МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

На правах рукопису

ПРИСТУПА ДМИТРО ВОЛОДИМИРОВИЧ

ПРИХОВУВАННЯ ТЕКСТОВОЇ ІНФОРМАЦІЇ В МЕДІА ФАЙЛАХ МЕТОДОМ PVD ЗАСОБАМИ МОВИ PYTHON

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня “бакалавр”

Науковий керівник:
ГОЛОВІН МИКОЛА БОРИСОВИЧ,
кандидат фізико-математичних наук,
доцент
кафедри комп’ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
Засідання кафедри комп’ютерних наук
та кібербезпеки від
_____2025 р.
Завідувач кафедри

(_____)Гришанович Т. О.

ЛУЦЬК 2025

ЗМІСТ

ВСТУП

Актуальність дослідження

В умовах стрімкого розвитку цифрових технологій та глобальної комп’ютеризації питання захисту інформації набуває все більшої актуальності. Величезна кількість даних щоденно передається через відкриті канали зв’язку, що створює потенційні ризики витоку конфіденційної інформації.

Традиційні методи шифрування не завжди можуть гарантувати збереження самого факту передачі секретних даних у таємниці. У такому контексті особливого значення набуває стеганографія — наука, що вивчає способи приховування інформації в носіях даних таким чином, щоб її існування залишалося непомітним для сторонніх. Одним з ефективних і широко досліджуваних підходів у стеганографії є метод PVD (Pixel Value Differencing), який передбачає використання різниці значень яскравості пікселів для вбудовування повідомлень. На відміну від простих методів на кшталт LSB (Least Significant Bit), метод PVD дозволяє досягти більшої глибини вбудовування без помітного спотворення зображення. Це робить його особливо привабливим для практичного використання в умовах, де збереження якості медіафайлу є критичним. Сучасна мова програмування Python завдяки своїй гнучкості, великій кількості бібліотек для роботи з мультимедіа, простоті у використанні та активній спільноті розробників є зручним інструментом для реалізації алгоритмів стеганографії. Таким чином, дослідження можливостей реалізації методу PVD у Python та створення відповідного програмного забезпечення має високу наукову, технічну й практичну цінність.

Мета і завдання дослідження

Метою кваліфікаційної роботи є дослідження, розробка та реалізація програмного інструменту для приховування текстової

інформації в медіафайлах із використанням стеганографічного методу Pixel Value Differencing на мові програмування Python.

Завдання роботи:

- проаналізувати сучасні підходи до стеганографії, класифікувати методи приховування інформації в медіафайлах;
- дослідити основи методу PVD та порівняти його з іншими популярними методами (LSB, DCT, DWT тощо);
- вивчити можливості мови Python та відповідних бібліотек для реалізації алгоритмів стеганографії та обробки медіа;
- розробити програмне забезпечення, яке дозволяє вбудовувати текстову інформацію в зображення, відео та інші медіа з використанням методу PVD;
- оцінити ефективність реалізованого методу за допомогою кількісних метрик (PSNR, MSE) та експериментального аналізу.

Об’єкт і предмет дослідження

Об’єктом дослідження є процес приховування інформації в цифрових медіафайлах за допомогою стеганографічних методів.

Предметом дослідження виступає стеганографічний метод Pixel Value Differencing (PVD) як один із способів реалізації приховування текстової інформації в медіафайлах.

Апробація роботи. Робота пройшла апробацію і була представлена на II Міжнародній науково-практичній конференції "Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем" виступом на тему " Програмна реалізація приховування секретного повідомлення у зображенні методом PVD" . (Луцьк-Світязь, 9–11 червня 2025р) [1]

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ СТЕГАНОГРАФІЇ ТА МЕТОДУ PVD

1.1. Основи стеганографії. Види стеганографій

Стеганографія – це мистецтво приховування інформації всередині інших даних. Ця техніка використовується для захисту конфіденційної інформації від сторонніх очей, дозволяючи передавати її через зображення, аудіо, відео та інші цифрові файли. За допомогою стеганографії можна замаскувати практично будь-який цифровий контент, включаючи текстові, графічні, відео або аудіо файли. Приховані дані потім витягуються в пункті призначення.

Часто перед тим, як сховати дані, їх спочатку шифрують, або обробляють так, щоб ускладнити виявлення. Це робить приховану інформацію ще більш захищеною.

Слово «стеганографія» походить від грецьких слів «steganos» (що означає «прихований») і «graphein» (що означає «писати»). Це мистецтво використовується протягом тисячоліть для забезпечення конфіденційності комунікацій. Наприклад, у давній Греції повідомлення вирізьблювали на дереві, а потім вкривали воском, щоб їх сховати. У Римі використовували невидимі чорнила, які ставали видимими під дією тепла або світла.

Стеганографія, як метод прихованої комунікації, іноді порівнюється з криптографією. Однак це не одне й те саме, оскільки стеганографія не передбачає шифрування інформації під час передачі або використання ключа для її дешифрування після отримання.

Види стеганографій

Стеганографія поділяється на такі види:

- стеганографія тексту;
- стеганографія зображення;
- відео стеганографія;
- аудіо стеганографія;
- мережева стеганографія.

Стеганографія тексту. Текстова стеганографія передбачає приховування інформації в текстових файлах. Це включає зміну формату існуючого тексту, зміну слів у тексті, використання контекстно-вільних граматик для створення читабельних текстів або генерування випадкових послідовностей символів.

Стеганографія зображення. Це передбачає приховування інформації у файлах зображень. У цифровій стеганографії зображення часто використовуються для приховування інформації, оскільки цифрове представлення зображення містить велику кількість елементів і існують різні способи приховати інформацію всередині зображення.

Аудіо стеганографія. Аудіо стеганографія включає секретні повідомлення, вбудовані в аудіосигнал, який змінює двійкову послідовність відповідного аудіофайлу. Приховування секретних повідомлень у цифровому звукі є складнішим процесом порівняно з іншими.

Відео стеганографія. Саме тут дані приховані в цифрових відеоформатах. Відеостеганографія дозволяє приховати великі обсяги даних у рухомому потоці зображень і звуків. Є два типи відеостеганографії. Перше, це вбудовування даних у нестиснене необроблене відео з наступним стисненням. Друге – вбудовування даних безпосередньо в стиснений потік даних.

Мережева стеганографія. Мережева стеганографія, іноді відома як

протокольна стеганографія, - це техніка вбудовування інформації в мережеві протоколи керування, які використовуються для передачі даних, наприклад TCP, UDP, ICMP тощо.

1.2. Популярні методи стеганографії

Метод найменш значущого біта (LSB). Цей метод передбачає вбудовування секретних даних у найменш значущі біти цифрового файлу. Наприклад: у зображеннях кожен піксель складається з трьох байтів даних, що відповідають червоному, зеленому та синьому кольорам. Деякі формати зображень також додають четвертий байт для прозорості, який називається «альфа».

