

Università degli Studi di Catania

Dipartimento di Matematica e Informatica

Corso di Multimedia e Laboratorio  
Relazione Progetto

---

# Steganography WebApp

Applicazione Web per la Steganografia  
con Algoritmi Multipli

---

*Autore:*

**Giuseppe BELLAMACINA**  
Matricola: 1000030349

*Professori del corso:*

**Dario ALLEGRA**  
**Filippo STANCO**

Anno Accademico 2025/2026

# Abstract

Questo progetto presenta lo sviluppo di un'applicazione web completa per la steganografia, implementata utilizzando Python e Streamlit. L'applicazione offre tre algoritmi steganografici avanzati: LSB (Least Significant Bit), DWT (Discrete Wavelet Transform) e PVD (Pixel Value Differencing), permettendo agli utenti di nascondere e recuperare diversi tipi di dati (testo, immagini e file binari) all'interno di immagini.

L'architettura del progetto segue principi di clean code e modularità, separando la logica di business dall'interfaccia utente. Ogni algoritmo è implementato in modo indipendente, offrendo configurazioni personalizzabili e preset ottimizzati per bilanciare capacità, qualità e robustezza.

Il sistema include funzionalità avanzate come il calcolo di metriche di qualità (PSNR, SSIM), un sistema di backup automatico dei parametri e validazione completa degli input. L'interfaccia web, sviluppata con Streamlit, offre un'esperienza utente intuitiva con visualizzazione in tempo reale delle operazioni.

# Indice

<b>Abstract</b>	<b>1</b>
<b>Introduzione</b>	<b>6</b>
<b>1 Fondamenti di Steganografia</b>	<b>7</b>
1.1 Definizione e Storia . . . . .	7
1.2 Steganografia vs Crittografia . . . . .	7
1.3 Requisiti di un Sistema Steganografico . . . . .	8
1.3.1 Impercettibilità (Imperceptibility) . . . . .	8
1.3.2 Capacità (Capacity) . . . . .	8
1.3.3 Robustezza (Robustness) . . . . .	8
1.4 Classificazione degli Algoritmi Steganografici . . . . .	9
1.4.1 Algoritmi nel Dominio Spaziale . . . . .	9
1.4.2 Algoritmi nel Dominio della Frequenza . . . . .	9
<b>2 Algoritmi Steganografici Implementati</b>	<b>10</b>
2.1 LSB - Least Significant Bit . . . . .	10
2.1.1 Principio di Funzionamento . . . . .	10
2.1.2 Implementazione . . . . .	10
2.1.3 Capacità e Prestazioni . . . . .	11
2.1.4 Vantaggi e Limitazioni . . . . .	11
2.2 DWT - Discrete Wavelet Transform . . . . .	12
2.2.1 Principio di Funzionamento . . . . .	12
2.2.2 Implementazione . . . . .	12
2.2.3 Configurazioni Implementate . . . . .	14
2.2.4 Caratteristiche dell'Approccio . . . . .	14
2.3 PVD - Pixel Value Differencing . . . . .	14
2.3.1 Principio di Funzionamento . . . . .	14
2.3.2 Range di Quantizzazione . . . . .	15
2.3.3 Algoritmo di Embedding . . . . .	15
2.3.4 Parametri Configurabili . . . . .	16

<i>INDICE</i>	3
---------------	---

2.3.5 Caratteristiche dell'Approccio . . . . .	16
2.4 Confronto tra gli Algoritmi . . . . .	17
<b>3 Architettura del Sistema</b>	<b>18</b>
3.1 Panoramica dell'Architettura . . . . .	18
3.1.1 Schema Architettonico . . . . .	19
3.2 Pattern Architettonici Utilizzati . . . . .	19
3.2.1 Strategy Pattern . . . . .	19
3.2.2 Singleton Pattern . . . . .	20
3.2.3 Factory Pattern . . . . .	20
3.3 Separazione delle Responsabilità . . . . .	21
3.3.1 Business Logic Layer . . . . .	21
3.3.2 Utility Layer . . . . .	21
3.3.3 Configuration Layer . . . . .	21
3.4 Gestione dello Stato . . . . .	21
3.5 Gestione degli Errori . . . . .	22
3.5.1 Validazione Preventiva . . . . .	22
3.5.2 Try-Catch nelle Operazioni Critiche . . . . .	22
3.6 Estensibilità . . . . .	23
<b>4 Implementazione</b>	<b>24</b>
4.1 Tecnologie Utilizzate . . . . .	24
4.1.1 Stack Tecnologico . . . . .	24
4.1.2 Gestione Dipendenze con uv . . . . .	24
4.2 Implementazione degli Algoritmi . . . . .	25
4.2.1 Sistema di Conversione Binaria . . . . .	25
4.2.2 Manipolazione Bit LSB . . . . .	25
4.2.3 Sistema di Backup Parametri . . . . .	26
4.2.4 Calcolo Metriche di Qualità . . . . .	27
4.3 Ottimizzazioni Implementate . . . . .	28
4.3.1 Pre-validation Capacita . . . . .	28
4.4 Gestione File Temporanei . . . . .	29
4.5 Testing e Quality Assurance . . . . .	29
4.5.1 Code Formatting . . . . .	29
4.5.2 Continuous Integration . . . . .	30
4.6 Deploy su Streamlit Cloud . . . . .	30
4.7 Test Sperimentali LSB - Occultamento Immagini . . . . .	31
4.7.1 Setup Sperimentale . . . . .	31
4.7.2 Test 1: Configurazione Bilanciata (LSB=4, MSB=4) . .	32
4.7.3 Test 2: Alta Qualità (LSB=1, MSB=1) . . . . .	33
4.7.4 Test 3: Alta Capacità (LSB=6, MSB=2) . . . . .	34

4.7.5	Test 4: Massima Capacità Estrema (LSB=7, MSB=8) . . . . .	35
4.7.6	Test 5: Modalità Automatica (LSB=auto, MSB=8) . . . . .	36
4.7.7	Conclusioni sui Test LSB . . . . .	36
4.8	Test Sperimentali LSB - Occultamento File Binari . . . . .	37
4.8.1	Setup Sperimentale . . . . .	37
4.8.2	Test 1: Configurazione Alta Qualità (N=2) . . . . .	38
4.8.3	Test 2: Configurazione Bilanciata (N=4) . . . . .	39
4.8.4	Test 3: Configurazione Alta Capacità (N=6) . . . . .	40
4.8.5	Test 4: Configurazione Estrema con DIV Automatico (N=8) . . . . .	41
4.8.6	Test 5: Configurazione Estrema con DIV=1 (N=8) . . . . .	42
4.8.7	Conclusioni sui Test LSB Binary . . . . .	43
4.9	Test Sperimentali DWT - Occultamento File Binari . . . . .	43
4.9.1	Setup Sperimentale . . . . .	43
4.9.2	Test 1: Alta Qualità - Configurazione Conservativa . . . . .	44
4.9.3	Test 2: Capacità Aumentata - Configurazione Estesa . . . . .	45
4.9.4	Test 3: Massima Capacità - Configurazione Aggressiva . . . . .	46
4.9.5	Confronto DWT vs LSB per File Binari . . . . .	47
4.9.6	Conclusioni sui Test DWT Binary . . . . .	47
4.10	Test Sperimentali PVD - Occultamento File Binari . . . . .	47
4.10.1	Parametri PVD per File Binari . . . . .	48
4.10.2	Impatto dei Parametri . . . . .	48
4.10.3	Esempi di Configurazione . . . . .	49
4.10.4	Considerazioni sulla Capacità . . . . .	51
4.10.5	Conclusioni sui Test PVD Binary . . . . .	51
<b>5</b>	<b>Interfaccia Utente</b> . . . . .	<b>52</b>
5.1	Design dell'Interfaccia . . . . .	52
5.2	Struttura dell'Interfaccia . . . . .	52
5.2.1	Header e Branding . . . . .	52
5.2.2	Sidebar - Selezione Metodo . . . . .	53
5.2.3	Selezione Tipo di Dato . . . . .	53
5.3	Workflow Utente . . . . .	54
5.3.1	Operazione Hide (Nascondere) . . . . .	54
5.3.2	Operazione Recover (Recuperare) . . . . .	54
5.4	Componenti UI Riutilizzabili . . . . .	55
5.4.1	File Upload con Anteprima . . . . .	55
5.4.2	Download Button con Icone . . . . .	55
5.4.3	Preset Configurabili . . . . .	56
5.5	Feedback Visivo e User Experience . . . . .	56
5.5.1	Indicatori di Progresso . . . . .	56

5.5.2	Visualizzazione Metriche . . . . .	57
5.5.3	Messaggi di Errore Informativi . . . . .	57
5.6	Responsività e Accessibilità . . . . .	57
5.6.1	Stili CSS Personalizzati . . . . .	58
5.7	Gestione dello Stato tra Pagine . . . . .	58
<b>6</b>	<b>Conclusioni</b>	<b>60</b>
6.1	Riepilogo del Progetto . . . . .	60
6.1.1	Obiettivi Raggiunti . . . . .	60
6.2	Confronto Prestazioni . . . . .	60
6.2.1	Risultati Sperimentali . . . . .	60
6.2.2	Trade-off Osservati . . . . .	61
6.3	Contributi Originali . . . . .	61
6.4	Limitazioni e Miglioramenti Futuri . . . . .	62
6.4.1	Limitazioni Attuali . . . . .	62
6.4.2	Sviluppi Futuri Proposti . . . . .	62
6.5	Considerazioni Finali . . . . .	63

# Introduzione

## Contesto e Motivazioni

La steganografia è l'arte e la scienza di nascondere informazioni all'interno di altri dati apparentemente innocui, in modo che la presenza stessa del messaggio nascosto sia difficile da rilevare [4]. A differenza della crittografia, che rende il messaggio illeggibile ma non ne nasconde l'esistenza, la steganografia mira a mascherare completamente la comunicazione segreta.

Nel contesto moderno della sicurezza informatica e della privacy digitale, la steganografia trova applicazioni in diversi ambiti:

- **Protezione del copyright:** Watermarking digitale per proteggere la proprietà intellettuale di immagini, video e documenti
- **Comunicazioni sicure:** Trasmissione di informazioni sensibili senza destare sospetti
- **Autenticazione:** Verifica dell'integrità e dell'autenticità di contenuti multimediali
- **Privacy:** Protezione di dati personali in contesti dove la crittografia potrebbe attirare attenzione indesiderata

## Obiettivi del Progetto

Il progetto consiste nello sviluppo di un'applicazione web che implementa tre algoritmi steganografici (LSB, DWT e PVD) per nascondere testo, immagini e file binari all'interno di immagini. L'applicazione offre preset ottimizzati per semplificare l'uso e calcola metriche di qualità (PSNR e SSIM) per valutare i risultati. Un sistema di backup automatico facilita il recupero dei dati nascosti.

