Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмного модуля завадостійкого кодування даних у системах передачі інформації

Виконав: студент 4 курсу
групи 2ПП-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Атаманенко В. М.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Рейда О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Кожем`яко А. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Атаманенко Владиславу Миколайовичу

1. Тема роботи – розробка програмного модуля завадостійкого кодування даних у системах передачі інформації.

Керівник роботи: Рейда Олександр Миколайович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки Python, операційна система Windows 11, полярне кодування.

4. Зміст розрахунково-пояснювальної записки: вступ, особливості кодування для систем передачі інформації, аналіз методів завадостійкого кодування даних, порівняльний аналіз аналогів, розробка методу кодування повідомлення, розробка методу проєктування полярних кодів, розробка методу декодування повідомлення, розробка методу симуляції каналу завадами, варіантний аналіз і обґрунтування вибору засобів для реалізації програмного модуля, розробка моделі та алгоритму застосунку, розробка модуля полярного кодування, тестування програмного забезпечення, розробка інструкції користувача, висновки, список використаних джерел, додатки.

5. Перелік ілюстративної частини: титульний слайд; актуальність; мета, завдання, предмет, і об'єкт дослідження; задачі; наукова новизна; практичне значення отриманих результатів; аналоги; порівняльний аналіз аналогів;методи та засоби розробки; діаграма класів; алгоритми кодування; приклади симуляції передачі; порівняння швикодії декодування з numba та без; висновки; публікації й апробації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Рейда О. М., к. т. н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз методів завадостійкого кодування даних	25.03.25 – 05.04.25	вик
2	Розробка методів для завадостійкого кодування	05.04.25 – 10.04.25	вик
3	Розробка моделі та алгоритму застосунку	10.04.25 – 28.04.25	вик
4	Розробка програмного модуля завадостійкого кодування	28.04.25 – 16.05.25	вик
5	Тестування застосунку	16.05.25 – 25.05.25	вик
6	Оформлення матеріалів до захисту БКР	25.05.25 – 30.05.25	вик

Студент _____ **В. М. Атаманенко**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **О. М. Рейда**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 61 сторінки формату А4, на яких є 29 рисунків, 1 таблиця, список використаних джерел містить 27 найменувань.

У бакалаврській кваліфікаційній роботі розроблено програмний модуль завадостійкого кодування даних у системах передачі інформації з використанням полярних кодів. Обґрунтовано доцільність їх застосування в умовах зашумлених каналів зв’язку. Розробку здійснено мовою Python із використанням бібліотек NumPy та Numba, що забезпечило баланс між продуктивністю й гнучкістю. Реалізовано алгоритми побудови полярного коду на основі меж Бхаттачар'ї, а також функції кодування, декодування та моделювання передачі через канал із адитивним білим гаусовим шумом. Реалізовано декодування методом послідовного виключення з використанням логарифмічних відношень правдоподібності. Розроблено графічний інтерфейс, що дозволяє користувачеві здійснювати налаштування та візуалізацію результатів. Тестування підтвердило ефективність реалізованого підходу.

Ключові слова: полярні коди, завадостійке кодування, системи передачі інформації, декодування, Python.

ABSTRACT

The bachelor's thesis consists of 61 A4 pages, including 29 figures, 1 table, and a list of 27 references.

The bachelor’s thesis presents the development of a software module for error-resistant data encoding in information transmission systems using polar codes. The feasibility of applying such codes under noisy channel conditions is substantiated. The development was carried out in Python using the NumPy and Numba libraries, providing a balance between performance and development flexibility. Algorithms for polar code construction based on Bhattacharyya bounds were developed, along with functions for encoding, decoding, and simulating transmission over an additive white Gaussian noise channel. Successive cancellation decoding with log-likelihood ratio computations was implemented. A graphical user interface was created to enable users to configure parameters and visualize results. Testing confirmed the effectiveness of the implemented approach.

Keywords: polar codes, error-resistant coding, information transmission systems, decoding, Python.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Останнім часом зростає потреба в ефективних і надійних системах цифрової передачі та зберігання даних, що пов’язано з бурхливим розвитком сучасних інформаційно-комунікаційних технологій. Цей попит значно зріс у зв’язку з розгортанням масштабних та високошвидкісних мереж нового покоління, які забезпечують обмін, обробку, передавання та довготривале зберігання великих обсягів цифрової інформації.

У 1948 році Клод Шеннон [1] у своїй знаковій праці довів, що за допомогою належного кодування можливо зменшити вплив шуму в каналі зв’язку або у середовищі зберігання до будь-якого бажаного рівня, не знижуючи при цьому швидкість передачі або зберігання інформації. З часу його відкриття було докладено значних зусиль до створення ефективних методів кодування та декодування, здатних забезпечити контроль помилок у шумному середовищі. Завдяки новітнім досягненням стало можливим досягати рівня надійності, який вимагають сучасні високошвидкісні цифрові комунікації та системи зберігання великої ємності, а застосування завадостійокого кодування стало стандартною практикою в їхньому проєктуванні.

Процеси передачі та зберігання цифрової інформації мають багато спільних рис – у кожному випадку дані передаються від джерела до приймача через певний канал. Джерелом інформації може бути як людина, так і пристрій. Вихідна інформація, призначена для передавання, може бути як неперервним сигналом, так і послідовністю дискретних символів.

Процес цифрової передачі або зберігання зазвичай складається з кількох етапів. Спочатку вихідне повідомлення, обробляється за допомогою кодера джерела, що перетворює його у зручну для передачі чи зберігання форму. Якщо сигнал є аналоговим – наприклад, мова або зображення – його додатково потрібно дискретизувати й квантизувати, щоб отримати цифровий вигляд у формі бітової послідовності.

Після цього застосовується завадостійке кодування, яке додає надлишкові біти, що дозволяють виявляти або виправляти помилки, які можуть виникнути під час передачі чи зберігання. Закодовану послідовність надсилають або записують у середовище, яке може піддаватися шумам або іншим спотворенням.

На боці отримувача система спочатку виконує завадостійке декодування із метою виявлення та виправлення помилок, а далі виконується декодування джерела для відновлення первинної інформації. Завдяки цим крокам забезпечується точне відтворення даних, навіть якщо в каналі були завади.

Ефективне програмне впровадження завадостійкого кодування залишається складною задачею. Основним викликом є зниження часу обробки, оскільки алгоритми декодування часто використовують рекурсію чи велику кількість ітерацій. Таким чином розробка програмних засобів завадостійкого кодування є актуальною задачею.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності використання обчислювальних ресурсів та зменшення часу обробки інформації за рахунок розробки програмного модуля завадостійкого кодування даних

Відповідно до поставленої мети в бакалаврській кваліфікаційній роботі потрібно вирішити такі **задачі**:

- провести аналіз існуючих методів завадостійкого кодування та їх застосування в системах передачі інформації;

- дослідити особливості побудови кодів з контролем і виправленням помилок та їх математичні основи;

- розробити алгоритми кодування та декодування для вибраного методу завадостійкого кодування;

- модифікувати метод завадостійкого кодування для підвищення швидкодії обробки даних;

- реалізувати програмну модель для перевірки ефективності алгоритмів у середовищі з завадами;

- провести тестування стійкості кодування до завад.

Об'єкт дослідження – процес розробки програмного модуля завадостійкого кодування даних у системах передачі інформації.

Предмет дослідження – методи та засоби розробки програмного модуля завадостійкого кодування даних у системах передачі інформації.

Методи дослідження. У ході дослідження застосовувалися методи та підходи, що ґрунтуються на теорії інформації та теорії кодування, а також: теорія чисел і чисельні методи для побудови та аналізу структур кодів; теорія алгоритмів для розробки ефективних процедур кодування та декодування; комп'ютерне моделювання – з метою аналізу стійкості кодів до завад, перевірки коректності реалізованих алгоритмів і підтвердження ефективності отриманих теоретичних результатів.

Новизна отриманих результатів:

Подальшого розвитку отримав метод завадостійкого кодування, у якому, на відміну від існуючих підходів, застосовуються засоби JIT-компіляції бібліотеки Numba, що дало можливість підвищити швидкодію декодування за рахунок оптимізації скомпільованих фрагментів Python-коду.

Практична цінність отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні модулі кодування завадостійкого кодування даних та симуляції передачі інформації через зашумлений канал.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У праці, опублікованих у співавторстві, студенту належать: [2] – проаналізовано сучасні методи завадостійкого кодування у системах передачі інформації.

Апробація результатів роботи. Основні положення бакалаврської кваліфікаційної роботи доповідалися на LIV Всеукраїнській науково-технічній

конференції факультету інформаційних технологій та комп'ютерної інженерії у 2025 році.

Публікації. За тематикою дослідження опубліковано одну наукову працю у збірниках матеріалів конференцій.

Структура та обсяг БКР. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на плагіат, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ЗАВАДОСТІЙКОГО КОДУВАННЯ ДАНИХ

1.1 Особливості кодування для систем передачі інформації

Передача інформації в реальних каналах зв'язку майже завжди супроводжується дією завад, що призводить до спотворення переданих даних. Для моделювання процесів передавання у системах цифрового зв'язку широко використовується модель каналу з додаванням білого гаусівського шуму [3]. У такій моделі прийнятий сигнал розглядається як сума переданого сигналу та шумової складової, що має нормальний розподіл із нульовим математичним сподіванням і дисперсією σ^2. Це дозволяє адекватно імітувати основні характеристики реальних каналів, у яких шум є випадковим та має незалежну природу [4].

Дія шуму на повідомлення може бути представлена формулою (1.1):

$$y = s + \xi, \tag{1.1}$$

де s – повідомлення, яке передається;

ξ – завада;

x – прийняте повідомлення

Оцінювання рівня завад у таких каналах зазвичай здійснюється за допомогою співвідношення енергії біта до спектральної щільності шуму E_b/N_0, що є стандартною мірою якості каналу в цифровому зв'язку [5].

Наявність шуму в каналі призводить до спотворення переданої інформації, що унеможливлює гарантування її достовірного приймання без спеціальних заходів. Згідно з основними положеннями теорії інформації, сформульованими К. Шенноном, існує межа пропускної здатності каналу, при якій можна забезпечити передачу інформації з довільно малим рівнем помилок за умови

застосування відповідного кодування [1]. Таким чином, завадостійке кодування є необхідним елементом системи цифрового зв'язку для забезпечення її надійності.

Методи завадостійкого кодування дозволяють збільшити ймовірність правильного приймання інформації навіть за наявності суттєвого шумового впливу. Основна ідея полягає у додаванні надлишкової інформації до переданого повідомлення, що дозволяє виявляти та виправляти помилки на приймальній стороні.

При моделюванні передачі інформації через зашумлені канали зазвичай враховується низка основних параметрів вхідних даних, серед яких:

- розмір блоку коду;
- розмір інформаційного блоку;
- рівень завад, заданий через параметр E_b/N_0.

Використання завадостійкого кодування дозволяє суттєво знизити ймовірність бітових помилок навіть у каналах із сильним шумовим впливом. Це має критичне значення для багатьох застосувань, таких як бездротові комунікаційні системи, супутниковий зв'язок, системи зберігання даних і передавання медіаконтенту, де висока надійність передавання є обов'язковою умовою.

1.2 Аналіз методів завадостійкого кодування даних

Сучасні комунікаційні технології вимагають високої надійності передачі інформації, що робить завадостійке кодування ключовим елементом цифрових систем зв’язку та зберігання даних.

Одним із класичних рішень є використання БЧХ-кодів, які забезпечують потужну корекцію декількох помилок на коротких кодових відстанях. Такі коди застосовуються переважно для невеликих об'ємів даних, наприклад у носіях інформації. Проте вони мають обмежену ефективність при великих об'ємах інформації та характеризуються складною реалізацією для довгих кодових слів.

Ще одним поширеним методом є використання кодів Ріда-Соломона. Ці коди особливо добре справляються з груповими помилками і широко використовуються у телекомунікаціях та системах зберігання даних. Однак робота у символічних полях збільшує обчислювальну складність, що робить ці коди менш привабливими для бітових каналів.

Коди Хеммінга також мають своє місце серед класичних методів захисту даних. Вони дозволяють виправляти одиничні помилки та виявляти подвійні. Перевагою кодів Хеммінга є їхня простота, але рівень захисту є недостатнім для сучасних систем передачі великих обсягів інформації через сильно зашумлені канали [6].

Хоча класичні методи корекції помилок, такі як коди Геммінга, коди БЧХ та коди Ріда-Соломона, продовжують використовуватися в багатьох сферах, вони мають обмеження щодо продуктивності та ефективності декодування при високих швидкостях передачі.

Розвиток обчислювальних потужностей дозволив створити більш ефективні методи, здатні працювати близько до межі Шеннона – теоретичної межі максимальної ефективності кодування. До таких методів належать LDPC-коди, турбокоди та полярні коди, які стали основою сучасних стандартів зв’язку, таких як 4G, 5G, супутникові та оптичні системи.

LDPC-коди були запропоновані Робертом Галлагером у 1962 році, але через обмеженість обчислювальних ресурсів вони залишалися практично невідомими до середини 1990-х років. Їхньою основною перевагою є використання розріджених матриць перевірок на парність, що дозволяє застосовувати ефективні ітеративні алгоритми декодування, такі як метод розповсюдження довіри. Це забезпечує високу ефективність роботи навіть для дуже довгих кодових слів, що робить LDPC-коди ідеальними для мобільного зв’язку, оптичних комунікацій та систем зберігання даних [7].

Турбокоди, запропоновані у 1993 році, стали першим класом кодів, які на практиці змогли працювати менш ніж на 1 дБ від межі Шеннона. Вони ґрунтуються на використанні двох або більше згорткових кодів, об'єднаних за

допомогою інтерлівера, що дозволяє значно покращити їхню ефективність. Турбокоди використовують ітеративне декодування, під час якого декодери обмінюються інформацією, уточнюючи оцінку переданого повідомлення. Цей підхід дозволяє отримувати низькі рівні помилок навіть при високих швидкостях передачі, що робить турбокоди популярними в мобільному зв'язку, супутникових системах та оптичних мережах [8].

Полярні коди, розроблені Ердалом Аріканом у 2008 році, стали першими кодами, для яких математично доведено, що вони можуть досягати межі Шеннона. Вони використовують процес «поляризації каналів», що дозволяє ефективно розподіляти інформацію між надійними та ненадійними символами. Однією з ключових переваг полярних кодів є низька обчислювальна складність декодування, яка становить O(N logN), що робить їх ефективними для застосування в мережах 5G [9].

З огляду на вищенаведений аналіз сучасних методів завадостійкого кодування, для реалізації програмного модуля було обрано полярні коди. Це рішення обґрунтоване тим, що саме полярні коди першими математично доведено досягають межі Шеннона, забезпечуючи високу ефективність при відносно низькій обчислювальній складності. Завдяки механізму поляризації каналів, вони дозволяють ефективно розділяти інформацію на надійні та ненадійні біти, що покращує якість відновлення повідомлень. До того ж, їхня сучасна апаратна та програмна підтримка у стандартах зв'язку 5G робить їх перспективними для широкого спектра застосувань.

1.3 Порівняльний аналіз аналогів

У процесі розробки програмного модуля для завадостійкого кодування важливим етапом є аналіз існуючих інструментів, які реалізують аналогічну функціональність. Серед найбільш відомих рішень варто виділити AFF3CT, GNU Radio та 5G Toolkit, кожен з яких має свої переваги та обмеження в контексті побудови, симуляції та аналізу систем каналового кодування.

AFF3CT [10] – це програмний інструментарій, призначений для роботи з методами прямої корекції помилок. Він реалізований мовою C++ та підтримує широкий спектр кодів: від широко застосовуваних турбокодів до новітніх полярних кодів, включаючи коди з низькою щільністю перевірок на парність.

Бібліотека складається з двох основних компонентів. Перший – це автономний симулятор, який дозволяє моделювати комунікаційні ланцюги за методом Монте-Карло. Другий – модульна бібліотека, доступ до якої здійснюється через добре задокументований API. Симулятор оптимізований для високошвидкісного моделювання, що досягається завдяки використанню сучасних паралельних обчислювальних технологій, зокрема SIMD-інструкцій, багатопотокового виконання та багатовузлових моделей програмування.

До основних мотивів створення симулятора належать потреба відтворити результати декодування, що відповідають найкращим сучасним методикам, дослідити різні конфігурації каналів кодування та знайти нові компроміси між продуктивністю й складністю. Також симулятор дозволяє швидко створювати прототипи апаратних реалізацій, зокрема з використанням фіксованої коми та інтеграцією у цикл розробки "hardware-in-the-loop", повторно використовувати надійні модулі та розширювати їх власними рішеннями. У випадках, коли використання MATLAB є надто повільним, AFF3CT може служити ефективною альтернативою.

Спочатку AFF3CT створювався винятково як симулятор, однак з часом зросла потреба в повторному використанні окремих модулів — так виникла бібліотека. Її застосування дозволяє реалізовувати користувацькі комунікаційні ланцюги, які не підтримуються симулятором, спрощувати створення прототипів апаратного забезпечення, а також інтегрувати окремі модулі в контекстах програмно-визначуваного радіо (SDR).

GNU Radio [11] – це вільне програмне середовище з відкритим вихідним кодом, призначене для створення програмно-визначених радіосистем (SDR). Воно забезпечує гнучку платформу для моделювання, тестування та реалізації цифрових систем обробки сигналів у реальному часі. GNU Radio активно

використовується в наукових дослідженнях, розробці прототипів телекомунікаційних систем, а також в освітніх проєктах.

Однією з ключових особливостей GNU Radio є модульність: користувачі можуть будувати обробку сигналу як ланцюг з окремих блоків, кожен з яких виконує певну функцію – наприклад, модуляцію, фільтрацію, перетворення Фур'є, декодування або корекцію помилок. Ці блоки можуть бути створені за допомогою C++ або Python, що забезпечує поєднання високої продуктивності та простоти розробки.

GNU Radio підтримує інтеграцію з широким спектром апаратного забезпечення, зокрема з пристроями USRP (Universal Software Radio Peripheral) від компанії Ettus Research, RTL-SDR, HackRF, LimeSDR та іншими. Завдяки цьому середовище дозволяє реалізовувати реальні системи безпосередньо на фізичному рівні, включаючи роботу з реальними радіочастотними сигналами.

У контексті завадостійкого кодування GNU Radio дозволяє експериментувати з різними методами кодування та декодування, реалізовувати власні схеми, а також інтегрувати сторонні бібліотеки, наприклад AFF3CT, для підвищення продуктивності. Таким чином, GNU Radio є потужним інструментом для дослідження, тестування та впровадження алгоритмів обробки сигналів, включаючи методи корекції помилок, у складних цифрових комунікаційних системах.

5G Toolkit [12] є потужним інструментом для моделювання та симуляції систем мобільного зв’язку п’ятого покоління відповідно до стандарту 3GPP 5G NR. Платформа реалізована мовою Python та орієнтована на наукові дослідження, прототипування та освітні потреби. Основна увага в архітектурі інструмента приділяється гнучкій реалізації всіх етапів фізичного рівня, включаючи передавач, канал та приймач. Особливу роль у структурі займає підтримка сучасних методів завадостійкого кодування, зокрема полярних кодів та LDPC-кодів.

У рамках модуля кодування реалізовано механізми, що відповідають вимогам стандарту 3GPP: LDPC-коди використовуються для кодових блоків

фізичного каналу передачі даних, а полярні коди – для каналів керування. У тулкіті передбачена підтримка повного циклу кодування та декодування, включаючи мапінг інформаційних бітів, CRC-контроль, вибір заморожених бітів та ітеративне декодування з використанням логарифмічних ймовірностей.

5G Toolkit відзначається гнучкістю конфігурації симуляційного середовища: користувач має змогу задавати параметри шуму, модульованого сигналу, пропускної здатності та топології передачі, що дозволяє аналізувати поведінку кодувальних алгоритмів у різних умовах. Симуляційне середовище може використовувати як адитивний білий гаусовий шум, так і більш складні моделі каналу, включно з багатопроменевими.

Результати аналізу наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця аналогів

Критерії оцінювання	AFF3CT	GNU Radio	5G Toolkit	Власна розробка
Безкоштовна ліцензія	1	1	0	1
Реалізація систематичного кодера	1	1	0	1
Відображення зашумленого зображення під час симуляції	0	0	0	1
Графічний інтерфейс симулятора	1	1	0	1
Застосування засобів оптимізації обчислень	1	0	1	1
Загальна оцінка	4	3	1	5

Виходячи з результатів аналізу аналогів, можна зробити висновок, що розробка програмного модуля для завадостійкого кодування, орієнтованої на систематичні полярні коди та реалізованої з використанням Python, є актуальною і технічно доцільною. У порівнянні з існуючими інструментами, такими як AFF3CT, GNU Radio та 5G Toolkit, запропонований модуль демонструє конкурентні переваги, зокрема завдяки візуалізації результатів, підтримці

моделювання каналу з шумом та ефективному використанню засобів прискорення обчислень. Такий підхід дозволяє не лише проводити дослідження в галузі кодування, але й застосовувати бібліотеку для навчання, прототипування та тестування рішень у сфері цифрового зв’язку.

1.4 Висновки

У першому розділі було проаналізовано сучасний стан галузі завадостійкого кодування та програмних засобів, що забезпечують моделювання і реалізацію відповідних алгоритмів. Проведений порівняльний аналіз існуючих рішень, таких як AFF3CT, GNU Radio та 5G Toolkit, засвідчив актуальність створення спеціалізованого інструменту для роботи з полярними кодами, що дозволяє моделювати передачу зашумлених зображень та візуалізувати результати декодування.

2 РОЗРОБКА МЕТОДІВ ДЛЯ ЗАВАДОСТІЙКОГО КОДУВАННЯ З ВИКОРИСТАННЯМ ПОЛЯРНИХ КОДІВ

2.1 Розробка методу кодування повідомлення

У якості твірної матриці полярного коду використовується спеціальний клас матриць, що утворюються шляхом багаторазового кронекерового добутку двійкової базової матриці розміру 2×2 і мають назву кронекерові матриці [13]. Вони відіграють ключову роль у структурі кодувальних перетворень, і тому їхні властивості мають фундаментальне значення для аналізу та реалізації полярних кодів. Базова матриця розміру 2×2 (2.1), має такий вигляд:

$$G_{2\times 2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.1}$$

Матриця G_2 відома як ядро у побудові полярних кодів.

Для спрощення надалі n-кратна кронекерова матриця позначається як $G(n)$ (2.2) та має розмірність $2^n \times 2^n$:

$$G(n) \triangleq G_2{}^{n}{}_{\times 2}{}^{n}. \tag{2.2}$$

$G(n)$ утворюється шляхом n-кратного кронекерового добутку базової матриці $G_{2\times 2}$ на себе саму. Отримана матриця використовується як матриця кодування для полярних кодів довжини $N = 2^n$, також часто позначається як $F^{\otimes N}$.

2-кратний кронекерів добуток ядра (2.1), що позначається як $G(2)$, є нижньотрикутним масивом розміру 2×2, який складається з трьох копій ядра $G(1)$ має вигляд (2.3):

$$G(2) = G(1) \otimes G(1) = \begin{bmatrix} G(1) & O \\ G(1) & G(1) \end{bmatrix}. \tag{2.3}$$

Заміна кожної копії ядра у матриці (2.3) на матрицю розміром 2×2, наведену у виразі (2.1), в результаті дає таку матрицю (2.4) розміру 4×4 над полем *GF*(2):

$$G(2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \quad (2.4)$$

Ця матриця є нижньотрикутною, в якій міститься 3^2=9 одиничних елементів, розташованих на головній діагоналі та під нею.

