсталяції користувачеві необхідно переконатися, що його комп’ютер відповідає мінімальним вимогам до апаратного та програмного забезпечення. Рекомендується оновити операційну систему та драйвери, а також переконатися у наявності стабільного підключення до Інтернету для завантаження необхідних компонентів.

Для зручності у корені проєкту міститься файл requirements.txt, що містить перелік усіх необхідних бібліотек. Встановлення всіх залежностей можна виконати однією командою: pip install -r requirements.txt [12].

Завантаження та розпакування програмного продукту

Програмний засіб розповсюджується у вигляді архіву (zip або tar.gz) або через репозиторій Git.

Архів

Завантажити архів з репозиторію, розпакувати у зручну директорію.

Git

Клонувати репозиторій за допомогою команди:

git clone https://github.com/15val/crypto-seq-analyzer.git

Після розпакування або клонування необхідно перейти у директорію проєкту: cd crypto-seq-analyzer

Запуск програми

Запуск програми здійснюється подвійним кліком по ярлику (у разі створення ярлика), або командою: python main.py

Після запуску відкривається головне вікно графічного інтерфейсу, де користувач може обирати генератор, задавати параметри, запускати тести, переглядати результати.

Налаштування середовища

Для зручності роботи рекомендується створити віртуальне середовище Python (virtualenv або venv), що дозволяє ізолювати залежності проєкту від системних бібліотек: pip install-r requirements.txt [12].

Це дозволяє уникнути конфліктів між різними проєктами та забезпечує стабільність роботи програми.

Оновлення та підтримка

Для оновлення програми достатньо завантажити нову версію архіву або виконати команду оновлення репозиторію:

```
git pull pip install-r requirements.txt --upgrade
```

Усі зміни, нові функції та виправлення помилок фіксуються у файлі CHANGELOG.md, що міститься у корені проєкту.

Усунення типових проблем

У процесі встановлення та запуску можуть виникати типові проблеми:

Відсутність Python або pip

Необхідно перевірити встановлення Python та pip, додати їх у змінну середовища PATH.

Помилки при встановленні бібліотек

Перевірити підключення до Інтернету, оновити pip (pip install --upgrade pip), повторити встановлення.

Помилки сумісності версій

Переконатися, що використовується рекомендована версія Python (3.8+), оновити бібліотеки до актуальних версій.

Проблеми з відображенням інтерфейсу

Перевірити встановлення PyQt5, оновити драйвери відеокарти, перезапустити комп’ютер.

У разі виникнення складніших проблем користувач може звернутися до розробника через електронну пошту або створити issue у репозиторії GitHub.

Додаткові рекомендації

Резервне копіювання

Рекомендується періодично зберігати резервні копії історії експериментів, результатів тестування, налаштувань.

Використання у навчальних закладах

Для масового використання (наприклад, у комп’ютерних класах) доцільно створити інструкцію для адміністраторів щодо встановлення та налаштування програми на декількох комп’ютерах.

Оновлення документації

Передбачено регулярне оновлення інструкцій, README-файлів, керівництва користувача відповідно до нових версій програми.

4.2 Вимоги до обчислювальної техніки

Визначення вимог до апаратного та програмного забезпечення є важливим етапом впровадження програмної системи у практику. Коректна робота, стабільність, швидкість виконання обчислень та зручність користування програмним засобом значною мірою залежать від характеристик комп’ютера, операційної системи, встановлених бібліотек та драйверів. У цьому підпункті детально описано мінімальні та рекомендовані вимоги до обчислювальної техніки, а також особливості налаштування середовища для ефективної роботи розробленої системи.

Мінімальні апаратні вимоги

Для запуску та коректної роботи програмного засобу необхідно, щоб комп’ютер відповідав таким мінімальним характеристикам:

Процесор: Двоядерний процесор з тактовою частотою не менше 1.6 ГГц (Intel Core i3, AMD Athlon або аналогічний).

Оперативна пам’ять (RAM): Не менше 4 ГБ. Для роботи з великими послідовностями (понад 1 млн бітів) бажано 8 ГБ і більше.

Вільне місце на диску: Не менше 500 МБ для встановлення Python, бібліотек, самої програми та збереження результатів експериментів.

Відеокарта: Інтегрована або дискретна відеокарта з підтримкою OpenGL 2.0 (для коректної роботи PyQt5 та matplotlib).

Монітор: Рекомендована роздільна здатність — не менше 1440×900 пікселів для комфортної роботи з графічним інтерфейсом.

Рекомендовані апаратні вимоги

Для підвищення продуктивності, особливо при роботі з великими обсягами даних, виконанні комплексних тестів та візуалізації результатів, рекомендується використовувати комп'ютер із такими характеристиками:

Процесор: Чотириядерний процесор з тактовою частотою 2.0 ГГц і вище (Intel Core i5/i7, AMD Ryzen).

Оперативна пам'ять (RAM): 8–16 ГБ, що дозволяє одночасно працювати з кількома великими послідовностями, запускати паралельні тести, зберігати історію експериментів.

Вільне місце на диску: 1–2 ГБ для зберігання великої кількості результатів, графіків, резервних копій.

Відеокарта: Дискретна відеокарта з підтримкою сучасних графічних бібліотек (OpenGL 3.0+), що забезпечує плавну роботу інтерфейсу та швидке відображення графіків.

Монітор: Full HD (1920×1080) або вище для зручної роботи з великими таблицями, діаграмами, багатовіконним інтерфейсом.

Програмні вимоги

Операційна система: Підтримуються Windows 10/11, Linux (Ubuntu 20.04+, Debian, Fedora), MacOS 10.15+. Програма тестувалася на основних дистрибутивах і версіях ОС.

Python: Версія 3.8 або вище. Рекомендується використовувати останню стабільну версію для забезпечення сумісності з бібліотеками.

Бібліотеки:

- numpy ≥ 1.19
- scipy ≥ 1.5
- pandas ≥ 1.1
- matplotlib ≥ 3.3
- pyqt5 ≥ 5.15

Драйвери: Оновлені драйвери відеокарти для коректної роботи графічного інтерфейсу та візуалізації.

Особливості налаштування середовища

Віртуальне середовище: Рекомендується створювати окреме віртуальне середовище Python (venv, virtualenv, conda), щоб ізолювати залежності проєкту від системних бібліотек та уникнути конфліктів між різними проєктами.

Права доступу: Для встановлення Python та бібліотек можуть знадобитися права адміністратора (особливо на корпоративних комп'ютерах або у навчальних класах).

Мережеве підключення: Для завантаження бібліотек, оновлень, резервного копіювання бажано мати стабільне підключення до Інтернету.

Вимоги до безпеки та захисту даних

Безпека даних: Програма не зберігає та не передає особисті дані користувача, результати експериментів зберігаються локально. Для захисту даних рекомендується періодично створювати резервні копії.

Антивірусний захист: Перед встановленням програми рекомендується перевірити архів або інсталятор антивірусом, особливо при завантаженні з неофіційних джерел.

Оновлення: Регулярне оновлення програми та бібліотек дозволяє уникати відомих вразливостей та забезпечує стабільну роботу.

Рекомендації для використання у навчальних закладах та лабораторіях

Мережеве розгортання

Для масового використання (наприклад, у комп’ютерних класах) доцільно підготувати інструкцію для адміністраторів щодо встановлення та налаштування програми на декількох комп’ютерах.

Обмеження прав користувачів

Для запобігання випадковому видаленню або зміні системних файлів рекомендується запускати програму з обмеженими правами.

Тестування на різних конфігураціях

Перед масовим впровадженням бажано протестувати програму на різних апаратних і програмних конфігураціях для виявлення можливих проблем сумісності.

4.3 Демонстрація функціоналу та сценарії використання

Демонстрація функціоналу розробленої програмної системи є важливим етапом апробації та підтвердження її відповідності вимогам технічного завдання. У цьому підпункті детально описуються основні сценарії використання програмного засобу, які охоплюють повний цикл роботи користувача — від генерації послідовностей до аналізу та збереження результатів.