LSB стеганографія змінює останній біт кожного з цих байтів, щоб приховати один біт інформації. Тому, для приховування одного мегабайта даних, потрібно вісім мегабайтів зображення. Зміна останнього біта пікселя не призводить до видимих змін у зображенні, тому навіть при порівнянні оригінального та стеганографічно зміненого зображення різниця не буде помітна.

Той самий підхід можна використовувати для інших цифрових медіа, таких як аудіо та відео файли, де дані приховуються в тих частинах, які викликають найменші зміни у звукових чи візуальних ефектах.

Wavelet Transform-Based Methods (DWT - Discrete Wavelet Transform)

Цей метод працює в частотній області, використовуючи дискретне вейвлет- перетворення для аналізу сигналу на різних рівнях деталізації.

Принцип роботи:

- Розбиття зображення: зображення ділиться на кілька рівнів (смуг частот) за допомогою вейвлет- перетворення. Найчастіше

виділяються чотири субсмуги: низькочастотна (LL) і три високочастотні (LH, HL, HH).

- Приховування даних: інформація вбудовується у високочастотні субсмуги (LH, HL, HH), оскільки ці компоненти менш помітні для людського ока. У низькочастотну субсмугу (LL) дані зазвичай не вставляють, щоб не погіршити якість зображення.
- Зворотне перетворення: після вбудовування даних проводиться зворотнє ейвлет-перетворення для відновлення зображення.

Переваги:

- Висока стійкість до втрати даних при стисненні (наприклад, JPEG).
- Можливість приховування великої кількості даних.
- Менша помітність змін для людського ока, оскільки використовуються частотні характеристики.

Недоліки:

- Складність у реалізації через необхідність роботи з частотними коефіцієнтами.
- Залежність якості від обраного типу вейвлета (наприклад, Haar, Daubechies тощо).
- Можливі помітні артефакти, якщо приховується занадто багато даних.

Discrete Cosine Transform (DCT)

DCT перетворює просторові дані (пікселі) в частотну область, де приховування інформації стає менш помітним. Цей метод активно використовується у форматах JPEG.

Принцип роботи:

- Розбиття зображення: зображення розбивається на блоки (зазвичай

8x8 пікселів).

- Перетворення блоків: до кожного блоку застосовується DCT, яка перетворює пікселі в коефіцієнти частот. Більшість інформації зосереджена в низькочастотних коефіцієнтах, тоді як високочастотні відповідають за деталі.
- Приховування даних: дані вбудовуються у високочастотні коефіцієнти, де зміни менш помітні. Для уникнення помітних спотворень часто змінюють лише коефіцієнти середнього діапазону частот.

Переваги:

- Висока стійкість до втрат даних при стисненні, особливо для форматів, що використовують DCT (JPEG).
- Менша помітність змін завдяки приховуванню в частотній області.

Недоліки:

- Вбудовування великих обсягів даних може спричинити помітні артефакти. Складність у реалізації через перетворення і обробку коефіцієнтів.
- Втрата прихованої інформації при перезаписі у форматах із сильним стисненням.

Pixel Value Differencing (PVD)

PVD використовує різницю між значеннями пікселів у парі для приховування даних, враховуючи властивості людського зору.

Принцип роботи:

- Розбиття пікселів на пари: зображення розділяється на непересічні пари пікселів.
- Обчислення різниці: визначається різниця між значеннями пари пікселів.

Якщо різниця велика, це означає, що область має високу текстуру, і

можна приховати більше даних. Якщо різниця мала, приховуються лише невеликі обсяги даних, щоб уникнути помітних змін.

- Відновлення: для вилучення даних аналізується різниця між пікселями.

Переваги методу PVD:

- Непомітність: Метод зберігає якість зображення завдяки мінімальному викривленню пікселів. Різниця у яскравості між сусідніми пікселями змінюється лише в межах, що залишаються невидимими для людського ока.
- Ємність: Кількість даних, що можуть бути приховані, залежить від рівня контрасту між пікселями. Пара з великим контрастом може містити більше біт.
- Гнучкість: Метод дозволяє налаштовувати піддіапазони залежно від вимог до якості зображення та обсягу даних.

Недоліки методу PVD

- Обмеження ємності: У пікселях з низьким контрастом може бути приховано менше інформації.
- Вразливість до атак: Якщо зловмисник знає алгоритм та параметри піддіапазонів, метод може бути зламаний.
- Можливі викривлення: Для висококонтрастних зображень метод може спричинити незначні артефакти при надмірному приховуванні даних.

1.3. Історія досягнень у методі Pixel Value Differencing (PVD)

Перший етап: Винахід методу

Метод Pixel Value Differencing (PVD) був запропонований у 2003 році. Основна ідея полягала в тому, щоб використовувати більшу різницю між сусідніми пікселями для приховування більшої кількості даних, зберігаючи при цьому якість зображення.

Еволюція методу: Модифікації та покращення

Адаптивний поріг: У 2011 році було запропоновано модифікації, які динамічно змінюють пороги в залежності від локальних характеристик зображення. Це дозволило підвищити стійкість до стеганографічного аналізу.

Приклад роботи: Adaptive PVD methods використовуються для мінімізації візуальних артефактів у висококонтрастних областях.

Блоковий PVD: Метод розширили, щоб оперувати блоками пікселів, а не окремими парами. Це забезпечило більшу ємність для приховування даних і кращий контроль якості зображення.

Гібридні підходи:

У дослідженнях 2017 року почали комбінувати PVD із методами Least Significant Bit (LSB) для збільшення ємності приховування. Гібридні підходи дозволяють використовувати LSB для однорідних областей і PVD для контрастних, тим самим зберігаючи баланс між прихованням і якістю зображення.

Сучасний стан і розширення

Застосування в інших форматах: У 2020-х роках метод PVD почали адаптувати для відео та тривимірної графіки. Це дозволило використовувати додаткові канали даних, наприклад, часовий вимір у відео.

Штучний інтелект у PVD: Сучасні дослідження інтегрують машинне навчання для аналізу оптимальних місць приховування

інформації, що додатково підвищує стійкість і ефективність алгоритму.

Досягнення

Метод PVD залишається одним із найпопулярніших способів приховування даних завдяки його ефективності та простоті.

Він використовується для створення безпечних каналів зв'язку, де стійкість до детекції відіграє ключову роль.

Інтеграція з сучасними технологіями (наприклад, штучним інтелектом) робить його актуальним і в наші дні.

1.4. Особливості методу Pixel Value Differencing (PVD). Алгоритм приховування даних у PVD

Метод Pixel Value Differencing (PVD) є одним із способів стеганографії, що дозволяє приховувати інформацію в цифрових зображеннях. Основною ідеєю цього методу є використання різниці значень сусідніх пікселів для визначення кількості інформації, яка може бути прихована.

Алгоритм приховування даних у PVD

1. Розбиття зображення на пари пікселів:

Зображення розбивається на групи пікселів, зазвичай на пари. Для кожної пари визначають різницю значень інтенсивності.

2. Обчислення різниці: Обчислюється різниця значень інтенсивності сусідніх пікселів у парі. Це дозволяє визначити текстурний тип області.