Трикратна кронекерова матриця, утворена на основі ядра *G*(1), визначається як кронекерів добуток ядра G(1) та 2-кратної кронекерової матриці *G*(2), і має вигляд (2.5):

$$G(3) = G(1) \otimes G(2) = \begin{bmatrix} G(2) & O \\ G(2) & G(2) \end{bmatrix} = \begin{bmatrix} G(1) & O & O & O \\ G(1) & G(1) & O & O \\ G(1) & O & G(1) & O \\ G(1) & G(1) & G(1) & G(1) \end{bmatrix}. \quad (2.5)$$

Ця трикратна Кронекерова матриця *G*(3) є трикутним масивом розміром 4×4, що містить 3^2=9 копій ядра *G*(1).

Заміна кожної копії ядра *G*(1) у матриці (2.5) на матрицю розміром 2×2, наведену у виразі (2.1) в результаті дає матрицю (2.6):

$$G(3) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (2.6)$$

Отримана матриця є нижньотрикутною матрицею розміру 8×8, яка містить 3^3=27 одиничних елементів, розташованих на головній діагоналі та нижче.

Чотирикратна кронекерова матриця, побудована на основі ядра G(1), визначається як кронекерів добуток ядра G(1) і трикратної кронекерової матриці (2.7):

$$G(4) = G(1) \otimes G(3) = \begin{bmatrix} G(3) & O \\ G(3) & G(3) \end{bmatrix}$$

$$= \begin{bmatrix} G(1) & O & O & O & O & O & O & O \\ G(1) & G(1) & O & O & O & O & O & O \\ G(1) & O & G(1) & O & O & O & O & O \\ G(1) & G(1) & G(1) & G(1) & O & O & O & O \\ G(1) & O & O & O & G(1) & O & O & O \\ G(1) & G(1) & O & O & G(1) & G(1) & O & O \\ G(1) & O & G(1) & O & G(1) & O & G(1) & O \\ G(1) & G(1) & G(1) & G(1) & G(1) & G(1) & G(1) & G(1) \end{bmatrix}. \quad (2.7)$$

Це – масив розміром 8×8, що складається з 3^3=27 копій ядра $G(1)$. Заміна кожної копії ядра $G(1)$ у матриці (2.7) на матрицю розміром 2×2, наведену у виразі (2.1) в результаті дає матрицю (2.8):

$$G(4) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (2.8)$$

Наведена послідовність побудови дво-, три- та чотириразових кронекерових матриць демонструє, що матриці цього типу, побудовані на основі ядра G(1), можуть формуватися рекурсивно. Для довільного цілого n ≥ 1, n-кратна кронекерова матриця G(n) визначається як Кронекерів добуток ядра G(1) та попередньої (n−1)-разової кронекерової матриці G(n−1). Загалом рекурсивне формування матриці кронекера можна виразити формулою (2.9):

$$G(n) = G(1) \otimes G(n\text{-}1) = \begin{bmatrix} G(\mathrm{n}-1) & O \\ G(n-1) & G(n-1) \end{bmatrix} \tag{2.9}$$

На основі наведеної конструкції випливає, що $G(n)$ є нижньотрикутним масивом розміру $2^{n-1}\times 2^{n-1}$, у якому розміщено 3^{n-1} копій ядра $G(1)$, розташованих на головній діагоналі та нижче. Повна матриця містить рівно 3^n одиничних елементів, що також лежать на головній діагоналі або під нею.

Маючи твірну матрицю можна кодувати повідомлення. Розглянемо алгоритм систематичного кодера полярних кодів [14], нехай інформаційна множина має вигляд $\mathcal{A} = \{a_0, a_1, ..., a_{k-1}\}$, впорядкована так, що $a_0 < a_1 < ... < a_{k-1}$. Систематичний кодер не лише містить вхідні біти u, а й розташовує їх відповідно до порядку в $\mathcal{A}$. На рисунку 2.1 зображено кодер для (8, 5)-коду з $\mathcal{A} = \{3, 4, 5, 6, 7\}$.

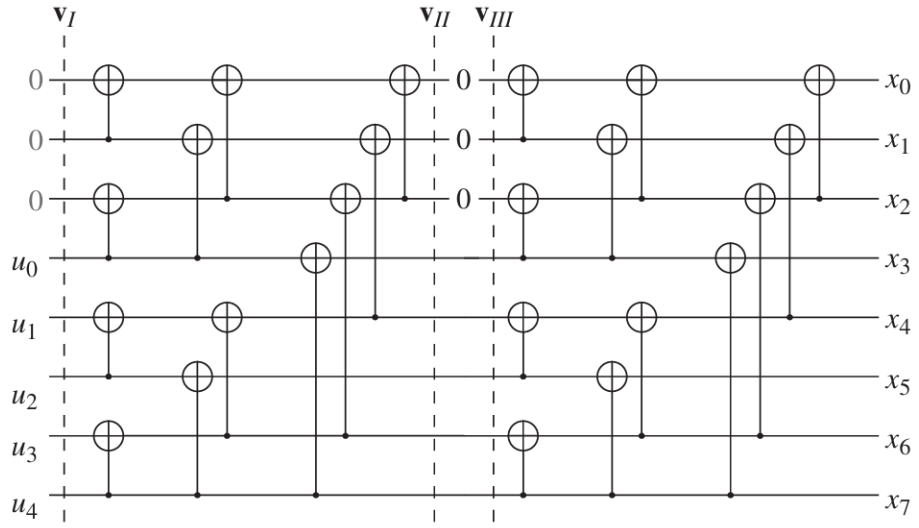


Рисунок 2.1 – Систематичний кодер для (8, 5)-коду

Відповідно до схеми на рисунку 2.1, процес кодування такий:

1. Вхідний вектор u = (u_0, u_1, ...,u_{K-1}) перетворюється у вектор v_I = ($v_{I,0}$, $v_{I,1}$, ..., $v_{I,N-1}$) довжини N, де $v_{I,ai} = u_i$ для i = 0, ..., K−1. Усі інші N-K елементів v_I встановлюються в 0.
2. Обчислюється $v_{II} = v_I \cdot F^{\otimes N}$.
3. Вектор v_{III} утворюється встановленням усіх елементів, не з $\mathcal{A}$, в 0.
4. Обчислюється вихідний вектор x = $v_{III} \cdot F^{\otimes N}$.

Позначимо цю операцію кодування як $\mathscr{E}$(u). Нехай G – це $K \times N$ матриця генератора рангу K. Кодування є лінійною операцією і може бути записане як (2.10):

$$\mathscr{E}(u) = u \cdot \Pi \cdot G, \tag{2.10}$$

де П — оборотна $K \times K$ матриця.

Кодування є систематичним, якщо вектори на позиціях з індексів $\mathcal{A}$ рівні u. Припускається, що $\mathcal{A}$ впорядкована ($a_0 < a_1 < ... < a_{k-1}$), тож u_i з'являється раніше, ніж u_j, якщо $i < j$.

Один зі способів реалізувати систематичне кодування – вибрати k лінійно незалежних стовпців з G, і взяти П як обернену до відповідної $K \times K$ підматриці.

Нехай E – $K \times N$ матриця, визначена як 2.11:

$$E_{i,j} = \begin{cases} 1, & j = a_i \\ 0, & j \neq a_i \end{cases}. \tag{2.11}$$

Для (8,5)-полярного коду з кодером із рис. 2.1 маємо (2.12):

$$E = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

Полярне кодування можна виразити через твірну матрицю (2.13):

$$G = E \cdot F^{\otimes N}, \tag{2.13}$$

Тобто, матриця E вибирає рядки матриці G, що відповідають інформаційним бітам. Застосування $u \cdot E$ дає вектор v_I довжини n, де $v_{I,aj} = u_j$.

Другий крок (2.14):

$$v_{II} = v_I \cdot F^{\otimes N} = u \cdot E \cdot F^{\otimes N}. \tag{2.14}$$

Третій крок (2.15):

$$v_{III} = v_{II} \cdot E^T \cdot E \tag{2.15}$$

утворює вектор, у якому елементи з індексами з множини $\mathcal{A}$ залишаються незмінними, а всі інші елементи встановлюються в 0.

Четвертий крок (2.16):

$$x = v_{III} \cdot F^{\otimes n} = u \cdot E \cdot F^{\otimes n} \cdot E^T \cdot E \, F^{\otimes n} \tag{2.16}$$

де $E \cdot F^{\otimes n} \cdot E = \Pi$, а $E \cdot F^{\otimes n} = G$.

Отже, x = $\mathscr{E}$(u) дає систематично закодоване повідомлення.

2.2 Розробка методу проєктування полярних кодів

Розглянемо базову ідею поляризації [15]. Нехай W – канал, що приймає бінарні входи. На рисунку 2.2, канал W використовується двічі: спочатку з вхідним символом u_0, а потім – з u_1. Відповідними виходами каналу є y_0 та y_1. Взаємна інформація між u_0 і y_0 позначається як I^0, а між u_1 і y_1 – як I^1. Оскільки використовується один і той самий канал, маємо: $I^0 = I^1 = I(W)$ – взаємна

інформація каналу *W*. Таким чином, обидва використання каналу є однаково надійними.

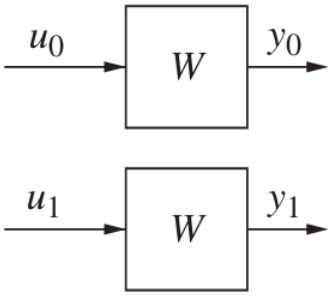


Рисунок 2.2 – Використання одного каналу двічі

На рисунку 2.3 канал W знову використовується двічі, але з попередньою обробкою вхідних бітів. У першому випадку в канал передається $u_0 \oplus u_1$, а в другому – u_1. Нехай I_c^0 позначає взаємну інформацію між u_0 та вектором виходів (y_0, y_1), а I_c^1 – взаємну інформацію між u_1 та вектором (y_0, y_1, u_0). Біт u_1 впливає на обидва виходи – y_0 і y_1, отже, про u_1 можна отримати трохи більше інформації, ніж про u_0. І навпаки, оскільки u_0 передається у зашифрованому вигляді через операцію XOR з u_1, інформації про нього стає трохи менше, ніж у звичайному випадку. Таким чином, можна встановити нерівність (2.17):

$$I_c^0 < I(W) < I_c^1. \qquad (2.17)$$

Отже, завдяки цій простій схемі кодування було штучно створено канал I_c^1, який кращий, ніж був без кодування, водночас створивши канал I_c^0, який гірший, ніж без кодування.

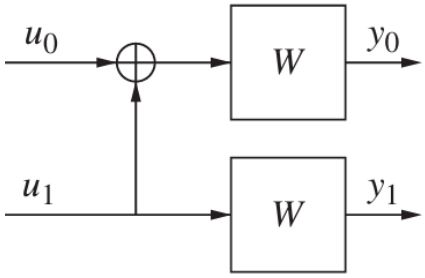


Рисунок 2.3 – Використання штучних каналів

Цю конструкцію можна застосовувати рекурсивно, що показано на рисунках 2.4 та 2.5, відповідно для N=4 та N=8. При збільшенні N, тобто кількості бітів у блоці, деякі з бітових каналів стають значно кращими, майже ідеальними для передачі інформації. Водночас інші канали втрачають здатність передавати надійну інформацію та стають надзвичайно шумними. Саме це явище і називається поляризацією каналів.

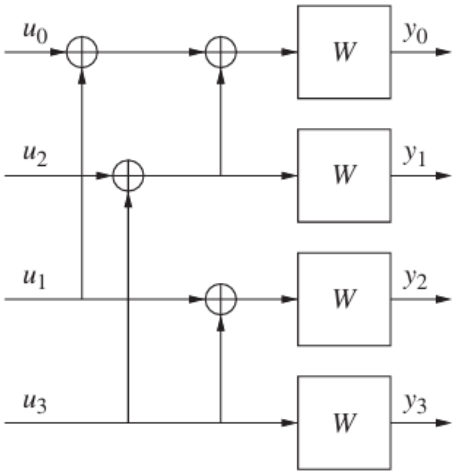


Рисунок 2.4 – Рекурсивне застосування побудови кодування при N=4

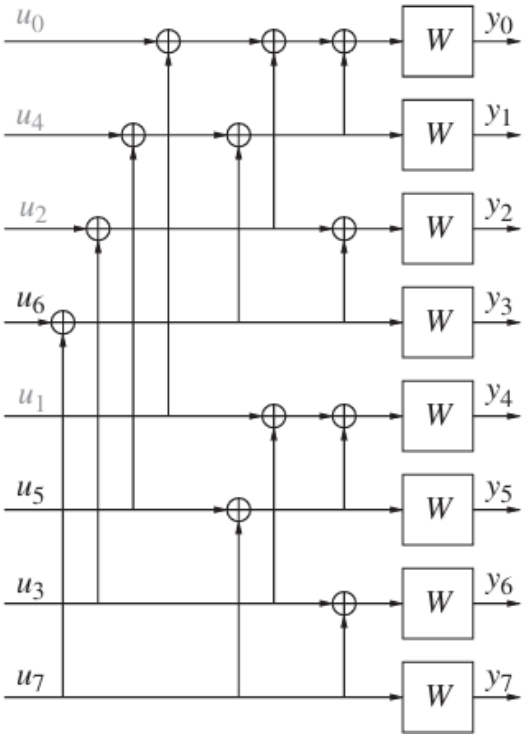


Рисунок 2.5 – Рекурсивне застосування побудови кодування при N=8 з деякими зафіксованими бітами

Найбільш значущим є той факт, що частка «ідеальних» каналів, які здатні передавати інформацію з майже одиничною надійністю, у межі наближається до пропускної здатності каналу W, тобто до величини $I(W)$. Полярне кодування використовує це явище наступним чином: щоб передавати дані з заданою швидкістю $R=K/N$, потрібно вибрати K найкращих каналів для передачі інформаційних бітів, а решту $N-K$ каналів «заморозити» − тобто встановити на фіксоване значення, найчастіше нуль. На рисунку 2.4 заморожені біти позначені сірим кольором, і видно, що біти u_3, u_5, u_6 та u_7 залишаються незамороженими – це інформаційні біти, які передають корисну інформацію.

Проєктування полярного коду полягає у виборі множини $\mathcal{A}^c$, тобто тих бітів, які будуть замороженими під час процесу кодування та декодування. У цьому контексті проєктуванням доцільно називати вибір множини заморожених бітів $\mathcal{A}^c$.

У загальному випадку для побудови ефективного полярного коду достатньо вибрати K найкращих синтетичних каналів серед N доступних, що відповідає швидкості коду $R=K/N$. Під «найкращими» каналами маються на увазі ті, для яких значення параметра Бхаттачар'ї $Z(W^{(i)}_N)$ є найменшими, оскільки це забезпечує мінімізацію верхньої межі ймовірності блочної помилки.

Важливим аспектом у проєктуванні полярних кодів є усвідомлення того, що полярні коди не є універсальними. Кожен код проєктується для певного каналу, зокрема – для певного рівня співвідношення сигналу до шуму SNR. Якщо полярний код, спроєктований для одного значення SNR, буде використано в умовах іншого SNR, його продуктивність може погіршитися. Рівень SNR, для якого створено полярний код, називається проектним SNR.

Проте на практиці код, розроблений для одного конкретного значення SNR, може зберігати високу ефективність у межах деякого діапазону значень SNR. Таким чином, немає потреби створювати окремий код для кожної можливої робочої точки каналу.

Практичний підхід, описаний у [13], полягає у наступному:

1. Розглянути набір значень SNR $\{S_1,S_2,\ldots,S_m$, які покривають діапазон значень SNR , що представляють інтерес для заданого каналу.

2. Для кожного значення S_i спроєктувати полярний код (тобто визначити множину заморожених бітів для коду).

3. Проаналізувати залежність бітової або блочної ймовірності помилки від SNR для кожного з розроблених полярних кодів.

4. На основі цього аналізу вибрати один код, що найкраще відповідає вимогам цільового застосування.

Межі Бхаттачар'ї є найпростішим способом побудови полярного коду та стали першим методом, запропонованим Аріканом [9]. Алгоритм побудови поступово розвиває параметри Бхаттачар'ї за допомогою схеми типу "метелика" та повертає верхні межі ймовірності блочної помилки для всіх N синтетичних каналів.

Межі Бхаттачар'ї можуть бути обчислені ітеративно за допомогою наступних рекурсивних співвідношень (2.18):

$$\begin{cases} Z_N^{(2i-1)} = Z_{N/2}^{(2i-1)} + Z_{N/2}^{(2i)} - Z_{N/2}^{(2i-1)} Z_{N/2}^{(2i)} \\ Z_N^{(2i)} = Z_{N/2}^{(2i-1)} Z_{N/2}^{(2i)} \end{cases}, \qquad (2.18)$$

де $Z^{(i)}{}_N - Z(W^{(i)}{}_N)$.

Рекурсивні взаємозв'язки алгоритмів в полярному кодуванні вимагають роботи в логарифмічній області, щоб уникнути помилок переповнення в їхніх реалізаціях. Доцільно використовувати функції, які додають та віднімають лінійні змінні в логарифмічній області. Нехай $x=ln(x)$ і $y=ln(y)$, тоді операції додавання (2.19) і віднімання (2.20) реалізуються наступним чином:

$$lsum(x,y) \triangleq \begin{cases} x + ln(1+e^{y-x}), & x \geq y \\ y + ln(1+e^{x-y}), & x < y \end{cases}, \qquad (2.19)$$

$$ldiff(x,y) \triangleq \begin{cases} x + ln(1-e^{y-x}), & x \geq y \\ y + ln(1-e^{x-y}), & x < y \end{cases}. \qquad (2.20)$$

Тоді межі Бхаттачар'ї в логарифмічній області обчислюються як (2.21):

$$\begin{cases} Z_N^{(2i-1)} = ldiff(lsum(Z_{N/2}^{(2i-1)}, Z_N^{(2i)}), Z_{N/2}^{(2i-1)} + Z_{N/2}^{(2i)}) \\ Z_N^{(2i)} = Z_{N/2}^{(2i-1)} + Z_{N/2}^{(2i)} \end{cases}, \qquad (2.21)$$

де $1 \leq i \leq N$;

$Z^{(i)}{}_1 = - E_b/N_0$ для каналу з білим шумом.

2.3 Розробка методу декодування повідомлення

Розглянемо декодування методом послідовного виключення [16] для довільного полярного коду з параметрами N, K, $\mathcal{A}$, $u_{\mathcal{A}c}$. Вектор джерела $u^{\mathrm{N}}{}_1$ складається з інформаційної частини $u_{\mathcal{A}}$ та замороженої частини $u_{\mathcal{A}c}$ Цей вектор передається через канал W_N, і на виході каналу отримується вихід $y^N{}_1$ з імовірністю $W_N(y^N{}_1|u^N{}_1)$.

Декодер спостерігає пару $y^N{}_1$ та $u_{\mathcal{A}c}$ і генерує оцінку $\hat{u}^N{}_1$ для вхідного вектора $u^N{}_1$. Можна уявити декодер як сукупність із N елементів прийняття рішень, по одному для кожного символу джерела u_i, елементи активуються послідовно від 1 до N.

Якщо $i \in \mathcal{A}^{\mathrm{c}}$, тобто u_i є замороженим, то відповідний елемент просто встановлює $\hat{u}_i = u_i$ і передає це значення всім наступним елементам.

Якщо $i \in \mathcal{A}$, то елемент очікує надходження усіх попередніх рішень $\hat{u}^{i-1}{}_1$, після чого обчислює відношення правдоподібності (2.22):

$$L_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1}\right) \triangleq \frac{W_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1}|1\right)}{W_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1}|0\right)} \qquad (2.22)$$

та приймає рішення за правилом (2.23):

$$\hat{u}_i = \begin{cases} 0, & L_N^{(i)}\left(y_N^1, \hat{u}_1^{i-1}\right) \geq 1 \\ 1, & L_N^{(i)}\left(y_N^1, \hat{u}_1^{i-1}\right) < 1 \end{cases}. \qquad (2.23)$$

Це однопрохідний алгоритм, тобто без перегляду вже прийнятих рішень. Його складність визначається переважно складністю обчислення відношень правдоподібності.

Відношення правдоподібності розраховуються як (2.24) і (2.25):

$$L_N^{(2i-1)}\left(y_1^N, \hat{u}_1^{2i-2}\right)=\frac{L_{N/2}^{(i)}\left(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}\right) L_{N/2}^{(i)}\left(y_{N/2+1}^{N}, \hat{u}_{1,o}^{2i-2}, \hat{u}_{1,e}^{2i-2}\right)}{L_{N/2}^{(i)}\left(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}\right)+L_{N/2}^{(i)}\left(y_{N/2+1}^{N}, \hat{u}_{1,o}^{2i-2}, \hat{u}_{1,e}^{2i-2}\right)} \quad (2.24)$$

$$L_N^{(2i)}\left(y_1^N, \hat{u}_1^{2i-1}\right) = \left[L_{N/2}^{(i)}\left(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}\right)\right]^{1-2\hat{u}_{2i-1}} \cdot L_{N/2}^{(i)}\left(y_{N/2+1}^{N}, \hat{u}_{1,o}^{2i-2}, \hat{u}_{1,e}^{2i-2}\right). \quad (2.25)$$

В логарифмічній області (2.24) виражається як (2.26), а (2.25) як (2.27):

$$L_N^{(2i-1)}\left(y_1^N, \hat{u}_1^{2i-2}\right) = ln\left(\frac{e^{L_1+L_2}+1}{e^{L_1}+e^{L_2}}\right) = lsum(L_1 + L_2, 0) - lsum(L_1, L_2) \quad (2.26)$$

$$L_N^{(2i)}\left(y_1^N, \hat{u}_1^{2i-1}\right) = \begin{cases} L_1 + L_2, & \hat{u}_{2i-1} = 0 \\ L_1 - L_2, & \hat{u}_{2i-1} = 1, \end{cases} \quad (2.27)$$

де $L_1 - L_{N/2}^{(i)}\left(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}\right)$;

$L_2 - L_{N/2}^{(i)}\left(y_{N/2+1}^{N}, \hat{u}_{1,o}^{2i-2}, \hat{u}_{1,e}^{2i-2}\right)$.

Таким чином, обчислення відношення правдоподібності довжини *N* зводиться до двох відношень довжини *N/2*.

2.4 Розробка методу симуляції каналу завадами

У цій роботі використовується схема модуляції BPSK [17], за якою біт «0» відображається у сигнал зі значенням $+\sqrt{E_b}$, а біт «1» $-\sqrt{E_b}$. Таким чином, кожне кодове слово модулюється за формулою (2.28):

$$\hat{x} = (2x - 1)\sqrt{E_b} \tag{2.28}$$

де $\hat{x}$ – бітове представлення кодового слова.

У програмному модулі моделюється канал із адитивним білим гаусівським шумом. Відповідно до (1.1), отриманий сигнал має вигляд: $y = \hat{x} + \xi$, де $\xi \sim \mathcal{N}(0,\ N_0/2)$, а N_0 – спектральна щільність потужності шуму. Для спрощення розрахунків зазвичай приймається N_0=1, тоді E_b визначає проектне співвідношення E_b/N_0.

Декодер потребує представлення сигналу у вигляді логарифмічних правдоподібностей, що забезпечує чисельну стабільність та зручність обчислень. Правдоподібність визначається як (2.29):

$$\mathcal{L} = ln\left(\frac{p(x=0)}{p(x=1)}\right) = -\frac{2y}{N_0}\sqrt{E_b} \tag{2.29}$$

де p – імовірність отримання сигналу.