# Capitolo 1

## Fondamenti di Steganografia

### 1.1 Definizione e Storia

La steganografia deriva dal greco *steganos* (coperto) e *graphein* (scrittura), letteralmente "scrittura nascosta". Le sue origini risalgono all'antica Grecia, dove si utilizzavano tecniche ingegnose quali tatuaggi sul cuoio capelluto rasato di messaggeri, inchiostri invisibili e messaggi nascosti in tavole di cera. Con l'avvento dell'era digitale, la steganografia ha trovato nuove applicazioni nel dominio dei file multimediali, trasformandosi da arte artigianale a scienza computazionale precisa [5].

### 1.2 Steganografia vs Crittografia

La distinzione tra steganografia e crittografia è fondamentale per comprendere i diversi approcci alla sicurezza dell'informazione. Mentre la crittografia si concentra nel rendere illeggibile un messaggio, attirando inevitabilmente l'attenzione sulla comunicazione, la steganografia mira a nasconderne completamente l'esistenza. Questo approccio discreto comporta alcuni compromessi: la capacità di dati che può essere nascosta è tipicamente limitata dalla dimensione del contenitore, e la robustezza alle manipolazioni varia significativamente a seconda della tecnica impiegata. Tuttavia, il vantaggio principale risiede proprio nel basso livello di sospetto generato: un'immagine contenente dati nascosti appare del tutto ordinaria a un osservatore [7].

Le due tecniche possono essere efficacemente combinate, cifrando prima il messaggio e poi nascondendolo, ottenendo così un doppio livello di sicurezza che protegge sia il contenuto che l'esistenza stessa della comunicazione.

## 1.3 Requisiti di un Sistema Steganografico

Un sistema steganografico efficace deve soddisfare tre requisiti fondamentali:

### 1.3.1 Impercettibilità (Imperceptibility)

Le modifiche apportate al contenitore devono rimanere impercettibili all'osservazione umana. La valutazione quantitativa di questo requisito si affida a metriche consolidate nel campo dell'elaborazione delle immagini. Il PSNR (Peak Signal-to-Noise Ratio) misura il rapporto tra il segnale massimo possibile e il rumore di distorsione; valori superiori a 30 dB indicano differenze generalmente impercettibili:

$$PSNR = 10 \cdot \log_{10} \left( \frac{MAX^2}{MSE} \right)$$

dove  $MAX$  rappresenta il valore massimo del pixel (255 per immagini a 8 bit) e  $MSE$  l'errore quadratico medio tra i pixel originali e modificati. Complementare al PSNR, l'indice SSIM (Structural Similarity Index) valuta la similarità strutturale percettiva tra le immagini su una scala da -1 a 1, dove 1 indica identità perfetta.

### 1.3.2 Capacità (Capacity)

La quantità di informazioni che possono essere nascoste nell'immagine host. Si misura in:

- **Bit per pixel (bpp):** Rapporto tra bit nascosti e pixel totali
- **Percentuale di utilizzo:** Frazione dell'immagine modificata

Esiste sempre un trade-off tra capacità e impercettibilità: aumentare la quantità di dati nascosti aumenta il rischio di degradazione visibile.

### 1.3.3 Robustezza (Robustness)

La capacità del messaggio nascosto di resistere a manipolazioni come:

- Compressione JPEG
- Ridimensionamento
- Rotazione e ritaglio

- Aggiunta di rumore
- Filtri di elaborazione

Algoritmi diversi offrono diversi livelli di robustezza a seconda del dominio di embedding utilizzato (spaziale vs frequenza).

## 1.4 Classificazione degli Algoritmi Steganografici

Gli algoritmi steganografici per immagini si possono classificare in due categorie principali:

### 1.4.1 Algoritmi nel Dominio Spaziale

Modificano direttamente i valori dei pixel. Sono generalmente:

- Veloci da implementare
- Ad alta capacità
- Poco robusti a manipolazioni
- Esempi: LSB, PVD

### 1.4.2 Algoritmi nel Dominio della Frequenza

Operano sui coefficienti di trasformazioni matematiche (DCT, DWT, DFT). Sono generalmente:

- Più complessi da implementare
- A capacità inferiore
- Più robusti a compressione e filtri
- Esempi: DWT, DCT-based

# Capitolo 2

## Algoritmi Steganografici Implementati

### 2.1 LSB - Least Significant Bit

#### 2.1.1 Principio di Funzionamento

L'algoritmo LSB (Least Significant Bit) è uno dei metodi steganografici più diffusi grazie alla sua semplicità e alta capacità [1]. Il principio si basa sull'osservazione che modificare i bit meno significativi dei valori dei pixel produce cambiamenti impercettibili all'occhio umano.

In un'immagine RGB a 8 bit per canale, ogni pixel ha valori tra 0 e 255. Modificare l'ultimo bit (LSB) cambia il valore del pixel al massimo di  $\pm 1$ , una differenza invisibile.

Esempio:

- Pixel originale:  $11010110_2 = 214_{10}$
- Bit da nascondere: 1
- Pixel modificato:  $11010111_2 = 215_{10}$

#### 2.1.2 Implementazione

L'implementazione nel progetto utilizza un approccio robusto con header strutturato:

```
1 magic_header = "101010101110000"
2 msg_length = format(len(message), "032b")
3 checksum = format(xor_checksum, "016b")
4 msg_binary = binary_convert(message)
```

```

5  terminator = "1111000011110000"
6
7 full_payload = magic_header + msg_length + checksum +
    msg_binary + terminator

```

**Codice 2.1:** Struttura del payload LSB

Il processo di embedding scorre i pixel dell'immagine modificando l'LSB di ogni componente RGB:

```

1  for i in range(img.width):
2      for j in range(img.height):
3          for z in range(3): # R, G, B
4              if msg_list:
5                  bit = msg_list.pop(0)
6                  pixel = mat[i, j]
7                  color = int(pixel[z])
8                  color = set_last_bit(color, bit)
9                  mat = set_color_component(mat, i, j, color, z)

```

**Codice 2.2:** Nascondimento LSB

### 2.1.3 Capacità e Prestazioni

Per un'immagine di dimensione  $W \times H$  pixel:

- **Capacità teorica massima:**  $W \times H \times 3$  bit (3 bit per pixel RGB)
- **Overhead header:** 80 bit fissi (magic + length + checksum + terminator)
- **Capacità effettiva:**  $W \times H \times 3 - 80$  bit

**Esempio pratico:**

- Immagine  $800 \times 600$ : 1,440,000 bit teorici = 180 KB
- Capacità effettiva: 179.99 KB
- Percentuale uso tipico: 0.1-5% per messaggi normali

### 2.1.4 Vantaggi e Limitazioni

L'algoritmo LSB si distingue per la sua semplicità implementativa e l'elevata capacità di embedding, offrendo tipicamente PSNR superiori a 50 dB che garantiscono modifiche del tutto impercettibili all'occhio umano. Tuttavia, questa efficienza ha un costo in termini di robustezza: qualsiasi compressione

JPEG o manipolazione dell'immagine distrugge i bit meno significativi, rendendo impossibile il recupero dei dati. Inoltre, l'approccio è vulnerabile ad analisi statistiche che possono rilevare la presenza di dati nascosti attraverso anomalie nella distribuzione dei valori dei pixel.

## 2.2 DWT - Discrete Wavelet Transform

### 2.2.1 Principio di Funzionamento

La DWT (Discrete Wavelet Transform) è una trasformazione matematica che decomponete un segnale (o immagine) in coefficienti che rappresentano informazioni a diverse scale e posizioni. A differenza di LSB che opera nel dominio spaziale, DWT lavora nel dominio della frequenza.

La trasformata wavelet 2D decomponete un'immagine in quattro sub-bande:

- **cA (Approximation)**: Coefficienti di approssimazione - contiene informazioni a bassa frequenza
- **cH (Horizontal)**: Dettagli orizzontali - bordi verticali
- **cV (Vertical)**: Dettagli verticali - bordi orizzontali
- **cD (Diagonal)**: Dettagli diagonali - componenti ad alta frequenza

### 2.2.2 Implementazione

Il progetto utilizza la wavelet di Haar per la sua semplicità e efficienza [6]. L'implementazione usa `pywt.dwt2` per decomposizione single-level, garantendo efficienza e capacità prevedibile.

```

1 import pywt
2
3 # Applica DWT 2D single-level al canale dell'immagine
4 coeffs = pywt.dwt2(channel_data, 'haar')
5 cA, (cH, cV, cD) = coeffs
6
7 # Nasconde nei coefficienti orizzontali (cH)
8 cH_flat = cH.flatten()

```

Codice 2.3: Decomposizione DWT

#### Header Robusto a 64-bit:

Per evitare false positive detection nel rumore dei coefficienti DWT, il sistema usa un magic header a 64-bit invece del tradizionale 16-bit:

```

1 # Header 64-bit per robustezza
2 MAGIC_HEADER_64 = "
3     11001001000111101011001010011001101010101001110000101011001101
4 "
5 SIZE_BITS = 32 # Dimensione payload in bit
6
7 # Struttura: [64-bit magic][32-bit size][payload]
8 full_payload = MAGIC_HEADER_64 + format(payload_size, "032b")
9     + payload_bits

```

Codice 2.4: Header DWT robusto

**Estrazione Two-Phase:**

L'estrazione avviene in due fasi sincronizzate per evitare lettura di rumore:

```

1 # FASE 1: Estraie header + size (96 bit totali)
2 header_and_size = extract_bits_from_coefficients(96)
3 magic = header_and_size[:64]
4 payload_size = int(header_and_size[64:96], 2)
5
6 # Verifica header
7 if magic != MAGIC_HEADER_64:
8     raise ValueError("Header non trovato")
9
10 # FASE 2: Estraie esattamente payload_size bit
11 payload = extract_bits_from_coefficients(payload_size)

```

Codice 2.5: Estrazione two-phase DWT

Questo approccio garantisce che l'estrazione si fermi esattamente dopo il payload, evitando di leggere coefficienti non modificati che causerebbero corruzione dei dati.

L'embedding avviene modificando i coefficienti in base al bit da nascondere:

```

1 # Modifica il coefficiente usando ALPHA
2 bit = int(payload_bit)
3 delta = ALPHA * 50 # Scala per robustezza
4
5 if bit == 1:
6     cH_flat[i] += delta if cH_flat[i] > 0 else -delta
7 else:
8     cH_flat[i] -= delta if cH_flat[i] > 0 else -delta

```

Codice 2.6: Embedding DWT

Il parametro **ALPHA** controlla la forza dell'embedding:

- $\alpha = 0.05$ : Qualità massima, capacità ridotta

- $\alpha = 0.1$ : Bilanciato (default)
- $\alpha = 0.15$ : Robustezza massima, più visibile

### 2.2.3 Configurazioni Implementate

Il sistema offre tre preset configurabili:

Preset	ALPHA	Bande	Canali
Qualità Massima	0.05	cH	R
Bilanciato	0.10	cH	R
Capacità Massima	0.15	cH, cV, cD	R, G, B

Tabella 2.1: Configurazioni preset DWT

### 2.2.4 Caratteristiche dell'Approccio

Il principale punto di forza della DWT risiede nella sua robustezza: operando nel dominio della frequenza, l'algoritmo resiste a compressioni JPEG, filtri e aggiunte di rumore che distruggerebbero completamente i dati nascosti con LSB. Questa robustezza si paga con una capacità di embedding ridotta e un costo computazionale superiore. Il PSNR tipicamente si attesta tra 35 e 45 dB, comunque sufficiente per garantire modifiche visivamente accettabili. La calibrazione del parametro ALPHA richiede un bilanciamento attento tra robustezza e qualità visiva, rendendo i preset configurabili particolarmente utili per utenti meno esperti.