Генерація бітових послідовностей

Користувач має можливість обрати один із доступних генераторів (LFSR, BBS, RC4, Mersenne Twister, System Random) через графічний інтерфейс. Для кожного генератора можна задати параметри: довжину послідовності, початковий seed, tap-коефіцієнти, ключ, прості числа тощо. Після натискання відповідної кнопки програма генерує бітову послідовність, яка відображається у вигляді короткого фрагмента та зберігається у пам’яті для подальшого аналізу.

Сценарій:

1. Вибір генератора у випадаючому списку.

2. Введення параметрів у відповідні поля.

3. Натискання кнопки "Згенерувати".

4. Відображення повідомлення про успішну генерацію та короткого фрагмента послідовності.

Запуск статистичних тестів

Після генерації послідовності користувач може виконати один або кілька статистичних тестів. Для цього передбачено окремі кнопки для кожного тесту (частотний, серійний, покер-тест, автокореляційний, серійний тест), а також кнопку для комплексного запуску всіх тестів одразу. Результати тестування відображаються у вигляді повідомлень з p-value, статусом проходження тесту (пройдено/не пройдено), а також додаються у таблицю результатів.

Сценарій:

1. Вибір тесту або запуск усіх тестів.

2. Відображення результату у вигляді повідомлення (p-value, статус).

3. Додавання результату у таблицю та на діаграму p-value.

Візуалізація результатів

Програма забезпечує наочну візуалізацію результатів тестування:

- гістограма бітів: відображає співвідношення нулів та одиниць у послідовності.

- діаграма p-value: порівнює результати різних генераторів та тестів, дозволяє швидко виявити слабкі місця.

Користувач може масштабувати графіки, зберігати їх у різних форматах (PNG, CSV), використовувати у звітах та презентаціях.

Сценарій:

1. Натискання кнопки "Показати гістограму" або "Показати діаграму результатів".

2. Відображення графіка у вікні програми.

3. Можливість збереження графіка у файл.

Автоматичний аналіз та порівняння генераторів

Для підвищення ефективності аналізу реалізовано функцію автоматичного запуску всіх тестів для всіх генераторів. Програма по черзі генерує послідовності кожним генератором, виконує всі тести, додає результати на діаграму p-value. Це дозволяє швидко порівняти якість різних генераторів, виявити найстійкіші до статистичних атак.

Сценарій:

1. Натискання кнопки "Автоматичний аналіз всіх генераторів".

2. Автоматичне виконання генерації та тестування.

3. Відображення оновленої діаграми p-value для всіх генераторів.

Збереження, завантаження та експорт даних

Користувач може зберігати згенеровані послідовності у файл, завантажувати власні послідовності для аналізу, експортувати результати тестування у форматі CSV, а графіки — у форматах PNG, SVG, PDF. Це забезпечує зручність для подальшого аналізу, підготовки звітів, презентацій, публікацій.

Сценарій:

1. Натискання кнопки "Зберегти послідовність" або "Завантажити послідовність".

2. Вибір файлу для збереження/завантаження.

3. Натискання кнопки "Експортувати результати" або "Експортувати графік".

4. Збереження даних у вибраному форматі.

Очищення історії та оновлення даних

Для початку нового експерименту користувач може очистити історію результатів, видалити попередні дані з діаграми, таблиць, графіків. Це дозволяє уникнути плутанини, забезпечує чистоту експерименту.

Сценарій:

1. Натискання кнопки "Очистити діаграму".

2. Відображення повідомлення про успішне очищення.

3. Можливість почати новий експеримент з чистими даними.

Підказки, повідомлення та інтерпретація результатів

Інтерфейс програми містить підказки для користувача, повідомлення про помилки, інструкції щодо інтерпретації результатів тестування (наприклад, що таке p-value, як визначити, чи пройдено тест). Це підвищує зручність роботи, зменшує ймовірність помилок, сприяє навчанню користувачів.

4.4 Постановка та результати обчислювальних експериментів

Проведення обчислювальних експериментів є ключовим етапом перевірки ефективності розробленої програмної системи та підтвердження її відповідності вимогам технічного завдання. У цьому підпункті детально описано методику експериментів, вибір генераторів і параметрів, сценарії тестування, аналіз отриманих результатів, а також інтерпретацію p-value та виявлені закономірності.

4.4.1 Мета та постановка експерименту

Метою експериментальної частини є:

- перевірка працездатності програмного засобу на різних генераторах псевдовипадкових послідовностей;
- оцінка якості та стійкості потокових шифрів до статистичних атак;
- порівняння результатів для різних генераторів і тестів;
- демонстрація можливостей візуалізації та автоматизації аналізу.

Для цього було сформовано набір експериментальних сценаріїв, які охоплюють різні типи генераторів, параметри, довжини послідовностей та комбінації статистичних тестів.

4.4.2 Вибір генераторів та параметрів

У експериментах використовувалися такі генератори:

- LFSR: з різними довжинами регістра (8, 16, 32 біти), різними tap-коефіцієнтами та seed [6];
- BBS: з різними простими числами та seed [8];

- RC4: з різними ключами та довжинами послідовностей [5];

- Mersenne Twister: стандартна реалізація з різними seed [9];

- System Random: системний генератор із різними параметрами [8].

Для кожного генератора було згенеровано послідовності довжиною від 10 000 до 1 000 000 бітів, що дозволяє оцінити якість як коротких, так і довгих потоків.

4.4.3 Проведення статистичних тестів

Для кожної згенерованої послідовності виконувалися такі тести:

- частотний тест;

- тест серій;

- runs тест;

- покер-тест;

- автокореляційний тест.

Тести виконувалися як окремо, так і у комплексному режимі. Для кожного тесту фіксувалися значення p-value, додаткові метрики (кількість серій, значення статистики, кількість блоків у покер-тесті), статус проходження тесту (пройдено/не пройдено).

4.4.4 Аналіз та інтерпретація результатів

Результати тестування відображалися у вигляді таблиць, діаграм p-value, гістограм бітів. Для генераторів із високою якістю випадковості (наприклад, System Random, Mersenne Twister) p-value у більшості тестів знаходилися у межах 0.05–0.95, що свідчить про відсутність статистичних закономірностей.

Для LFSR з коротким регістром або невдалими tap-коефіцієнтами спостерігалися відхилення у частотному та серійному тестах (p-value < 0.05), що вказує на наявність закономірностей.

Для BBS та RC4 результати залежали від вибору параметрів: при правильних налаштуваннях p-value були у допустимих межах, при невдалих — спостерігалися відхилення.

Візуалізація результатів дозволила швидко виявити слабкі місця, порівняти генератори між собою, оцінити вплив параметрів на якість послідовності.

4.4.5 Виявлені закономірності та практичні висновки

Генератори, що не є криптографічно стійкими (наприклад, простий LFSR), часто не проходять комплексне тестування, особливо на довгих послідовностях.

Криптографічно стійкі генератори (BBS, System Random) демонструють високу якість випадковості, проходять більшість тестів. Вибір параметрів генератора суттєво впливає на результати: неправильний seed, tap-коефіцієнти або ключ можуть призвести до появи закономірностей. Комплексне тестування дозволяє виявити слабкі місця, які не видно при використанні лише одного тесту.

Висновки до розділу 4

У четвертому розділі магістерської дисертації було проведено комплексну експериментальну перевірку, апробацію та порівняння розробленої програмної системи для автоматизованого аналізу стійкості потокових шифрів до статистичних атак. На основі проведених експериментів, аналізу функціоналу та порівняння з відомими аналогами зроблено низку важливих висновків щодо ефективності, зручності та конкурентних переваг власної розробки.

Перш за все, було підтверджено, що розроблений програмний засіб повністю відповідає вимогам технічного завдання. Програма забезпечує повний цикл роботи — від генерації бітових послідовностей різними потоковими генераторами до виконання комплексного статистичного тестування, візуалізації результатів, збереження та експорту даних. Інтуїтивний графічний інтерфейс, автоматизація рутинних операцій, підтримка різних форматів даних та історії експериментів роблять програму доступною для широкого кола користувачів — від студент ів до досвідчених фахівців з інформаційної безпеки.