Ймовірність отримання сигналу у при переданому біті x задається гаусівською щільністю (2.30):

$$p(x) = p(y \mid \hat{x}) = \frac{1}{\sqrt{\pi N_0}} exp\{(y - \hat{x})/N_0, \hat{x} \in \{\pm\sqrt{E_b}\} \tag{2.30}$$

Щоб мати змогу коректно порівнювати різні коди, проектне співвідношення SNR має бути нормалізоване на швидкість коду $R=K/N$. Це означає, що повинна справджуватися рівність (2.31):

$$\left(\frac{E_b}{N_0}\right)_{\text{код}} = \frac{1}{R}\left(\frac{E_b}{N_0}\right)_{\text{повідомлення}} \tag{2.31}$$

Таким чином, передана енергія повідомлення для будь-якого кодового слова залишається сталою.

2.5 Висновки

У другому розділі було сформовано теоретичне підґрунтя для побудови системи завадостійкого кодування на основі полярних кодів. Розглянуто механізм поляризації каналів, метод визначення заморожених бітів за допомогою меж Бхаттачар'ї, а також принципи систематичного кодування. Наведено математичну модель передачі даних через канал із адитивним білим гаусівським шумом із використанням BPSK-модуляції. Описано основи декодування методом послідовного виключення з рекурсивним оновленням логарифмічних правдоподібностей. Отримані теоретичні положення стали основою для подальшої реалізації програмного модуля.

3 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ДЛЯ ЗАВАДОСТІЙКОГО КОДУВАННЯ ДАНИХ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного модуля

У процесі проєктування модуля полярного завадостійкого кодування було проведено аналіз можливих варіантів реалізації з огляду на ключові вимоги до програмного забезпечення. До таких вимог належать: висока обчислювальна ефективність при роботі з великими масивами даних, зручність реалізації складних алгоритмів, підтримка векторизованих та паралельних обчислень, доступність засобів візуалізації та швидкість розробки прототипів.

Серед основних мов програмування, які розглядалися для реалізації, були C++, C, Java, MATLAB, та Python.

Мови C та C++ традиційно вважаються найпродуктивнішими засобами для реалізації алгоритмів нижнього рівня, зокрема тих, що використовуються в засобах цифрової обробки сигналів [18]. Їхні переваги включають прямий контроль над пам’яттю, високу швидкодію та широкі можливості для оптимізації. Проте використання цих мов значно ускладнює розробку, налагодження і супровід коду, особливо коли йдеться про реалізацію ітеративних алгоритмів, що змінюються в процесі експериментів. Висока складність низькорівневої реалізації не є виправданою на ранніх етапах дослідження або для задач, де важлива гнучкість змін і швидка адаптація.

Java забезпечує кращу безпеку пам’яті та дещо простішу розробку порівняно з C++, однак поступається у продуктивності та має меншу популярність у галузі чисельного моделювання та наукових обчислень. Крім того, для реалізації алгоритмів, що потребують тісного контролю над ресурсами та оптимізацією чисельних операцій, Java не є пріоритетною [19].

MATLAB є потужним середовищем для наукових обчислень і часто використовується для моделювання систем зв’язку, зокрема для перевірки

алгоритмів завадостійкого кодування. Перевагою MATLAB є зручність синтаксису для векторних операцій і доступність вбудованих функцій. Однак комерційна ліцензія, обмеження щодо інтеграції в сторонні проєкти та нижча продуктивність при обробці великих масивів даних роблять його менш привабливим для створення готового рішення [20].

У цьому контексті Python виявився найкращим компромісом між швидкістю розробки, зручністю налагодження та обчислювальною ефективністю. Завдяки своїй простій та виразній синтаксичній структурі Python значно пришвидшує розробку ітеративних прототипів та реалізацію складних математичних моделей. Його популярність у галузі машинного навчання, цифрової обробки сигналів та наукових досліджень підтверджується широким спектром бібліотек і готових рішень [21].

Основою обчислювального ядра реалізованого модуля стала бібліотека NumPy [22], що забезпечує ефективну роботу з багатовимірними масивами. Для прискорення критичних ділянок коду було застосовано компілятор Numba [23], що дозволив компілювати окремі функції у високопродуктивний машинний код з використанням Just-In-Time підходу. Завдяки поєднанню Python, NumPy і Numba вдалося досягти рівня продуктивності, порівнянного з рішеннями на більш низькорівневих мовах, при цьому зберігаючи зручність реалізації та можливість швидкої адаптації алгоритмів.

Отже, з огляду на всі перелічені фактори, Python став найбільш доцільним вибором для реалізації модуля полярного завадостійкого кодування. Обрана платформа забезпечує оптимальний баланс між простотою розробки, високою обчислювальною ефективністю та можливістю масштабування, що є критичним для експериментального дослідження алгоритмів у системах цифрового зв’язку.

3.2 Розробка моделі та алгоритму застосунку

Для забезпечення зручної та наочної демонстрації можливостей програмного модуля було розроблено інтуїтивний графічний інтерфейс. Робота

із застосунком починається з того, що користувач має можливість обрати зображення для подальшої передачі. Це можна зробити двома способами: або вручну ввести шлях до файлу у відповідне текстове поле, або скористатися кнопкою «Огляд», яка відкриває стандартне діалогове вікно вибору файлу. Такий підхід дозволяє швидко та зручно знаходити потрібне зображення на комп’ютері.

Після вибору зображення користувач переходить до налаштування параметрів кодування. У відповідних полях інтерфейсу необхідно вказати довжину коду N, кількість інформаційних бітів K та проектний та канальний рівень відношення SNR. Ці параметри визначають характеристики полярного коду, який буде використано для передачі зображення через зашумлений канал. Інтерфейс дозволяє легко змінювати ці значення, що дає змогу експериментувати з різними налаштуваннями та досліджувати їх вплив на якість передачі.

Після заповнення всіх необхідних полів користувач натискає кнопку «Симулювати». У цей момент застосунок переходить у режим обробки. Якщо під час введення даних було допущено помилку, наприклад, некоректний шлях до файлу, неправильний формат зображення або недопустимі значення параметрів, система автоматично повідомляє про це за допомогою діалогового вікна з описом помилки. Після цього користувач може виправити дані та повторити спробу.

Після успішної обробки зображення результати автоматично відображаються у головному вікні програми. Користувач бачить три зображення: оригінал, зашумлену версію, яка імітує передачу через канал із шумом, та виправлену версію, яка отримана після декодування. Зображення розташовані поруч, що дозволяє легко порівнювати їх між собою та оцінювати ефективність застосування полярних кодів для захисту інформації. Також відображається статистика відновлення у відсотках.

Користувач може змінювати параметри кодування та повторювати експеримент стільки разів, скільки потрібно, що робить застосунок зручним інструментом для навчання, досліджень та демонстрації роботи сучасних методів кодування. У будь-який момент можна повернутися до початкового

стану, обрати інше зображення або змінити налаштування, не перезапускаючи програму.

Для кращого розуміння роботи модулів програмного засобу було розроблено діаграму діяльності застосунку, що зображена на рисунку 3.1.

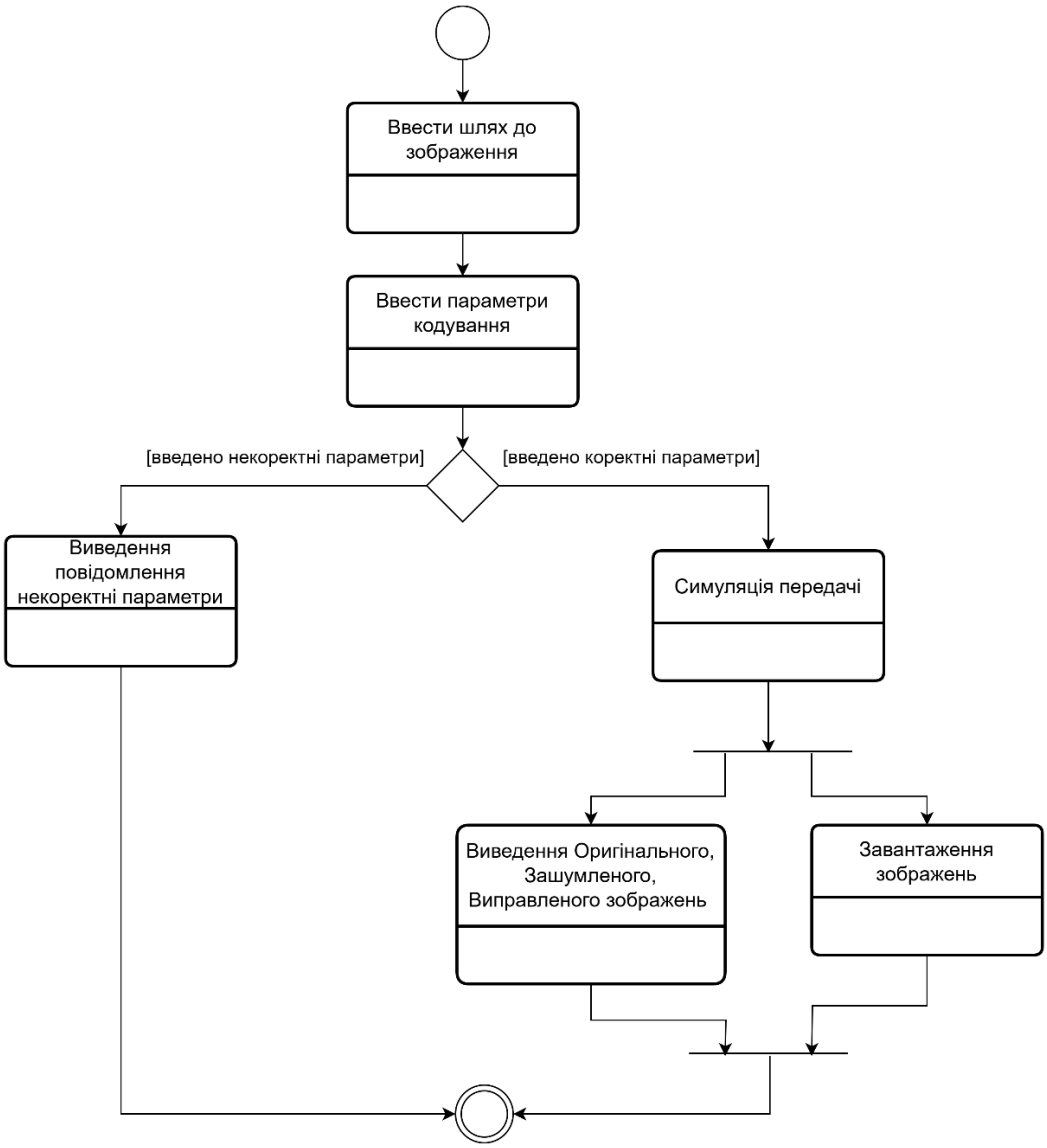


Рисунок 3.1 – Діаграма діяльності програмного застосунку

Також було розроблено блок-схему функції, яка відповідає за основний цикл обробки зображення у програмному застосунку. Ця функція реалізує послідовність дій, необхідних для підготовки зображення до передачі через зашумлений канал із використанням полярних кодів, а також для подальшого відновлення та візуалізації результатів. Зокрема, вона здійснює зчитування

зображення, кодування, симуляцію передачі через канал, декодування та формування трьох зображень для відображення у графічному інтерфейсі: оригінального, зашумленого та виправленого після декодування.

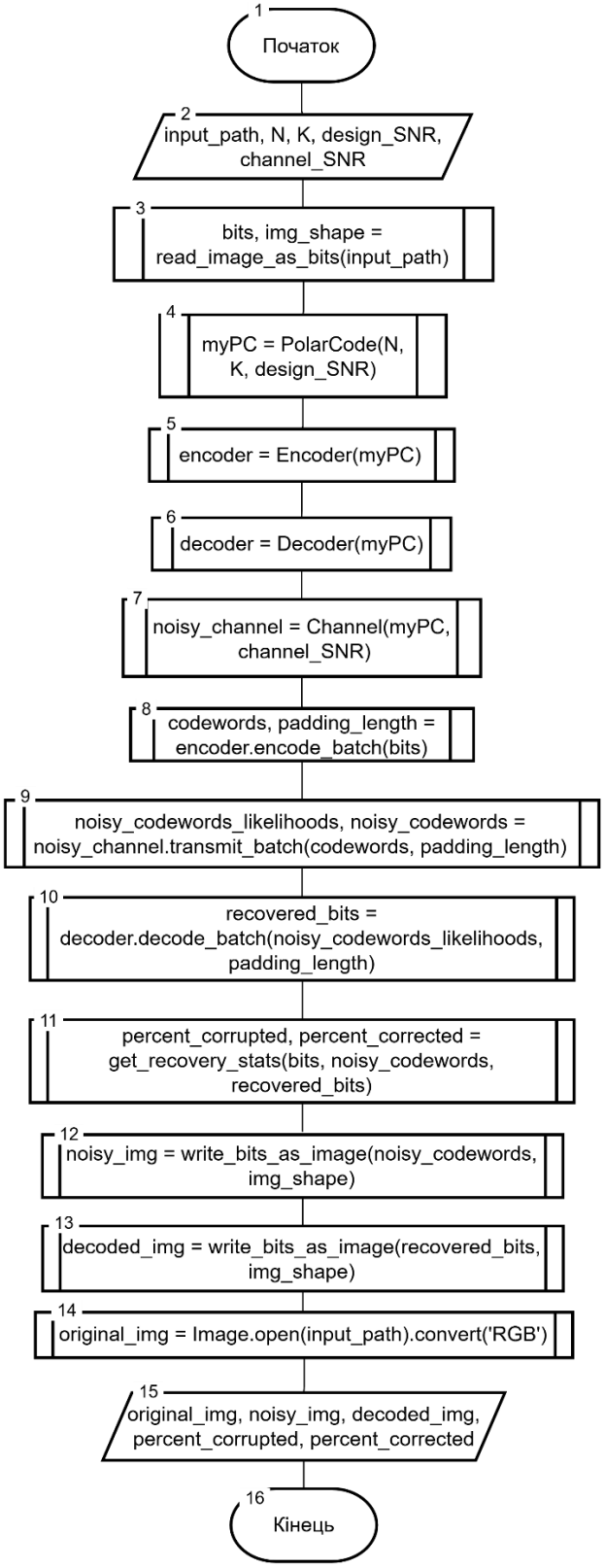


Рисунок 3.2 – Алгоритм функції обробки зображення

На рисунку 3.3 наведено UML-діаграму класів основних компонентів програмного забезпечення для високопродуктивної передачі зображень із використанням полярних кодів. Діаграма ілюструє структуру проєкту, взаємозв’язки між класами та основні атрибути й методи кожного класу.

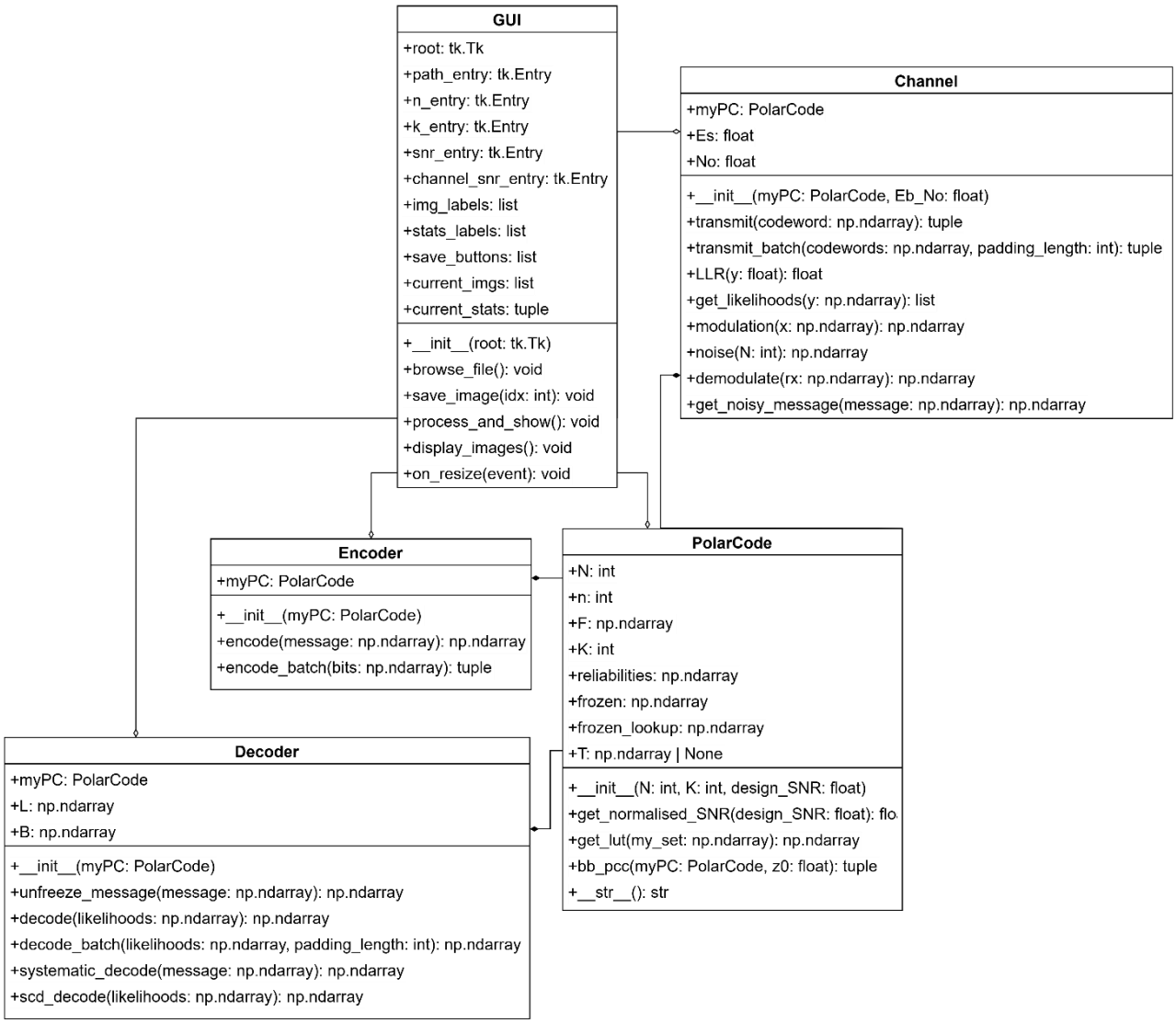


Рисунок 3.3 – Діаграма класів

Клас PolarCode відповідає за побудову та параметризацію полярного коду. Містить методи для генерації матриці Арікана, визначення заморожених та інформаційних бітів.

Клас Encoder реалізує процес кодування інформаційної послідовності згідно з параметрами полярного коду. Забезпечує формування кодових слів для

подальшої передачі через канал.

Клас Decoder відповідає за декодування прийнятих даних із використанням алгоритму послідовного виключення. Відновлює інформаційні біти з ймовірностей, отриманих після проходження каналу.

Клас Channel моделює канал передачі із додаванням шуму. Забезпечує модуляцію, додавання шуму, демодуляцію та обчислення логарифмічних відношень правдоподібності для подальшого декодування.

Клас GUI реалізує графічний інтерфейс користувача. Дозволяє обирати зображення, задавати параметри кодування, запускати процес обробки та переглядати результати у зручному вигляді.

3.3 Розробка модуля полярного кодування

Для побудови полярного коду була розроблена функція побудови на основі оцінки надійності бітових каналів методом параметра Бхаттачарії. Вона є ключовим етапом ініціалізації полярного коду, оскільки дозволяє визначити, які біти будуть інформаційними, а які – замороженими.

На вхід функція отримує початковий рівень надійності каналів, що визначається проектним E_b/N_0 та параметрами коду. Далі вона рекурсивно обчислює параметри надійності для кожного бітового каналу за допомогою формули (2.21) на основі спеціальних співвідношень, що відображають структуру поляризації каналів. На кожному рівні рекурсії відбувається розгалуження каналів на більш надійні та менш надійні, що дозволяє поступово сформувати повний розподіл надійності для всіх позицій у кодовому слові.

Після завершення обчислень функція впорядковує біти за ступенем їхньої надійності, визначає множину заморожених бітів, які отримають фіксоване значення нуля. Отримані результати використовуються для формування структури полярного коду, що забезпечує оптимальний розподіл інформаційних та заморожених бітів відповідно до умов передачі.

Блок-схема розробленої функції зображена на рисунку 3.4.

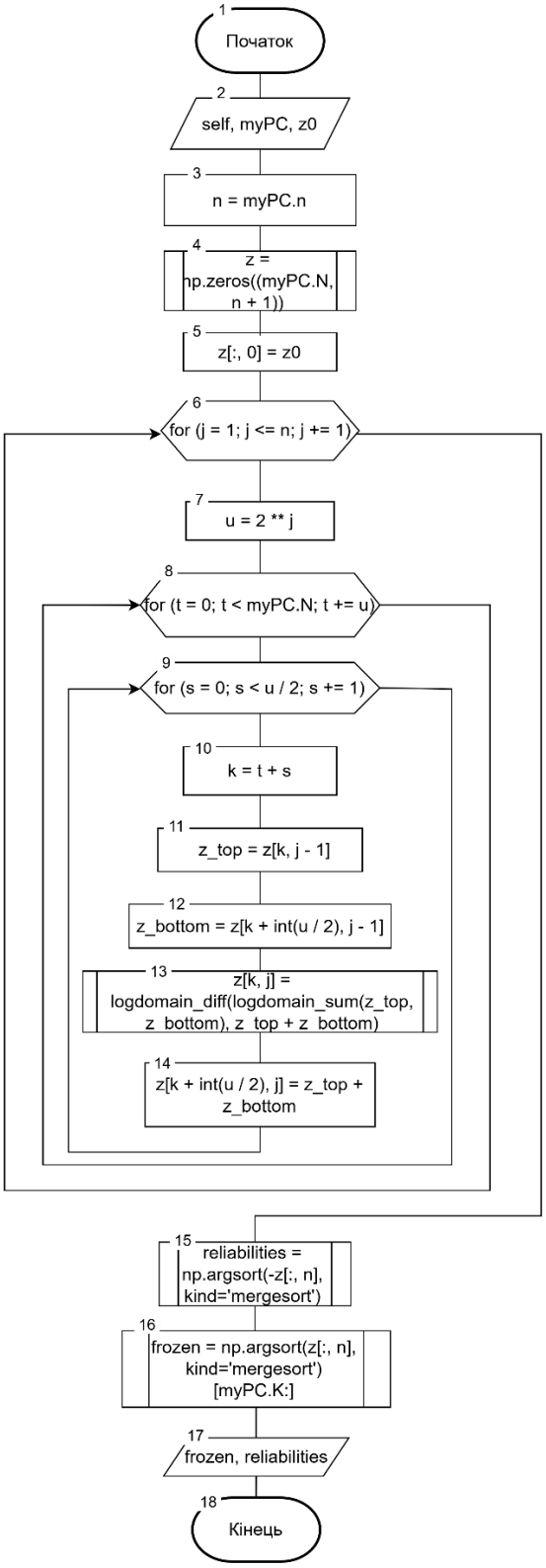


Рисунок 3.4 – Алгоритм побудови полярного коду

На рисунку 3.5 представлено блок-схему алгоритму, який відповідає за формування закодованої послідовності на основі вхідних інформаційних бітів відповідно до структури полярного коду. Даний алгоритм розміщує

інформаційні біти у визначені позиції, формує повний вектор для кодування, виконує необхідні перетворення та отримує кодове слово, яке може бути передане через канал зв’язку.