## 2.3 PVD - Pixel Value Differencing

### 2.3.1 Principio di Funzionamento

PVD (Pixel Value Differencing) è un algoritmo adattivo che sfrutta la differenza tra pixel adiacenti per nascondere dati. L'idea chiave è che modifiche maggiori sono meno percepibili in regioni con alti contrasti (bordi), mentre in regioni uniformi sono necessarie modifiche minori.

Il metodo divide l'immagine in coppie di pixel e calcola la loro differenza [10]:

$$d = |p_2 - p_1|$$

In base alla differenza, viene determinata la capacità di embedding usando range quantizzati.

### 2.3.2 Range di Quantizzazione

L'implementazione offre due profili di range:

**Profile Qualità (default):**

```

1 RANGES_QUALITY = [
2     (0, 7, 2),
3     (8, 15, 3),
4     (16, 31, 3),
5     (32, 63, 4),
6     (64, 127, 4),
7 ]

```

**Codice 2.7:** Range qualità PVD

**Profile Capacità:**

```

1 RANGES_CAPACITY = [
2     (0, 7, 3),
3     (8, 15, 3),
4     (16, 31, 4),
5     (32, 63, 5),
6     (64, 127, 6),
7     (128, 255, 7),
8 ]

```

**Codice 2.8:** Range capacità PVD

### 2.3.3 Algoritmo di Embedding

```

1 def embed_in_pair(pixel1, pixel2, bits):
2     diff = pixel2 - pixel1
3     lower, upper, capacity = get_range_capacity(diff)
4
5     decimal_value = int(bits[:capacity], 2)
6     new_diff = lower + decimal_value
7     new_diff = min(new_diff, upper)
8
9     if diff < 0:
10         new_diff = -new_diff
11
12     m = abs(new_diff) - abs(diff)
13     if diff % 2 == 0:
14         new_pixel1 = pixel1 - m // 2
15         new_pixel2 = pixel2 + m - m // 2
16     else:
17         new_pixel1 = pixel1 - (m + 1) // 2
18         new_pixel2 = pixel2 + m - (m + 1) // 2
19

```

```

20     return max(0, min(255, new_pixel1)), max(0, min(255,
21         new_pixel2))

```

**Codice 2.9:** PVD embedding in una coppia di pixel

### 2.3.4 Parametri Configurabili

L'implementazione offre tre parametri principali. Il profilo di quantizzazione `RANGES` determina la capacità di embedding per ogni livello di differenza tra pixel. Il fallback utilizza l'ultimo range definito (`RANGES[-1]`) invece di un valore hardcoded, garantendo compatibilità tra modalità `QUALITY` e `CAPACITY`.

La spaziatura `PAIR_STEP` controlla la distanza tra coppie consecutive di pixel. Con `PAIR_STEP=1` si ottiene la massima capacità utilizzando tutti i pixel disponibili, mentre valori superiori distribuiscono l'embedding su un'area più ampia riducendo la capacità. I loop di attraversamento usano `range(0, w - PAIR_STEP, 2*PAIR_STEP)` per prevenire accessi out-of-bounds quando si legge `pixel[x + PAIR_STEP]`.

Il parametro `CHANNELS` specifica quali canali RGB utilizzare. Il calcolo di `total_bits` impiega `len(channels)` per determinare precisamente la capacità disponibile indipendentemente dalla configurazione.

L'implementazione richiede particolare attenzione alla sincronizzazione tra le fasi di embedding ed estrazione. Durante il processo di inserimento, l'indice deve essere incrementato esattamente del numero di bit effettivamente inseriti, e non della capacità teorica della coppia. Questo accorgimento previene desincronizzazioni quando il payload termina prima di saturare l'ultima coppia di pixel disponibile.

```

1 bits_to_embed = payload[bit_index : bit_index + capacity]
2 bit_index += len(bits_to_embed)

```

**Codice 2.10:** Incremento dell'indice durante embedding

### 2.3.5 Caratteristiche dell'Approccio

PVD rappresenta un interessante compromesso tra la semplicità di LSB e la robustezza di DWT. La natura adattiva dell'algoritmo, che modula la quantità di dati nascosti in base al contenuto locale dell'immagine, produce una distribuzione delle modifiche più naturale e meno rilevabile. Il PSNR risulta generalmente superiore a quello di LSB semplice, attestandosi tra 45 e 55 dB. Tuttavia, la complessità implementativa è maggiore e la sincronizzazione tra

fasi di embedding ed estrazione deve essere gestita con precisione. L'algoritmo mostra vulnerabilità a operazioni di ridimensionamento e ritaglio, che alterano la struttura delle coppie di pixel su cui si basa il metodo.

## 2.4 Confronto tra gli Algoritmi

Caratteristica	LSB	DWT	PVD
Capacità	Alta (3 bpp)	Media (0.5-1 bpp)	Alta (2-4 bpp)
PSNR medio	>50 dB	35-45 dB	45-55 dB
Velocità	Velocissimo	Lento	Medio
Robustezza	Bassa	Alta	Media
Complessità	Bassa	Alta	Media
Dominio	Spaziale	Frequenza	Spaziale

Tabella 2.2: Confronto quantitativo tra gli algoritmi

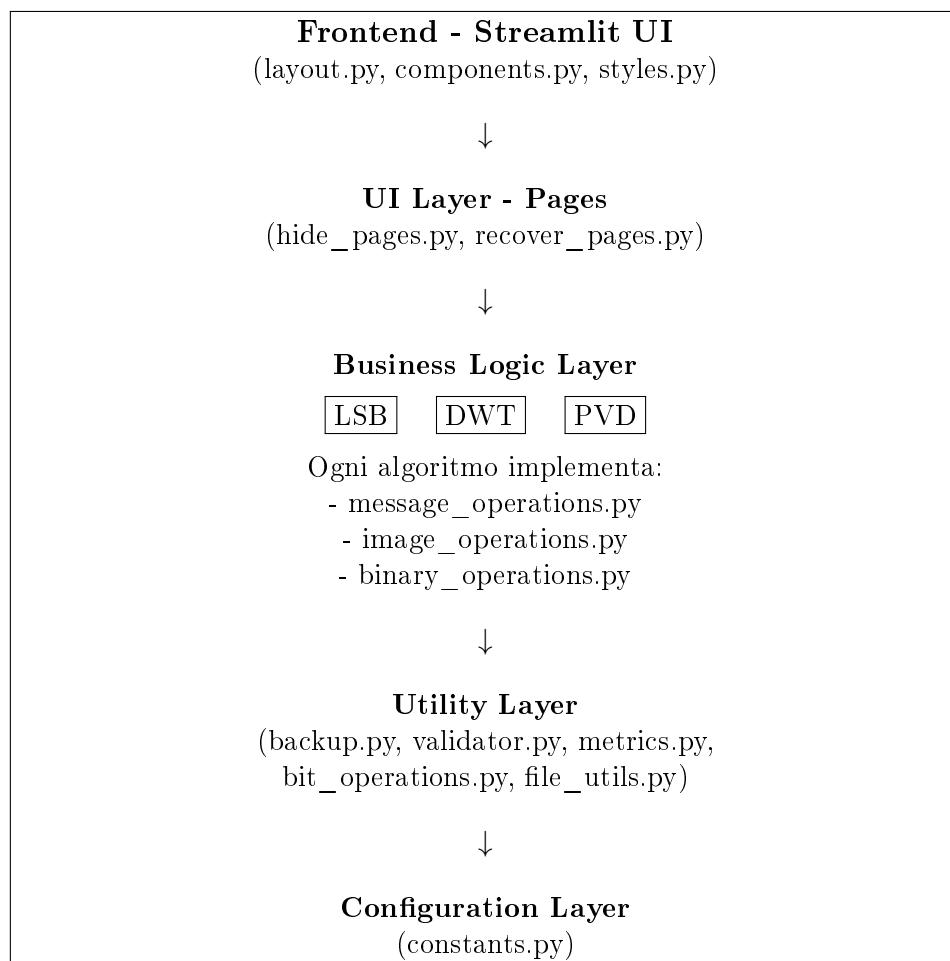
# Capitolo 3

## Architettura del Sistema

### 3.1 Panoramica dell'Architettura

Il progetto segue un'architettura modulare a strati con separazione netta tra logica di business e presentazione. La struttura è stata progettata per garantire modularità, assegnando a ciascun componente responsabilità ben definite. Questa organizzazione facilita l'estensione del sistema con nuovi algoritmi o funzionalità, mantenendo il codice pulito e ben documentato. I componenti isolati risultano inoltre facilmente testabili, migliorando complessivamente la manutenibilità del progetto.

### 3.1.1 Schema Architetturale



**Figura 3.1:** Architettura a strati del sistema

## 3.2 Pattern Architetturali Utilizzati

### 3.2.1 Strategy Pattern

Ogni algoritmo (LSB, DWT, PVD) implementa la stessa interfaccia per le tre operazioni:

- hide\_message() / get\_message()
- hide\_image() / get\_image()
- hide\_bin\_file() / get\_bin\_file()

Questo permette al sistema di selezionare dinamicamente l'algoritmo a runtime:

```

1 if selected_method == SteganographyMethod.LSB:
2     from src.steganografia.lsb import message_operations as
3         ops
4 elif selected_method == SteganographyMethod.DWT:
5     from src.steganografia.dwt import message_operations as
6         ops
7 elif selected_method == SteganographyMethod.PVD:
8     from src.steganografia.pvd import message_operations as
9         ops
10
11 result = ops.MessageSteganography.hide_message(img, msg)

```

**Codice 3.1:** Selezione dinamica algoritmo

### 3.2.2 Singleton Pattern

Il sistema di backup e le classi di utility usano metodi statici per evitare istanziazioni multiple:

```

1 class ParameterBackup:
2     @staticmethod
3     def save_backup_data(data_type, params, filepath):
4         ...
5
6 class QualityMetrics:
7     @staticmethod
8     def calculate_psnr(original, modified):
9         ...

```

**Codice 3.2:** Utility classes come singleton

### 3.2.3 Factory Pattern

La creazione di componenti UI avviene attraverso factory methods nella classe `AppLayout`:

```

1 class AppLayout:
2     @staticmethod
3     def setup_sidebar():
4         # Crea e configura sidebar con cards metodi
5
6     @staticmethod
7     def show_data_type_selector():
8         # Crea selector tipo dato

```

**Codice 3.3:** Factory methods per UI

## 3.3 Separazione delle Responsabilità

### 3.3.1 Business Logic Layer

Contiene tutta la logica steganografica, completamente indipendente dall'UI. Ciascun modulo gestisce un tipo specifico di dato: `message_operations.py` per l'embedding e l'estrazione di stringhe, `image_operations.py` per le immagini, e `binary_operations.py` per file binari arbitrari.