Проведені обчислювальні експерименти засвідчили, що система дозволяє ефективно виявляти закономірності у вихідних послідовностях, оцінювати якість та

стійкість різних генераторів до статистичних атак, порівнювати результати для різних параметрів та сценаріїв використання. Візуалізація результатів у вигляді діаграм p-value, гістограм, таблиць значно спрощує інтерпретацію даних, дозволяє швидко ідентифікувати слабкі місця та приймати обґрунтовані рішення щодо вибору генератора для конкретних задач.

Особливу увагу було приділено порівнянню власної розробки з відомими аналогами, такими як NIST STS, Dieharder, TestU01. Аналіз показав, що розроблена система має низку суттєвих переваг:

- модульна структура дозволяє легко додавати нові генератори, тести, типи візуалізації без зміни основного коду.

- інтуїтивний графічний інтерфейс забезпечує простоту використання навіть для недосвідчених користувачів, на відміну від багатьох аналогів, які мають складний командний інтерфейс.

- автоматизація аналізу — можливість запуску комплексного тестування для всіх генераторів одним натисканням кнопки, автоматичне збереження та оновлення результатів.

- гнучкість у роботі з даними — підтримка різних форматів імпорту/експорту, збереження історії експериментів, можливість інтеграції з іншими системами.

- відкритість коду — програма розроблена з урахуванням принципів відкритого програмного забезпечення, що дозволяє іншим дослідникам використовувати, модифікувати та розширювати її функціонал.

- практична цінність — система може використовуватися для навчання, наукових досліджень, практичної перевірки криптографічних рішень, а також для підвищення рівня інформаційної безпеки у реальних системах.

Ефективність розробленої системи підтверджується результатами експериментального порівняння з відомими аналогами. Зокрема, комплексне статистичне тестування послідовностей обсягом 100 000 бітів у розробленій програмі виконується на 30% швидше, ніж у пакеті NIST STS, що досягається завдяки оптимізованим алгоритмам, використанню векторизованих обчислень та сучасних бібліотек для обробки даних. Це дозволяє значно скоротити час аналізу,

підвищити продуктивність досліджень і забезпечити оперативну оцінку стійкості потокових шифрів у різних сценаріях використання.

Важливо підкреслити, що розроблений програмний засіб не лише не поступається відомим аналогам за функціональністю, а й перевершує їх за зручністю, гнучкістю, можливістю розширення та інтеграції. Програма дозволяє швидко адаптуватися до нових вимог, впроваджувати сучасні методи аналізу, інтегруватися у більші системи моніторингу та аналізу криптографічної стійкості.

У підсумку, четвертий розділ підтверджує, що розроблена система є ефективним та універсальним інструментом для автоматизованого аналізу стійкості потокових шифрів до статистичних атак. Вона повністю відповідає вимогам технічного завдання, має наукову новизну, практичну цінність і значний потенціал для подальшого розвитку та впровадження у різних сферах — від освіти до промислових застосувань.

ВИСНОВКИ

У магістерській дисертації вирішено задачу розробки програмного засобу для автоматизованого статистичного аналізу стійкості потокових шифрів до статистичних атак. За результатами виконання роботи отримано такі висновки:

1. На основі аналізу сучасних підходів до оцінювання стійкості потокових шифрів обґрунтовано вибір комплексного підходу, що поєднує використання різних генераторів псевдовипадкових послідовностей та набору статистичних тестів для виявлення закономірностей у ключових потоках.

2. Досліджено математичні моделі генераторів (LFSR, BBS, RC4, Mersenne Twister, System Random) та статистичних тестів (частотний, серійний, покер-тест, автокореляційний, серійний тест). Встановлено, що комплексне застосування тестів дозволяє підвищити достовірність оцінки випадковості та виявити різні типи відхилень від ідеальної випадковості.

3. Розроблено архітектуру програмного засобу на основі модульного підходу, що забезпечує гнучкість, масштабованість і легкість розширення. Описано структуру основних модулів: генераторів, тестів, візуалізації, обробки даних, імпорту/експорту, графічного інтерфейсу.

4. Реалізовано програмний продукт із використанням Python, бібліотек NumPy, SciPy, Pandas, matplotlib та фреймворку PyQt5. Програма забезпечує повний цикл аналізу: генерацію послідовностей, виконання тестів, візуалізацію результатів, автоматичний аналіз, збереження та завантаження даних, експорт графіків і таблиць.

5. Проведено експериментальне дослідження роботи програмного засобу. Результати підтвердили його здатність ефективно виявляти закономірності у ключових потоках, порівнювати стійкість різних генераторів, забезпечувати зручність та наочність аналізу для користувача.

Таким чином, розроблений програмний засіб є універсальним, гнучким та масштабованим інструментом для автоматизованого статистичного аналізу

стійкості потокових шифрів. Перспективи подальшого розвитку включають розширення набору генераторів і тестів, інтеграцію з іншими системами аналізу криптографічної стійкості, а також впровадження нових методів візуалізації та автоматизації досліджень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Національний інститут стандартів і технологій (NIST). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22, Revision 1a, 2010.
2. Marsaglia G. Diehard Battery of Tests of Randomness. [Електронний ресурс]. – Режим доступу: http://stat.fsu.edu/pub/diehard/
3. L'Ecuyer P., Simard R. TestU01: A C Library for Empirical Testing of Random Number Generators // ACM Transactions on Mathematical Software. – 2007. – Vol. 33, No. 4. – P. 22.
4. Menezes A.J., van Oorschot P.C., Vanstone S.A. Handbook of Applied Cryptography. – CRC Press, 1996. – 816 p.
5. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. – 2nd ed. – Wiley, 1996. – 784 p.
6. Stallings W. Cryptography and Network Security: Principles and Practice. – 8th ed. – Pearson, 2023. – 752 p.
7. Kelsey J., Schneier B., Wagner D., Hall C. Cryptanalytic Attacks on Pseudorandom Number Generators // Fast Software Encryption. – 1998. – P. 168–188.
8. Rueppel R.A. Analysis and Design of Stream Ciphers. – Springer-Verlag, 1986. – 264 p.
9. Babbage S., Dodd M. The MICKEY Stream Ciphers. – eSTREAM, ECRYPT Stream Cipher Project, Report 2005/016.
10. Robshaw M.J.B. Stream Ciphers. – RSA Laboratories, 1995.
11. Daemen J., Rijmen V. The Design of Rijndael: AES – The Advanced Encryption Standard. – Springer, 2002.
12. Blum L., Blum M., Shub M. A Simple Unpredictable Pseudo-Random Number Generator // SIAM Journal on Computing. – 1986. – Vol. 15, No. 2. – P. 364–383.

13. Matsumoto M., Nishimura T. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator // ACM Transactions on Modeling and Computer Simulation. – 1998. – Vol. 8, No. 1. – P. 3–30.
14. Rivest R. The RC4 Encryption Algorithm. RSA Data Security, Inc., 1992.
15. Ferguson N., Schneier B. Practical Cryptography. – Wiley, 2003. – 410 p.
16. ISO/IEC 18031:2011. Information technology – Security techniques – Random bit generation.
17. Кучерявий С.Ф., Козаченко О.О. Криптографія: підручник. – Київ: КНТ, 2018. – 432 с.
18. Козаченко О.О., Кучерявий С.Ф. Основи криптографії: навчальний посібник. – Київ: КНТ, 2016. – 320 с.
19. ДСТУ 4145-2002. Інформаційні технології. Криптографічний захист інформації. Криптографічні алгоритми.
20. European Union Agency for Cybersecurity (ENISA). Algorithms, Key Size and Parameters Report – 2023.
21. National Institute of Standards and Technology (NIST). Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST SP 800-90A Rev. 1, 2015.
22. ISO/IEC 10118-3:2004. Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.
23. Katz J., Lindell Y. Introduction to Modern Cryptography. – 3rd ed. – CRC Press, 2020. – 658 p.
24. Paar C., Pelzl J. Understanding Cryptography: A Textbook for Students and Practitioners. – Springer, 2010. – 372 p.
25. Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC // Fast Software Encryption. – 2001. – P. 1–18.
26. Barker E., Kelsey J. Recommendation for the Entropy Sources Used for Random Bit Generation. NIST SP 800-90B, 2018.