3.Визначення інтервалу різниць: Знайдена різниця порівнюється з попередньо заданими інтервалами. Кожен інтервал відповідає певній кількості бітів, які можна приховати.

4. Приховування даних: витягується стільки бітів секретного повідомлення, скільки дозволяє інтервал; нове значення різниці

обчислюється шляхом корекції, вихідної різниці відповідно до прихованих бітів;

5. Модифікація пікселів: На основі нової різниці модифікуються значення інтенсивності пікселів, щоб приховані дані залишилися в текстурній області. При цьому враховується, щоб зміни не виходили за межі діапазону кольорів.

6. Відновлення зображення: Модифіковані значення пікселів об’єднуються, формуючи нове зображення з прихованими даними.

7. Розшифрування даних: Для вилучення даних повторюється процес обчислення різниці між пікселями, визначення інтервалу та витягування прихованої інформації.

Метод PVD забезпечує високий рівень приховування даних у зображенні без значних візуальних змін, що робить його ефективним для стеганографічних застосувань.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЗПЕЧЕННЯ З ПРИХОВУВАННЯ ТЕКСТУ В МЕДІАФАЙЛАХ МЕТОДОМ PVD

2.1. Реалізація алгоритму на Python із впровадження повідомлення у зображення

Програма із впровадження повідомлення у зображення представлена в роботі [1]. Нижче, функція впровадження повідомлення з цієї програми

```
from PIL import Image
def pvd_encode(cover_image_path, secret_message, stego_image_path):
    img = Image.open(cover_image_path).convert('RGB')
    width, height = img.size
    binary_message = ''.join(format(ord(char), '08b') for char in secret_message)
    message_len = len(binary_message)
    data_index = 0
    for i in range(0, height - 1, 2):
        for j in range(0, width - 1, 2):
            if data_index < message_len:
                p1, p2 = img.getpixel((j, i))[0], img.getpixel((j + 1, i))[0]
                diff = abs(p1 - p2)
                if   diff <= 7:
                    n_bits= 3
                    if data_index +n_bits> message_len: n_bits= message_len - data_index
                    binary_chunk = binary_message[data_index:data_index + n_bits]
                    data_index +=n_bits
                    new_diff = int(binary_chunk, 2)
                    sp=(p1+p2)//2
                    if p1 < p2:
                        p1_new = sp-new_diff//2
                        p2_new = sp+(new_diff-new_diff//2)
                    else:
                       p1_new = sp+(new_diff-new_diff//2)
                       p2_new = sp-new_diff//2
                    img.putpixel((j,i),(p1_new,img.getpixel((j,i))[1],img.getpixel((j,i))[2]))
                    img.putpixel((j+1,i),(p2_new,img.getpixel((j+1,i))[1],img.getpixel((j+1,i))[2]))
    img.save(stego_image_path)
```

Процес впровадження повідомлення у зображення має наступні етапи. [1,2]

1. Відкриття вхідного зображення (порожній контейнер) та перетворення його на двомірний масив значень пікселів, що має три складових Red, Green, Blue.
2. Визначення ширини і висоти зображення
3. Перетворення текстового повідомлення в бітовий масив [2]
4. Визначення довжини бітового масиву
5. Відкриття відповідних циклів сканування зображення за висотою і шириною та перебір за пікселями парами.
6. Розрахунок різниці між червоними компонентами двох сусідніх пікселів (diff).
7. Пошук за допомогою розгалуження if diff <= 7: пар пікселів для яких ця різниця була б така, щоб число бітів (n_bits), які можна було б закодувати, не перевищувало трьох.
8. Вирізаємо бінарний шматок коду для впровадження в зображення
9. Переводимо бінарний шматок коду в десятковий код
10. Визначаємо нову різницю міри червоності двох сусідніх пікселів
11. Знаходимо середне значення червоності між пікселями.
12. Якщо рначення червоності пікселей p1 < p2. Обчислюємо нові
13. значення червоності пікселів так:
    p1_new = sp-new_diff//2;
    p2_new=sp+(new_diff-new_diff//2)
14. Якщо рначення червоності пікселей p1 > p2. Обчислюємо нові
15. значення червоності сусідніх пікселів так:
    p1_new = sp+(new_diff-new_diff//2) ;
    p2_new = sp-new_diff//2
16. Впровадити нові значення червоності пікселів в зображення [1,2]

2.2. Реалізація алгоритму на Python із вилучення повідомлення з зображення

Програма із вилучення повідомлення з зображення представлена в роботі [1]. Нижче, функція вилучення повідомлення та запуск цієї програми

```
def pvd_decode(stego_image_path):
    img = Image.open(stego_image_path).convert('RGB')
    width, height = img.size
    binary_message = ''
    for i in range(0, height - 1, 2):
        for j in range(0, width - 1, 2):
            p1, p2 = img.getpixel((j, i))[0], img.getpixel((j + 1, i))[0]
            diff = abs(p1 - p2)
            if diff <= 7:
                n_bits= 3
                binary_chunk = bin(diff)[2:].zfill(n_bits)
                binary_message += binary_chunk
    decoded_message = ''
    for i in range(0, len(binary_message), 8):
        if i + 8 > len(binary_message): break
        byte = binary_message[i:i + 8]
        if '#'!=chr(int(byte, 2)):
            decoded_message += chr(int(byte, 2))
        else: break
    return decoded_message
if __name__ == '__main__':
    cover_image_path = 'cover.png' stego_image_path = 'stego_image.png'
    secret_message = 'This is a secret message! #'
    pvd_encode(cover_image_path, secret_message, stego_image_path)
    decoded_message = pvd_decode(stego_image_path)
    print("Decoded message:", decoded_message)
```

Процес роботи програми повідомлення з зображення має наступні етапи. [1,2]

1. Завантаження фотографії, що є заповненим контейнером
2. Визначення ширини і висоти зображення
3. Резервування строкової змінної для бінарного коду
4. Сканування двома циклами картинки рядами та стовбчиками пікселів
5. Визначення міри червоності двох сусідніх пікселів p1, p2

6. Визначення різниці міри червоності двох сусідніх пікселів
7. Пошук пікселів міра різниці червоності diff, яких менша або рівна 7, тобто вкладається в трьох розрядне двійкове число
8. Перетворення diff в двійкове три розрядне число
9. Формування тексту в бінарному форматі
10. Розбиття повідомлення в бінарному форматі на байтові фрагменти
11. Перетворення тексту в бінарному форматі в звичайний текст. [2]

2.3. Аналіз роботи програми по приховуванню тексту в картинці по технології PVD.

Приховування в картинці тексту «ACE» . Знизу представлена роздруківка роботи програми, що приховує інформацію в різницях значень червоного коліру сусідніх пікселів картинки. Приховування відбувається тільки в тих різницях значень пікселів, які не перевищують десяткове 7 або двійкове 111. Тобто в пікселях, які мають малу різницю значень червоного коліру і різниця вкладається в трьох розрядне двійкове число.

Якщо різниця перевищує зазор в значеннях кольорів, змін не відбувається.