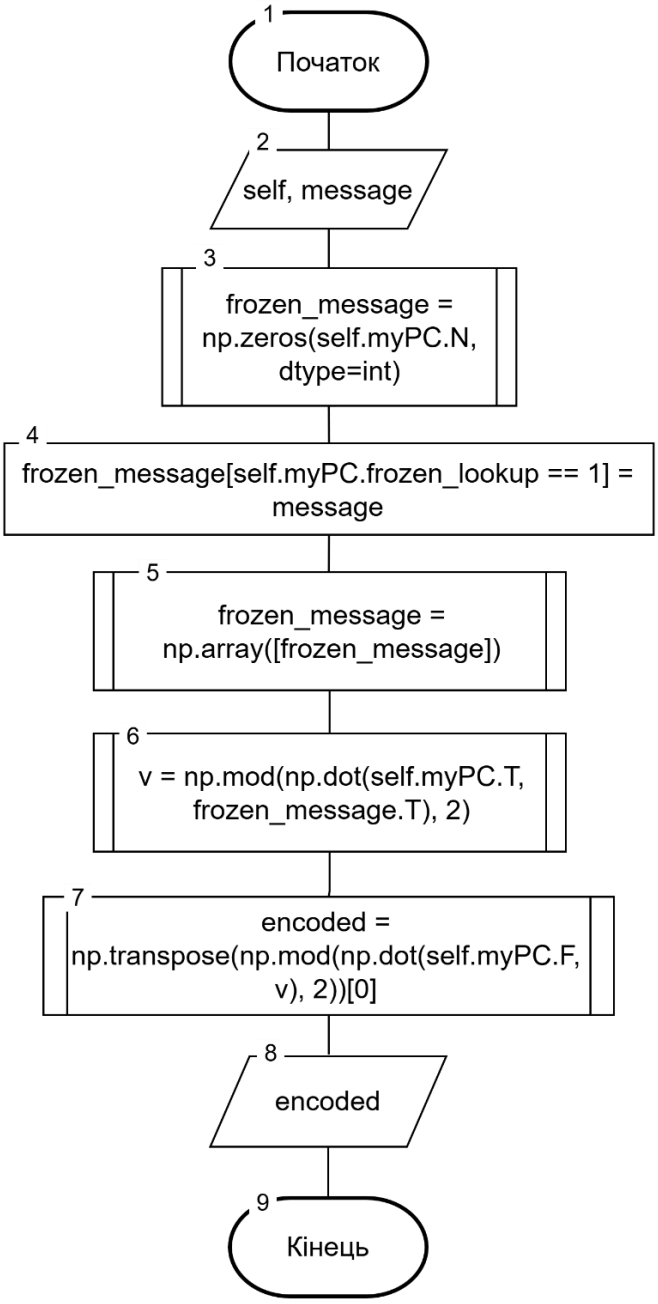


Рисунок 3.5 – Алгоритм кодування

На рисунку 3.6 представлено блок-схему алгоритму декодування, який використовується для поетапного відновлення інформаційної послідовності з отриманих після передачі через канал ймовірностей. Даний алгоритм реалізує

послідовне прийняття рішень щодо кожного біта на основі попередньо обчислених логарифмічних відношень правдоподібності та вже відновлених бітів. На кожному кроці визначається, чи є поточна позиція інформаційною або замороженою, після чого приймається відповідне рішення щодо її значення. В результаті роботи алгоритму формується відновлена послідовність бітів, яка використовується для подальшого відновлення переданого повідомлення.

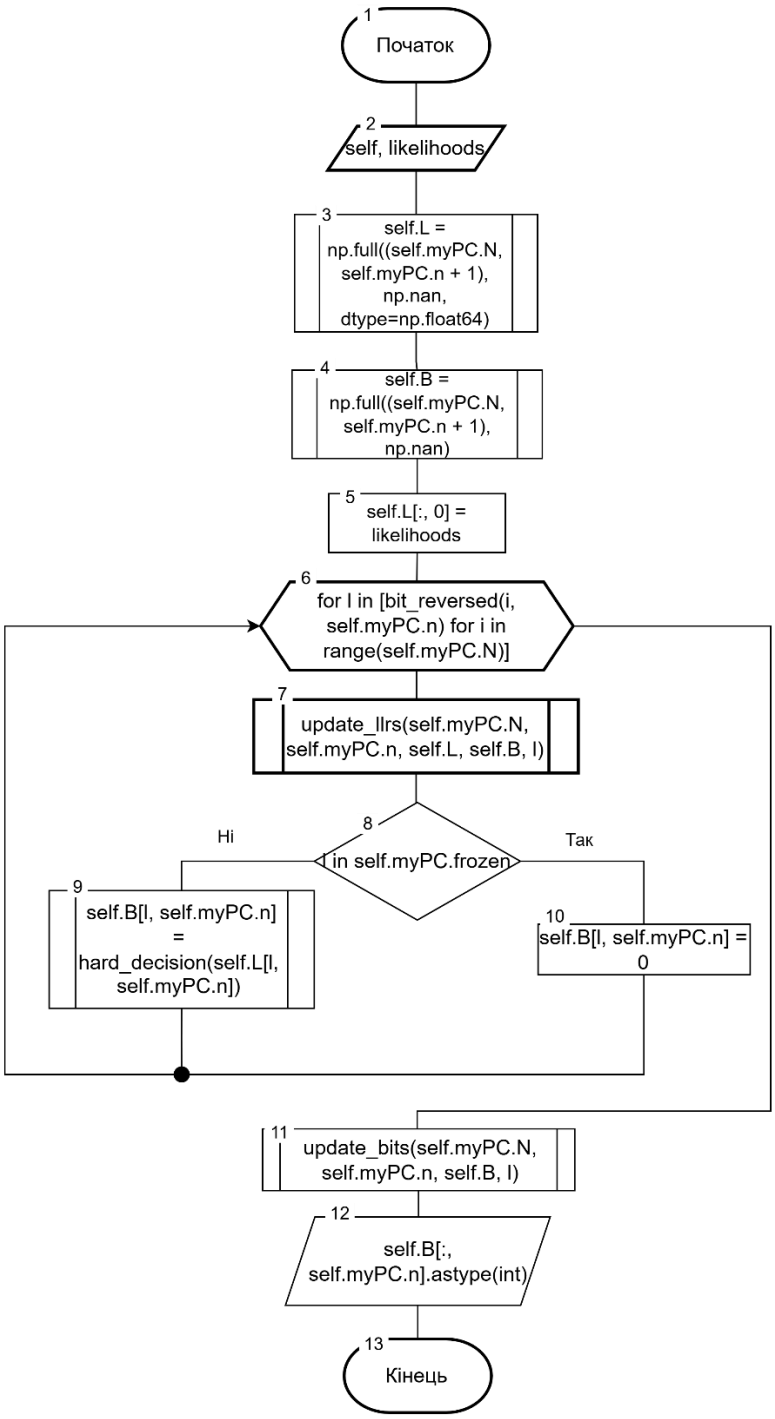


Рисунок 3.6 – Алгоритм декодування послідовного виключення

На рисунку 3.7 представлено блок-схему алгоритму, який відповідає за рекурсивне оновлення проміжних ймовірностей для декодування інформаційної послідовності. Даний алгоритм використовується для поетапного обчислення допоміжних значень, необхідних для прийняття рішень щодо кожного біта під час декодування за допомогою формул (2.26) та (2.27). На кожному кроці враховується структура коду та вже відновлені біти, що дозволяє забезпечити коректне та ефективне відновлення переданої інформації.

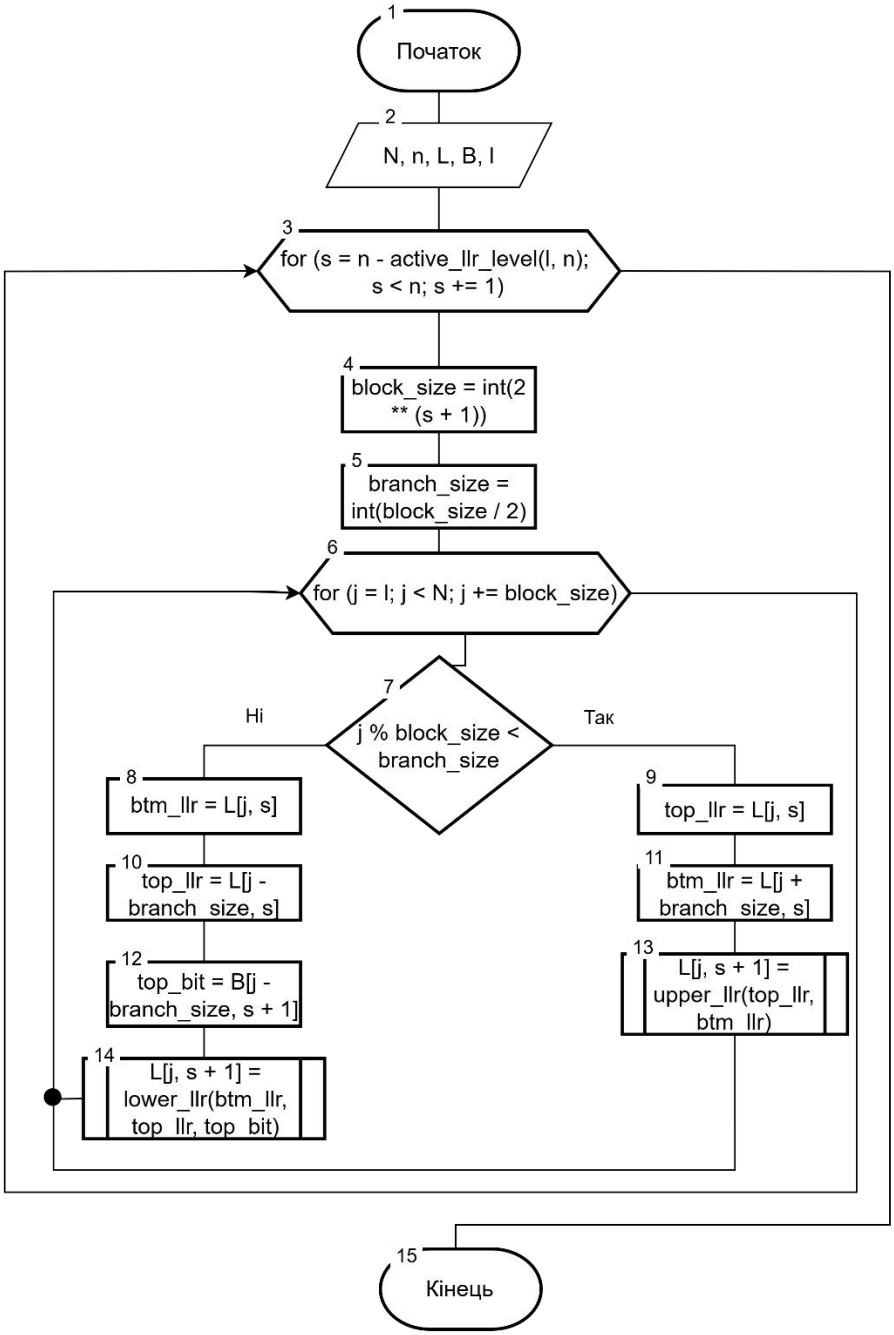


Рисунок 3.7 – Алгоритм оновлення оновлення проміжних ймовірностей

На рисунку 3.8 представлено блок-схему алгоритму, який відповідає за оновлення проміжних значень відновлених бітів під час декодування. Даний алгоритм використовується для коректного поширення отриманих рішень по дереву декодування, враховуючи ієрархічну структуру коду. Це дозволяє забезпечити правильне формування інформаційної послідовності на основі вже прийнятих рішень щодо окремих бітів.

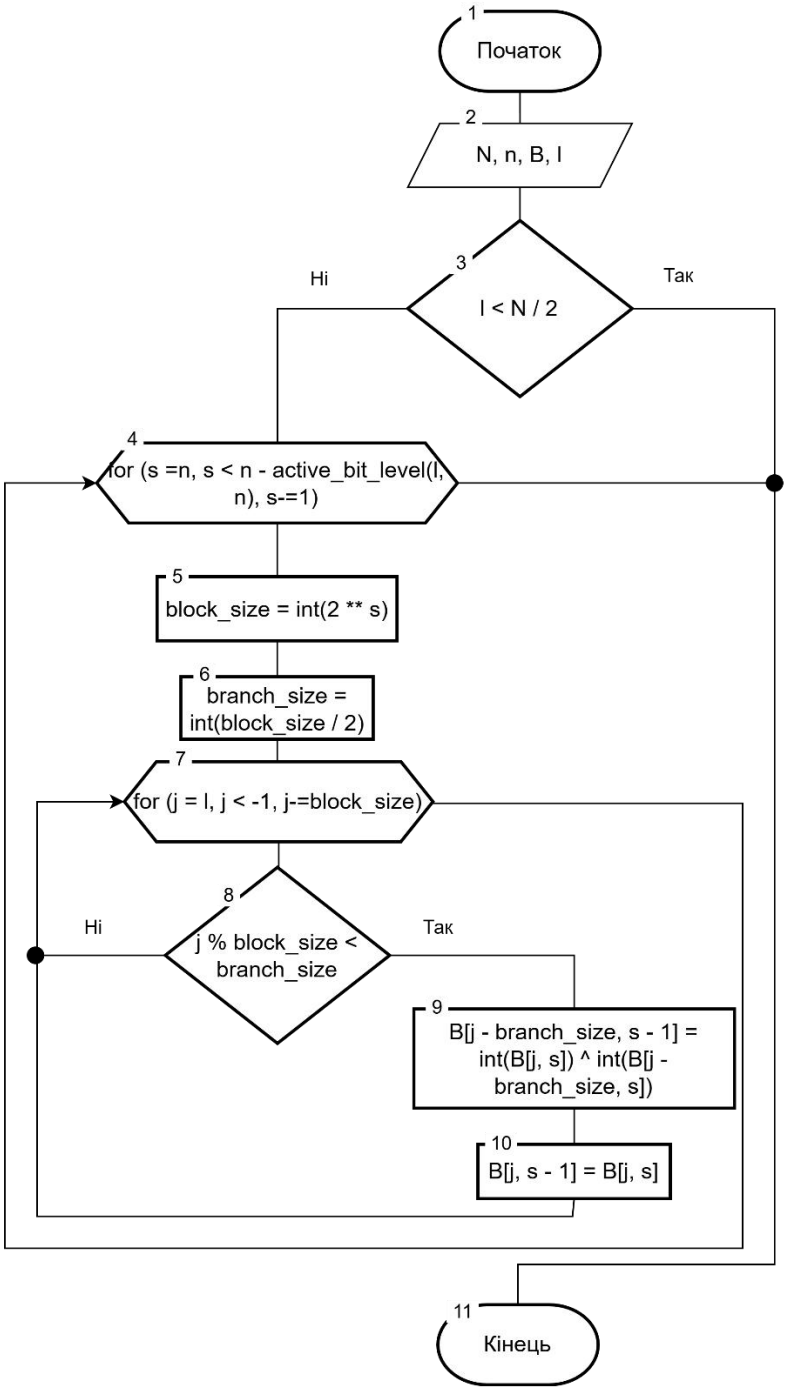


Рисунок 3.8 – Алгоритм оновлення проміжних значень відновлених бітів

На рисунку 3.9 представлено блок-схему алгоритму, який відповідає за пакетне кодування інформаційних бітів у кодові слова відповідно до заданої структури полярного коду. Даний алгоритм здійснює підготовку вхідної послідовності, розбиває її на блоки необхідної довжини, виконує кодування кожного блоку та формує масив закодованих слів для подальшої передачі через канал.

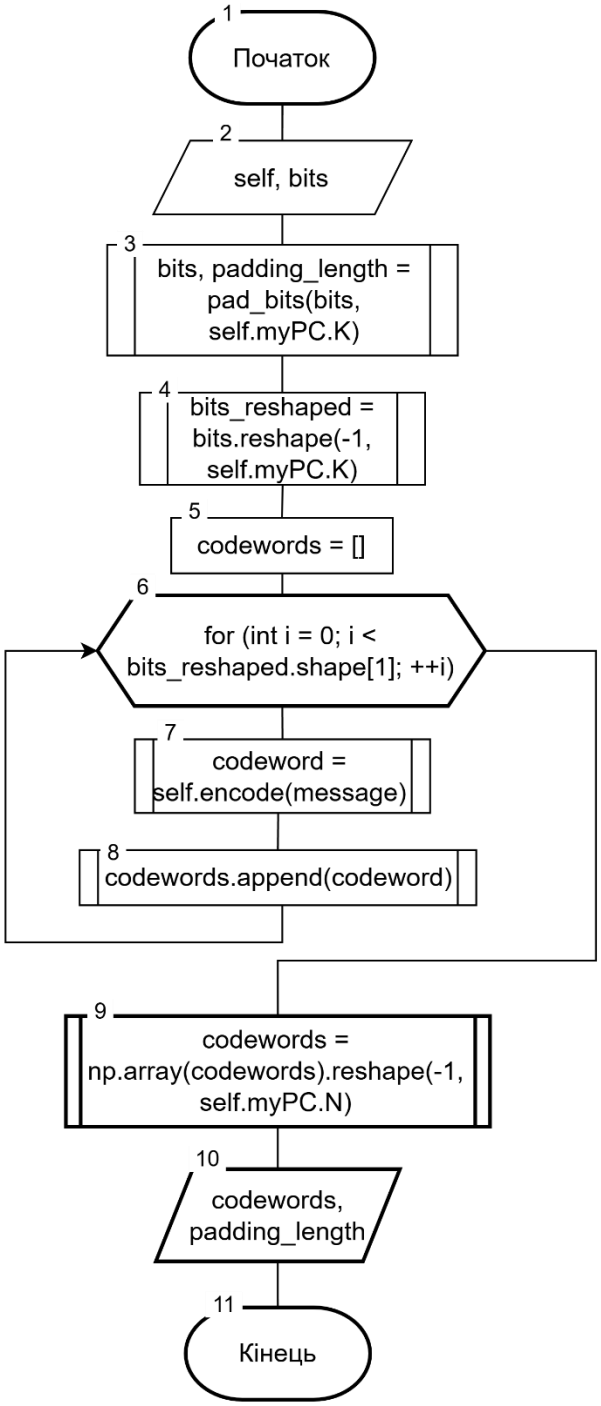


Рисунок 3.9 – Алгоритм пакетного кодування

На рисунку 3.10 представлено блок-схему алгоритму, який відповідає за симуляцію пакетної передачі закодованих даних через зашумлений канал. Даний алгоритм поетапно обробляє кожне кодове слово: здійснює його передачу через канал із додаванням шуму, формує відповідні ймовірності для декодування та накопичує результати для всієї послідовності. Після завершення обробки забезпечується коректне формування вихідних даних для подальшого декодування та аналізу.

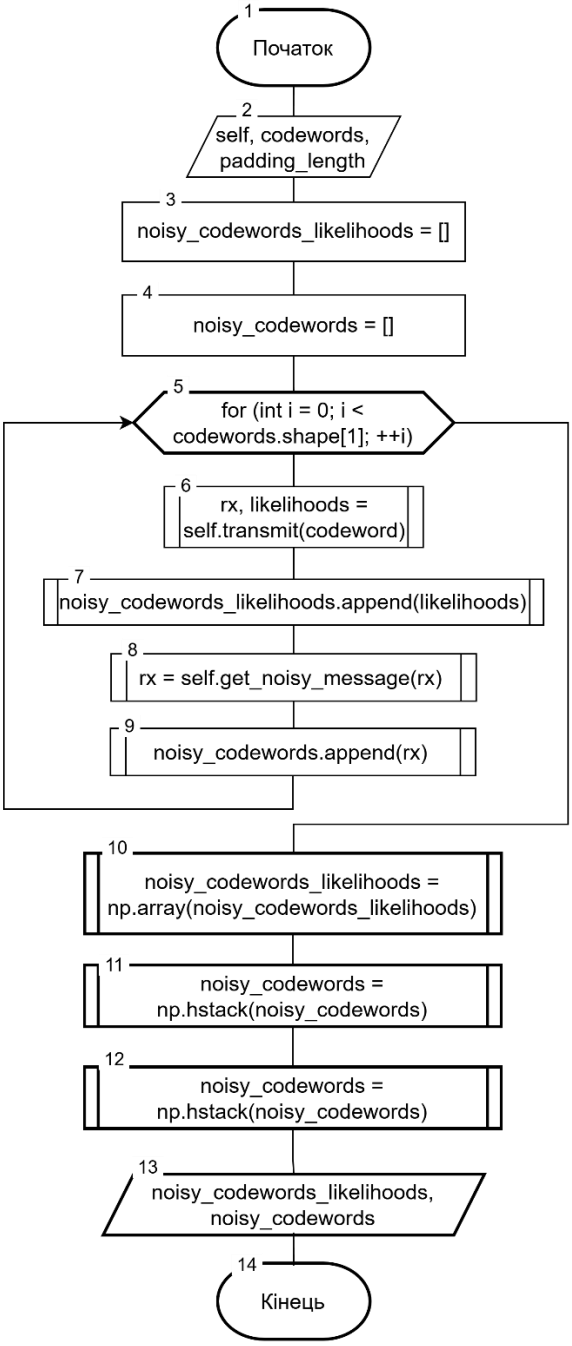


Рисунок 3.10 – Алгоритм симуляції пакетної передачі даних

На рисунку 3.11 представлено блок-схему алгоритму, який відповідає за пакетне декодування отриманих після передачі через канал ймовірностей. Даний алгоритм поетапно обробляє кожен набір вхідних даних, виконує відновлення інформаційної послідовності для кожного блоку, об’єднує результати та забезпечує коректне видалення службових бітів, що були додані під час кодування. Це дозволяє отримати відновлену бітову послідовність для подальшого формування зображення.

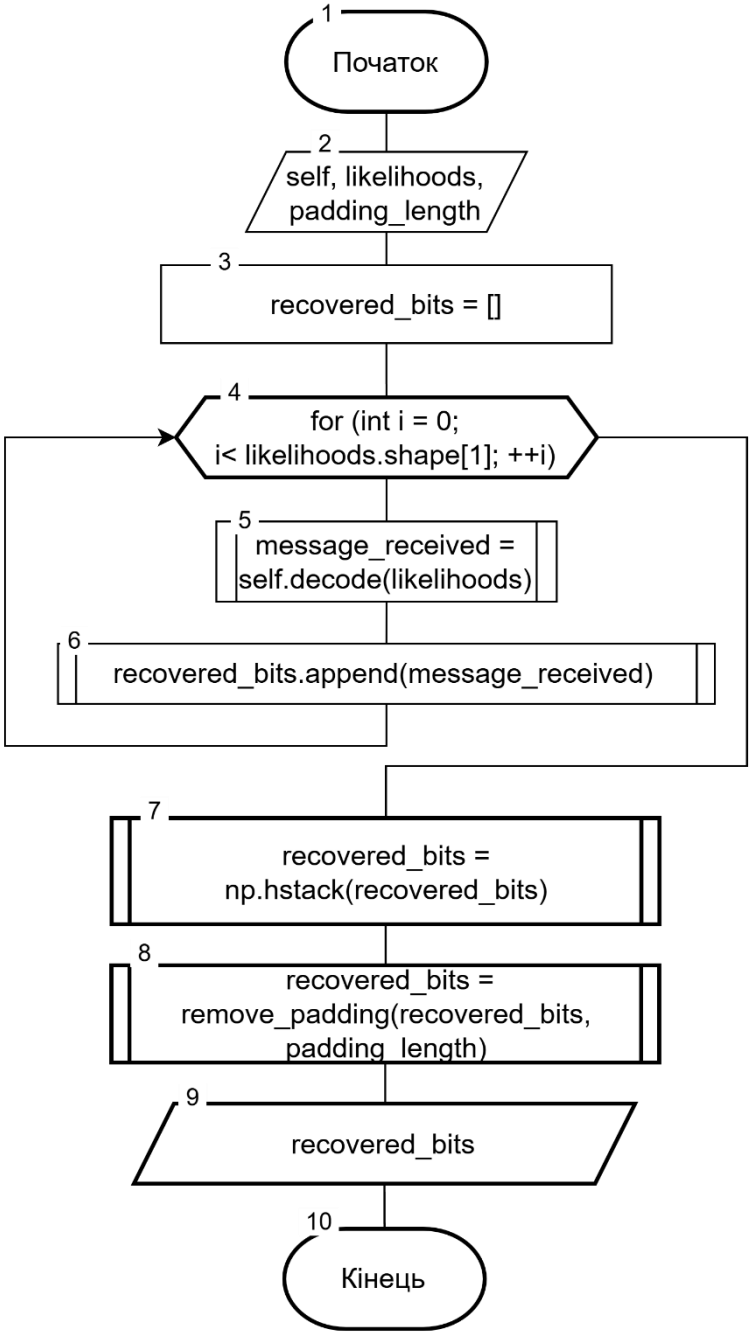


Рисунок 3.11 – Алгоритм пакетного декодування

3.4 Висновки

У третьому розділі було розроблено алгоритм роботи програмного модуля завадостійкого кодування даних. На основі проведеного варіантного аналізу обґрунтовано вибір мови програмування Python та бібліотек NumPy, Numba, що дозволило досягти високої продуктивності обчислень при збереженні гнучкості реалізації. У розділі також було описано блок-схеми основних алгоритмів, а також діаграму класів, яка ілюструє взаємозв’язки між класами системи.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування програмного забезпечення

Тестування є невід'ємною складовою життєвого циклу його розробки, адже воно дає змогу виявити і своєчасно усунути помилки, а також підтвердити відповідність функціональності продукту встановленим вимогам і технічним специфікаціям [24]. Його мета полягає у забезпеченні якості програмної системи шляхом перевірки функціональних і нефункціональних характеристик, зокрема надійності, ефективності, зручності використання, продуктивності та безпеки. Крім того, тестування охоплює процеси верифікації та валідації програмного продукту, які забезпечують відповідність очікуванням користувача.