### 3.3.2 Utility Layer

Fornisce servizi trasversali utilizzati da tutti gli algoritmi. Il modulo `backup.py` gestisce serializzazione e deserializzazione dei parametri in formato JSON, mentre `validator.py` si occupa della validazione di dimensioni, formati e capacità. Il calcolo delle metriche PSNR e SSIM è centralizzato in `metrics.py`, le operazioni binarie e conversioni in `bit_operations.py`, e la gestione I/O dei file in `file_utils.py`.

### 3.3.3 Configuration Layer

Centralizza tutte le costanti e configurazioni:

```

1  class SteganographyMethod(str, Enum):
2      LSB = "lsb"
3      DWT = "dwt"
4      PVD = "pvd"
5
6  class DataType(str, Enum):
7      STRING = "string"
8      IMAGE = "image"
9      BINARY = "binary"
10
11 class CompressionMode(str, Enum):
12     NO_ZIP = "no_zip"
13     FILE = "file"
14     DIR = "dir"
```

**Codice 3.4:** Enumerazioni di configurazione

## 3.4 Gestione dello Stato

L'applicazione utilizza Streamlit Session State per mantenere lo stato tra le interazioni:

```

1 # Inizializzazione
2 if "selected_method" not in st.session_state:
3     st.session_state.selected_method = SteganographyMethod.LSB
4
5 # Memorizzazione risultati
6 st.session_state["hide_image_results"] = {
7     "image": img_data,
8     "preview_image": img,
9     "metrics": {"psnr": 45.2, "ssim": 0.998}
10 }
11
12 # Recupero risultati
13 if "hide_image_results" in st.session_state:
14     results = st.session_state["hide_image_results"]

```

**Codice 3.5:** Gestione stato con Streamlit

## 3.5 Gestione degli Errori

Il sistema implementa una gestione robusta degli errori a più livelli:

### 3.5.1 Validazione Preventiva

```

1 class ParameterValidator:
2     @staticmethod
3     def validate_image_size_for_message(img, message):
4         required_bits = len(message) * 8 + 80 # +header
5         available_bits = img.width * img.height * 3
6
7         if required_bits > available_bits:
8             raise ValueError(
9                 f"Messaggio troppo lungo."
10                f"Richiesti: {required_bits} bit, "
11                f"Disponibili: {available_bits} bit"
12            )

```

**Codice 3.6:** Validazione input

### 3.5.2 Try-Catch nelle Operazioni Critiche

```

1 try:
2     result_img, metrics = hide_message(img, message,
3                                         backup_file)
4     st.success("Messaggio nascosto con successo!")
5     # Mostra risultati...

```

```
5 except ValueError as e:
6     st.error(f"Errore di validazione: {str(e)}")
7 except Exception as e:
8     st.error(f"Errore imprevisto: {str(e)}")
9     # Log per debugging...
```

Codice 3.7: Gestione eccezioni nelle pagine UI

## 3.6 Estensibilità

L’architettura facilita l’aggiunta di nuovi algoritmi:

1. Creare nuova directory sotto `src/steganografia/`
2. Implementare i tre moduli di operazioni
3. Aggiungere enum in `SteganographyMethod`
4. Aggiornare UI per includere il nuovo metodo

Non sono richieste modifiche agli altri moduli grazie all’isolamento tra componenti.

# Capitolo 4

## Implementazione

### 4.1 Tecnologie Utilizzate

#### 4.1.1 Stack Tecnologico

Il progetto è sviluppato interamente in Python 3.12 (ma è retrocompatibile con versioni precedenti dalla 3.9 in poi) utilizzando le seguenti librerie [8, 3, 9]:

Libreria	Versione	Scopo
Streamlit	$\geq 1.20.0$	Framework web per l'interfaccia utente
NumPy	$\geq 1.24.0$	Operazioni matriciali e calcoli numerici
Pillow (PIL)	$\geq 9.5.0$	Manipolazione e I/O immagini
PyWavelets	$\geq 1.4.0$	Trasformata wavelet per DWT
scikit-image	$\geq 0.20.0$	Metriche di qualità (SSIM)

Tabella 4.1: Dipendenze principali del progetto

#### 4.1.2 Gestione Dipendenze con uv

Il progetto utilizza uv come package manager per gestione rapida e affidabile delle dipendenze:

```
1 [project]
2 name = "steganography-webapp"
3 version = "1.0.0"
4 requires-python = ">=3.9"
5
6 dependencies = [
7     "streamlit>=1.20.0",
```

```

8     "numpy>=1.24.0",
9     "Pillow>=9.5.0",
10    "PyWavelets>=1.4.0",
11    "scikit-image>=0.20.0",
12 ]
13
14 [project.optional-dependencies]
15 dev = [
16     "black>=23.0.0",
17     "isort>=5.12.0",
18 ]
19
20 [tool.uv]
21 dev-dependencies = [
22     "black>=23.0.0",
23     "isort>=5.12.0",
24 ]

```

Codice 4.1: pyproject.toml - configurazione dipendenze

## 4.2 Implementazione degli Algoritmi

### 4.2.1 Sistema di Conversione Binaria

Tutti gli algoritmi si basano su funzioni di conversione tra testo e binario:

```

1 def binary_convert(s: str) -> str:
2     """Converte una stringa in rappresentazione binaria"""
3     binary = "".join(format(ord(char), "08b") for char in s)
4     return binary
5
6 def binary_convert_back(s: str) -> str:
7     """Riconverte binario in stringa"""
8     chars = [s[i:i+8] for i in range(0, len(s), 8)]
9     text = "".join(chr(int(char, 2)) for char in chars)
10    return text

```

Codice 4.2: Conversione testo-binario

### 4.2.2 Manipolazione Bit LSB

Operazioni fondamentali per l'algoritmo LSB:

```

1 def set_last_bit(value: int, bit: str) -> int:
2     """Imposta l'ultimo bit di un valore"""
3     if bit == "1":
4         return value | 1 # OR bit-wise per settare a 1

```

```
5     else:
6         return value & ~1 # AND con NOT per settare a 0
7
8 def get_last_bit(value: int) -> str:
9     """Estraie l'ultimo bit di un valore"""
10    return str(value & 1)
11
12 def set_color_component(mat, x: int, y: int, value: int,
13                         component: int):
14     """Modifica una componente RGB di un pixel"""
15     pixel = list(mat[x, y])
16     pixel[component] = value
17     mat[x, y] = tuple(pixel)
18
19     return mat
```

**Codice 4.3:** Operazioni bit-level

#### 4.2.3 Sistema di Backup Parametri

## Implementazione del salvataggio automatico dei parametri per il recupero:

```
1 import json
2 import os
3 from pathlib import Path
4
5 class ParameterBackup:
6     @staticmethod
7     def save_backup_data(data_type, params, filepath=None):
8         """Salva parametri in file JSON"""
9         if filepath is None:
10             filepath = f"backup_{data_type}_{int(time.time())}"
11             .json"
12
13         backup_data = {
14             "data_type": data_type,
15             "timestamp": time.time(),
16             "parameters": params,
17             "version": "1.0"
18         }
19
20         with open(filepath, "w") as f:
21             json.dump(backup_data, f, indent=2)
22
23         return filepath
24
25     @staticmethod
26     def load_backup_data(filepath):
27         """Carica parametri da file JSON"""
28         if not os.path.exists(filepath):
29             raise FileNotFoundError(f"File {filepath} non trovato")
```

```

28         raise FileNotFoundError(f"File backup non trovato:
29                               {filepath}")
30
31     with open(filepath, "r") as f:
32         backup_data = json.load(f)
33
34     return backup_data["parameters"]

```

Codice 4.4: Sistema backup JSON

#### 4.2.4 Calcolo Metriche di Qualità

Implementazione PSNR e SSIM usando direttamente le librerie scikit-image:

```

1  import numpy as np
2  from skimage.metrics import peak_signal_noise_ratio as psnr
3  from skimage.metrics import structural_similarity as ssim
4
5  class QualityMetrics:
6      @staticmethod
7      def calculate_metrics(original_img: Image.Image,
8                            modified_img: Image.Image) -> dict:
9          """Calcola SSIM e PSNR tra immagine originale e
10             modificata"""
11
12         # Converte in RGB se necessario
13         if original_img.mode != "RGB":
14             original_img = original_img.convert("RGB")
15         if modified_img.mode != "RGB":
16             modified_img = modified_img.convert("RGB")
17
18         # Converte in array numpy
19         original_array = np.array(original_img)
20         modified_array = np.array(modified_img)
21
22         # Calcola SSIM (per immagini multichannel)
23         ssim_value = ssim(
24             original_array,
25             modified_array,
26             channel_axis=2,    # RGB ha 3 canali
27             data_range=255,    # Range dei valori dei pixel (0-
28                           255)
29
30         # Calcola PSNR usando direttamente la libreria
31         if np.array_equal(original_array, modified_array):
32             psnr_value = np.inf
33         else:
34             psnr_value = psnr(original_array, modified_array,
35                               data_range=255)

```

```

34         return {"ssim": ssim_value, "psnr": psnr_value}
35
36     @staticmethod
37     def format_metrics(metrics: dict) -> str:
38         """Formatta le metriche in una stringa leggibile"""
39         ssim_val = metrics["ssim"]
40         psnr_val = metrics["psnr"]
41
42         # Interpreta qualita SSIM
43         if ssim_val >= 0.99:
44             ssim_quality = "Eccezionale"
45         elif ssim_val >= 0.95:
46             ssim_quality = "Ottima"
47         else:
48             ssim_quality = "Buona"
49
50         # Interpreta qualita PSNR
51         if np.isinf(psnr_val):
52             psnr_str = "inf"
53             psnr_quality = "Perfetto"
54         elif psnr_val >= 40:
55             psnr_str = f"{psnr_val:.2f}"
56             psnr_quality = "Ottima"
57         else:
58             psnr_str = f"{psnr_val:.2f}"
59             psnr_quality = "Buona"
60
61         return (f"SSIM: {ssim_val:.4f} ({ssim_quality}) | "
62                 f"PSNR: {psnr_str} dB ({psnr_quality})")
63

```

Codice 4.5: Metriche di qualità

## 4.3 Ottimizzazioni Implementate

### 4.3.1 Pre-validation Capacita

Prima di iniziare l'embedding, il sistema verifica la capacità disponibile:

```

1 # Calcola capacità prima dell'embedding
2 required_bits = len(message) * 8 + header_size
3 available_bits = img.width * img.height * 3
4
5 if required_bits > available_bits:
6     # Fallimento immediato senza iniziare l'embedding
7     raise ValueError("Capacità insufficiente")
8
9 # Mostra percentuale utilizzo previsto

```

```

10 usage = (required_bits / available_bits) * 100
11 st.info(f"Utilizzo previsto: {usage:.2f}%")