Перетворення тексту «ACE» в двійковий код. 010000010100001101000101 (код A - 65 або 01000001. алф. A,B,C,D,E,F,…)

Нижче приведена роздруківка роботи програми при умові, що на екран виводяться:

- значення RED кольорів двох сусідніх пікселів;
- різниця значень двох сусідніх пікселів;
- бінарний фрагмент тексту, що впроваджується;
- нова різниця значень двох сусідніх пікселів;
- нові значення кольорів пікселів.

Роздруківка виконання програми:

Значення RED кольорів двох сусідніх пікселів: 24 , 29 ; різниця значень 5.

Бінарний фрагмент тексту, що впроваджується 010;

нова різниця 2; нові значення кольорів 25, 27

Значення RED кольорів двох сусідніх пікселів: 51 , 42 ; різниця значень 9.

Бінарний фрагмент тексту, що впроваджується ------;

нова різниця 9; нові значення кольорів 51, 42

Значення RED кольорів двох сусідніх пікселів: 40, 43 ; різниця значень 3 .

Бінарний фрагмент тексту, що впроваджується 000 ;

нова різниця 0 ;нові значення кольорів 41, 41

Значення RED кольорів двох сусідніх пікселів: 52, 33 ; різниця значень 19 .

Бінарний фрагмент тексту, що впроваджується ------ ;

нова різниця 19 нові значення кольорів 52 , 33

Значення RED кольорів двох сусідніх пікселів: 33, 73; різниця значень 40 .

Бінарний фрагмент тексту, що впроваджується ------ ;

нова різниця 40 ; нові значення кольорів 33, 73

Значення RED кольорів двох сусідніх пікселів: 70, 49 ; різниця значень 21 .

Бінарний фрагмент тексту, що впроваджується ------- ;

нова різниця 21 ; нові значення кольорів 70, 49

Значення RED кольорів двох сусідніх пікселів: 51, 60; різниця значень 9 .

Бінарний фрагмент тексту, що впроваджується - ;

нова різниця 9 ; нові значення кольорів 51 , 60

Значення RED кольорів двох сусідніх пікселів: 87, 64; різниця значень 23 .

Бінарний фрагмент тексту, що впроваджується ------- ;

нова різниця 23 ; нові значення кольорів 87, 64

Значення RED кольорів двох сусідніх пікселів: 62, 65 ; різниця значень 3 .

Бінарний фрагмент тексту, що впроваджується 010 ;

нова різниця 2 ; нові значення кольорів 62 ,64

Значення RED кольорів двох сусідніх пікселів: 97, 105; різниця значень 8

Бінарний фрагмент тексту, що впроваджується -------- ;

нова різниця 8 ; нові значення кольорів пікселів 97 , 105

Значення RED кольорів двох сусідніх пікселів: 77, 49; різниця значень 28 .

Бінарний фрагмент тексту, що впроваджується -------- ;

нова різниця 28 нові значення кольорів 77 , 49 Значення RED кольорів двох сусідніх пікселів: 35, 29; різниця значень 6 .

Бінарний фрагмент тексту, що впроваджується 100 ;

нова різниця 4 ; нові значення кольорів 34 , 30 Значення RED кольорів двох сусідніх пікселів: 28, 24; різниця значень 4 .

Бінарний фрагмент тексту, що впроваджується 001 ;

нова різниця 1 ; нові значення кольорів 27 , 26

Значення RED кольорів двох сусідніх пікселів: 20, 27; різниця значень 7 .

Бінарний фрагмент тексту, що впроваджується 101 ;

нова різниця 5 ; нові значення кольорів 21, 26

Значення RED кольорів двох сусідніх пікселів: 16, 19; різниця значень 3 .

Бінарний фрагмент тексту, що впроваджується 000 ;

нова різниця 0 ; нові значення кольорів 1 , 17

Значення RED кольорів двох сусідніх пікселів: 8, 16; різниця значень 8 .

Бінарний фрагмент тексту, що впроваджується -------- ;

нова різниця 8 нові значення кольорів 8 , 16 Значення RED кольорів двох сусідніх пікселів: 21, 26; різниця значень 5 .

Бінарний фрагмент тексту, що впроваджується 101 ;

нова різниця 5 ; нові значення кольорів 21 , 26

З роздруківки видно, що впровадження бінарного коду відбувається не в усі рядом розташовані пікселя, а лише в ті, де різниця червоної складової менше або рівна 7. В випадку, якщо впровадження не відбувалось виводяться прочерки «------». Видно, що програма працює корректно.

2.4. Аналіз роботи програми по вилученню тексту з картинки, що був впроваджений по технології PVD.

Вилучення з картинки тексту «ACE» . Знизу представлена роздруківка роботи програми, що вилучає інформацію з різниць значень червоного коліру сусідніх пікселів картинки. Вилучення відбувається тільки з тих різниць значень пікселів, які не перевищують десяткове 7 або двійкове 111. Тобто вилучення відбувається тільки з тих пікселів, які мають малу різницю значень червоного коліру і різниця вкладається в трьох розрядне двійкове число.

Якщо різниця перевищує зазор в значеннях кольорів, вилучення не відбувається.

Нижче приведена роздруківка роботи програми при умові, що на екран виводяться:

- значення RED кольорів двох сусідніх пікселів;
- різниця значень двох сусідніх пікселів;
- бінарний фрагмент тексту, що вилучається;

Роздруківка виконання програми:

Значення RED кольорів двох сусідніх пікселів: 25, 27; різниця значень 2.
Бінарний фрагмент тексту, що вилучається 010
Значення RED кольорів двох сусідніх пікселів: 51, 42 ; різниця значень 9.
Бінарний фрагмент тексту, що вилучається -------
Значення RED кольорів двох сусідніх пікселів: 41 , 41 ; різниця значень 0 .
Бінарний фрагмент тексту, що вилучається 000
Значення RED кольорів двох сусідніх пікселів: 52 , 33 ; різниця значень 19 .
Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 33, 73 ; різниця значень 40 .
Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 70 , 49 ; різниця значень 21 .

Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 51 , 60 ; різниця значень 9 .
Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 87 , 64 ; різниця значень 23 .
Бінарний фрагмент тексту, що вилучається -
Значення RED кольорів двох сусідніх пікселів: 62, 64 ; різниця значень 2 .
Бінарний фрагмент тексту, що вилучається 010
Значення RED кольорів двох сусідніх пікселів: 97, 105; різниця значень 8 .
Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 77, 49; різниця значень 28 .
Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 34, 30 ; різниця значень 4 .
Бінарний фрагмент тексту, що вилучається 100
Значення RED кольорів двох сусідніх пікселів: 27, 26; різниця значень 1 .
Бінарний фрагмент тексту, що вилучається 001
Значення RED кольорів двох сусідніх пікселів: 21, 26; різниця значень 5 .
Бінарний фрагмент тексту, що вилучається 101
Значення RED кольорів двох сусідніх пікселів: 17 , 17 ; різниця значень 0.
Бінарний фрагмент тексту, що вилучається 000
Значення RED кольорів двох сусідніх пікселів: 8, 16; різниця значень 8 .
Бінарний фрагмент тексту, що вилучається --------
Значення RED кольорів двох сусідніх пікселів: 21 , 26 ; різниця значень 5 .
Бінарний фрагмент тексту, що вилучається 101

З роздруківки видно, що вилучення бінарного коду відбувається не з усіх рядом розташованих пікселів, а лише з тих, де різниця червоної складової менше або рівна 7. В випадку, якщо вилучення не відбувалось виводяться прочерки «------». Вино, що програма працює корректно.