Верифікація передбачає перевірку того, наскільки реалізація системи відповідає її технічним вимогам і специфікації. Для цього застосовуються як статичні методи – аналіз коду, рецензування, так і динамічні – виконання тестів з подальшим аналізом результатів. Основна мета верифікації – переконатися, що реалізація програмного забезпечення відповідає попередньо визначеним технічним критеріям.

Натомість валідація спрямована на визначення того, чи задовольняє система реальні потреби кінцевого користувача. Це включає оцінку функціональності, зручності інтерфейсу, відповідності практичним умовам експлуатації та загальній придатності до використання. Таким чином, валідація засвідчує, що створене програмне забезпечення може бути ефективно застосоване за призначенням [25].

Методи тестування класифікуються залежно від ступеня обізнаності тестувальника про внутрішню структуру програмної системи [26]. Тестування методом «чорної скриньки» передбачає перевірку зовнішньої поведінки системи без доступу до її внутрішньої реалізації. Такий підхід дозволяє виявити помилки в логіці функціонування, структурі даних і реалізації інтерфейсів, розглядаючи систему виключно з точки зору користувача.

Тестування методом «білої скриньки», навпаки, базується на повному розумінні коду програми. Воно дозволяє аналізувати логіку виконання, шляхи проходження програми та внутрішні залежності. Такий підхід забезпечує максимальне покриття тестами та ефективне виявлення логічних помилок, збоїв або вразливостей, що залишилися після етапу реалізації.

Компромісним варіантом є тестування методом «сірої скриньки», коли тестувальник має часткову інформацію про реалізацію, що дозволяє йому більш точно визначити критичні ділянки для перевірки. Цей підхід поєднує переваги двох попередніх, дозволяючи проводити гнучке тестування з високою ефективністю.

Для перевірки коректності функціонування реалізованого програмного модуля в межах цієї роботи було обрано метод «білої скриньки». Тестування проводилося у середовищі Windows 11.

На першому етапі була перевірена поведінка програмного модуля при введенні некоректних вхідних даних, що дозволило виявити потенційні помилки обробки. На рисунку 4.1 зображено реакцію розробленого програмного модуля на встановлення довжини коду N меншою за довжину інформаційної послідовності K.

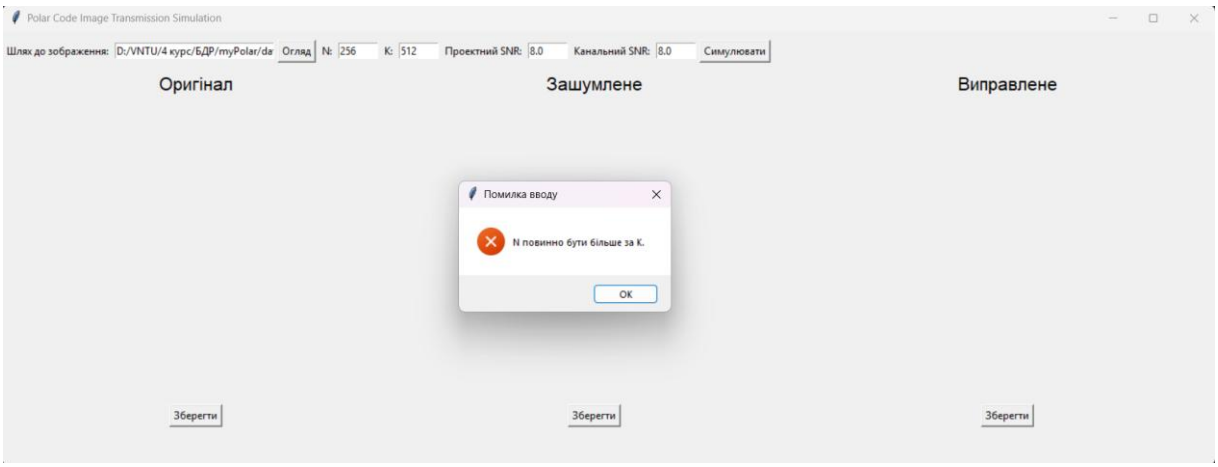


Рисунок 4.1 – Встановлення N меншим ніж K

Якщо користувач вводить значення N, яке менше або дорівнює кількості інформаційних бітів K, система повідомляє про помилку. Це попереджає

користувача, що довжина коду повинна бути більшою за кількість інформаційних бітів, інакше кодування неможливе.

На рисунку 4.2 показано реакцію програмного забезпечення на введення некоректного значення параметра *N*, яке не є степенем двійки або є недопустимо малим.

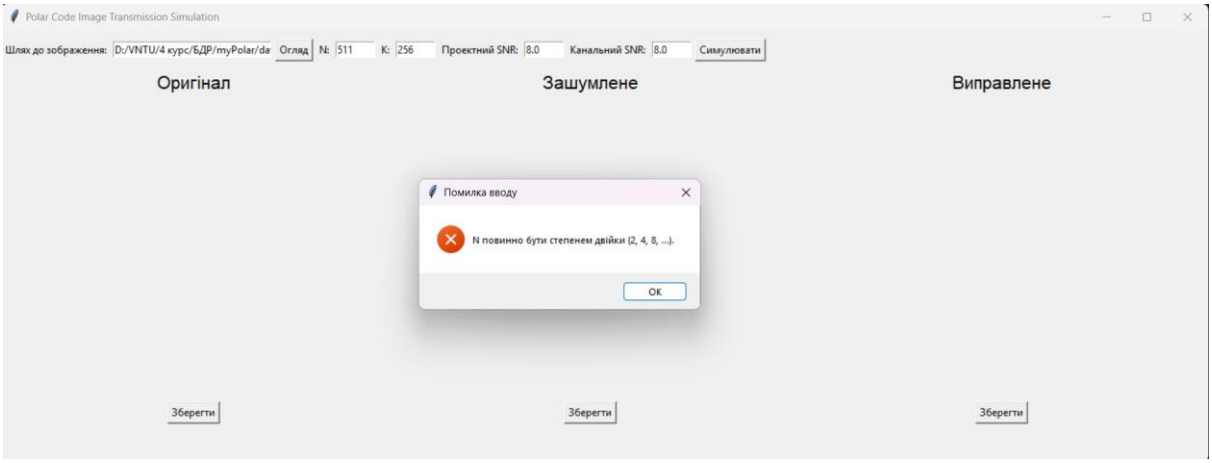


Рисунок 4.2 – Повідомлення про помилку при некоректному значенні *N*

У разі спроби ввести значення *N*, яке не є степенем або менше чи дорівнює нулю, система автоматично видає повідомлення про помилку. Це інформує користувача про необхідність ввести коректне значення *N* (2, 4, 8, ...), що запобігає некоректній роботі алгоритму кодування.

На рисунку 4.3 показано реакцію програмного забезпечення на введення від’ємних або нульових значень параметрів SNR.

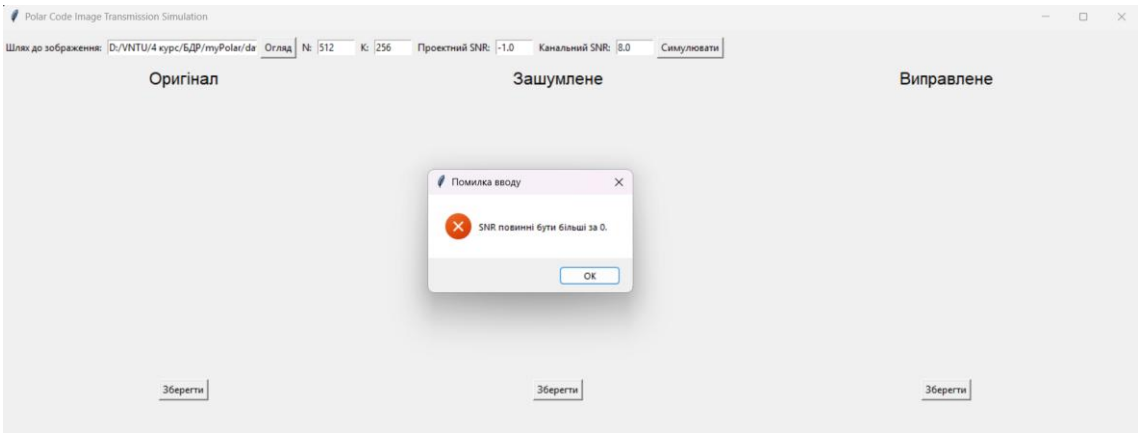


Рисунок 4.3 – Повідомлення про помилку при некоректних SNR

Якщо користувач вводить від’ємні або нульові значення для проектного чи канального SNR, система видає відповідне повідомлення про помилку. Це забезпечує коректність моделювання каналу та попереджає про необхідність введення додатних значень.

Наступним етапом було проведено тестування програмного забезпечення під час симуляції передачі та відновлення даних з урахуванням потенційного шуму або викривлень. Усі тести проводяться на одному зображенні – фотографії тигра, яка містить широкий спектр контрастних елементів, структури різного розміру, ділянки симетрії та присутні деталі, чутливі до спотворень. Зображення проходить процес кодування, модуляції, додавання завади, демодуляції та декодування. Для оцінки ефективності результати представляються у вигляді візуального порівняння трьох зображень: оригінального, зашумленого та результату декодування, відображуючи при цьому відсоток пошкоджених бітів та виправлених помилок. Це дозволяє наочно простежити ступінь збереження або втрати інформації в умовах різних значень спектральної щільності шуму. На рисунку 4.4 зображено результат тестування при N – 512, K – 256, помірному значенні проєктного та канального SNR 8,0.

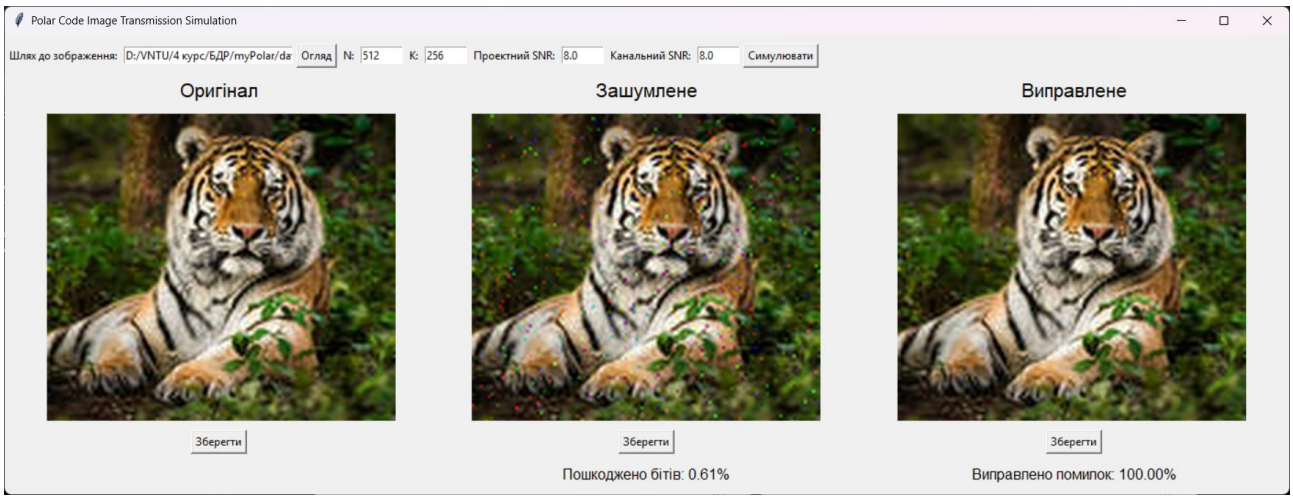


Рисунок 4.4 – Результат тестування при помірному SNR

Результат показує здатність кодування повністю відновити зображення в заданих умовах.

Наступним кроком було тестування застосування коду спроєктованого для помірного SNR – 8,0 для передачі в каналі з гіршим відношенням – 3,0. Результат зображено на рисунку 4.5.

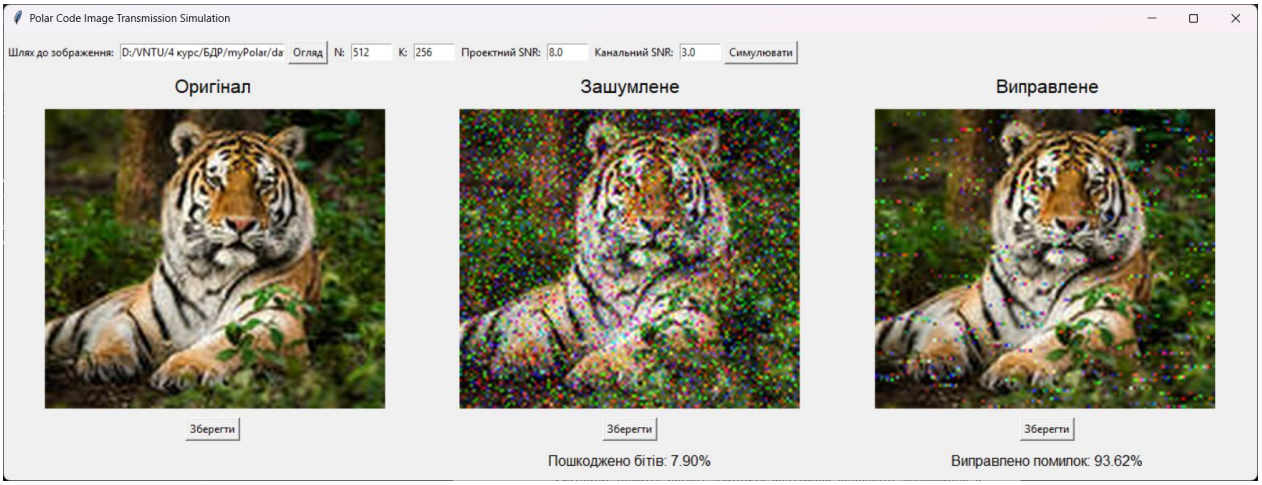


Рисунок 4.5 – Результат тестування при невідповідності проєктного та канального SNR

Результат показує погану здатність кодування відновити зображення в заданих умовах. На рисунку 4.6 зображено результат тестування при проєктуванні коду для низького SNR – 3,0.

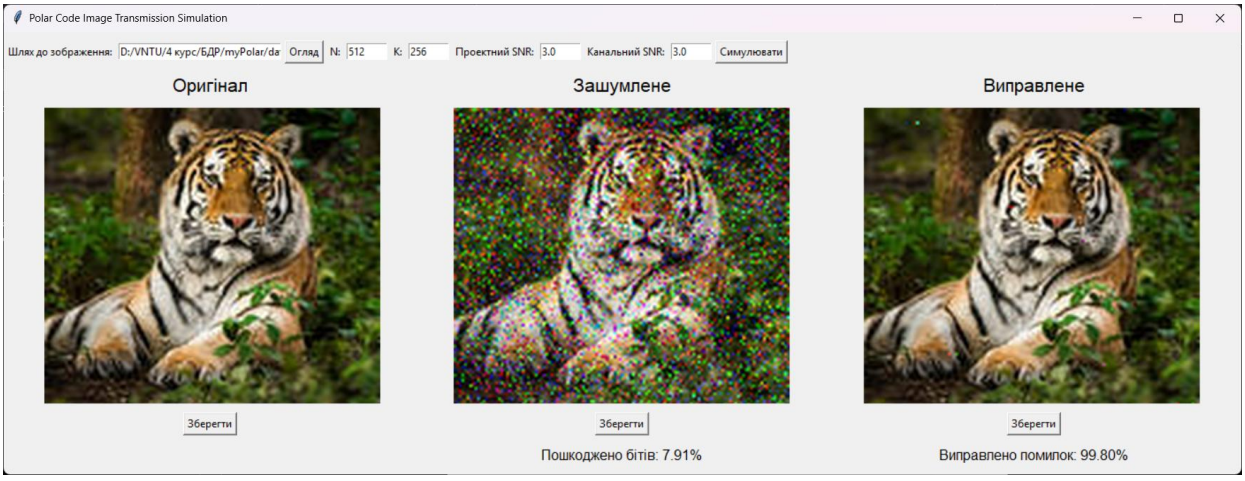


Рисунок 4.6 – Результат тестування в поганому каналі з відповідним проєктним SNR

Результат тестування показує підвищення здатності відновлення

зображення, що доводить суттєвість вибору множини заморожених бітів для коректності декодування.

Наступним кроком було проведено тестування для попередніх умов каналу зі збільшенням параметрів довжини коду. На рисунку 4.7 зображено результат тестування з N – 2048, K – 1024.

Рисунок 4.7 – Результат тестування зі збільшенням параметрів довжини

Результат тестування показує збільшення здатності відновлення зі збільшенням довжини блоку коду.

На рисунку 4.8 показано результат тестування передачі в дуже поганому каналі при SNR – 2,0.

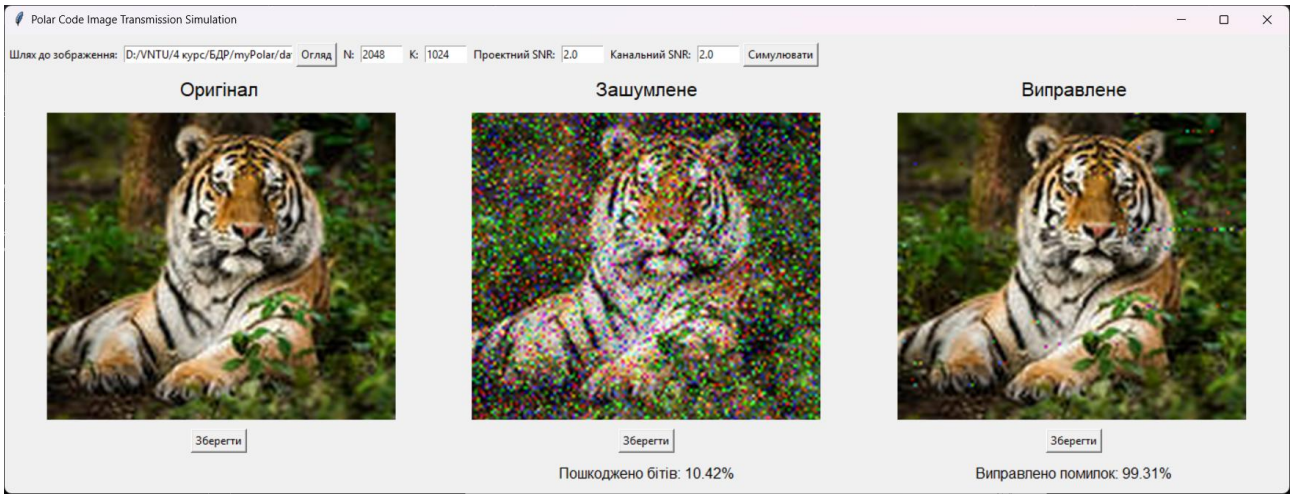


Рисунок 4.9 – Результат тестування передачі в дуже поганому каналі

Результат тестування показує не повне відновлення зображення однак виправлення 99,31% помилок при 10,42% пошкоджених бітів є досить непоганим результатом.

Також було проведено тестування для попередніх умов каналу зі збільшенням довжини зменшенням швидкості коду. На рисунку 4.9 зображено результат тестування з N – 4096, K – 1024.

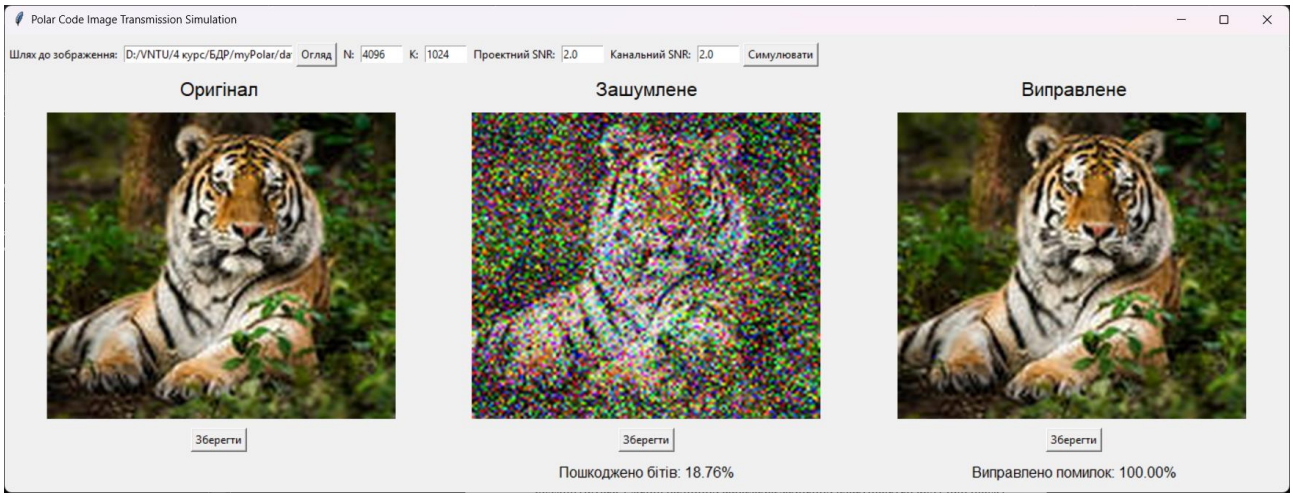


Рисунок 4.9 – Результат тестування передачі в
дуже поганому каналі при R = 1/4

Результат тестування демонструє збільшення здатності до відновлення даних зі зменшенням швидкості коду. Швидкість 1/4 є досить низькою, однак може застосовуватися у космічному зв’язку [27], де важливішою є надійність, а не швидкість передачі.

Також можна спостерігати, що при меншій швидкості частка пошкоджених бітів може зрости. Це пояснюється тим, що інформаційні біти розміщуються у менш надійних каналах,а захисних властивостей коду при цьому недостатньо для повної компенсації впливу шуму при збільшенні довжини коду.