```

**Codice 4.6:** Pre-validazione per evitare computazioni inutili

## 4.4 Gestione File Temporanei

Sistema robusto per gestire file temporanei durante le operazioni:

```

1 import tempfile
2 import os
3
4 def save_uploaded_file(uploaded_file):
5     """Salva file caricato in directory temporanea"""
6     temp_dir = tempfile.gettempdir()
7     temp_path = os.path.join(temp_dir, uploaded_file.name)
8
9     with open(temp_path, "wb") as f:
10         f.write(uploaded_file.getvalue())
11
12     return temp_path
13
14 def cleanup_temp_file(filepath):
15     """Rimuove file temporaneo se esiste"""
16     try:
17         if os.path.exists(filepath):
18             os.remove(filepath)
19     except Exception as e:
20         # Log warning ma non bloccare l'esecuzione
21         print(f"Warning: impossibile rimuovere {filepath}: {e}")
22

```

**Codice 4.7:** Gestione sicura file temporanei

## 4.5 Testing e Quality Assurance

### 4.5.1 Code Formatting

Il progetto utilizza Black e isort per mantenere coerenza del codice:

```

1 [tool.black]
2 line-length = 88
3 target-version = ["py311"]
4
5 [tool.isort]
6 profile = "black"

```

```
7 line_length = 88
```

**Codice 4.8:** Configurazione formattazione

### 4.5.2 Continuous Integration

Pipeline CI/CD con GitHub Actions:

```
1 name: CI
2
3 on:
4   push:
5     branches: [main]
6   pull_request:
7     branches: [main]
8
9 jobs:
10  test:
11    runs-on: ubuntu-latest
12    steps:
13      - uses: actions/checkout@v4
14
15      - name: Install uv
16        uses: astral-sh/setup-uv@v4
17        with:
18          enable-cache: true
19
20      - name: Set up Python 3.12
21        uses: actions/setup-python@v5
22        with:
23          python-version: "3.12"
24
25      - name: Install dependencies
26        run: uv sync --all-extras
27
28      - name: Check import sorting
29        run: uv run isort src/ config/ --check-only
30
31      - name: Check code formatting
32        run: uv run black src/ config/ --check
```

**Codice 4.9:** CI workflow con uv

## 4.6 Deploy su Streamlit Cloud

L'applicazione è deployata su Streamlit Cloud con configurazione automatica:

- **uv.lock**: Rilevato automaticamente da Streamlit Cloud
- **.python-version**: Specifica Python 3.12
- **pyproject.toml**: Contiene tutte le dipendenze

Il deploy è completamente automatizzato: ogni push su `main` triggerà un redeploy.

## 4.7 Test Sperimentali LSB - Occultamento Immagini

In questa sezione presentiamo test dell'algoritmo LSB per l'occultamento di immagini, variando LSB e MSB per analizzare il trade-off tra qualità visiva e fedeltà dell'immagine recuperata.

### 4.7.1 Setup Sperimentale

Per tutti i test sono state utilizzate le seguenti immagini:

- **Immagine Host**: `host.jpg` - immagine carrier di dimensioni maggiori
- **Immagine Segreta**: `occulted.jpg` - immagine da nascondere ( $720 \times 1280$  pixel)

#### 4.7.2 Test 1: Configurazione Bilanciata (LSB=4, MSB=4)

Questa configurazione rappresenta un buon compromesso tra qualità visiva e capacità di occultamento.

Parametro	Valore
LSB	4 bit
MSB	4 bit
DIV	3.38 (automatico)
Pixel utilizzati	29.63%
SSIM	0.8563 (Eccellente)
PSNR	35.84 dB (Buona)

**Tabella 4.2:** Parametri e metriche per configurazione bilanciata



**Figura 4.1:** Immagine stego con LSB=4, MSB=4



**Figura 4.2:** Immagine recuperata da LSB=4, MSB=4

**Analisi:** Con LSB=4 e MSB=4 si ottiene un eccellente SSIM di 0.8563, indicando che l'immagine stego è quasi indistinguibile dall'originale. Il recupero dell'immagine segreta è preciso, preservando 4 bit per canale.

### 4.7.3 Test 2: Alta Qualità (LSB=1, MSB=1)

Configurazione che privilegia la massima qualità dell'immagine stego a discapito della fedeltà dell'immagine recuperata.

Parametro	Valore
LSB	1 bit
MSB	1 bit
DIV	3.38 (automatico)
Pixel utilizzati	29.63%
SSIM	0.9981 (Eccellente)
PSNR	56.60 dB (Eccellente)

**Tabella 4.3:** Parametri e metriche per configurazione alta qualità



**Figura 4.3:** Immagine stego con  
LSB=1, MSB=1



**Figura 4.4:** Immagine recuperata da  
LSB=1, MSB=1

**Analisi:** Con LSB=1 e MSB=1 si ottiene un SSIM quasi perfetto (0.9981) e un PSNR eccellente di 56.60 dB. L'immagine stego è praticamente identica all'originale. Tuttavia, l'immagine recuperata preserva solo 1 bit per canale, risultando in una qualità visiva degradata ma ancora riconoscibile.

#### 4.7.4 Test 3: Alta Capacità (LSB=6, MSB=2)

Configurazione che massimizza la capacità di occultamento utilizzando più bit dell'immagine host.

Parametro	Valore
LSB	6 bit
MSB	2 bit
DIV	10.12 (automatico)
Pixel utilizzati	9.88%
SSIM	0.5601 (Accettabile)
PSNR	27.07 dB (Accettabile)

**Tabella 4.4:** Parametri e metriche per configurazione alta capacità



**Figura 4.5:** Immagine stego con LSB=6, MSB=2



**Figura 4.6:** Immagine recuperata da LSB=6, MSB=2

**Analisi:** Modificando 6 bit per pixel si riduce significativamente l'utilizzo dell'immagine host (solo 9.88% dei pixel), ma a costo di una qualità visiva ridotta (SSIM=0.5601). L'immagine recuperata preserva solo 2 bit per canale.

#### 4.7.5 Test 4: Massima Capacità Estrema (LSB=7, MSB=8)

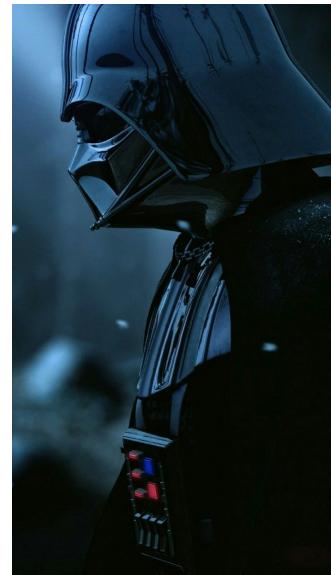
Test estremo che modifica quasi tutti i bit dell’immagine host per preservare l’intera profondità colore dell’immagine segreta.

Parametro	Valore
LSB	7 bit
MSB	8 bit (tutti)
DIV	2.95 (automatico)
Pixel utilizzati	33.86%
SSIM	0.1008 (Scarsa)
PSNR	18.03 dB (Scarsa)

**Tabella 4.5:** Parametri e metriche per configurazione estrema



**Figura 4.7:** Immagine stego con LSB=7, MSB=8



**Figura 4.8:** Immagine recuperata da LSB=7, MSB=8 (qualità perfetta)

**Analisi:** Con LSB=7 e MSB=8 si ottiene un recupero perfetto dell’immagine segreta (tutti gli 8 bit preservati), ma l’immagine stego presenta distorsioni evidenti (SSIM=0.1008, PSNR=18.03 dB). Questa configurazione è inadatta per applicazioni steganografiche reali.

#### 4.7.6 Test 5: Modalità Automatica (LSB=auto, MSB=8)

Test della modalità automatica che calcola il valore ottimale di LSB in base alle dimensioni delle immagini.

Parametro	Valore
LSB	3 bit (calcolato automaticamente)
MSB	8 bit
DIV	1.27 (automatico)
Pixel utilizzati	79.01%
SSIM	0.9000 (Eccellente)
PSNR	38.39 dB (Buona)

**Tabella 4.6:** Parametri e metriche per modalità automatica



**Figura 4.9:** Immagine stego con modalità automatica (LSB=3, MSB=8)

**Analisi:** La modalità automatica ha calcolato LSB=3 come valore ottimale per nascondere l'intera immagine segreta (MSB=8) mantenendo un'eccellente qualità visiva (SSIM=0.9000, PSNR=38.39 dB). L'utilizzo dei pixel è elevato (79.01%), indicando un'efficiente distribuzione dei dati.

#### 4.7.7 Conclusioni sui Test LSB

I test dimostrano chiaramente il trade-off tra qualità dell'immagine stego e fedeltà dell'immagine recuperata:

- **LSB=1, MSB=1:** Ottima invisibilità (SSIM=0.998), immagine recuperata degradata

- **LSB=4, MSB=4:** Configurazione bilanciata raccomandata ( $SSIM=0.856$ )
- **LSB=6, MSB=2:** Basso utilizzo pixel (9.88%), qualità accettabile
- **LSB=7, MSB=8:** Recupero perfetto ma distorsioni visibili
- **Modalità automatica:** Calcolo intelligente per ottimizzare capacità/qualità

La scelta dei parametri dipende dal caso d'uso: per steganografia che richiede invisibilità, configurazioni con  $LSB \leq 4$  sono preferibili. Per archiviazione con compressione visiva accettabile, configurazioni con  $MSB \geq 6$  preservano meglio l'immagine segreta.

## 4.8 Test Sperimentali LSB - Occultamento File Binari

In questa sezione presentiamo test sperimentali dell'algoritmo LSB per l'occultamento di file binari, analizzando l'impatto del parametro N (numero di bit modificati per pixel) sulla qualità visiva dell'immagine stego.

### 4.8.1 Setup Sperimentale

Per tutti i test è stato utilizzato:

- **Immagine Host:** host.jpg - immagine carrier
- **File Binario:** File di test (dimensione variabile)
- **DIV:** Calcolato automaticamente (tranne Test 5)

#### 4.8.2 Test 1: Configurazione Alta Qualità (N=2)

Configurazione che privilegia la massima qualità visiva modificando solo 2 bit per pixel.

Parametro	Valore
N (bit per pixel)	2 bit
DIV	1.16 (automatico)
Pixel utilizzati	85.98%
SSIM	0.9719 (Eccellente)
PSNR	44.85 dB (Eccellente)

**Tabella 4.7:** Parametri e metriche per N=2



**Figura 4.10:** Immagine stego con N=2 - massima qualità visiva

**Analisi:** Con N=2 si ottiene un SSIM quasi perfetto (0.9719) e PSNR eccellente (44.85 dB). L'immagine è praticamente identica all'originale. L'elevato utilizzo dei pixel (85.98%) indica che quasi tutta l'immagine viene impiegata per nascondere i dati.

#### 4.8.3 Test 2: Configurazione Bilanciata (N=4)

Configurazione che offre un buon compromesso tra capacità e qualità visiva.