2.5. Реалізація алгоритму на Python із впровадження повідомлення у звук по технології PVD

Нижче представлений приклад Python-коду для приховування текстової інформації в WAV-файлі методом PVD [1, 3].

```
import wave
def text_to_bits(text):
    return ''.join(f"{ord(c):08b}" for c in text)
def embed_pvd(wav_in, wav_out, text):
    bits = text_to_bits(text) + '00000000'  # маркер завершення
    bit_index = 0
    with wave.open(wav_in, 'rb') as w:
        params = w.getparams()
        frames = bytearray(w.readframes(w.getnframes()))
    for i in range(0, len(frames) - 1, 2):
        if bit_index >= len(bits):   break
        p1, p2 = frames[i], frames[i + 1]
        diff = abs(p1 - p2)
        # Визначення місткості (кількості біт для вставки)
        if diff <= 7:  nbits = 3
        if bit_index + nbits > len(bits):   nbits = len(bits) - bit_index
        val = int(bits[bit_index:bit_index + nbits], 2)
        new_p2 = (p1 + val) % 256
        frames[i + 1] = new_p2
        bit_index += nbits
    with wave.open(wav_out, 'wb') as w:
        w.setparams(params)
        w.writeframes(frames)
    print("Приховування завершено.")
# Приклад використання
embed_pvd("input.wav", "output_pvd.wav", "SECRET MESSAGE ")
```

В цій програмі відбувається аналіз різниць між сусідніми сэмплами у WAV-файлі. Ця різниця визначає місткість (кількість бітів), яку можна вставити в семпли пари. В цій програмі, як і у попередній текст в бітовому форматі впроваджується в ті парі де різниця між сэмплами менша 7, аналогічно тому, як це було зроблено в попередніх програмах.

Ця програма може працювати з 8-бітним моно звукам в WAV форматі (1 канал, sample width = 1 байт). Для 16-бітного звуку або стерео, код потребує адаптації.

Програма впровадження повідомлення у звук по технології PVD була перевірена на коректность роботи. Програма показала коректну роботу з 8-бітним моно звуком в WAV форматі. Для перевірки використовувались звуки саксофона, що вирізняються від інших високою чистою звучання. Порівняння звуку без впровадженого тексту з впровадженим текстом не виявило різниці, також не вивлено було ніяких сторонніх шумів.

2.6. Реалізація алгоритму на Python із вилучення повідомлення з звуку по технології PVD

```
import wave
def bits_to_text(bits):
    chars = []
    for i in range(0, len(bits), 8):
        byte = bits[i:i+8]
        if byte == '00000000': break
        chars.append(chr(int(byte, 2)))
    return ''.join(chars)
def extract_pvd(wav_file):
    bits = ''
    with wave.open(wav_file, 'rb') as w:
        frames = bytearray(w.readframes(w.getnframes()))
    for i in range(0, len(frames) - 1, 2):
        p1, p2 = frames[i], frames[i + 1]
        diff = abs(p1 - p2)
        if diff <= 7:    nbits = 3
        val = (p2 - p1) % 256
        bits += f"{val:0{nbits}b}"
        if '00000000' in bits[-16:]:  break # останні 2 байти = 0
    return bits_to_text(bits)
# Витягування прихованого повідомлення
msg = extract_pvd("output_pvd.wav")
print("Витягнуте повідомлення:", msg)
```

Програма вилучення повідомлення з звуку, що був впроваджений туди по технології PVD, була перевірена на коректность роботи. Експерименти проводились з різними за розміром текстами та музикальними звуками. Повідомлення успішно вилучались. Наявність текстових закладок в звукові файли не проявляли себе ніякіми спотвореннями звуку.

2.7. Програма впровадження тексту в відео файл. Архітектура програми, використані бібліотеки, інтерфейс

У реалізації було використано наступні бібліотеки:

- OpenCV (cv2) — використовується для обробки відео: відкриття, зчитування кадрів, конвертація кольорів, збереження відео;
- Pillow (PIL) — застосовується для конвертації кадрів у зображення, сумісні з Tkinter;
- Tkinter — стандартна бібліотека Python для створення графічного інтерфейсу користувача (GUI);
- SQLite (sqlite3) — вбудована СУБД, використовується для зберігання інформації про відеофайли та закодовані повідомлення;

Таким чином, обрані інструменти дозволили реалізувати зручний графічний застосунок, здатний:

- приховувати текстову інформацію у відеофайлах методом PVD;
- зчитувати повідомлення з відео;
- зберігати та завантажувати текст;
- керувати відтворенням відео;
- зберігати історію кодування у базі даних.

Архітектура розробленої програмної системи побудована з урахуванням модульності, зручності використання та масштабованості. Вона складається з декількох функціональних компонентів, які взаємодіють між собою для забезпечення приховування текстових повідомлень у відеофайлах методом PVD (Pixel Value Differencing).

Основні компоненти системи: Графічний інтерфейс користувача (GUI): Реалізований з використанням бібліотеки Tkinter. Містить:

- Полотно для відтворення відео.
- Текстове поле для введення або перегляду повідомлення.

- Кнопки управління: вибір відео, запуск/пауза, кодування, витягування повідомлення, збереження/завантаження тексту.

Модуль обробки відео заснований на бібліотеці OpenCV забезпечує:

- Завантаження відеофайлу.
- Побудову циклу відтворення кадрів.
- Застосування методу PVD для вбудовування або зчитування повідомлень з кадрів.

Модуль стеганографії (PVD): Реалізує логіку вбудовування текстової інформації в пікселі відеокадрів за методом Pixel Value Differencing. Операції включають:

- Перетворення тексту у двійковий формат.
- Роботу з парами пікселів для визначення і зміни різниці яскравості.
- Відновлення повідомлення з бітів після аналізу пікселів.

База даних (SQLite): Забезпечує локальне збереження інформації про відео та повідомлення:

- Назва та шлях до закодованого відеофайлу.
- Вміст повідомлення.
- Дата та час кодування.
- Модулі збереження/завантаження тексту: Дозволяють:
- Зберігати введене повідомлення у текстовий файл.
- авантажувати повідомлення з текстового файлу у вікно програми.

Для реалізації приховування текстової інформації в медіафайлах методом Pixel Value Differencing (PVD) було створено Python-застосунок, який об’єднує обробку відео та графічний інтерфейс користувача. Програма забезпечує повний цикл: вибір відео, кодування/декодування повідомлення, відображення результатів, роботу з базою даних та файлами.