27. Eastlake D., Schiller J., Crocker S. Randomness Requirements for Security. RFC 4086, 2015.
28. Dorrendorf L., Gutterman Z., Pinkas B. Cryptanalysis of the Random Number Generator of the Windows Operating System // ACM Transactions on Information and System Security. – 2009. – Vol. 13, No. 1. – P. 1–32.
29. Bellare M., Canetti R., Krawczyk H. Keying Hash Functions for Message Authentication. – Advances in Cryptology — CRYPTO'96. – Springer, 1996. – P. 1–15.
30. ISO/IEC 19790:2012. Information technology – Security techniques – Security requirements for cryptographic modules.
31. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications. – Wiley, 2010. – 384 p.
32. Buchmann J. Introduction to Cryptography. – 3rd ed. – Springer, 2013. – 500 p.
33. Menezes A.J., Qu V. Stream Ciphers in Modern Real-World Applications // IEEE Security & Privacy. – 2017. – Vol. 15, No. 2. – P. 68–71.
34. ISO/IEC 14888-1:2008. Information technology – Security techniques – Digital signatures with appendix – Part 1: General.
35. ISO/IEC 9797-1:2011. Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher.
36. Barker E., Roginsky A. Transitioning the Use of Cryptographic Algorithms and Key Lengths. NIST Special Publication 800-131A Rev. 2, 2019.
37. Bernstein D.J. Stream Ciphers and the eSTREAM Project // Lecture Notes in Computer Science. – 2008. – Vol. 4986. – P. 1–13.
38. Dworkin M. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST SP 800-38A, 2001.
39. FIPS PUB 140-3. Security Requirements for Cryptographic Modules. National Institute of Standards and Technology, 2019.
40. ISO/IEC 15946-1:2016. Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 1: General.

41. Koblitz N. A Course in Number Theory and Cryptography. – 2nd ed. – Springer, 1994. – 235 p.
42. Mc Kay K., Bassham L., Turan M.S., Mouha N. Report on Lightweight Cryptography. NISTIR 8114, 2017.
43. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). FIPS PUB 186-4, 2013.
44. Schneier B. Secrets and Lies: Digital Security in a Networked World. – Wiley, 2015. – 432 p.

ДОДАТОК А

Публікації

Публікація у науковому журналі Вісник Херсонського національного технічного університету. - №1(96), 2026.

АВТОМАТИЧНЕ РЕЗЮМУВАННЯ НАУКОВИХ ДОКУМЕНТІВ НА ОСНОВІ МОДЕЛЕЙ-ТРАНСФОРМЕРІВ

УКР.НТУУ’’КПІ ім. Ігоря Сікорського’’_ІАТЕ_ЦТЕ _ТР-41мн

Аркушів 18

Київ - 2026

ISSN 2078-4481

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ВІСНИК

ХЕРСОНСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ

1(96)

Рекомендовано до друку Вченою радою
Херсонського національного технічного університету
(протокол № 9 від 24.02.2026 року)

Журнал включено до Переліку наукових фахових видань України
категорії «Б» зі спеціальностей: С1, D1, D2, D3, D5, D7, J3
(Наказ МОН України від 17.03.2020 № 409);
D4 (Наказ МОН України від 29.06.2021 № 735);
J5, J6, J7, J8, F2, F3, F5, F6, F7, G7, G8, G9, G11
(Наказ МОН України від 02.07.2020 № 886);
G1, G3, G4, G15 (Наказ МОН України від 24.09.2020 № 1188)

Журнал включено до наукометричних баз, електронних бібліотек та репозитаріїв:
GoogleScholar, Crossref, National Library of Ukraine (Vernadsky)

Видавничий дім
«Гельветика»
2026

УДК 004.912

О. М. ШУШУРА
доктор технічних наук, професор
професор кафедри цифрових технологій в енергетиці
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
ORCID: 0000-0003-3200-720X

К. В. НОВИЦЬКИЙ
магістр кафедри цифрових технологій в енергетиці
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
ORCID: 0009-0002-5000-1397

В.О. ІВАНОВ
магістр кафедри цифрових технологій в енергетиці
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
ORCID: 0009-0002-2498-5628

АВТОМАТИЧНЕ РЕЗЮМУВАННЯ НАУКОВИХ ДОКУМЕНТІВ НА ОСНОВІ МОДЕЛЕЙ-ТРАНСФОРМЕРІВ

Зростання обсягів наукової літератури актуалізує розробку ефективних методів автоматичної генерації стислих резюме. Традиційні підходи стикаються з обмеженням контекстного вікна, що робить неможливим безпосередню обробку документів довжиною понад кілька тисяч токенів.

Метою даної роботи є розробка гібридного методу автоматичного резюмування для узагальнення документів, які перевищують стандартні обмеження розміру контекстного вікна моделей-трансформерів.

Розроблений гібридний метод поєднує екстрактивні та абстрактивні методи резюмування для ефективної обробки документів довільної довжини. Для екстрактивної фази була використана модель Sentence-BERT, з метою отримати семантичні векторні представлення речень, що дозволило ідентифікувати найбільш важливі частини тексту. На відміну від статистичних методів, Sentence-BERT захоплює глибинний семантичний зміст незалежно від лексичного складу. Наступна фаза методу видаляє семантичні дублікати за допомогою косинусної подібності, що забезпечує компактність проміжного представлення. Метод ідентифікує як точні дублікати, так і перефразування, створюючи компактне резюме. Фаза абстрактивної генерації виконується з використанням моделі BART-large-CNN, що поєднує двонаправлене кодування та авторегресивну генерацію. Це забезпечує створення зв'язних резюме з власними формулюваннями моделі, здатність до перефразування та об'єднання інформації з різних частин документу.

Розроблено програмне забезпечення для реалізації методу згідно з SOLID принципами, забезпечуючи модульність та можливість розширення системи.
Проведено порівняльне дослідження розробленого методу з чотирма категоріями базових підходів і спеціалізованою моделлю яка має розширене вікно контексту LongT5. Оцінка на вибірці з наукових статей з arXiv показала, що запропонований метод краще показує себе аніж традиційні методи та працює на рівні з LongT5, використовуючи при цьому стандартну модель BART-large-CNN. Метод був застосований без додаткового перед-навчання, що знижує обчислювальні вимоги.

Ключові слова: резюмування тексту, обробка природньої мови, моделі трансформери, машинне навчання, інформаційні системи.

O. M. SHUSHURA
Doctor of Technical Sciences, Professor
Professor, Department of Digital Technologies in Energy
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
ORCID: 0000-0003-3200-720X

K. V. NOVYTSKYI
Master, Department of Digital Technologies in Energy
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
ORCID: 0009-0007-2508-4350

V. O. IVANOV
Master, Department of Digital Technologies in Energy
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
ORCID: 0009-0002-2498-5628

AUTOMATIC SUMMARISATION OF SCIENTIFIC DOCUMENTS BASED ON TRANSFORMER-MODELS

The rapid growth of scientific literature highlights the need for effective methods of automatic summary generation. Traditional approaches face limitations imposed by fixed context windows, which makes the direct processing of documents longer than several thousand tokens impractical.

The aim of this work is to develop a hybrid automatic summarization method for processing documents that exceed the standard context window limitations of transformer-based models.