Наступним тестом було проведено порівняння швидкодії декодування за допомогою розробленого модуля з використанням прискорення через JIT-компіляцію за допомогою бібліотеки Numba та без застосування цього прискорення. Результати експерименту наведено на рисунку 4.10

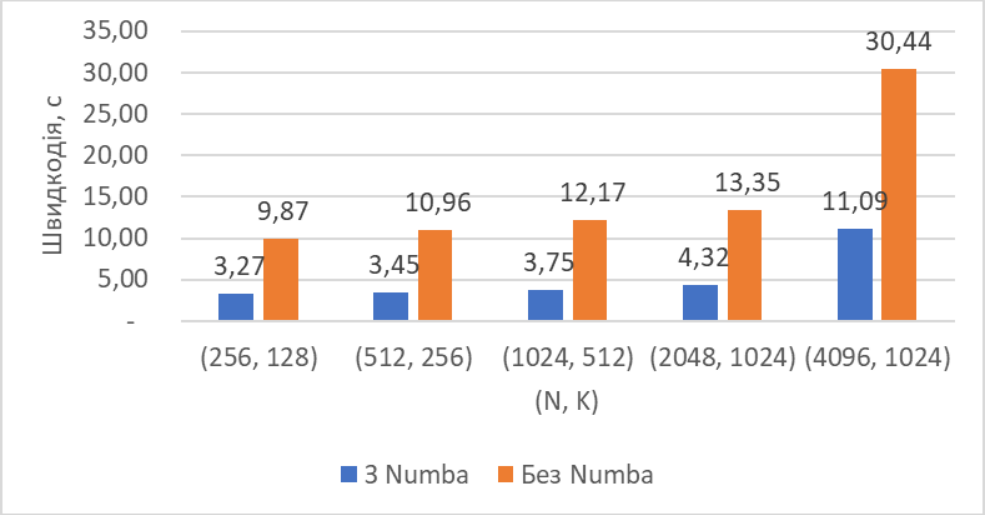


Рисунок 4.10 – Графік порівняння швидкодії

Як видно з графіку, використання засобів JIT-компіляції Numba, дозволяє досягти суттєвого пришвидшення – у середньому в 2.5–3 рази. Це значно підвищує ефективність виконання операцій декодування, особливо при обробці великих обсягів даних.

Отже, запропонований програмний модуль не лише функціонує коректно, а й демонструє високу продуктивність при застосуванні оптимізаційних засобів. Це підтверджує його придатність для дослідження продуктивності полярного кодування в різних експериментальних умовах.

4.2 Розробка інструкції користувача

Розробка програмного забезпечення полярного завадостійкого кодування передбачає створення інструкції користувача, яка є важливим компонентом для ефективної експлуатації розробленого продукту. Така інструкція покликана забезпечити користувача зрозумілою інформацією щодо основного функціоналу системи, особливостей налаштування параметрів кодування та декодування, а також послідовності дій для отримання коректного результату при передачі зображень через зашумлений канал.

При запуску програмного модуля відкривається вікно, зображене на рисунку 4.11

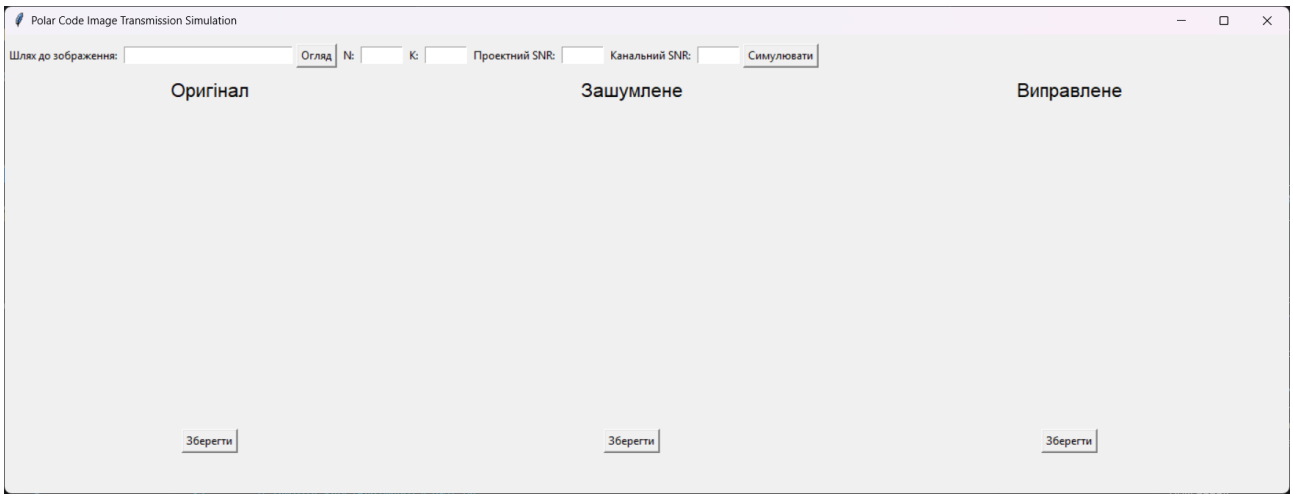


Рисунок 4.11 – Вікно програмного модуля після запуску

Для симуляції передачі зображення через зашумлений канал користувач може ввести такі параметри як:

- «Шлях до зображення»;
- «N» – довжина блоку коду;
- «K» – довжина інформаційної послідовності;
- «Проєктний SNR»;
- «Канальний SNR».

Приклад заповнення вхідних параметрів системи зображено на рисунку 4.12.

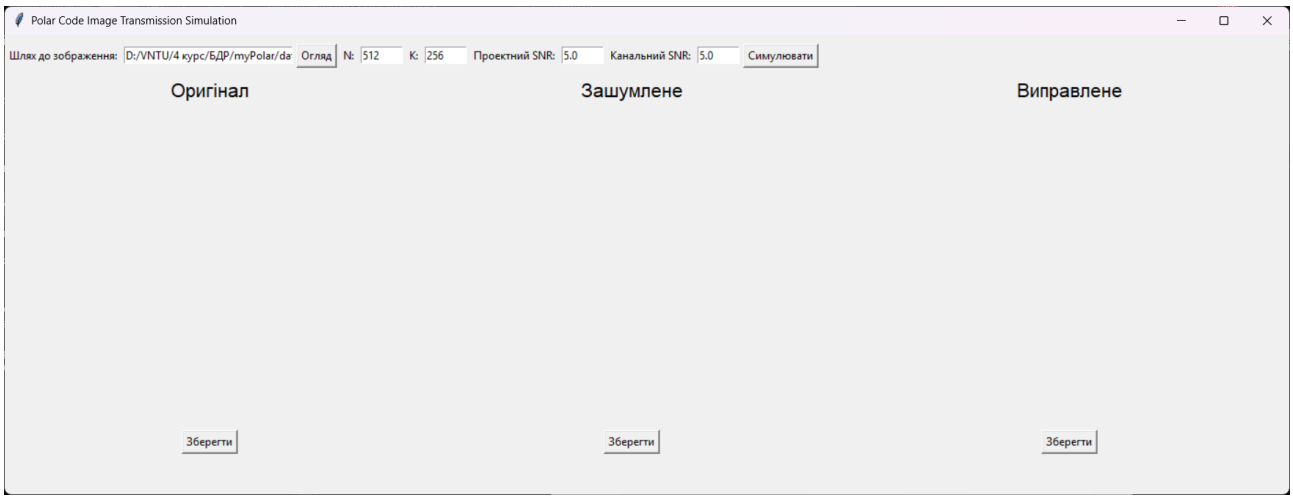


Рисунок 4.12 – Приклад заповнення вхідних параметрів системи

Після заповнення параметрів необхідно натиснути на кнопку «Симулювати». Результат симуляції з заданими параметрами зображено на рисунку 4.13.

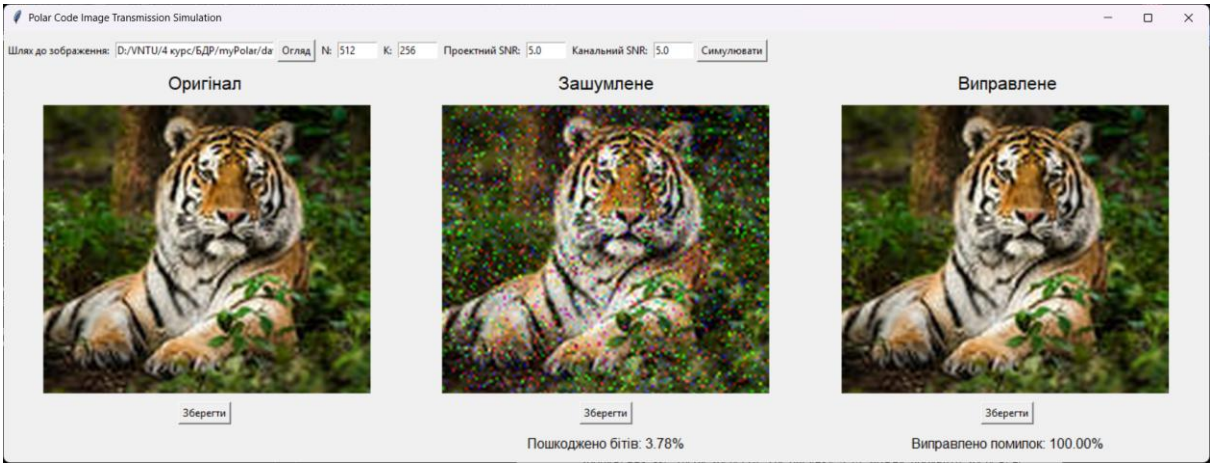


Рисунок 4.13 – Результат симуляції з заданими параметрами

Для збереження зображення потрібно натиснути кнопку «Зберегти» під зображення яке треба зберегти. На рисунку 4.14 можна побачити збережене «Зашумлене» зображення.

Рисунок 4.14 – Збережене «Зашумлене» зображення

За потреби можна змінювати параметри та повторювати симуляції до завершення роботи.

4.3 Висновки

У четвертому розділі було проведено тестування програмного модуля завадостійкого кодування даних. У рамках тестування перевірено стійкість програмного забезпечення до помилкових вхідних даних. Проведено серію тестів для різних значень SNR та довжин кодових блоків, що дозволило оцінити вплив параметрів на ефективність декодування і відновлення інформації, а також порівняти швидкодію розробленого модуля з використанням прискорення через JIT-компіляцію за допомогою бібліотеки Numba та без застосування цього прискорення. Також було розроблено інструкцію користувача по початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено програмний модуль завадостійкого кодування даних в системах передачі інформації. Було обґрунтовано доцільність застосування полярних кодів у задачах захисту інформації під час передачі через зашумлені канали зв'язку. На основі проведеного аналізу засобів реалізації обрано мову програмування Python та бібліотеки NumPy і Numba, які забезпечили баланс між обчислювальною ефективністю та гнучкістю розробки.

У роботі розв'язано такі задачі: проведено аналіз існуючих методів завадостійкого кодування та їх застосування в системах передачі інформації; досліджено особливості побудови кодів з контролем і виправленням помилок та їх математичні основи; розроблено алгоритми кодування та декодування для вибраного методу завадостійкого кодування; модифіковано метод завадостійкого кодування для підвищення швидкодії обробки даних; реалізовано програмну модель для перевірки ефективності алгоритмів у середовищі завадами; провести тестування стійкості реалізованих кодів до завад.

Подальшого розвитку отримав метод завадостійкого кодування, у якому, на відміну від існуючих, використано інструментарій JIT-компіляції бібліотеки Numba, що дало можливість компілювати фрагменти Python-коду у високопродуктивний машинний код, підвищуючи швидкодію обчислень.

У першому розділі було проаналізовано сучасний стан галузі завадостійкого кодування та програмних засобів, що забезпечують моделювання і реалізацію відповідних алгоритмів. Проведений порівняльний аналіз існуючих рішень, таких як AFF3CT, GNU Radio та 5G Toolkit, засвідчив актуальність створення спеціалізованого інструменту для роботи з полярними кодами, що дозволяє моделювати передачу зашумлених зображень та візуалізувати результати декодування.

У другому розділі було сформовано теоретичне підґрунтя для побудови системи завадостійкого кодування на основі полярних кодів. Розглянуто

механізм поляризації каналів, метод визначення заморожених бітів за допомогою меж Бхаттачар'ї, а також принципи систематичного кодування. Наведено математичну модель передачі даних через канал із адитивним білим гаусівським шумом із використанням BPSK-модуляції. Описано основи декодування методом послідовного виключення з рекурсивним оновленням логарифмічних правдоподібностей. Отримані теоретичні положення стали основою для подальшої реалізації програмного модуля.

У третьому розділі було розроблено алгоритм роботи програмного модуля завадостійкого кодування даних. На основі проведеного варіантного аналізу обґрунтовано вибір мови програмування Python та бібліотек NumPy, Numba, що дозволило досягти високої продуктивності обчислень при збереженні гнучкості реалізації. У розділі також було описано блок-схеми основних алгоритмів, а також діаграму класів, яка ілюструє взаємозв'язки між класами системи.

У четвертому розділі було проведено тестування програмного модуля завадостійкого кодування даних. У рамках тестування перевірено стійкість програмного забезпечення до помилкових вхідних даних. Проведено серію тестів для різних значень SNR та довжин кодових блоків, що дозволило оцінити вплив параметрів на ефективність декодування і відновлення інформації, а також порівняти швидкодію розробленого модуля з використанням прискорення через JIT-компіляцію за допомогою бібліотеки Numba та без застосування цього прискорення. Також було розроблено інструкцію користувача по початковій експлуатації програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shannon C. A Mathematical Theory of Communication (1948) [Електронний ресурс] / Claude Shannon // Ideas That Created the Future. – [Б. м.], 2021. – p. 121–134. – Режим доступу: https://doi.org/10.7551/mitpress/12274.003.0014

2. Атаманенко В. М. Сучасні методи завадостійкого кодування даних у системах передачі інформації [Електронний ресурс] // LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) Режим доступу https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23938

3. Chitode J. S. Information Theory and Coding: Information, Source Coding and Channel Coding / J. S. Chitode; TECHNICAL PUBLICATIONS, 2020. – 534 p

4. Майданюк, В. П. Основи теорії інформації та кодування : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / Майданюк В. П., Романюк О. Н., Тужанський С. Є. – Вінниця : ВНТУ, 2022. – 133 с. Режим доступу: https://jetiq.vntu.edu.ua/repository/card.php?lang=uk&id=5107

5. Sklar B. Digital Communications: Fundamentals and Applications. Pearson Education, Limited, 2021.

6. Івашко А. В. Теорія інформації та кодування в прикладах і задачах : навч. посібник / А. В. Івашко, В. А. Крилова ; Нац. техн. ун-т "Харків. політехн. ін-т". – Харків : НТУ "ХПІ", 2022. – 317 с.

7. Low-Density Parity-Check Codes [Електронний ресурс] // Channel Codes. – 2-ге вид. – [Б. м.], 2024. – p. 235–280. – Режим доступу: https://doi.org/10.1017/9781009335928.007

8. Turbo Codes [Електронний ресурс] // Channel Codes. – 2-ге вид. – [Б. м.], 2024. – p. 322–364. – Режим доступу: https://doi.org/10.1017/9781009335928.009

9. Polarization and Polar Coding [Електронний ресурс] // Information Theoretic Perspectives on 5G Systems and Beyond. – [Б. м.], 2022. – p. 267–298. – Режим доступу: https://doi.org/10.1017/9781108241267.008

10. AFF3CT [Електронний ресурс]. – Режим доступу https://aff3ct.github.io/index.html

11. GNU Radio [Електронний ресурс]. – Режим доступу: https://www.gnuradio.org/

12. 5G Toolkit [Електронний ресурс]. – Режим доступу: https://gigayasa.com/5g-toolkit/

13. Fundamentals of Classical and Modern Error-Correcting Codes. – [S. l.] : University of Cambridge ESOL Examinations, 2021. – 840 p

14. Moon T. K. Error Correction Coding: Mathematical Methods and Algorithms / Todd K. Moon. – [S. l.] : Wiley & Sons, Incorporated, John, 2020. – 1008 p.

15. Pilanci M. Computational Polarization: An Information-Theoretic Method for Resilient Computing [Електронний ресурс] / Mert Pilanci // IEEE Transactions on Information Theory. – 2022. – Т. 68, № 4. – С. 2211–2238. – Режим доступу: https://doi.org/10.1109/tit.2021.3139009

16. Аналіз завадостійкості захищеної системи зв'язку 5g з полярним кодуванням [Електронний ресурс] / Ілля Пятін [та ін.] // Measuring and computing devices in technological processes. – 2023. – № 3. – С. 154–163. – Режим доступу: https://doi.org/10.31891/2219-9365-2023-75-18

17. Конспект лекцій з дисципліни «Теорія передавання інформації» для здобувачів вищої освіти другого (магістерського) рівня зі спеціальності - 172 «Електронні комунікації та радіотехніка». /Укл.: Литвиненко В.А., - Кам’янське; ДДТУ, 2024 р. – 77 с.

18. C++ Programming language [Електронний ресурс]. – Режим доступу https://www.geeksforgeeks.org/c-plus-plus/

19. Java [Електронний ресурс]. – Режим доступу: https://www.java.com/en/

20. Matlab [Електронний ресурс]. – Режим доступу: https://www.mathworks.com/products/matlab.html

21. Python [Електронний ресурс]. — Режим доступу: https://www.python.org/

22. NumPy [Електронний ресурс]. — Режим доступу: https://numpy.org/

23. Numba [Електронний ресурс]. — Режим доступу: https://numba.pydata.org/

24. Що таке тестування програмного забезпечення? [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/shho-taketestuvannya-programnogo-zabezpechennya/

25. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://mate.academy/blog/qa/functional-non-functional-testing

26. White/black/grey box-тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/

27. CCSDS 131.0-B-5. TM SYNCHRONIZATION AND CHANNEL CODING [Електронний ресурс], 2023. – 100 р. – Режим доступу: https://ccsds.org/Pubs/131x0b5.pdf#page=58&zoom=100,208,129

ДОДАТКИ

Додаток А – Технічне завдання

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., проф. О. Н. Романюк

"25" березня 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу

«Розробка програмного модуля завадостійкого кодування даних у системах передачі інформації»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:

к.т.н., доц. каф. ПЗ Рейда О.М.

"24" березня 2025 р.

Виконав: студент гр. 2ПІ-21Б

Атаманенко В. М.

"24" березня 2025 р.

Вінниця - 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: Розробка програмного модуля завадостійкого кодування даних у системах передачі інформації.

Галузь застосування – системи передачі інформації.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від « 18 » лютого 2025 р.

3. Мета та призначення розробки.

Метою дослідження є розробка програмного модуля завадостійкого кодування даних, який дозволить ефективно використовувати обчислювальні ресурси та зменшити час обробки інформації.

Призначення роботи – розробка програмного модуля завадостійкого кодування даних у системах передачі інформації.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Майданюк, В. П. Основи теорії інформації та кодування : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / Майданюк В. П., Романюк О. Н., Тужанський С. Є. – Вінниця : ВНТУ, 2022. – 133 с. Режим доступу: https://jetiq.vntu.edu.ua/repository/card.php?lang=uk&id=5107

2. Moon T. K. Error Correction Coding: Mathematical Methods and Algorithms / Todd K. Moon. – [S. l.] : Wiley & Sons, Incorporated, John, 2020. – 1008 p.

5. Технічні вимоги

Тип програми – комп’ютеринй застосунок; модель розробки каскадна; вхідні дані: параметри коду та повідомлення; вихідні дані: закодовані та декодовані повідомлення; засоби розробки – MS Visual Studio Code; мова програмування – Python.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. пояснювальна записка до БКР;
2. технічне завдання;
3. лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз методів завадостійкого кодування даних	25.03.25 – 05.04.25
2	Розробка методів для завадостійкого кодування	05.04.25 – 10.04.25
3	Розробка моделі та алгоритму застосунку	10.04.25 – 28.04.25
4	Розробка програмного модуля завадостійкого кодування	28.04.25 – 16.05.25
5	Тестування застосунку	16.05.25 – 25.05.25
6	Оформлення матеріалів до захисту БКР	25.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного модуля завадостійкого кодування даних у системах передачі інформації

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,2 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Рейда О. М., к. т. н., доцент кафедри ПЗ
(прізвище, ініціали, посада)

Здобувач (підпис) Атаманенко В. М.
(прізвище, ініціали)

Додаток В – Лістинг програмного забезпечення

Лістинг файлу PolarCode.py

```
import numpy as np
from utils import *
import json

class PolarCode:
    def __init__(self, N, K, design_SNR):

        self.N = N
        self.n = int(np.log2(self.N))
        self.F = arikan_gen(self.n)
        self.K = K
        self.reliabilities = np.array([])
        self.frozen = np.array([])
        self.frozen_lookup = np.array([])
        self.reliabilities, self.frozen = self.bb_pcc(self, -
self.get_normalised_SNR(design_SNR))
        self.frozen_lookup = self.get_lut(self.frozen)
        self.T = np.eye(self.N, dtype=int)
        A = inverse_set(self.frozen, self.N)
        self.T[A, :] = self.F[A, :]



    def __str__(self):
        output = '=' * 10 + " Polar Code " + '=' * 10 + '\n'
        output += "N: " + str(self.N) + '\n'
        output += "M: " + str(self.M) + '\n'
        output += "K: "+ str(self.K) + '\n'
        output += "Ordered Bits (least reliable to most reliable): " + str(self.reliabilities)
+ '\n'
        output += "Frozen Bits: " + str(self.frozen) + '\n'
        return output

    def get_normalised_SNR(self, design_SNR):
        Eb_No_dB = design_SNR
        Eb_No = 10 ** (Eb_No_dB / 10)
        Eb_No = Eb_No * (self.K / self.N)
        return Eb_No
```

```
    def get_lut(self, my_set):
        my_lut = np.ones(self.N, dtype=int)
        my_lut[my_set] = 0
        return my_lut

    def bb_pcc(self, myPC, z0):
        n = myPC.n
        z = np.zeros((myPC.N, n + 1))
        z[:, 0] = z0

        for j in range(1, n + 1):
            u = 2 ** j
            for t in range(0, myPC.N, u):
                for s in range(int(u / 2)):
                    k = t + s
                    z_top = z[k, j - 1]
                    z_bottom = z[k + int(u / 2), j - 1]
                    z[k, j] = logdomain_diff(logdomain_sum(z_top, z_bottom), z_top +
z_bottom)
                    z[k + int(u / 2), j] = z_top + z_bottom

        reliabilities = np.argsort(-z[:, n], kind='mergesort')
        frozen = np.argsort(z[:, n], kind='mergesort')[myPC.K:]
        return reliabilities, frozen
```

Лістинг файлу Encoder.py

```
import numpy as np
from PolarCode import PolarCode
from utils import *

class Encoder:
    def __init__(self, myPC):
        self.myPC = myPC

    def encode(self, message):
        frozen_message = np.zeros(self.myPC.N, dtype=int)
        frozen_message[self.myPC.frozen_lookup == 1] = message
        frozen_message = np.array([frozen_message])
        v = np.mod(np.dot(self.myPC.T, frozen_message.T), 2)
        encoded = np.transpose(np.mod(np.dot(self.myPC.F, v), 2))[0]
```

```
        return encoded

    def encode_batch(self, bits):
        bits, padding_length = pad_bits(bits, self.myPC.K)

        bits_reshaped = bits.reshape(-1, self.myPC.K)
        codewords = []
        for message in bits_reshaped:
            codeword = self.encode(message)
            codewords.append(codeword)
        codewords = np.array(codewords).reshape(-1, self.myPC.N)
        return codewords, padding_length
```