Код впровадження тексту в відео файл виглядає так:

```
def embed_message_pvd(frame, binary_message):
    message_index = 0
    rows, cols, _ = frame.shape
    for x in range(rows):
        for y in range(0, cols - 1, 2):  # Обробляємо пари пікселів
            if message_index >= len(binary_message):
                return frame
            pixel1 = frame[x, y]
            pixel2 = frame[x, y + 1]
            diff = abs(int(pixel1[0]) - int(pixel2[0]))
            if diff < 8:    n_bits = 3
             if message_index + n_bits > len(binary_message):
                n_bits = len(binary_message) - message_index
             hidden_bits = binary_message[message_index:message_index + n_bits]
            hidden_value = int(hidden_bits, 2)
             new_diff = hidden_value
             if pixel1[0] > pixel2[0]:
                pixel1[0] = pixel2[0] + new_diff
            else:
                pixel2[0] = pixel1[0] + new_diff
             frame[x, y] = pixel1
            frame[x, y + 1] = pixel2
            message_index += n_bits
     return frame
```

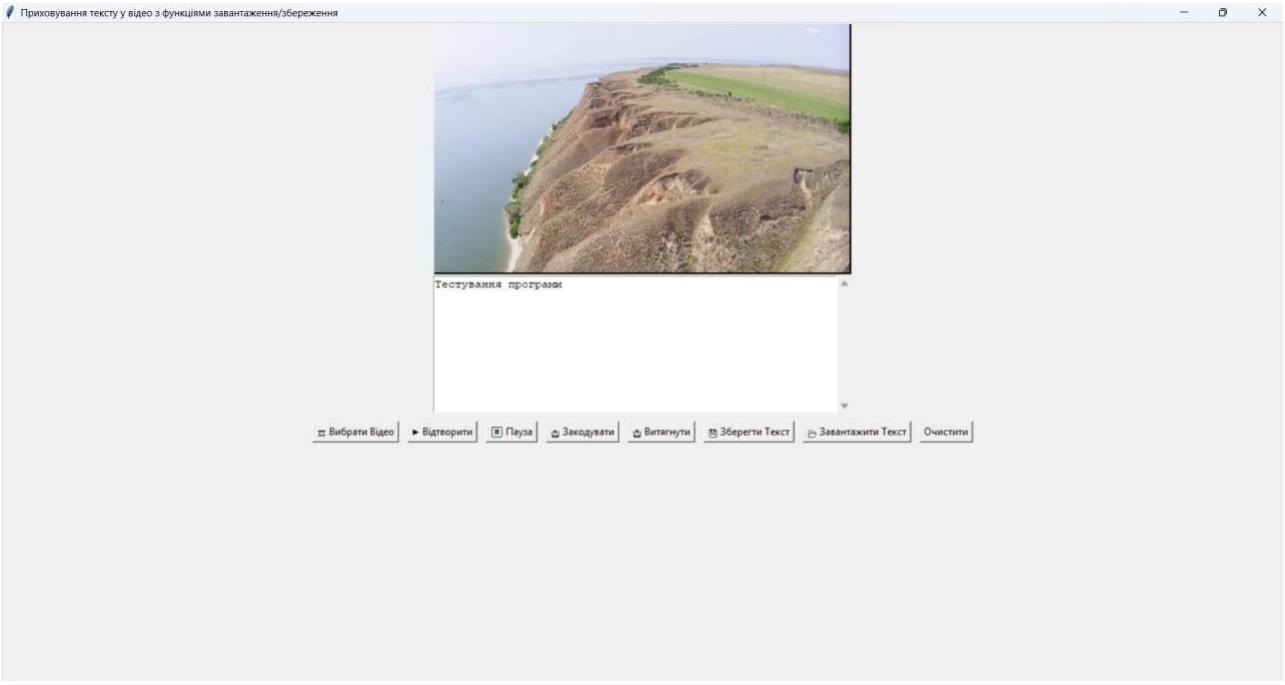
Код видобовування тексту з відео виглядає так:

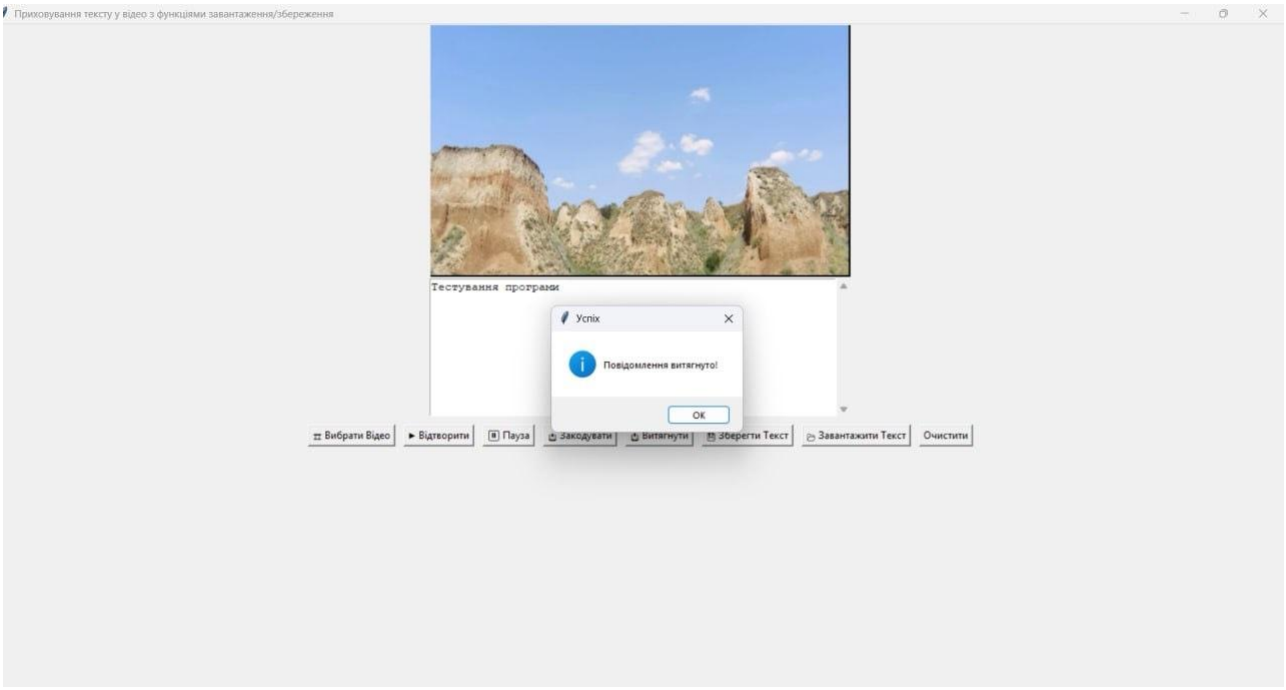
```
def extract_message_pvd(video_path):
    cap = cv2.VideoCapture(video_path)
    binary_message = ""
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        rows, cols, _ = frame.shape
        for x in range(rows):
            for y in range(0, cols - 1, 2):
                pixel1 = frame[x, y]
                pixel2 = frame[x, y + 1]
                diff = abs(int(pixel1[0]) - int(pixel2[0]))
                if diff < 8:    n_bits = 3
                value = diff
                binary_segment = format(value, f'0{n_bits}b')
                binary_message += binary_segment
                if "1111111111111110" in binary_message:
                    cap.release()
                    bits = binary_message.split("1111111111111110")[0]
                    return bits_to_message(bits)
```

Повний текст програми представлений в додатку А.