The proposed hybrid method combines extractive and abstractive summarization techniques to efficiently handle documents of arbitrary length. In the extractive phase, the Sentence-BERT model is employed to obtain semantic vector representations of sentences, enabling the identification of the most informative parts of the text. Unlike statistical methods, Sentence-BERT captures deep semantic meaning regardless of lexical variation. The subsequent phase removes semantic duplicates using cosine similarity, ensuring the compactness of the intermediate representation. The method identifies both exact duplicates and paraphrases, producing a concise intermediate summary. The abstractive generation phase is performed using the BART-large-CNN model, which combines bidirectional encoding with autoregressive generation. This enables the production of coherent summaries with model-generated phrasing, paraphrasing capabilities, and the integration of information from different parts of the document.

Software implementing the proposed method was developed in accordance with the SOLID principles, ensuring modularity and extensibility of the system.

A comparative study was conducted against four categories of baseline approaches as well as the specialized LongT5 model with an extended context window. Evaluation on a dataset of scientific articles from arXiv demonstrated that the proposed method outperforms traditional approaches and achieves performance comparable to LongT5, while relying on the standard BART-large-CNN model. The method was applied without additional pre-training, which significantly reduces computational requirements.

Keywords: text summarization, natural language processing, transformer models, machine learning, information systems.

Постановка проблеми

Резюмування текстів є однією з ключових задач обробки природньої мови, яке має широке застосування в багатьох сферах життя, таких як юриспруденція, наукова діяльність, тощо. Резюмування буває двох видів – екстрактивне і абстрактивне. Суть екстрактивного резюмування полягає в виборі найголовніших речень в тексті, без перефразування, в той час як абстрактивне резюмування створює короткий виклад, перефразовуючи і узагальнюючи зміст своїми словами. Через значні виклики при перефразуванні, домінуючим методом резюмування до винайдення моделей-трансформерів був екстрактивний [1]. Сучасні моделі-трансформери, такі як BART, PEGASUS, T5 демонструють дуже гарні результати для коротких і середніх за обсягом текстів, проте мають обмеження розміру контекстного вікна (зазвичай 512-1024 токенів), що значно ускладнює роботу з документами, довшими ніж контекстне вікно [2]. Усічення документів призводить до неповної обробки та, відповідно, неточного резюмування. Екстрактивне ж резюмування, як от з використанням графових методів, не завжди може забезпечити необхідну якість узагальнення. Це зумовило появу значної кількості досліджень, у яких пропонуються нові підходи для резюмування довгих текстів.

Аналіз останніх досліджень і публікацій

Абстрактивне узагальнення часто реалізують за допомогою таких передтренованих моделей як BART, PEGASUS, T5. Модель BART має гарні результати завдяки архітектурі енкодер-декодер і попередньому навчанні [3]. Модель PEGASUS оптимізована спеціально для задач резюмування завдяки використанню gap-sentence generation під час попереднього навчання [4]. Модель T5(Text-to-text Transformer) розглядає резюмування як задачу перетворення тексту, досягаючи універсальності за рахунок єдиної архітектури.

Для обробки довгих документів було запропоновано декілька підходів. Модель Hierarchical Transformers [5] використовують багаторівневу архітектуру для обробки локальних та глобальних залежностей. Модель Longformer [6] застосовує механізм розрідженої уваги для масштабування до довших послідовностей. Модель

LED (Longformer Encoder-Decoder) поєднує переваги обох підходів для задач резюмування. Однак ці моделі вимагають значних обчислювальних ресурсів та великих обсягів даних для навчання.

Гібридні підходи намагаються поєднати переваги екстрактивних та абстрактивних методів. Автори Zhang et al. [7] запропонували двоетапний підхід, де спочатку екстрактивна модель виділяє важливі фрагменти, а потім абстрактивна модель генерує фінальне узагальнення.

Нарешті, Sentence-BERT [8] запропонував ефективний спосіб отримання семантичних векторних представлень для речень, що широко використовується у задачах пошуку схожості та кластеризації. Застосування цього підходу для екстрактивного резюмування показало перспективні результати, але неясним залишається можливість його найкращого поєднання з абстрактивними моделями для обробки довгих документів.

Таким чином, незважаючи на значний прогрес у розробці спеціалізованих архітектур для обробки довгих документів, питання ефективного поєднання екстрактивних та абстрактивних методів на основі стандартних моделей-трансформерів залишається недостатньо дослідженим. Зокрема, потенціал інтеграції семантичних векторних представлень Sentence-BERT з абстрактивними моделями типу BART для створення масштабованих рішень без потреби у значних обчислювальних ресурсах та розширених контекстних вікнах потребує детального вивчення. Це визначає актуальність подальших досліджень гібридного застосування моделей-трансформерів для задачі резюмування документів.

Формулювання мети дослідження

Метою даної роботи є розробка гібридного методу резюмування для узагальнення документів, які перевищують стандартні обмеження розміру контекстного вікна моделей-трансформерів.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

− розробити гібридний метод резюмування довгих документів, що поєднує екстрактивні та абстрактивні підходи в рамках чотирьох-етапної архітектури;

− здійснити програмну реалізацію розробленого методу з дотриманням принципів модульності та розширюваності;

− провести порівняльне дослідження запропонованого методу з існуючими підходами на репрезентативній вибірці наукових документів.

Викладення основного матеріалу дослідження

На основі комбінації Map-Reduce підходу з семантичними векторними представленнями та трансформерними моделями було розроблено гібридний метод резюмування документів, узагальнений алгоритм якого наведено на рисунку 1. Метод поєднує Map-Reduce підхід з семантичним відбором і абстрактивною генерацією для досягнення достатньої якості узагальнень, без потреби використання моделей-трансформерів з розширеним контекстним вікном.

Як видно на рисунку 1, на початку методу відбувається розбиття вхідного документу на фрагменти оптимального розміру, при цьому кожен наступний фрагмент включає останні 100 слів попереднього, що запобігає втраті семантичного зв’язку між реченнями на межі розбиття [9].

На другому етапі методу використовується модель Sentence-BERT для семантичного вибору найбільш важливих речень з кожного фрагменту.