Лістинг файлу Decoder.py

```
import numpy as np
from utils import *
from decoder_utils import *

class Decoder:
    def __init__(self, myPC):
        self.myPC = myPC

    def unfreeze_message(self, message):
        message = self.systematic_decode(message)
        return message[self.myPC.frozen_lookup == 1]

    def decode(self, likelihoods):
        decoded = self.scd_decode(likelihoods)
        return self.unfreeze_message(decoded)

    def decode_batch(self, likelihoods, padding_length):
        recovered_bits = []
        for likelihoods in likelihoods:
            message_received = self.decode(likelihoods)
            recovered_bits.append(message_received)
        recovered_bits = np.hstack(recovered_bits)

        recovered_bits = remove_padding(recovered_bits, padding_length)
        return recovered_bits

    def systematic_decode(self, message):
```

```
        x = np.array([message], dtype=int)
        return np.transpose(np.mod(np.dot(self.myPC.T, x.T), 2))[0]

    def scd_decode(self, likelihoods):

        self.L = np.full((self.myPC.N, self.myPC.n + 1), np.nan, dtype=np.float64)
        self.B = np.full((self.myPC.N, self.myPC.n + 1), np.nan)
        self.L[:, 0] = likelihoods

        for l in [bit_reversed(i, self.myPC.n) for i in range(self.myPC.N)]:
            update_llrs(self.myPC.N, self.myPC.n, self.L, self.B, l)

            if l in self.myPC.frozen:
                self.B[l, self.myPC.n] = 0
            else:
                self.B[l, self.myPC.n] = hard_decision(self.L[l, self.myPC.n])

            update_bits(self.myPC.N, self.myPC.n, self.B, l)
        return self.B[:, self.myPC.n].astype(int)
```

Лістинг файлу Channel.py

```
import matplotlib.pyplot as plt
import numpy as np
from utils import remove_padding

class Channel:
    def __init__(self, myPC, Eb_No):
        self.myPC = myPC
        self.Es = myPC.get_normalised_SNR(Eb_No)
        self.No = 1

    def transmit(self, codeword):
        tx = self.modulation(codeword)
        rx = tx + self.noise(self.myPC.N)
        likelihoods = np.array(self.get_likelihoods(rx), dtype=np.float64)
        rx = self.demodulate(rx)
        return rx, likelihoods

    def transmit_batch(self, codewords, padding_length):
        noisy_codewords_likelihoods = []
        noisy_codewords = []
```

```
        for codeword in codewords:
            rx, likelihoods = self.transmit(codeword)
            noisy_codewords_likelihoods.append(likelihoods)
            rx = self.get_noisy_message(rx)
            noisy_codewords.append(rx)
        noisy_codewords_likelihoods = np.array(noisy_codewords_likelihoods)
        noisy_codewords = np.hstack(noisy_codewords)

        noisy_codewords = remove_padding(noisy_codewords, padding_length)

        return noisy_codewords_likelihoods, noisy_codewords

    def LLR(self, y):
      return -2 * y * np.sqrt(self.Es) / self.No

    def get_likelihoods(self, y):
        return [self.LLR(y[i]) for i in range(len(y))]

    def modulation(self, x):
        return 2 * (x - 0.5) * np.sqrt(self.Es)

    def noise(self, N):
        s = np.random.normal(0, np.sqrt(self.No / 2), size=N)
        return s

    def demodulate(self, rx):
        return (rx > 0).astype(np.uint8)

    def get_noisy_message(self, message):
      return message[self.myPC.frozen_lookup == 1]
```

Лістинг файлу decoder_utils.py

```
import numpy as np
from utils import *
from numba import njit

def hard_decision(y):
    if y >= 0:
        return 0
    else:
        return 1
```

```
@njit(cache=True)
def upper_llr(l1, l2):
    return logdomain_sum(l1 + l2, 0) - logdomain_sum(l1, l2)

@njit(cache=True)
def lower_llr(l1, l2, b):
    if b == 0:
        return l1 + l2
    elif b == 1:
        return l1 - l2
    return np.nan

@njit(cache=True)
def active_llr_level(i, n):
    mask = 2**(n-1)
    count = 1
    for k in range(n):
        if (mask & i) == 0:
            count += 1
            mask >>= 1
        else:
            break
    return min(count, n)

@njit(cache=True)
def active_bit_level(i, n):
    mask = 2**(n-1)
    count = 1
    for k in range(n):
        if (mask & i) > 0:
            count += 1
            mask >>= 1
        else:
            break
    return min(count, n)

@njit(cache=True)
def update_llrs(N, n, L, B, l):
        for s in range(n - active_llr_level(l, n), n):
            block_size = int(2 ** (s + 1))
```

```
            branch_size = int(block_size / 2)
            for j in range(l, N, block_size):
                if j % block_size < branch_size:
                    top_llr = L[j, s]
                    btm_llr = L[j + branch_size, s]
                    L[j, s + 1] = upper_llr(top_llr, btm_llr)
                else:
                    btm_llr = L[j, s]
                    top_llr = L[j - branch_size, s]
                    top_bit = B[j - branch_size, s + 1]
                    L[j, s + 1] = lower_llr(btm_llr, top_llr, top_bit)

@njit(cache=True)
def update_bits(N, n, B, l):
    if l < N / 2:
        return

    for s in range(n, n - active_bit_level(l, n), -1):
        block_size = int(2 ** s)
        branch_size = int(block_size / 2)
        for j in range(l, -1, -block_size):
            if j % block_size >= branch_size:
                B[j - branch_size, s - 1] = int(B[j, s]) ^ int(B[j - branch_size, s])
                B[j, s - 1] = B[j, s]
```

Лістинг файлу file_utils.py

```
import numpy as np
from pathlib import Path
from PIL import Image

def read_image_as_bits(input_path):
    img = Image.open(input_path).convert('RGB')
    img_data = np.array(img)
    original_shape = img_data.shape

    flat_img = img_data.flatten()
    bits = np.unpackbits(flat_img)
    return bits, original_shape

def write_bits_as_image(bits, shape, output_path = None):
    n_bits = len(bits)
    padding_length = (-n_bits) % 8
```

```
    if padding_length > 0:
        bits = np.concatenate([bits, np.zeros(padding_length, dtype=np.uint8)])

    bytes_arr = np.packbits(bits)
    expected_bytes = np.prod(shape)
    if len(bytes_arr) < expected_bytes:
        raise ValueError(f"Недостатньо байтів ({len(bytes_arr)}) щоб виконати
перетворення {shape}")
    bytes_arr = bytes_arr[:expected_bytes]

    img_data = bytes_arr.reshape(shape)
    img = Image.fromarray(img_data.astype(np.uint8))
    if output_path is not None:
        img.save(output_path)
    return img
```

Лістинг файлу utils.py

```
import numpy as np
from numba import njit

@njit(cache=True)
def bit_reversed(x, n):
    result = 0
    for i in range(n):
        if (x & (1 << i)):
            result |= (1 << (n - 1 - i))
    return result

@njit(cache=True)
def logdomain_diff(x, y):
    if x > y:
        z = x + np.log1p(-np.exp(y - x))
    else:
        z = y + np.log1p(-np.exp(x - y))
    return z

@njit(cache=True)
def logdomain_sum(x, y):
    if x > y:
        z = x + np.log1p(np.exp(y - x))
    else:
```

```
        z = y + np.log1p(np.exp(x - y))
    return z

@njit(cache=True)
def inverse_set(F, N):
    n = int(np.log2(N))
    not_F = []
    for i in range(N):
        if i not in F:
            not_F.append(i)
    return np.array(not_F)

@njit(cache=True)
def arikan_gen(n):
    F = np.array([[1, 1], [0, 1]])
    F_n = F
    for i in range(n - 1):
        F_n = np.kron(F, F_n)
    return F_n

def pad_bits(bits, K):
    padding_length = (-len(bits)) % K
    if padding_length > 0:
        bits = np.concatenate([bits, np.zeros(padding_length, dtype=np.uint8)])
    return bits, padding_length

def remove_padding(bits, padding_length):
    if padding_length > 0:
        bits = bits[:-padding_length]
    return bits

def get_recovery_stats(bits, noisy_codewords, recovered_bits):
    total_bits = len(bits)
    noisy_bits = noisy_codewords[:total_bits].flatten()
    recovered_bits_flat = recovered_bits[:total_bits].flatten()
    bits_flat = bits.flatten()

    corrupted = (bits_flat != noisy_bits).sum()
    percent_corrupted = corrupted / total_bits * 100

    corrected = ((bits_flat != noisy_bits) & (bits_flat == recovered_bits_flat)).sum()
```

```
    percent_corrected = (corrected / corrupted * 100) if corrupted > 0 else 0.0

    return percent_corrupted, percent_corrected
```

Лістинг файлу GUI.py

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
from file_utils import read_image_as_bits, write_bits_as_image
from utils import get_recovery_stats
from PolarCode import PolarCode
from Channel import Channel
from Encoder import Encoder
from Decoder import Decoder

def process_image(input_path, N, K, design_SNR, channel_SNR):
    bits, img_shape = read_image_as_bits(input_path)

    myPC = PolarCode(N, K, design_SNR)
    encoder = Encoder(myPC)
    decoder = Decoder(myPC)
    noisy_channel = Channel(myPC,  channel_SNR)

    codewords, padding_length = encoder.encode_batch(bits)

    noisy_codewords_likelihoods, noisy_codewords =
noisy_channel.transmit_batch(codewords, padding_length)

    recovered_bits = decoder.decode_batch(noisy_codewords_likelihoods,
padding_length)

    percent_corrupted, percent_corrected = get_recovery_stats(bits, noisy_codewords,
recovered_bits)

    noisy_img = write_bits_as_image(noisy_codewords, img_shape)
    decoded_img = write_bits_as_image(recovered_bits, img_shape)
    original_img = Image.open(input_path).convert('RGB')
    return original_img, noisy_img, decoded_img, percent_corrupted,
percent_corrected

class GUI:
```

```
    def __init__(self, root):
        self.root = root
        self.root.title("Polar Code Image Transmission Simulation")


        self.root.geometry("1400x500")

        input_frame = tk.Frame(root)
        input_frame.grid(row=0, column=0, columnspan=3, pady=10, sticky="w")

        tk.Label(input_frame, text="Шлях до зображення:").pack(side=tk.LEFT,
padx=2)
        self.path_entry = tk.Entry(input_frame, width=30)
        self.path_entry.pack(side=tk.LEFT, padx=2)
        tk.Button(input_frame, text="Огляд",
command=self.browse_file).pack(side=tk.LEFT, padx=2)

        tk.Label(input_frame, text="N:").pack(side=tk.LEFT, padx=2)
        self.n_entry = tk.Entry(input_frame, width=7)
        self.n_entry.pack(side=tk.LEFT, padx=2)

        tk.Label(input_frame, text="K:").pack(side=tk.LEFT, padx=2)
        self.k_entry = tk.Entry(input_frame, width=7)
        self.k_entry.pack(side=tk.LEFT, padx=2)

        tk.Label(input_frame, text="Проектний SNR:").pack(side=tk.LEFT, padx=2)
        self.snr_entry = tk.Entry(input_frame, width=7)
        self.snr_entry.pack(side=tk.LEFT, padx=2)

        tk.Label(input_frame, text="Канальний SNR:").pack(side=tk.LEFT, padx=2)
        self.channel_snr_entry = tk.Entry(input_frame, width=7)
        self.channel_snr_entry.pack(side=tk.LEFT, padx=2)

        tk.Button(input_frame, text="Симулювати",
command=self.process_and_show).pack(side=tk.LEFT, padx=2)

        self.img_labels = []
        self.save_buttons = []
        for i, title in enumerate(["Оригінал", "Зашумлене", "Виправлене"]):
            tk.Label(root, text=title, font=("Arial", 16)).grid(row=1, column=i)
            lbl = tk.Label(root)
```

```
            lbl.grid(row=2, column=i, padx=10, pady=10, sticky="nsew")
            self.img_labels.append(lbl)
            root.grid_columnconfigure(i, weight=1)

            btn = tk.Button(root, text="Зберегти", command=lambda idx=i:
self.save_image(idx))
            btn.grid(row=3, column=i, pady=(0, 10))
            self.save_buttons.append(btn)

        self.stats_labels = []
        for i in range(3):
            stat_lbl = tk.Label(root, text="", font=("Arial", 12))
            stat_lbl.grid(row=4, column=i, pady=(0, 10))
            self.stats_labels.append(stat_lbl)

        root.grid_rowconfigure(2, weight=1)

        self.current_imgs = None
        self.current_stats = (0.0, 0.0)

        self.root.bind("<Configure>", self.on_resize)

    def browse_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Image files",
"*.png;*.jpg;*.jpeg;*.bmp")])
        if file_path:
            self.path_entry.delete(0, tk.END)
            self.path_entry.insert(0, file_path)

    def save_image(self, idx):
        if not self.current_imgs or idx >= len(self.current_imgs):
            return
        img = self.current_imgs[idx]
        filetypes = [
            ("PNG files", "*.png"),
            ("JPEG files", "*.jpg;*.jpeg"),
            ("BMP files", "*.bmp"),
            ("All files", "*.*")
        ]
        default_names = ["original.png", "noisy.png", "decoded.png"]
        file_path = filedialog.asksaveasfilename(
```

```
            defaultextension=".png",
            filetypes=filetypes,
            initialfile=default_names[idx]
        )
        if file_path:
            try:
                img.save(file_path)
                messagebox.showinfo("Збережено", f"Зображення збережено в
{file_path}")
            except Exception as e:
                messagebox.showerror("Помилка збереження", str(e))

    def process_and_show(self):
        path = self.path_entry.get()
        try:
            N = int(self.n_entry.get())
            K = int(self.k_entry.get())
            design_SNR = float(self.snr_entry.get())
            channel_SNR = float(self.channel_snr_entry.get())

            if N <= K:
                messagebox.showerror("Помилка вводу", "N повинно бути більше за
K.")
                return
            if N & (N - 1) != 0 or N <= 0:
                messagebox.showerror("Помилка вводу", "N повинно бути степенем
двійки (2, 4, 8, ...).")
                return
            if design_SNR <= 0 or channel_SNR <= 0:
                messagebox.showerror("Помилка вводу", " SNR повинні бути більші за
0.")
                return
        except Exception:
            messagebox.showerror("Помилка вводу", "Будь ласка введіть коректні
значення.")
            return

        try:
            original_img, noisy_img, decoded_img, percent_corrupted, percent_corrected
= process_image(path, N, K, design_SNR, channel_SNR)
        except Exception as e:
```

```
            messagebox.showerror("Помилка симуляції", str(e))
            return

        self.current_imgs = [original_img, noisy_img, decoded_img]
        self.current_stats = (percent_corrupted, percent_corrected)
        self.display_images()

    def display_images(self):
        if not self.current_imgs:
            return

        win_width = self.root.winfo_width()
        win_height = self.root.winfo_height()
        min_margin = 10
        n_images = 3

        cell_width = (win_width - (n_images + 1) * min_margin) // n_images
        cell_height = win_height - 120

        tk_imgs = []
        for img in self.current_imgs:
            img = img.copy()
            img_ratio = img.width / img.height
            cell_ratio = cell_width / cell_height

            if img_ratio > cell_ratio:

                new_width = cell_width
                new_height = int(cell_width / img_ratio)
            else:

                new_height = cell_height
                new_width = int(cell_height * img_ratio)

            img = img.resize((new_width, new_height), Image.LANCZOS)
            tk_imgs.append(ImageTk.PhotoImage(img))

        for i, (lbl, tk_img) in enumerate(zip(self.img_labels, tk_imgs)):
            lbl.configure(image=tk_img)
            lbl.image = tk_img
            if i == 0:
```

```
                padx = (min_margin, min_margin // 2)
            elif i == n_images - 1:
                padx = (min_margin // 2, min_margin)
            else:
                padx = (min_margin // 2, min_margin // 2)
            lbl.grid_configure(padx=padx, pady=10, sticky="n")

        percent_corrupted, percent_corrected = self.current_stats
        self.stats_labels[0].configure(text="")
        self.stats_labels[1].configure(
            text=f"Пошкоджено бітів: {percent_corrupted:.2f}%"
        )
        self.stats_labels[2].configure(
            text=f"Виправлено помилок: {percent_corrected:.2f}%"
        )

    def on_resize(self, event):
        if self.current_imgs:
            self.display_images()

if __name__ == "__main__":
    root = tk.Tk()
    app = GUI(root)
    root.mainloop()
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ЗАВАДОСТІЙКОГО КОДУВАННЯ ДАНИХ У СИСТЕМАХ ПЕРЕДАЧІ ІНФОРМАЦІЇ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

"Розробка програмного модуля завадостійкого кодування даних у системах передачі інформації"

СТУДЕНТ: ст. гр. 2ПІ-21Б
Атаманенко В. М.
КЕРІВНИК: к.т.н., доцент
Рейда О.М.

Вінниця 2025

Рисунок Г.1 – Титульний слайд

Актуальність

Останнім часом зростає потреба в ефективних і надійних системах цифрової передачі та зберігання даних, що пов'язано з бурхливим розвитком сучасних інформаційно-комунікаційних технологій. Цей попит значно зріс у зв'язку з розгортанням масштабних та високошвидкісних мереж нового покоління, які забезпечують обмін, обробку, передавання та довготривале зберігання великих обсягів цифрової інформації.

Ефективне програмне впровадження завадостійкого кодування залишається складною задачею. Основним викликом є зниження часу обробки, оскільки алгоритми декодування часто використовують рекурсію чи велику кількість ітерацій. Таким чином розробка програмних засобів завадостійкого кодування є актуальною задачею.

2

Рисунок Г.2 – Актуальність

Мета і завдання дослідження

Метою дослідження є підвищення ефективності використання обчислювальних ресурсів та зменшення часу обробки інформації за рахунок розробки програмного модуля завадостійкого кодування.

Об'єкт дослідження – процес розробки програмного модуля завадостійкого кодування даних у системах передачі інформації.

Предмет дослідження – методи та засоби розробки програмного модуля завадостійкого кодування даних у системах передачі інформації.

3

Рисунок Г.3 – Мета, завдання, предмет, і об’єкт дослідження

Задачі

1. провести аналіз існуючих методів завадостійкого кодування та їх застосування в системах передачі інформації;
2. дослідити особливості побудови кодів з контролем і виправленням помилок та їх математичні основи;
3. розробити алгоритми кодування та декодування для вибраного методу завадостійкого кодування;
4. модифікувати метод завадостійкого кодування для підвищення швидкодії обробки даних;
5. реалізувати програмну модель для перевірки ефективності алгоритмів у середовищі з завадами;
6. провести тестування стійкості кодування до завад.

4

Рисунок Г.4 – Задачі

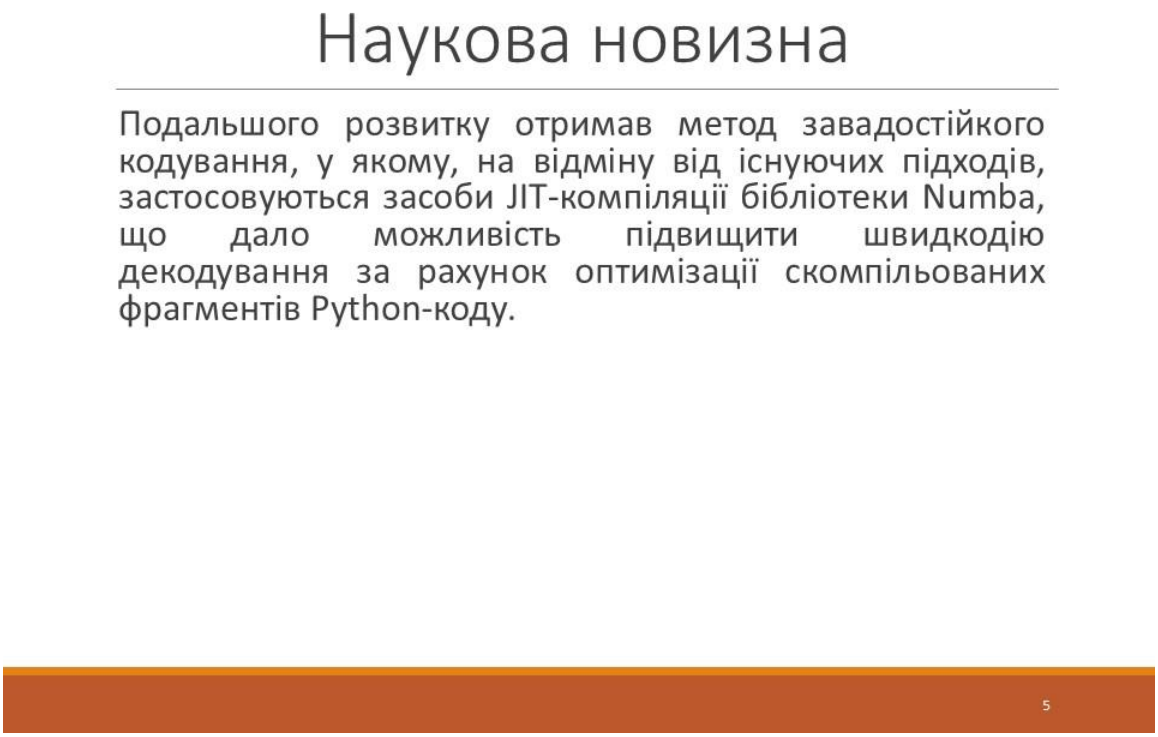


Рисунок Г.5 – Наукова новизна

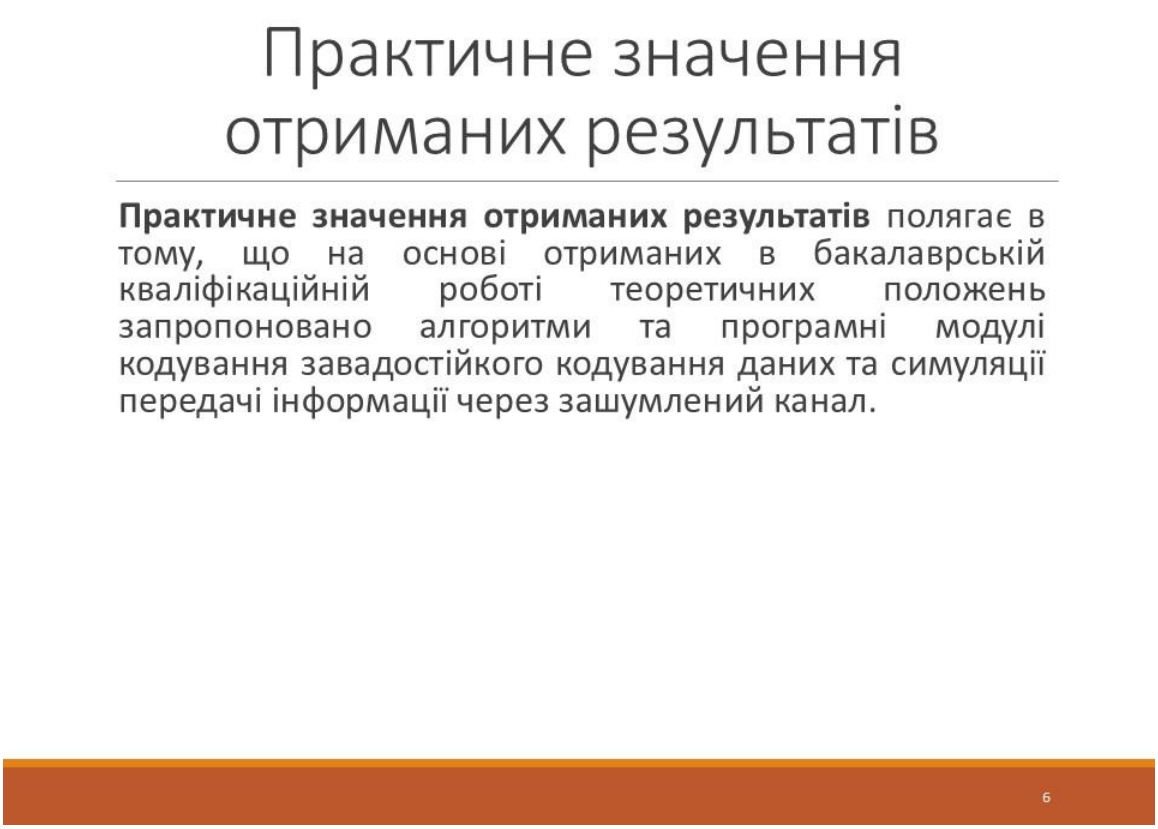


Рисунок Г.6 – Практичне значення отриманих результатів

Аналоги

AFF3CT [10] – це програмний інструментарій, призначений для роботи з методами прямої корекції помилок. Він реалізований мовою C++ та підтримує широкий спектр кодів: від широко застосовуваних турбокодів до новітніх полярних кодів, включаючи коди з низькою щільністю перевірок на парність.