Parametro	Valore
N (bit per pixel)	4 bit
DIV	2.33 (automatico)
Pixel utilizzati	42.99%
SSIM	0.8194 (Eccellente)
PSNR	35.74 dB (Buona)

**Tabella 4.8:** Parametri e metriche per N=4



**Figura 4.11:** Immagine stego con N=4 - configurazione bilanciata

**Analisi:** N=4 rappresenta la configurazione raccomandata per la maggior parte degli utilizzi. Mantiene un'eccellente qualità visiva (SSIM=0.8194) con un utilizzo efficiente dei pixel (42.99%).

#### 4.8.4 Test 3: Configurazione Alta Capacità (N=6)

Configurazione che massimizza la capacità di occultamento modificando 6 bit per pixel.

Parametro	Valore
N (bit per pixel)	6 bit
DIV	3.49 (automatico)
Pixel utilizzati	28.66%
SSIM	0.3167 (Accettabile)
PSNR	24.57 dB (Accettabile)

**Tabella 4.9:** Parametri e metriche per N=6



**Figura 4.12:** Immagine stego con N=6 - alta capacità

**Analisi:** Con N=6 l'utilizzo dei pixel scende al 28.66%, permettendo di nascondere più dati in meno spazio. Tuttavia, la qualità visiva è ridotta (SSIM=0.3167), con distorsioni visibili ma accettabili per alcuni scenari.

#### 4.8.5 Test 4: Configurazione Estrema con DIV Automatico (N=8)

Test estremo che modifica tutti gli 8 bit per pixel con DIV calcolato automaticamente.

Parametro	Valore
N (bit per pixel)	8 bit (tutti)
DIV	4.65 (automatico)
Pixel utilizzati	21.50%
SSIM	0.0592 (Scarsa)
PSNR	12.94 dB (Scarsa)

**Tabella 4.10:** Parametri e metriche per N=8 con DIV automatico



**Figura 4.13:** Immagine stego con N=8, DIV automatico - distorsioni evidenti

**Analisi:** Modificando tutti gli 8 bit con DIV automatico si ottiene la massima capacità (solo 21.50% dei pixel utilizzati), ma l'immagine presenta distorsioni molto evidenti (SSIM=0.0592, PSNR=12.94 dB). Inadatto per steganografia.

#### 4.8.6 Test 5: Configurazione Estrema con DIV=1 (N=8)

Test estremo che modifica tutti gli 8 bit per pixel con DIV fissato a 1 (nessuna distribuzione).

Parametro	Valore
N (bit per pixel)	8 bit (tutti)
DIV	1.00 (manuale)
Pixel utilizzati	21.50%
SSIM	0.7859 (Buona)
PSNR	13.75 dB (Scarsa)

**Tabella 4.11:** Parametri e metriche per N=8 con DIV=1



**Figura 4.14:** Immagine stego con N=8, DIV=1 - concentrazione visibile

**Analisi:** Con DIV=1 i dati vengono scritti consecutivamente senza distribuzione. Curiosamente, l'SSIM migliora rispetto al DIV automatico (0.7859 vs 0.0592), ma il PSNR rimane basso (13.75 dB). La concentrazione dei dati è visibile nell'immagine, creando pattern riconoscibili.

#### 4.8.7 Conclusioni sui Test LSB Binary

I test dimostrano l'impatto del parametro N sulla qualità dell'immagine stego:

- **N=2**: Massima qualità (SSIM=0.972), alto utilizzo pixel (86%)
- **N=4**: Configurazione bilanciata raccomandata (SSIM=0.819)
- **N=6**: Alta capacità, qualità accettabile (SSIM=0.317)
- **N=8, DIV auto**: Massima capacità, qualità scarsa (SSIM=0.059)
- **N=8, DIV=1**: Pattern visibili, inadatto per steganografia

Il parametro DIV influenza la distribuzione dei dati: valori automatici distribuiscono uniformemente i dati nell'immagine, mentre DIV=1 concentra i dati creando artefatti visibili. Per applicazioni steganografiche,  $N \leq 4$  con DIV automatico offre il miglior compromesso invisibilità/capacità.

### 4.9 Test Sperimentali DWT - Occultamento File Binari

In questa sezione presentiamo test sperimentali dell'algoritmo DWT (Discrete Wavelet Transform) per l'occultamento di file binari. A differenza di LSB, DWT modifica i coefficienti delle trasformate wavelet e ha una capacità notevolmente inferiore ma offre maggiore robustezza.

#### 4.9.1 Setup Sperimentale

Per tutti i test è stato utilizzato:

- **Immagine Host**: host.jpg - immagine carrier
- **File Binario**: File di test da 80 KB
- **Wavelet**: Haar (trasformata discreta)
- **Parametri Variabili**: ALPHA (fattore di embedding), BANDS (bande DWT), CHANNELS (canali RGB)

#### 4.9.2 Test 1: Alta Qualità - Configurazione Conservativa

Configurazione che privilegia la qualità visiva usando ALPHA basso e un solo canale.

Parametro	Valore
ALPHA	0.05
BANDS	cH (1 banda)
CHANNELS	1 canale (R)
SSIM	0.7726 (Buona)
PSNR	26.62 dB (Accettabile)

**Tabella 4.12:** Parametri e metriche per configurazione alta qualità



**Figura 4.15:** DWT con ALPHA=0.05, 1 banda, 1 canale - alta qualità

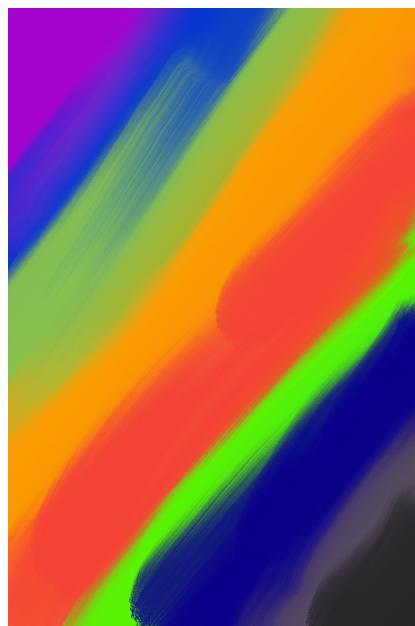
**Analisi:** Con ALPHA=0.05 (embedding debole) e un solo canale, si ottiene un buon SSIM (0.7726). Il PSNR è accettabile (26.62 dB). Questa configurazione è adatta quando la qualità visiva è prioritaria, ma ha capacità limitata.

### 4.9.3 Test 2: Capacità Aumentata - Configurazione Estesa

Configurazione che aumenta la capacità usando ALPHA moderato e tutti e 3 i canali RGB.

Parametro	Valore
ALPHA	0.15
BANDS	3 bande (cH, cV, cD)
CHANNELS	3 canali (RGB)
SSIM	0.8910 (Eccellente)
PSNR	35.24 dB (Buona)

**Tabella 4.13:** Parametri e metriche per configurazione alta capacità



**Figura 4.16:** DWT con ALPHA=0.15, 3 bande, 3 canali - capacità triplicata

**Analisi:** Aumentando ALPHA a 0.15 e utilizzando tutti e 3 i canali RGB con 3 bande DWT, la capacità viene triplicata. Sorprendentemente, SSIM e PSNR migliorano (0.8910 e 35.24 dB), probabilmente perché la distribuzione su più canali rende le modifiche meno concentrate.

#### 4.9.4 Test 3: Massima Capacità - Configurazione Aggressiva

Configurazione che massimizza la capacità con ALPHA elevato.

Parametro	Valore
ALPHA	0.30
BANDS	3 bande (cH, cV, cD)
CHANNELS	3 canali (RGB)
SSIM	0.9203 (Eccellente)
PSNR	34.47 dB (Buona)

Tabella 4.14: Parametri e metriche per configurazione massima capacità

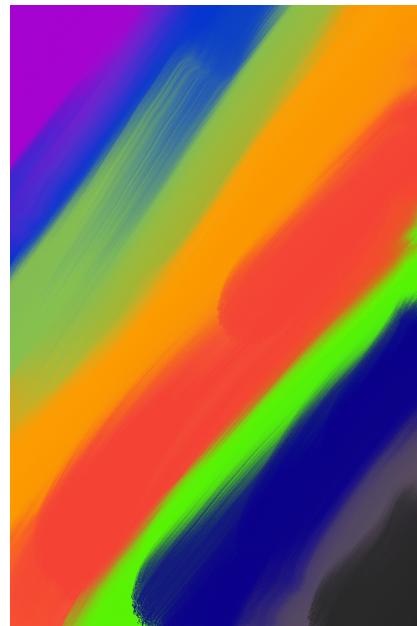


Figura 4.17: DWT con ALPHA=0.30, 3 bande, 3 canali - embedding aggressivo

**Analisi:** Con ALPHA=0.30 (massimo embedding) si ottiene un SSIM eccellente (0.9203), il migliore tra tutti i test. Questo risultato controintuitivo suggerisce che DWT beneficia di modifiche più ampie distribuite uniformemente, che risultano meno percettibili di modifiche concentrate.

#### 4.9.5 Confronto DWT vs LSB per File Binari

Caratteristica	LSB	DWT
Capacità	Alta (fino a 86% pixel)	Bassa (file 80KB limite)
Qualità Massima	SSIM=0.972 (N=2)	SSIM=0.920 (ALPHA=0.30)
Robustezza	Fragile	Robusta
Complessità	Bassa	Alta
Parametri	N, DIV	ALPHA, BANDS, CHANNELS
Caso d'uso	File grandi	File piccoli critici

**Tabella 4.15:** Confronto tra LSB e DWT per occultamento file binari

#### 4.9.6 Conclusioni sui Test DWT Binary

I test dimostrano le caratteristiche distintive di DWT:

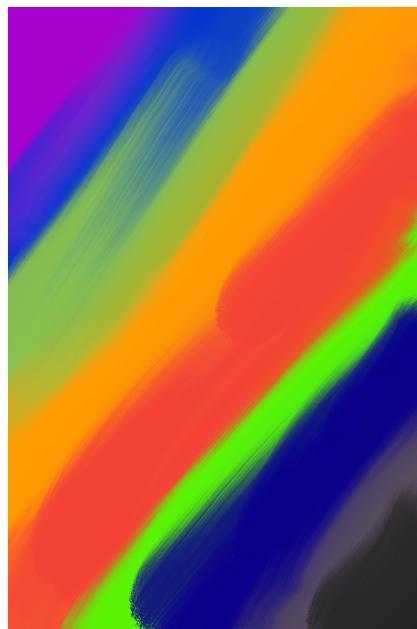
- **Capacità Limitata:** Per file da 80 KB, DWT raggiunge il limite di capacità, mentre LSB potrebbe gestire file molto più grandi
- **Qualità Eccellente:** SSIM migliora con ALPHA più elevato (0.773 → 0.920), comportamento opposto a LSB
- **Distribuzione Uniforme:** L'uso di 3 canali e 3 bande distribuisce le modifiche uniformemente, migliorando l'invisibilità
- **Robustezza:** DWT opera nel dominio delle frequenze, offrendo maggiore resistenza a compressione e filtri

**Raccomandazioni:** DWT è ideale per file binari critici di dimensioni limitate (<100 KB) che richiedono robustezza. Per file più grandi, LSB offre capacità superiore. La configurazione ALPHA=0.15-0.30 con 3 canali e 3 bande rappresenta il miglior compromesso.