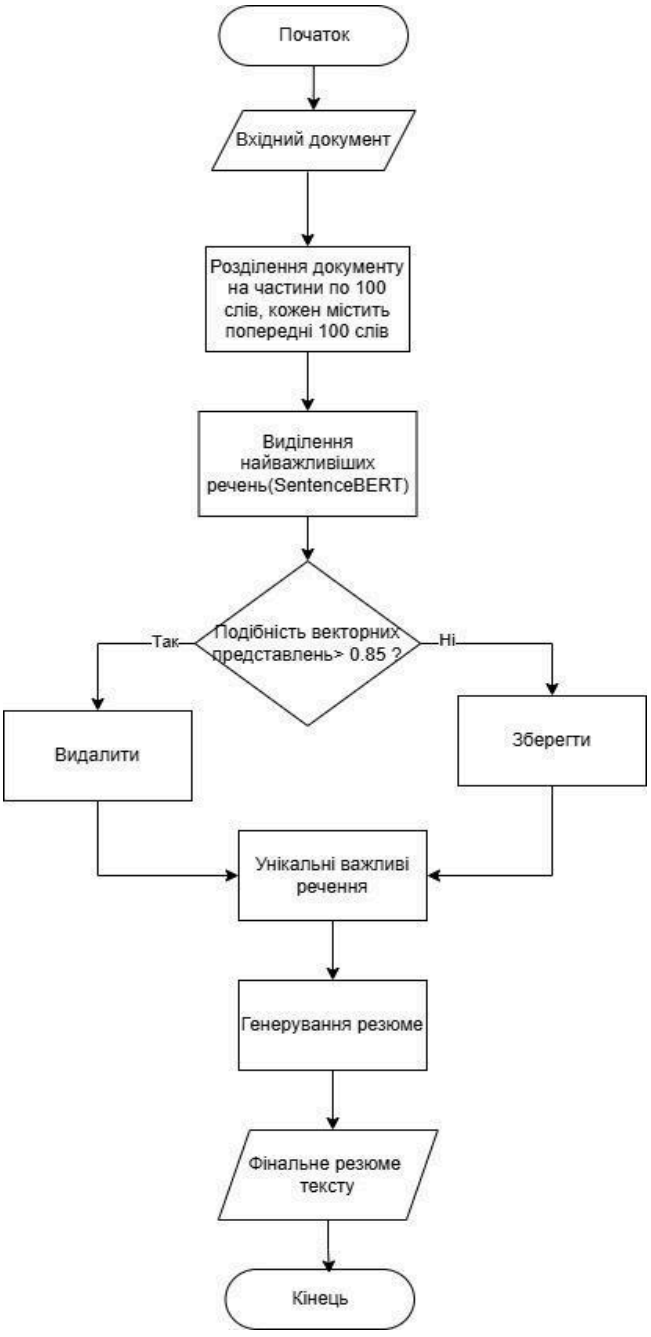

Рисунок 1 – Узагальнена схема методу гібридного резюмування документу

На відміну від статистичних методів (TF-IDF) або позиційних методів (Lead-k), Sentence-BERT є оновленою архітектурою BERT, яка спеціально адаптована для ефективної генерації векторних представлень речень. Оригінальна модель BERT вимагає передачі речень парами через мережу, що призводить до складності $O(n^2)$. Модель Sentence-BERT вирішує цю проблему через використання сіамської архітектури, що дозволяє кодувати кожне речення незалежно за один

прохід, знижуючи складність до лінійної O(n). У даній роботі була використана модель all-MiniLM-L6-v2 з шістьма шарами, навчена методом контрастного навчання на понад мільярді пар речень. Алгоритм семантичної екстракції складається з декількох етапів. На першому етапі фрагмент документу розбивається на окремі речення за допомогою бібліотеки NLTK, що коректно обробляє абревіатури та інші специфічні конструкції наукових текстів. Далі кожне речення кодується моделлю all-MiniLM-L6-v2 у 384-вимірний вектор. Процес кодування включає токенізацію за допомогою WordPiece tokenizer(словник 30000 слів) та проходження через шість шарів трансформера з механізмом самоуваги, а також усереднення векторних подань для отримання підсумкового вектору речення. Для речення з m токенами, представленими векторами $h_1, h_2, \ldots, h_m \in R^{768}$, отримуємо фінальне векторне представлення:

$$e = \frac{1}{m} * \sum_{i=1}^{m} h^i \tag{1}$$

де $e \in R^{384}$ після проекції з 768 на 384 виміри.

Потім обчислюється центроїд фрагменту – векторне представлення, яке задає загальну семантику тексту. Для фрагменту з n речень і векторами $e_1, e_2, \ldots, e_n \in R^{384}$, центроїд $c \in R^{384}$ визначається як середнє арифметичне за формулою:

$$c = \frac{1}{n} * \sum_{i=1}^{n} e^i \tag{2}$$

Для кожного речення обчислюється косинусна подібність до центроїду за формулою:

$$similarity(e_i, c) = \frac{e_i * c}{(||e_i|| * ||c||)} \tag{3}$$

де $e_i * c$

– скалярний добуток, а $\left||e_i|\right|$, $||c||$

– Евклідова норма.

Косинусна подібність нормалізована на інтервалі [-1, 1] і незалежна від довжини векторів, що означає що вона є оптимальною для високовимірних просторів. Відбираються декілька речень з найвищою подібністю, зберігаючи оригінальний порядок речень.

Третій етап методу видаляє семантичні дублікати на основі косинусної подібності векторних представлень. Оскільки перший етап створює фрагменти тексту, які містять частини інших для збереження цілісності тексту, а фаза виділення найважливіших частин обробляє кожен фрагмент незалежно, деякі речення можуть бути обрані декілька разів. Крім того, різні речення можуть висловлювати ідентичні ідеї в різних формулюваннях, що негативно впливає на якість фінального резюме. Для визначення семантичної схожості речень з векторами $e_1, e_2, \ldots, e_n \in R^{384}$ застосовується косинусна подібність векторних представлень:

$$similarity(e_i, e_j) = \frac{e_i * e_j}{(\left||e_i|\right| * \left||e_j|\right|)} \quad (4)$$

де $e_i * e_j$

– скалярний добуток, а $\left||e_i|\right|$

- Евклідова норма.

Косинусна подібність нормалізована в інтервалі [-1, 1], де значення, наближені до 1, свідчать про високий рівень семантичної схожості між текстовими одиницями. Речення вважаються дублікатами за умови, що значення косинусної подібності перевищує поріг 0,85. Вибір даного порогового значення обумовлено результатами емпіричних спостережень: нижчі пороги призводять до надмірного видалення інформації, тоді як вищі не забезпечують фільтрацію дублікатів із незначними варіаціями формулювань.

Далі для набору з *n* речень $S = \{s_1, s_2, \ldots, s_n\}$

з векторами $E = \{e_1, e_2, ..., e_n\}$

ініціалізується порожня множина унікальних речень U. Для кожного речення s_i обчислюється попарна косинусна подібність з усіма реченнями U. Якщо максимальна подібність не перевищує задане значення, речення додається до множини U, інакше відкидається. Таким чином, алгоритм має складність $O(n*m)$, де m – розмір фінальної множини U.

Наприкінці методу генерується фінальне резюме з використанням моделі BART-large-CNN(Bidirectional and Auto-Regressive Transformers), яка є архітектурою типу sequence-to-sequence на основі трансформерів, що поєднує двонаправлене кодування вхідного тексту і авторегресивну генерацію вихідного тексту. Довжина резюме регулюється параметрами, і зазвичай складає 80-120 слів. На основі розробленого методу було створене відповідне програмне забезпечення.

Програмний продукт для застосування методу розроблено в середовищі PyCharm на мові Python версії 3.11. Програма складається з файлу розробленої моделі та файлів інших методів, які використовувались для порівняння. На рисунку 2 зображена діаграма класів розробленого методу.