2.8. Тестування програми

В інтерфейсі програми обираємо потрібне нам відео і пишемо повідомлення яке ми хочемо приховати у відеофайлі.





Після того зберігаємо змінений файл на наш комп’ютер, обираємо змінений файл, і витягаємо текст.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто задачу приховування текстової інформації в медіафайлах за допомогою методу Pixel Value Differencing (PVD). Цей підхід виявився ефективним для стеганографії у відео, оскільки дозволяє адаптивно вбудовувати дані з мінімальними візуальними спотвореннями.

У ході роботи:

- обґрунтовано вибір методу PVD як більш гнучкого та стійкого до виявлення;
- спроєктовано архітектуру прикладної програми з модульною структурою;
- реалізовано застосунок мовою Python з підтримкою OpenCV, Pillow, Tkinter і SQLite;
- реалізовано зручний інтерфейс для вибору відео, введення повідомлень, їх кодування/витягнення, а також збереження тексту у файли;
- збереження інформації про кодування організовано через локальну базу даних.
- Результати роботи можна застосовувати у практичних системах інформаційного захисту, а також використовувати як основу для подальших досліджень і розробки гібридних або більш стійких стеганографічних методів.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Микола Головін, Дмитро Гузачов, Дмитро Приступа Програмна реалізація приховування секретного повідомлення у зображенні методом PVD. II Міжнародна науково-практична конференція "Проблеми комп'ютерних наук, програмного моделювання та безпеки цифрових систем" Луцьк-Світязь 9–11 червня 2025р. Тези доповідей Луцьк – 2025, С.130-131 https://apcssm.vnu.edu.ua/index.php/conf/article/view/242/161
2. Mykola Holovin, Nina Holovina Educational example of masking textual information in a photographic signal. Journal «ScienceRise: Pedagogical Education» No4(49)2022. P. 24-28 (Index Copernicus) http://journals.uran.ua/sr_edu/article/view/261051/258566
3. Головін М.Б. Навчальний приклад маскування інформації в акустичному сигналі / М.Б.Головін, Н.А.Головіна // Наукові записки Бердянського державного педагогічного університету. Серія : Педагогічні науки : зб. наук. пр. – Вип. 2. – Бердянськ : БДПУ, 2021. – С.203-211 https://evnuir.vnu.edu.ua/handle/123456789/20108
4. Що таке стеганографія, принципи та застосування в кібербезпеці. URL: https://hackyourmom.com/osvita/shho-take-steganografiya-prynczypy-ta-zastosuvannya-v-kiberbezpeczi/
5. Pixel value differencing steganography techniques: Analysis and open challenge URL: https://ieeexplore.ieee.org/abstract/document/7216859
6. Документація Pillow. URL: https://pillow.readthedocs.io/en/stable/
7. tkinter. URL: https://docs.python.org/3/library/tkinter.html
8. OS Module. URL: https://docs.python.org/3/library/os.html
9. Neil F. Johnson, Zoran Duric, Sushil Jajodia. Information Hiding: Steganography and Watermarking—Attacks and Countermeasures
10. Provos N., Honeyman P. Hide and Seek: An Introduction to Steganography
11. Chan C.-K., Cheng L.-M. Hiding data in images by simple LSB substitution

12. Kaur M., Kaur A., Kaur K. Steganographic approach for hiding image in DCT domain
13. Wang R.-Z., Lin C.-F., Lin J.-C. Image hiding by optimal LSB substitution and genetic algorithm
14. OpenCV Documentation. URL: https://docs.opencv.org/
15. Lutz M. Learning Python. — O'Reilly Media, 5th edition, 2013.
16. SQLite Documentation. URL: https://sqlite.org/docs.html
17. NumPy Documentation. URL: https://numpy.org/doc/

АНОТАЦІЇ

У кваліфікаційній роботі розглянуто задачу приховування текстової інформації у відеофайлах за допомогою стеганографічного методу Pixel Value Differencing (PVD). Актуальність теми обумовлена зростанням потреб у захисті конфіденційної інформації в умовах відкритих цифрових мереж.

У теоретичній частині роботи проаналізовано основні підходи до стеганографії, визначено її відмінності від криптографії, а також обґрунтовано вибір методу PVD як адаптивного й малопомітного способу вбудовування даних у цифрові зображення та відео.

У практичній частині реалізовано програмну систему засобами мови Python, що дозволяє:

- вибирати відеофайли та відтворювати їх у графічному інтерфейсі;
- кодувати текстові повідомлення у відео методом PVD;
- витягувати приховані повідомлення;
- зберігати й завантажувати повідомлення у текстових файлах;
- вести облік операцій за допомогою локальної бази даних SQLite.

Проведено тестування програмного забезпечення, яке підтвердило його функціональність та збереження візуальної якості відео після приховування повідомлень.

Обсяг роботи – 32 сторінки, кількість використаних джерел – 17, кількість додатків – 1.

This thesis addresses the problem of hiding textual information in video files using the steganographic method known as Pixel Value Differencing (PVD). The relevance of the topic is driven by the increasing need for protecting confidential information in open digital environments.

The theoretical section explores the fundamentals of steganography, compares it to cryptography, and justifies the use of the PVD method as an adaptive and imperceptible approach for embedding data into digital images and videos.

In the practical part, a software system was developed in Python, which provides the following functionality:

- selecting and playing video files via a graphical interface;
- embedding textual messages into video frames using the PVD method;
- extracting hidden messages from encoded video;
- saving and loading messages from text files;
- logging encoding activity using a local SQLite database.

The application was tested and shown to successfully embed and recover messages while preserving high video quality and minimizing visible distortion.

Total volume – 32 pages, number of references – 17, appendices – 1.