GNU Radio [11] – це вільне програмне середовище з відкритим вихідним кодом, призначене для створення програмно-визначених радіосистем (SDR). Воно забезпечує гнучку платформу для моделювання, тестування та реалізації цифрових систем обробки сигналів у реальному часі. GNU Radio активно використовується в наукових дослідженнях, розробці прототипів телекомунікаційних систем, а також в освітніх проєктах.

5G Toolkit [12] є потужним інструментом для моделювання та симуляції систем мобільного зв'язку п'ятого покоління відповідно до стандарту 3GPP 5G NR. Платформа реалізована мовою Python та орієнтована на наукові дослідження, прототипування та освітні потреби. Основна увага в архітектурі інструмента приділяється гнучкій реалізації всіх етапів фізичного рівня, включаючи передавач, канал та приймач. Особливу роль у структурі займає підтримка сучасних методів завадостійкого кодування, зокрема полярних кодів та LDPC-кодів.

7

Рисунок Г.7 – Аналоги

Порівняльний аналіз аналогів

Критерії оцінювання	AFF3CT	GNU Radio	5G Toolkit	Власна розробка
Безкоштовна ліцензія	1	1	0	1
Реалізація систематичного кодера	1	1	0	1
Відображення зашумленого зображення під час симуляції	0	0	0	1
Графічний інтерфейс симулятора	1	1	0	1
Застосування засобів оптимізації обчислень	1	0	1	1
Загальна оцінка	4	3	1	5

8

Рисунок Г.8 – Порівняльний аналіз аналогів

Методи ти засоби розробки

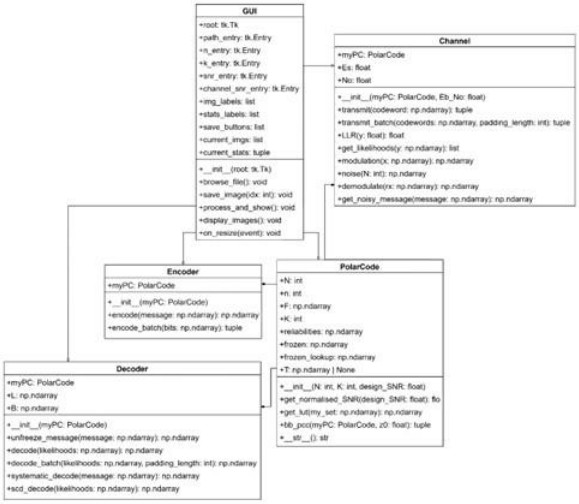
Python виявився найкращим компромісом між швидкістю розробки, зручністю налагодження та обчислювальною ефективністю. Завдяки своїй простій та виразній синтаксичній структурі Python значно пришвидшує розробку ітеративних прототипів та реалізацію складних математичних моделей. Його популярність у галузі машинного навчання, цифрової обробки сигналів та наукових досліджень підтверджується широким спектром бібліотек і готових рішень [21].

Основою обчислювального ядра реалізованого модуля стала бібліотека NumPy [22], що забезпечує ефективну роботу з багатовимірними масивами. Для прискорення критичних ділянок коду було застосовано компілятор Numba [23], що дозволив компілювати окремі функції у високопродуктивний машинний код з використанням Just-In-Time підходу. Завдяки поєднанню Python, NumPy і Numba вдалося досягти рівня продуктивності, порівнянного з рішеннями на більш низькорівневих мовах, при цьому зберігаючи зручність реалізації та можливість швидкої адаптації алгоритмів.

9

Рисунок Г.9 – Методи та засоби розробки

Діаграма класів



10

Рисунок Г.10 – Діаграма класів

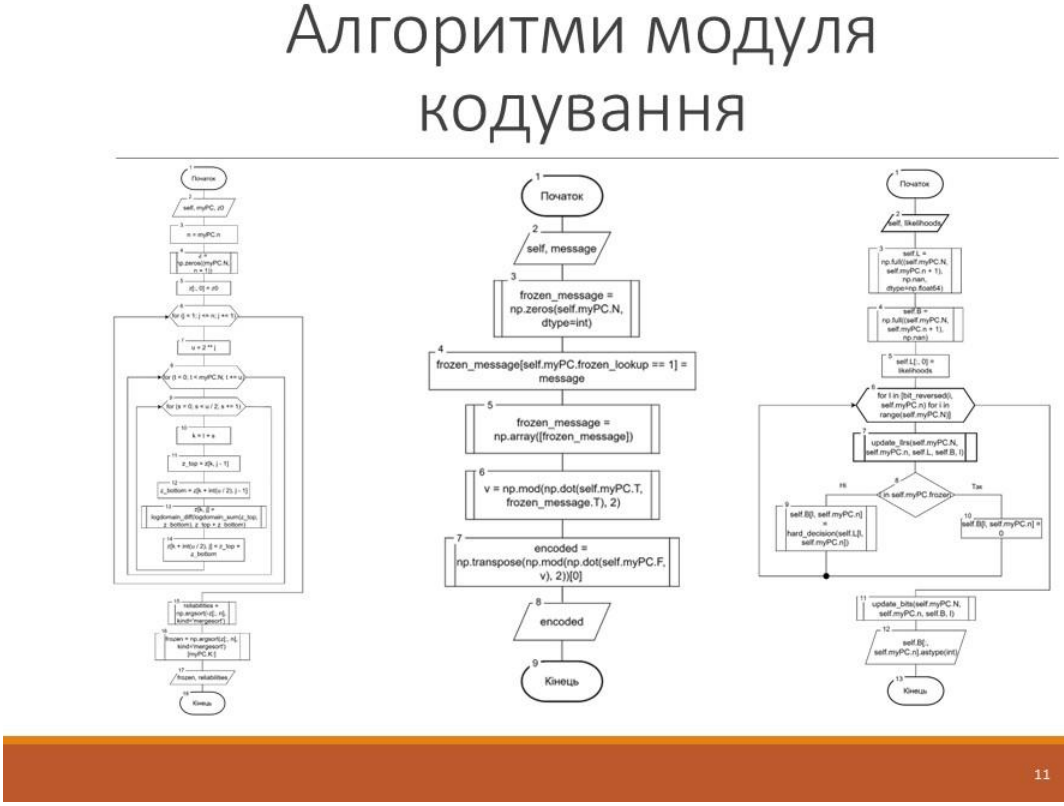


Рисунок Г.11 – Алгоритми кодування

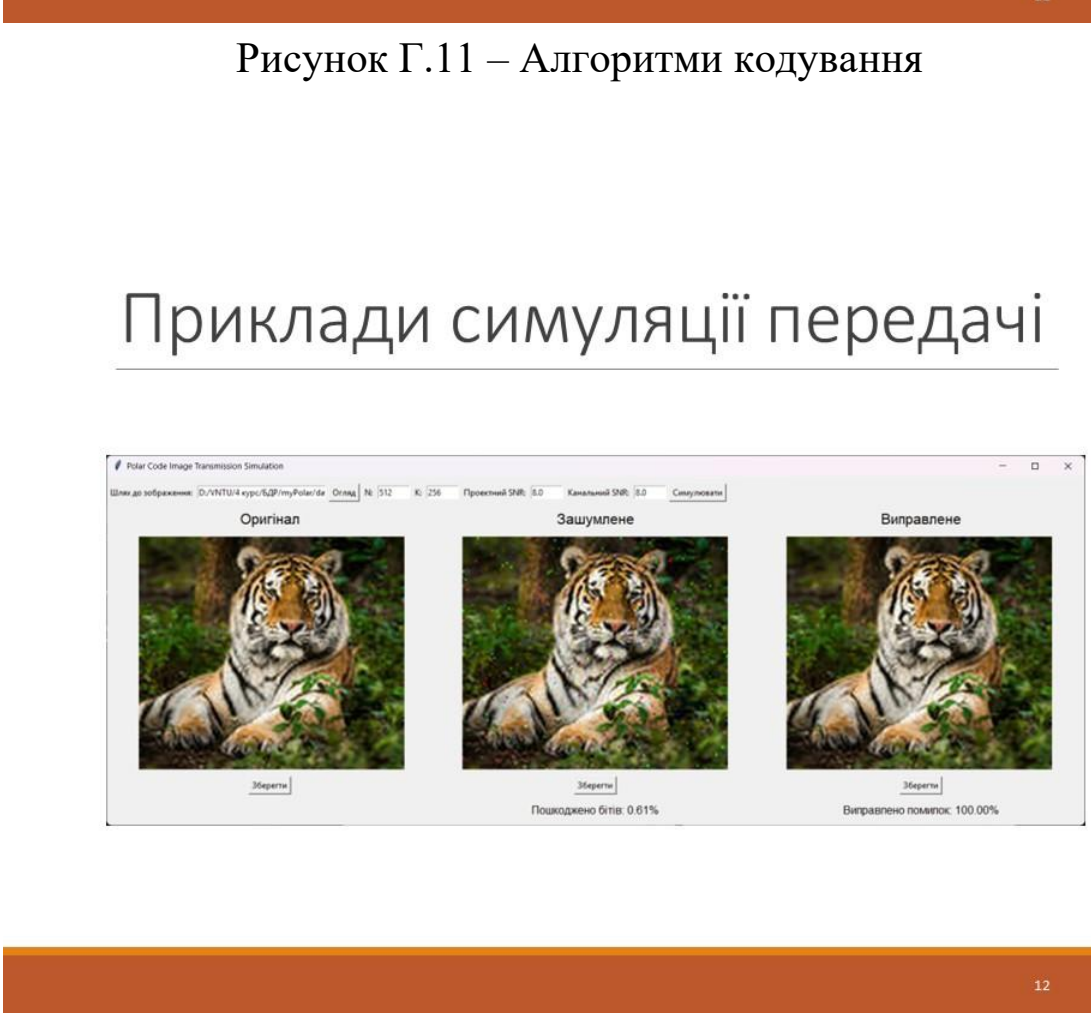


Рисунок Г.12 – Приклади симуляції передачі

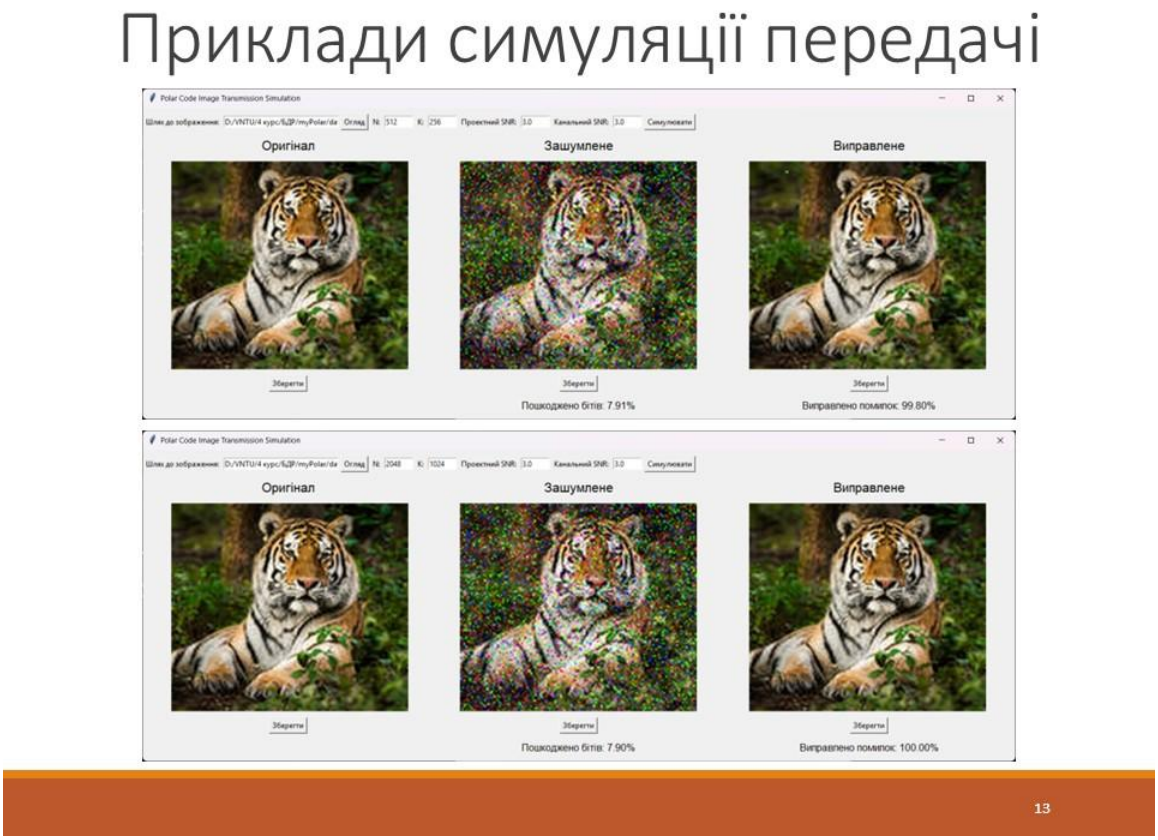


Рисунок Г.13 – Приклади симуляції передачі

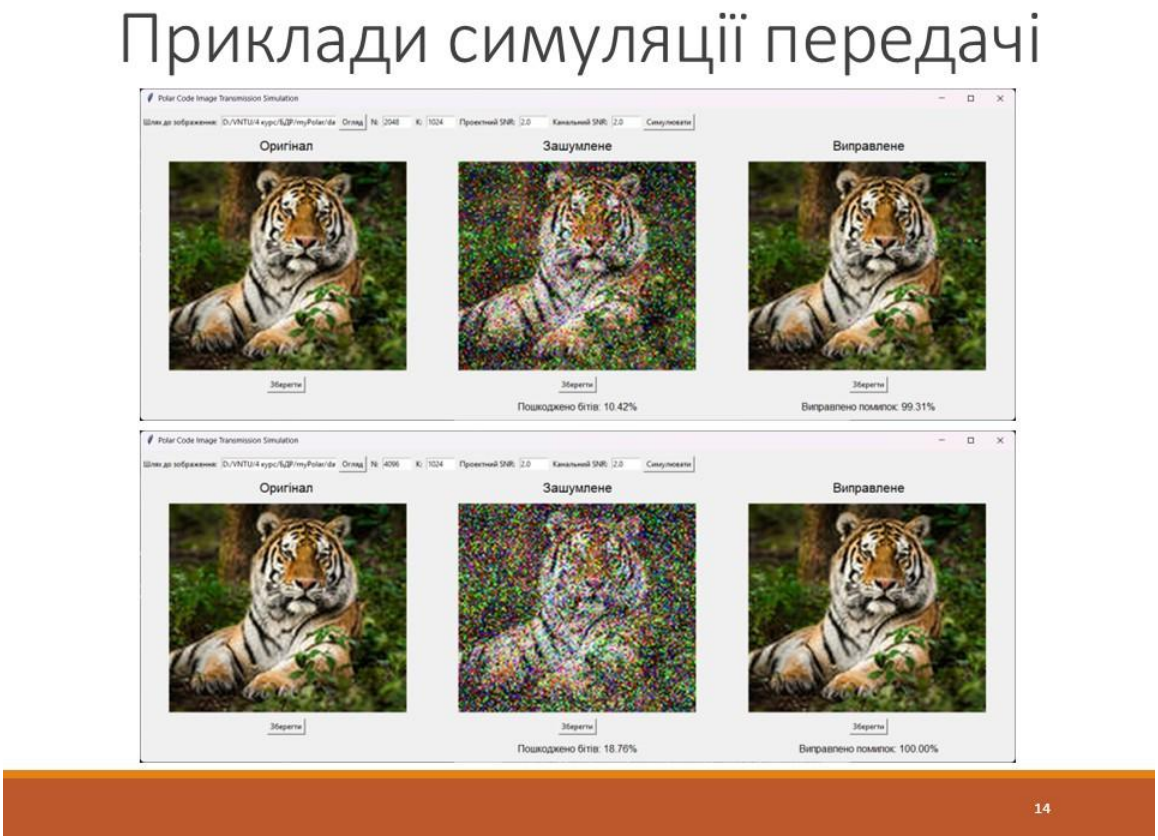
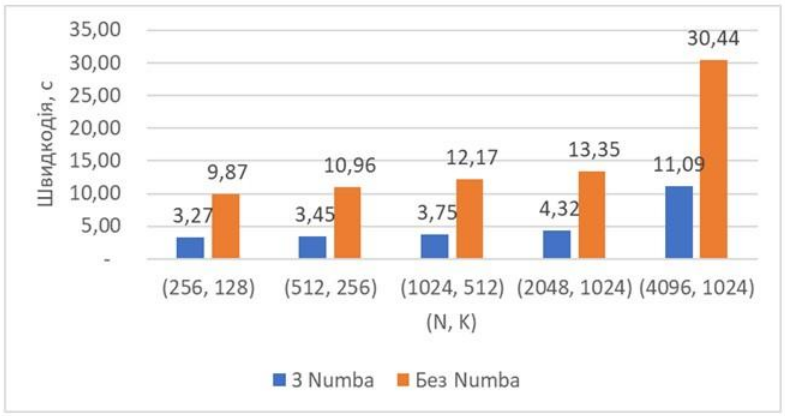


Рисунок Г.14 – Приклади симуляції передачі

Порівняння швикодії декодування з Numba та без



15

Рисунок Г.15 – Порівняння швикодії декодування з Numba та без

Висновки

У бакалаврській кваліфікаційній роботі було розроблено програмний модуль завадостійкого кодування даних в системах передачі інформації. Було обґрунтовано доцільність застосування полярних кодів у задачах захисту інформації під час передачі через зашумлені канали зв'язку. На основі проведеного аналізу засобів реалізації обрано мову програмування Python та бібліотеки NumPy і Numba, які забезпечили баланс між обчислювальною ефективністю та гнучкістю розробки.

Подальшого розвитку отримав метод завадостійкого кодування, у якому, на відміну від існуючих, використано інструментарій JIT-компіляції бібліотеки Numba, що дало можливість компілювати фрагменти Python-коду у високопродуктивний машинний код, підвищуючи швидкодію обчислень.

16

Рисунок Г.16 – Висновки

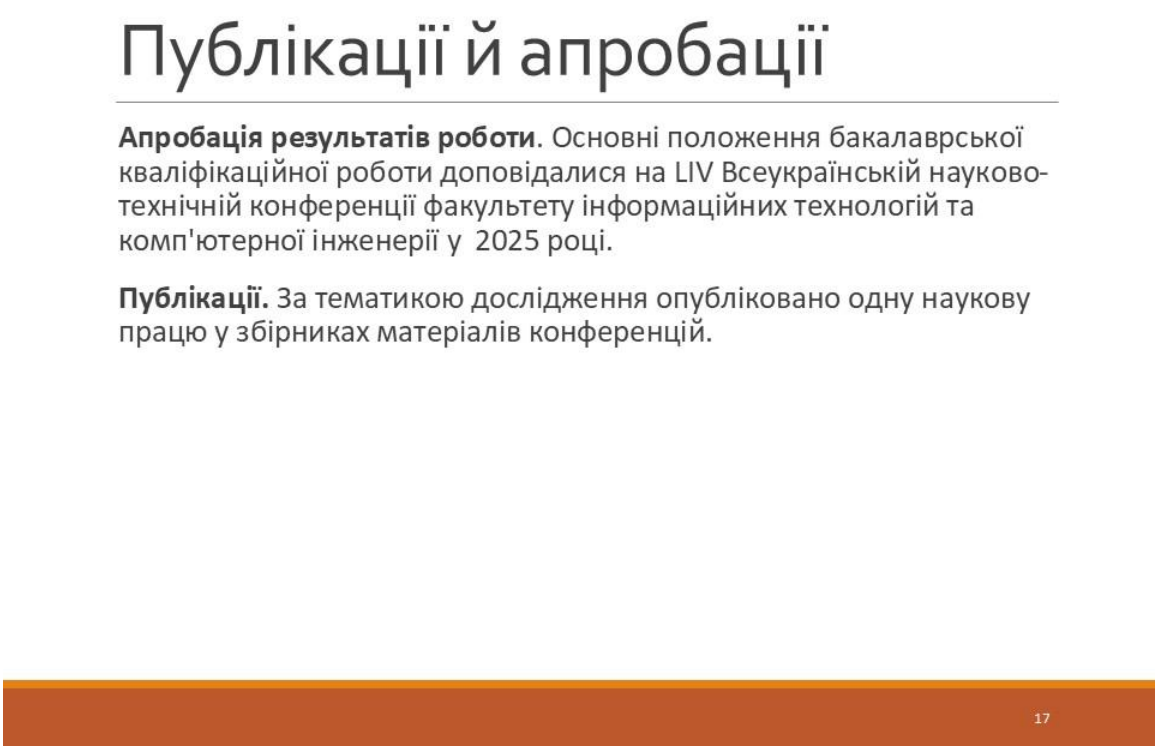


Рисунок Г.17 – Публікації й апробації

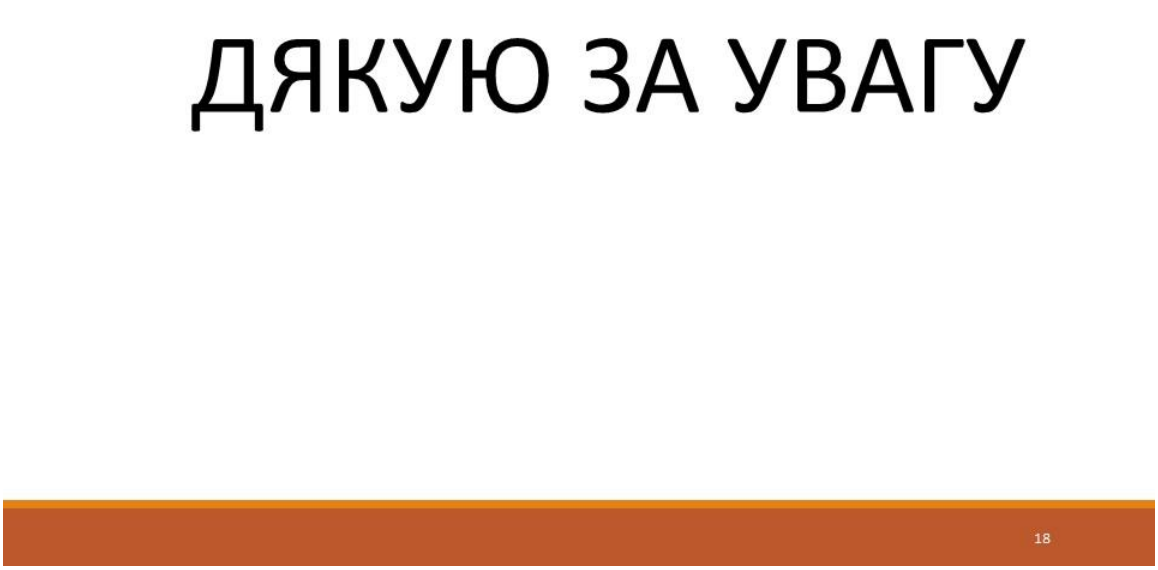


Рисунок Г.18 – Фінальний слайд з подякою