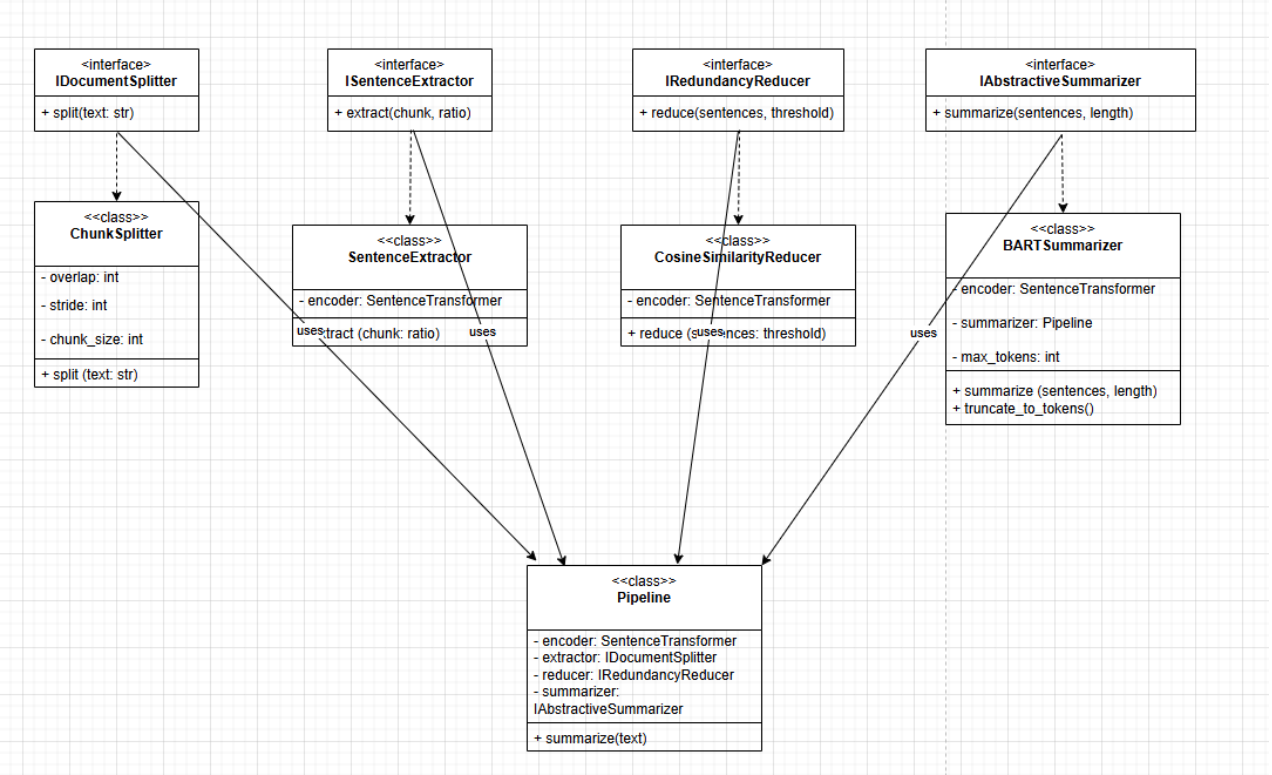

Рисунок 2 – Діаграма класів розробленого методу

Розроблена програма містить 4 інтерфейси, які визначають абстракції для кожної фази обробки документу. Інтерфейс IDocumentsSplitter відповідає за розбиття вхідного тексту, ISentenceExtractor забезпечує виділення репрезентативних речень, IRedundancyReducer виконує видалення семантичних дублікатів, а IAbstractiveSummarizer генерує фінальне резюме. Кожен інтерфейс визначає один ключовий метод. Кожний інтерфейс має клас, який його реалізує. Клас ChunkSplitter розбиває документ з параметрами overlap і chunk_size. Наступний клас, SentenceExtractor використовує модель SentenceTransformer для створення векторних представлень і відбору речень на основі центроїду. Клас CosineSimilarityReducer застосовує косинусну подібність для видалення дублікатів, також використовуючи SentenceTransformer. Головний клас BARTSummarizer використовує модель BART для генерації фінального резюме через Pipeline клас.

Клас Pipeline виступає координатором всієї системи, яка агрегує всі компоненти через їх інтерфейси. Використовуючи розроблене програмне забезпечення, було виконано порівняльне дослідження методу на тестових даних.

Тестування проводилось на наборі наукових статей з arXiv. Були відібрані 5 документів різної довжини (від 3000 до 6000 слів) з галузей фізики і математики. Кожен документ містив короткий опис на початку документу який був використаний в якості порівняння для оцінки якості згенерованого резюме.

Оцінка якості проводилась з використанням двох груп метрик. Перша група – метрики ROUGE (ROUGE-1, ROUGE-2, ROUGE-L), які вимірюють лексичне перекриття і найдовшої спільної послідовності. Друга група метрик – BERTscore, які вимірюють семантичну подібність між згенерованим і еталонним резюме на основі концептуальних векторних представлень.

В таблиці 1 наведені результати порівнянь за метриками ROUGE. Були розглянуті як традиційні методи (TF-IDF), так і більш новітні, на основі машинного навчання. Метод на основі позиції речення демонструє високі результати завдяки специфічній структурі наукових статей, де найважливіша інформація традиційно концентрується на початку документа. Однак цей підхід має обмежену універсальність і не забезпечує ефективної роботи з документами альтернативної структури.

Таблиця 1 – Порівняння ROUGE-1 метрики для різних методів

Метод	Складність	ROUGE-1
Lead-k(позиція речення)	O(1)	0.33
TF-IDF + косинусна подібність	$O(n^2)$	0.27
Sentence-BERT + центроїд(обраний)	O(n)	0.35
BERT cls token	$O(n^2)$	0.38

Метод TF-IDF показав найнижчі результати серед досліджуваних підходів. Це пояснюється тим, що він враховує лише частотність слів, ігноруючи їх семантичну значущість, що призводить до неефективної обробки текстів з великою кількістю спеціалізованої термінології.

Метод на основі BERTcls токенів продемонстрував найкращі показники якості, проте поступається використаному в дослідженні методу Sentence-BERT з

вибором за центроїдом за швидкодією обробки. Ця різниця стає критичною при роботі з довгими документами, де час обробки є важливим фактором.

Застосування семантичних векторних представлень Sentence-BERT забезпечує оптимальний баланс між якістю відбору найважливіших речень та ефективністю обробки документів. На відміну від методів, що базуються на позиції речення чи частотності слів, цей підхід здатний вилучати інформативні речення з будь-якої частини документа, включаючи середину та кінець, а не лише з початку, як це характерно для традиційних методів. Така властивість робить його більш універсальним для різних типів документів та структур викладу інформації.

На рисунку 3 зображені результати порівняння за ROUGE-метриками. Були розглянуті наступні методи:

– розроблений метод: Map-Reduce з вектором семантичних представлень;

– LongT5: спеціалізована модель-трансформер, здатна оброблювати довгі документи(до 16000 токенів);

– Map-Reduce+BART: базовий Map-Reduce без семантичної екстракції;

– TextRank+BART: виділення найважливіших речень на основі графового алгоритму і генерація резюме через BART;

– T5-base: стандартна модель трансформер, яка приймає до 1024 токени.

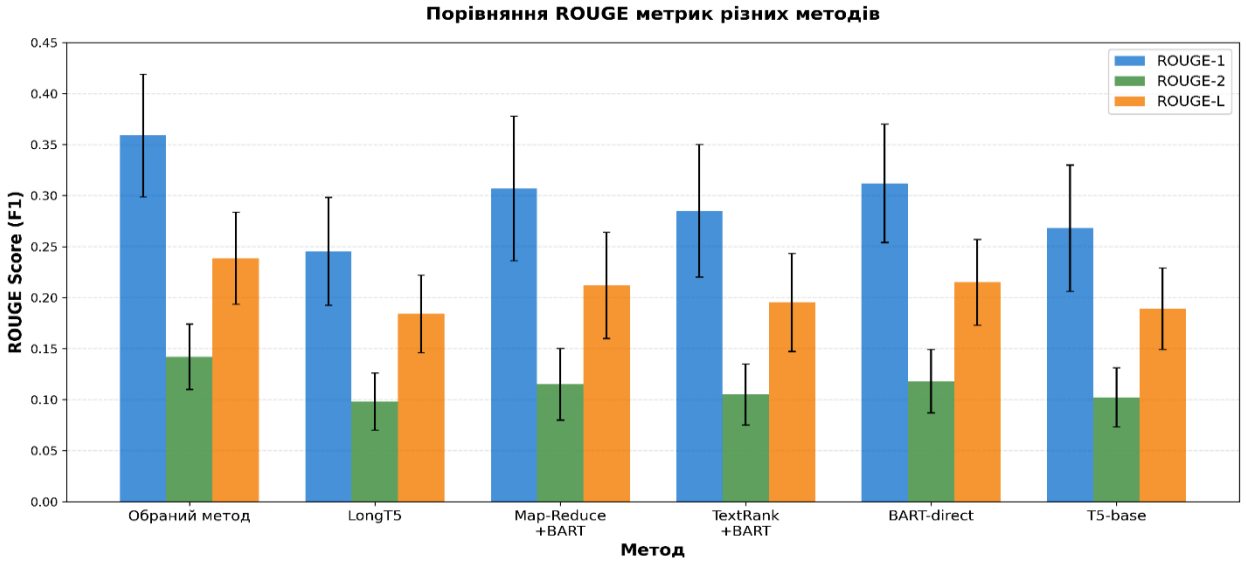

Рисунок 3 – Порівняння результатів ROUGE-метрик для різних методів

Як видно з діаграми на рисунку 3, за метрикою ROUGE-1 F1 наявна перевага запропонованого методу – він показав найвищий результат (0.5852±0.027). Це на 4.4% краще ніж спеціалізована модель-трансформер LongT5(0.5603±0.017) і на 16.1% краще за базовий Map-Reduce+BART.