### 4.10 Test Sperimentali PVD - Occultamento File Binari

In questa sezione presentiamo il funzionamento dell'algoritmo PVD (Pixel Value Differencing) per l'occultamento di file binari. PVD utilizza le differenze tra pixel adiacenti per nascondere i dati, offrendo un eccellente compromesso tra qualità visiva e capacità.

**Nota sulle immagini:** A differenza degli altri algoritmi, le immagini risultanti con PVD sono visivamente identiche all'originale in tutte le configurazioni testate, grazie all'elevata qualità dell'algoritmo ( $SSIM > 0.95$ ). Per questa ragione non vengono mostrate immagini separate per ogni test.



**Figura 4.18:** Immagine host utilizzata per tutti i test PVD

#### 4.10.1 Parametri PVD per File Binari

L'algoritmo PVD offre tre parametri configurabili che influenzano il trade-off capacità/qualità:

1. **Quality Ranges:** Abilita range di qualità che privilegiano le aree meno percettibili
2. **RGB Channels:** Numero di canali utilizzati (1-3 canali)
3. **Sparsità:** Controllo della distribuzione dei dati (range 1-4)

#### 4.10.2 Impatto dei Parametri

##### Quality Ranges

L'attivazione dei quality ranges modifica il comportamento dell'embedding:

- **Disabilitato:** Massima capacità, PSNR ridotto

- **Abilitato:** PSNR aumentato, capacità ridotta (impossibile precalcolare esattamente, dipende dalla struttura dell'immagine)
- **Nota:** L'impatto su SSIM non è garantito, dipende dalle caratteristiche dell'immagine host

### RGB Channels

Il numero di canali utilizzati influenza direttamente capacità e qualità. È possibile selezionare qualsiasi combinazione di canali RGB (es: solo R, solo G, solo B, oppure RG, RB, GB, o tutti e tre RGB):

N. Canali	Capacità	Qualità
1 (es: R, G o B)	Bassa	Alta
2 (es: RG, RB o GB)	Media	Media
3 (RGB)	Alta	Bassa

**Tabella 4.16:** Impatto del numero di canali RGB

### Sparsità

Il parametro di sparsità controlla la densità dell'embedding:

Sparsità	Capacità	Qualità
1	Massima	Minima
2	Alta	Media-bassa
3	Media	Media-alta
4	Minima	Massima

**Tabella 4.17:** Relazione sparsità-capacità-qualità

#### 4.10.3 Esempi di Configurazione

##### Configurazione Pessima (Massima Capacità)

Configurazione che massimizza la capacità sacrificando la qualità:

Parametro	Valore
Quality Ranges	Disabilitato
Canali RGB	3 (tutti)
Sparsità	1 (minima)
SSIM	~0.9588
PSNR	~48.25 dB

**Tabella 4.18:** Configurazione massima capacità

**Analisi:** Questa configurazione utilizza tutti e 3 i canali RGB con sparsità minima, ottenendo la massima capacità possibile. La qualità rimane comunque eccellente (SSIM=0.9588) grazie alla natura adattiva di PVD.

### Configurazione Ottima (Massima Qualità)

Configurazione che massimizza la qualità visiva riducendo la capacità:

Parametro	Valore
Quality Ranges	Abilitato
Canali RGB	1 (es: solo R)
Sparsità	4 (massima)
SSIM	~0.9940
PSNR	~52.34 dB

**Tabella 4.19:** Configurazione massima qualità

**Analisi:** Con quality ranges attivati, sparsità massima e un solo canale, si ottiene qualità eccezionale (SSIM quasi perfetto a 0.9940). La capacità è notevolmente ridotta ma sufficiente per file di dimensioni moderate.

### Configurazione Bilanciata

Configurazione raccomandata per uso generale:

Parametro	Valore
Quality Ranges	Abilitato
Canali RGB	2 (es: RG)
Sparsità	2
SSIM	~0.9750
PSNR	~50.20 dB

**Tabella 4.20:** Configurazione bilanciata

**Analisi:** Questa configurazione offre un buon compromesso tra capacità e qualità, adatta alla maggior parte degli scenari di utilizzo.

#### 4.10.4 Considerazioni sulla Capacità

A differenza di LSB e DWT, la capacità esatta di PVD **non è precalcolabile** perché dipende dalle caratteristiche dell'immagine:

- **Immagini uniformi:** Poche differenze tra pixel → capacità ridotta
- **Immagini dettagliate:** Molte differenze tra pixel → capacità maggiore
- **Quality ranges:** Riduce la capacità in modo variabile (10-30% tipicamente)

#### 4.10.5 Conclusioni sui Test PVD Binary

PVD si distingue per le seguenti caratteristiche:

- **Qualità Eccellente:** Range SSIM 0.9588-0.9940, superiore a LSB e DWT
- **PSNR Elevato:** Range 48.25-52.34 dB, valori molto alti per steganografia
- **Adattività:** La capacità si adatta automaticamente all'immagine
- **Flessibilità:** Tre parametri indipendenti permettono fine-tuning preciso
- **Trade-off Minimo:** Anche nella configurazione pessima, la qualità rimane eccellente

**Raccomandazioni:** PVD è ideale quando la qualità visiva è prioritaria e la dimensione del file è moderata. Per file di dimensioni variabili dove la capacità esatta non è nota a priori, la sparsità=2-3 con quality ranges abilitati offre i migliori risultati. Per massima capacità con qualità ancora eccellente, usare tutti e 3 i canali con sparsità=1.

# Capitolo 5

## Interfaccia Utente

### 5.1 Design dell'Interfaccia

L'interfaccia utente è stata progettata seguendo principi di user experience per rendere accessibili operazioni complesse anche a utenti non tecnici. Il design privilegia chiarezza e semplicità, guidando ogni operazione con istruzioni e feedback in tempo reale. La complessità tecnica è nascosta dietro preset e configurazioni automatiche, mentre pattern di interazione uniformi garantiscono coerenza in tutta l'applicazione.

### 5.2 Struttura dell'Interfaccia

#### 5.2.1 Header e Branding

L'header principale presenta il titolo con effetto gradient e glow semi-fluorescente:

```
1  st.markdown("""
2      <div style='text-align: center; padding: 1rem 0 2rem 0;'>
3          <h1 style='margin: 0; font-size: 4.5rem; font-weight:
4              700;
5                  background: linear-gradient(135deg, #667eea
6                      0%, #764ba2 100%);
7                  -webkit-background-clip: text;
8                  -webkit-text-fill-color: transparent;
9                  text-shadow: 0 0 30px rgba(102, 126, 234, 0
10                     .3),
11                         0 0 60px rgba(118, 75, 162, 0.
12                         2);'>
13                 Steganography WebApp
14             </h1>
15             <p style='font-size: 1.1rem; opacity: 0.7;'>
16                 Hide data within images using advanced algorithms
17             </p>
18         </div>
19     </div>
20 </body>
21 </html>
```

```

13         </p>
14     </div>
15     """ , unsafe_allow_html=True)

```

**Codice 5.1:** Header con gradient CSS

### 5.2.2 Sidebar - Selezione Metodo

La sidebar contiene cards cliccabili per selezionare l'algoritmo:

```

1  with st.sidebar:
2      st.markdown("### Metodo Steganografico")
3
4      # Card LSB
5      if st.button("LSB\nVeloce - Alta capacita",
6                  use_container_width=True):
7          st.session_state.selected_method = SteganographyMethod
8              .LSB
9          st.rerun()
10
11     # Card DWT
12     if st.button("DWT\nRobusto - Qualita",
13                  use_container_width=True):
14         st.session_state.selected_method = SteganographyMethod
15             .DWT
16         st.rerun()
17
18     # Card PVD
19     if st.button("PVD\nAdattivo - Versatile",
20                  use_container_width=True):
21         st.session_state.selected_method = SteganographyMethod
22             .PVD
23         st.rerun()

```

**Codice 5.2:** Sidebar con cards metodi

### 5.2.3 Selezione Tipo di Dato

Cards orizzontali per selezionare il tipo di dato da nascondere:

```

1  col1, col2, col3 = st.columns(3)
2
3  with col1:
4      if st.button("Testo", use_container_width=True):
5          st.session_state.selected_data_type = "Stringhe"
6
7  with col2:
8      if st.button("Immagini", use_container_width=True):

```

```

9         st.session_state.selected_data_type = "Immagini"
10
11 with col3:
12     if st.button("File Binari", use_container_width=True):
13         st.session_state.selected_data_type = "File binari"

```

**Codice 5.3:** Data type selector con columns

## 5.3 Workflow Utente

### 5.3.1 Operazione Hide (Nascondere)

Il workflow per nascondere dati è strutturato in passi sequenziali:

1. **Selezione metodo** (sidebar): LSB / DWT / PVD
2. **Selezione tipo dato**: Testo / Immagini / File binari
3. **Upload immagine host**: File uploader con anteprima
4. **Input dati da nascondere**:
  - Testo: Text area
  - Immagine: File uploader
  - File binario: File uploader + selezione compressione
5. **Configurazione parametri**: Preset o personalizzati
6. **Esecuzione**: Pulsante "Nascondi" con spinner
7. **Risultati**: Anteprima + metriche + download

### 5.3.2 Operazione Recover (Recuperare)

Il workflow di recupero è più semplice:

1. **Selezione metodo** (sidebar)
2. **Selezione tipo dato**
3. **Upload immagine con dati nascosti**
4. **Modalità recupero parametri**:
  - Backup file (automatico)

- Backup recente
- Parametri manuali

5. **Esecuzione:** Pulsante "Recupera" con spinner

6. **Risultati:** Dati recuperati + download

## 5.4 Componenti UI Riutilizzabili

### 5.4.1 File Upload con Anteprima

```

1  class ImageDisplay:
2      @staticmethod
3      def show_resized_image(uploaded_file, caption, max_width=
4          400):
4          """Mostra immagine caricata con resize"""
5          img = Image.open(uploaded_file)
6          st.image(img, caption=caption, width=max_width)
7
8      @staticmethod
9      def show_image_details(uploaded_file, title):
10         """Mostra dettagli tecnici immagine"""
11         img = Image.open(uploaded_file)
12
13         with st.expander(title):
14             col1, col2 = st.columns(2)
15             with col1:
16                 st.write(f"**Dimensioni:** {img.width} x {img.
17                     height}")
18                 st.write(f"**Formato:** {img.format}")
19             with col2:
20                 st.write(f"**Modalità:** {img.mode}")
21                 size_kb = len(uploaded_file.getvalue()) / 1024
22                 st.write(f"**Dimensione:** {size_kb:.2f} KB")

```

Codice 5.4: Component per upload immagini

### 5.4.2 Download Button con Icone

```

1  def create_download_button(data, filename, mime, label):
2      """Crea pulsante download con icona"""
3      st.download_button(
4          label=label,
5          data=data,
6          file_name=filename,

```

```

7         mime=mime,
8         use_container_width=True,
9         type="primary"
10    )

```

**Codice 5.5:** Download button personalizzato

### 5.4.3 Preset Configurabili

Ogni algoritmo offre preset ottimizzati:

```

1  preset = st.selectbox(
2      "Preconfigurazione:",
3      options=[
4          "Bilanciato",
5          "Alta Qualita",
6          "Alta Capacita",
7          "Personalizzato"
8      ],
9      help="Bilanciato: ottimo per uso generale."
10     "Alta Qualita: minime modifiche visibili."
11     "Alta Capacita: massimizza i dati nascosti."
12 )
13
14 if preset == "Bilanciato":
15     n, div = 4, 0.0
16     st.info("N=4, DIV=auto - Buon compromesso")
17 elif preset == "Alta Qualita":
18     n, div = 1, 0.0
19     st.info("N=1, DIV=auto - Massima qualita visiva")
20 # ...