Додаток А

Код програми

```
import sqlite3
import os
import cv2
from tkinter import Tk, filedialog, Text, Button, Canvas, Scrollbar, messagebox, END
from tkinter.ttk import Frame
from PIL import Image, ImageTk
import threading
import time

# Ініціалізація бази даних SQLite
def init_db():
    conn = sqlite3.connect("video_steganography.db")
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS videos (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        video_name TEXT,
                        video_path TEXT,
                        encoded_message TEXT,
                        encoded_date TEXT
                      )''')
    conn.commit()
    return conn, cursor
```

```
# Кодування повідомлення в біти
def message_to_bits(message):
    return ''.join(format(ord(char), '08b') for char in message)

# Перетворення бітів у текст
def bits_to_message(bits):
    chars = [chr(int(bits[i:i + 8], 2)) for i in range(0, len(bits), 8)]
    return ''.join(chars)

# PVD метод: вбудовування повідомлення у відео
def embed_message_pvd(frame, binary_message):
    message_index = 0
    rows, cols, _ = frame.shape
    for x in range(rows):
        for y in range(0, cols - 1, 2):  # Обробляємо пари пікселів
            if message_index >= len(binary_message):
                return frame
            pixel1 = frame[x, y]
            pixel2 = frame[x, y + 1]
            diff = abs(int(pixel1[0]) - int(pixel2[0]))
            if diff < 8:
                n_bits = 3
            if message_index + n_bits > len(binary_message):
                n_bits = len(binary_message) - message_index
            hidden_bits = binary_message[message_index:message_index + n_bits]
            hidden_value = int(hidden_bits, 2)
```

```
            new_diff = hidden_value

            if pixel1[0] > pixel2[0]:
                pixel1[0] = pixel2[0] + new_diff
            else:
                pixel2[0] = pixel1[0] + new_diff

            frame[x, y] = pixel1
            frame[x, y + 1] = pixel2
            message_index += n_bits

    return frame

# Основний клас програми
class SteganographyVideoApp:
    def __init__(self, root, cursor, conn):
        self.root = root
        self.conn = conn
        self.cursor = cursor
        self.media_path = None
        self.playing = False
        self.paused = False
        self.video_thread = None

        # Інтерфейс
        self.canvas = Canvas(root, width=500, height=300, bg='black')
        self.canvas.pack()
```

```
        # Додавання текстового поля з прокруткою
        text_frame = Frame(root)
        text_frame.pack()
        self.scrollbar = Scrollbar(text_frame)
        self.scrollbar.pack(side="right", fill="y")
        self.message_text = Text(text_frame, width=60, height=10, wrap="word",
yscrollcommand=self.scrollbar.set)
        self.message_text.pack(side="left", fill="both")
        self.scrollbar.config(command=self.message_text.yview)


        # Кнопки
        self.btn_frame = Frame(root)
        self.btn_frame.pack(pady=10)
        Button(self.btn_frame, text="      Вибрати Відео",
command=self.choose_video).pack(side="left", padx=5)
        Button(self.btn_frame, text="▶ Відтворити", command=self.play_video).pack(side="left",
padx=5)
        Button(self.btn_frame, text="    Пауза", command=self.pause_video).pack(side="left",
padx=5)
        Button(self.btn_frame, text="  Закодувати",
command=self.hide_message).pack(side="left", padx=5)
        Button(self.btn_frame, text="  Витягнути",
command=self.extract_message).pack(side="left", padx=5)
        Button(self.btn_frame, text="  Зберегти Текст",
command=self.save_text_to_file).pack(side="left", padx=5)
        Button(self.btn_frame, text="  Завантажити Текст",
command=self.load_text_from_file).pack(side="left", padx=5)
        Button(self.btn_frame, text="Очистити", command=self.clear_fields).pack(side="left",
padx=5)
```

```
    # Вибір відео
    def choose_video(self):
        file_path = filedialog.askopenfilename(title="Виберіть відео", filetypes=[("Video Files", "*.mp4;*.avi;*.mkv")])
        if file_path:
            self.media_path = file_path
            self.playing = False
            self.paused = False
            self.canvas.delete("all")

    # Відтворення відео
    def play_video(self):
        if not self.media_path:
            messagebox.showerror("Помилка", "Виберіть відео для відтворення!")
            return
        if not self.playing:
            self.playing = True
            self.video_thread = threading.Thread(target=self.video_loop)
            self.video_thread.start()
        else:
            self.paused = False

    # Пауза відео
    def pause_video(self):
        if self.playing:
            self.paused = True
```

```
# Цикл відтворення відео
def video_loop(self):
    cap = cv2.VideoCapture(self.media_path)
    while cap.isOpened() and self.playing:
        if self.paused:
            time.sleep(0.1)
            continue
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = cv2.resize(frame, (500, 300))
        img = ImageTk.PhotoImage(Image.fromarray(frame))

        self.canvas.create_image(250, 150, image=img)
        self.canvas.image = img
        time.sleep(1 / 30)  # ~30 FPS
    cap.release()
    self.playing = False

def hide_message(self):
    if not self.media_path:
        messagebox.showerror("Помилка", "Виберіть відео!")
        return

    message = self.message_text.get("1.0", END).strip()
```

```
        if not message:
            messagebox.showerror("Помилка", "Введіть текст для кодування!")
            return

        binary_message = message_to_bits(message) + '1111111111111110'
        cap = cv2.VideoCapture(self.media_path)
        save_path = filedialog.asksaveasfilename(defaultextension=".mp4", filetypes=[("MP4
Files", "*.mp4")])

        if not save_path:
            return

        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(save_path, fourcc, 30.0, (int(cap.get(3)), int(cap.get(4))))

        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            frame = embed_message_pvd(frame, binary_message)
            out.write(frame)

        cap.release()
        out.release()

        # Збереження в базу даних
        self.cursor.execute(
```

```
            "INSERT INTO videos (video_name, video_path, encoded_message, encoded_date)
VALUES (?, ?, ?, datetime('now'))",
            (os.path.basename(save_path), save_path, message)
        )
        self.conn.commit()
        messagebox.showinfo("Успіх", "Повідомлення закодовано у відео!")

    def extract_message(self):
        if not self.media_path:
            messagebox.showerror("Помилка", "Виберіть відео для витягнення повідомлення!")
            return

        # Витягуємо повідомлення з бази
        self.cursor.execute("SELECT encoded_message FROM videos WHERE video_path = ?",
(self.media_path,))
        row = self.cursor.fetchone()

        if row:
            extracted_message = row[0]
        else:
            extracted_message = "Повідомлення не знайдено в базі."

        self.message_text.delete("1.0", END)
        self.message_text.insert("1.0", extracted_message)
        messagebox.showinfo("Успіх", "Повідомлення витягнуто!")

    def save_text_to_file(self):
```

```
        text = self.message_text.get("1.0", END).strip()
        if not text:
            messagebox.showerror("Помилка", "Немає тексту для збереження!")
            return
        save_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files",
"*.txt")])
        if save_path:
            with open(save_path, "w", encoding="utf-8") as f:
                f.write(text)
            messagebox.showinfo("Успіх", "Текст збережено у файл!")

    def load_text_from_file(self):
        file_path = filedialog.askopenfilename(title="Завантажити текст", filetypes=[("Text Files",
"*.txt")])
        if file_path:
            with open(file_path, "r", encoding="utf-8") as f:
                text = f.read()
            self.message_text.delete("1.0", END)
            self.message_text.insert("1.0", text)
            messagebox.showinfo("Успіх", "Текст завантажено у текстове поле!")

    def clear_fields(self):
        self.message_text.delete("1.0", END)
        self.canvas.delete("all")
        self.media_path = None
        self.playing = False
        self.paused = False
```

```
# Запуск програми
if __name__ == "__main__":
    root = Tk()
    root.title("Приховування тексту у відео з функціями завантаження/збереження")
    conn, cursor = init_db()  # Ініціалізація бази даних
    app = SteganographyVideoApp(root, cursor, conn)  # Створення екземпляра класу
    root.mainloop()
```