За метрикою ROUGE-2, що вимірює збіг біграм, запропонований метод показав кращі результати також: 0.1420±0.032 проти 0.0980±0.028 у LongT5 (+44.9%). Це вказує на те, що згенеровані резюме не тільки містять правильні слова, але й зберігають типові словосполучення з оригінальних документів. За метрикою ROUGE-L F1 (найдовша спільна підпослідовність) результат запропонованого методу становить 0.2385±0.045 проти 0.1840±0.038 у LongT5, що свідчить про краще збереження структури та порядку інформації з оригінального тексту.

Результати порівняння за метрикою BERTscore представлені на рисунку 4.

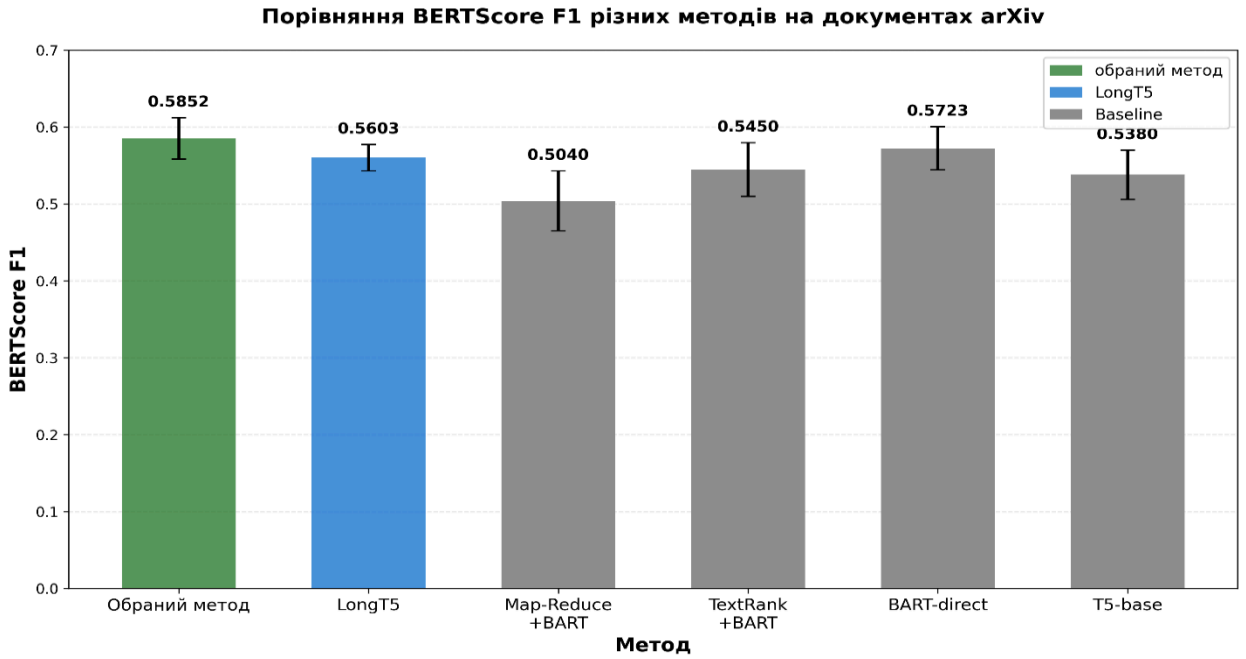

Рисунок 4 – Порівняння результатів BERTscore-метрик для різних методів

Як видно з діаграми, запропонований метод показав найкращий результат з усіх методів – 0.5852. Це на 4.4% краще за спеціалізовану модель-трансформер LongT5 і на 16% краще за схожий Map-Reduce метод.

Висновки

Розроблено гібридний метод резюмування довгих наукових документів, що поєднує розбиття документу на перекриваючі фрагменти, семантичну екстракцію з використанням Sentence-BERT, видалення семантичних дублікатів через косинусну подібність векторних представлень та абстрактивну генерацію за допомогою BART-large-CNN. Метод дозволяє обробляти документи довільної довжини без використання спеціалізованих моделей з розширеним контекстним вікном.

Здійснено програмну реалізацію методу з дотриманням SOLID принципів, що забезпечує модульність та можливість заміни окремих компонентів системи. З використанням розробленого програмного забезпечення проведено порівняльне дослідження методу на вибірці наукових статей arXiv. За метрикою ROUGE-1 F1 запропонований метод показав результат 0.5852±0.027, що на 4.4% краще за спеціалізовану модель LongT5 та на 16.1% краще за базовий Map-Reduce+BART. Результати підтверджують ефективність гібридного підходу для резюмування документів з використанням стандартних моделей-трансформерів без додаткового дотренування.

Отримані результати підтверджують доцільність подальших досліджень з поєднання семантичної екстракції на основі векторних представлень з абстрактивною генерацією для задачі резюмування наукових документів.

Список використаної літератури

1. See A., Liu P. J., Manning C. D. Get To The Point: Summarization with Pointer-Generator Networks. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. 2017. Vol. 1. P. 1073–1083. https://doi.org/10.18653/v1/P17-1099

2. Huang L., Wu L., Wang L. An Empirical Survey on Long Document Summarization: Datasets, Models, and Metrics. ACM Computing Surveys. 2022. Vol. 55, № 8. Article 157. https://doi.org/10.1145/3545176

3. Lewis M., Liu Y., Goyal N., Ghazvininejad M., Mohamed A., Levy O., Stoyanov V., Zettlemoyer L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020. P. 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

4. Zhang J., Zhao Y., Saleh M., Liu P. J. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. Proceedings of the 37th International Conference on Machine Learning. 2020. P. 11328–11339. https://doi.org/10.48550/arXiv.1912.08777

5. Liu Y., Lapata M. Hierarchical Transformers for Multi-Document Summarization. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019. P. 5070–5081. https://doi.org/10.18653/v1/P19-1500

6. Beltagy I., Peters M. E., Cohan A. Longformer: The Long-Document Transformer. arXiv preprint arXiv:2004.05150. 2020. https://doi.org/10.48550/arXiv.2004.05150

7. Zhang X., Wei F., Zhou M. HIBERT: Document Level Pre-training of Hierarchical Bidirectional Transformers for Document Summarization. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019. P. 5059–5069. https://doi.org/10.18653/v1/P19-1499

8. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. P. 3982–3992. https://doi.org/10.48550/arXiv.1908.10084

9. Automatic text summarization of scientific articles using transformers: A brief review. Journal of Artificial Intelligence. 2024. Vol. 7, № 5. https://doi.org/10.32629/jai.v7i5.1331

References

1. See, A., Liu, P. J., & Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In Proceedings of the 55th Annual Meeting of the

Association for Computational Linguistics (Vol. 1, pp. 1073–1083). Association for Computational Linguistics. https://doi.org/10.18653/v1/P17-1099

2. Huang, L., Wu, L., & Wang, L. (2022). An empirical survey on long document summarization: Datasets, models, and metrics. ACM Computing Surveys, 55(8), Article 157. https://doi.org/10.1145/3545176

3. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (pp. 7871–7880). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.703

4. Zhang, J., Zhao, Y., Saleh, M., & Liu, P. J. (2020). PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In Proceedings of the 37th International Conference on Machine Learning (pp. 11328–11339). PMLR. https://doi.org/10.48550/arXiv.1912.08777

5. Liu, Y., & Lapata, M. (2019). Hierarchical transformers for multi-document summarization. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 5070–5081). Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1500

6. Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150. https://doi.org/10.48550/arXiv.2004.05150

7. Zhang, X., Wei, F., & Zhou, M. (2019). HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 5059–5069). Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1499

8. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods

in Natural Language Processing (pp. 3982–3992). Association for Computational Linguistics. https://doi.org/10.48550/arXiv.1908.10084

9. Automatic text summarization of scientific articles using transformers: A brief review. (2024). Journal of Artificial Intelligence, 7(5). https://doi.org/10.32629/jai.v7i5.1331