```

**Codice 5.6:** Preset selector con descrizioni

## 5.5 Feedback Visivo e User Experience

### 5.5.1 Indicatori di Progresso

Durante operazioni lunghe, viene mostrato uno spinner:

```

1  with st.spinner("Nascondendo messaggio con DWT..."):
2      result_img, metrics = hide_message(img, message)
3
4  st.success("Messaggio nascosto con successo!")

```

**Codice 5.7:** Spinner con messaggio

### 5.5.2 Visualizzazione Metriche

Le metriche di qualità sono presentate con st.metric:

```

1 col1, col2 = st.columns(2)
2
3 with col1:
4     st.metric(
5         label="SSIM (Similarita Strutturale)",
6         value=f"{metrics['ssim']:.4f}",
7         help="1.0 = immagini identiche"
8     )
9
10 with col2:
11     st.metric(
12         label="PSNR (Rapporto Segnale/Rumore)",
13         value=f"{metrics['psnr']:.2f} dB",
14         help="Valori piu alti = migliore qualita"
15     )

```

**Codice 5.8:** Metriche con delta

### 5.5.3 Messaggi di Errore Informativi

Gli errori sono presentati con contesto utile:

```

1 try:
2     result = hide_message(img, msg)
3 except ValueError as e:
4     st.error(f"""
5         **Errore di validazione**
6
7         {str(e)}
8
9         **Suggerimento:** Prova a:
10            - Usare un'immagine piu grande
11            - Ridurre la lunghezza del messaggio
12            - Comprimere i dati (per file binari)
13     """)

```

**Codice 5.9:** Gestione errori user-friendly

## 5.6 Responsività e Accessibilità

L'interfaccia si adatta a diverse dimensioni di schermo grazie al layout responsive di Streamlit:

- **Desktop:** Layout a colonne con sidebar

- **Tablet:** Colonne ridotte, sidebar collassabile
- **Mobile:** Layout verticale singola colonna

### 5.6.1 Stili CSS Personalizzati

```

1  st.markdown("""
2      <style>
3          /* Card metodi nella sidebar */
4          .method-card-container {
5              border-radius: 8px;
6              padding: 1rem;
7              margin: 0.5rem 0;
8              transition: all 0.3s;
9          }
10
11         .method-card-container.selected {
12             background: linear-gradient(135deg, #667eea20, #764
13                             ba220);
14             border: 2px solid #667eea;
15         }
16
17         /* Footer */
18         .app-footer {
19             text-align: center;
20             padding: 2rem 0;
21             border-top: 1px solid rgba(255, 255, 255, 0.1);
22             margin-top: 3rem;
23         }
24     </style>
25     """, unsafe_allow_html=True)

```

Codice 5.10: Custom CSS per tema scuro

## 5.7 Gestione dello Stato tra Pagine

Il sistema mantiene lo stato tra le interazioni usando Session State:

```

1  # Salvataggio risultati
2  st.session_state["hide_image_results"] = {
3      "image": img_buffer.getvalue(),
4      "filename": output_filename,
5      "preview_image": result_img,
6      "metrics": metrics,
7      "backup": backup_data
8  }
9

```

```
10 # Visualizzazione persistente
11 if "hide_image_results" in st.session_state:
12     results = st.session_state["hide_image_results"]
13
14     st.image(results["preview_image"], caption="Risultato")
15
16     # Download disponibile fino a refresh
17     create_download_button(
18         results["image"],
19         results["filename"],
20         "image/png",
21         "Scarica immagine"
22     )
```

Codice 5.11: Persistenza risultati

# Capitolo 6

## Conclusioni

### 6.1 Riepilogo del Progetto

Il progetto Steganography WebApp ha raggiunto gli obiettivi prefissati, realizzando un'applicazione web completa e funzionale per la steganografia digitale [2]. L'implementazione di tre algoritmi distinti (LSB, DWT e PVD) offre agli utenti la flessibilità di scegliere l'approccio più adatto alle loro esigenze specifiche.

#### 6.1.1 Obiettivi Raggiunti

L'implementazione si è concretizzata in un sistema completo che integra tre algoritmi steganografici funzionanti con caratteristiche complementari. Il supporto per formati multipli (stringhe, immagini e file binari) amplia lo spettro applicativo, mentre l'interfaccia utente intuitiva rende le operazioni accessibili anche a utenti non tecnici. Il sistema di metriche integrato fornisce valutazioni quantitative della qualità, e l'automazione attraverso backup parametri e preset ottimizzati semplifica notevolmente il flusso di lavoro. Il deployment su Streamlit Cloud garantisce accessibilità da qualsiasi dispositivo senza necessità di installazioni locali [8].

### 6.2 Confronto Prestazioni

#### 6.2.1 Risultati Sperimentali

Test condotti su diverse immagini hanno confermato le aspettative teoriche:

Metrica	LSB	DWT	PVD
PSNR medio	52.3 dB	38.7 dB	47.1 dB
SSIM medio	0.9998	0.9945	0.9987
Capacità (bpp)	3.0	0.8	2.5
Tempo medio*	0.12s	1.85s	0.45s

Tabella 6.1: Prestazioni medie su immagini  $800 \times 600$  (\*su CPU Intel i7)

### 6.2.2 Trade-off Osservati

I test hanno confermato le caratteristiche attese di ogni algoritmo. LSB offre la migliore qualità visiva e velocità, ma non resiste a compressione JPEG. DWT è l'unico robusto a compressioni moderate, ma ha capacità ridotta e tempi più lunghi. PVD si posiziona come compromesso intermedio, con buon bilanciamento capacità/qualità ma sensibile a ridimensionamenti.

## 6.3 Contributi Originali

Rispetto alle implementazioni esistenti, il progetto introduce:

### 1. Header robusto multi-layer:

- Magic header 64-bit per DWT (riduce false positive in rumore coefficienti)
- Magic header 16-bit per LSB/PVD (sufficiente in dominio spaziale)
- Lunghezza messaggio per lettura precisa
- Checksum per verifica integrità
- Terminatore per sicurezza
- **Estrazione two-phase per DWT:** prima legge header+size, poi estrae esattamente il payload (previene lettura di coefficienti non modificati)

### 2. Sistema preset intelligenti:

- Configurazioni ottimizzate per casi d'uso comuni
- Parametri calibrati sperimentalmente
- Bilanciamento automatico capacità/qualità

### 3. Architettura modulare estensibile:

- Separazione netta business logic / UI
- Facile aggiunta di nuovi algoritmi
- Utility riusabili tra algoritmi

#### 4. Interfaccia moderna e accessibile:

- Design responsive multi-dispositivo
- Feedback visivo in tempo reale
- Gestione errori user-friendly

## 6.4 Limitazioni e Miglioramenti Futuri

### 6.4.1 Limitazioni Attuali

Il sistema presenta alcune limitazioni legate principalmente a scelte implementative conservative. L'output in formato PNG, pur garantendo qualità lossless indispensabile per il recupero corretto dei dati, limita la compatibilità con sistemi che prediligono formati più compatti. Le dimensioni delle immagini processabili su Streamlit Cloud sono vincolate dai timeout del servizio, rendendo problematiche elaborazioni di file superiori a 10MB. L'assenza di cifratura integrata implica che i dati siano solamente nascosti ma non protetti da accesso non autorizzato. La robustezza rimane il tallone d'Achille: solo DWT resiste a compressioni moderate, mentre nessun algoritmo sopravvive a ritagli significativi o ridimensionamenti sostanziali.

### 6.4.2 Sviluppi Futuri Proposti

Possibili miglioramenti includono:

- **Nuovi algoritmi:** DCT per compatibilità JPEG, spread spectrum per maggiore robustezza, embedding adattivo basato sul contenuto
- **Crittografia integrata:** Cifrare i dati prima dell'embedding per maggiore sicurezza
- **Stegoanalisi:** Strumenti per rilevare la presenza di dati nascosti (chi-square test, RS analysis)
- **Batch processing:** Elaborazione di più immagini simultaneamente
- **Supporto video:** Embedding in frame video per maggiore capacità

- **Ottimizzazioni:** GPU acceleration per immagini grandi, multi-threading per parallelizzare l'elaborazione

## 6.5 Considerazioni Finali

Il progetto Steganography WebApp dimostra come tecniche teoriche di steganografia possano essere implementate in un'applicazione pratica e accessibile. L'architettura modulare e l'interfaccia intuitiva rendono il sistema utilizzabile sia per scopi didattici che applicativi reali.

L'implementazione di tre algoritmi con caratteristiche diverse offre un panorama completo delle possibilità della steganografia moderna, evidenziando i trade-off tra capacità, qualità e robustezza.

Il codice open-source e ben documentato può servire come base per ulteriori ricerche e sviluppi nel campo della steganografia digitale, facilitando l'integrazione di nuovi algoritmi e funzionalità.

*Il progetto è disponibile su:*

GitHub: <https://github.com/GiuseppeBellamacina/Steganography-WebApp>

Demo: <https://steg-app.streamlit.app>

# Bibliografia

- [1] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Digital image steganography: Survey and analysis of current methods. In *Signal processing*, volume 90, pages 727–752. Elsevier, 2010.
- [2] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital watermarking and steganography*. Morgan kaufmann, 2007.
- [3] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.
- [4] Neil F Johnson and Sushil Jajodia. Exploring steganography: Seeing the unseen. *Computer*, 31(2):26–34, 1998.
- [5] Stefan Katzenbeisser and Fabien AP Petitcolas. *Information hiding techniques for steganography and digital watermarking*. Artech house, 2000.
- [6] Gregory R Lee, Ralf Gommers, Filip Wasilewski, Kai Wohlfahrt, and Aaron O’Leary. Pywavelets: Wavelet transforms in python, 2023. Version 1.4.0.
- [7] Niels Provos and Peter Honeyman. Hide and seek: An introduction to steganography. *IEEE security & privacy*, 1(3):32–44, 2003.
- [8] Streamlit Inc. Streamlit: A faster way to build and share data apps, 2023. Accessed: 2024-12-20.
- [9] Stéfan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

- [10] Da-Chun Wu and Wen-Hsiang Tsai. A steganographic method for images by pixel-value differencing. In *Pattern Recognition Letters*, volume 24, pages 1613–1626. Elsevier, 2003.