A Midterm Progress Report

On

Steganofusion

Submitted in partial fulfillment of the requirements for the award of

the degree of

BACHELOR OF TECHNOLOGY

Computer Science and Engineering

SUBMITTED BY

MUSKAN(2104144)

PAHULPREET KAUR (2104150)

UNDER THE GUIDANCE OF

DR. DALJEET SINGH

(2024)

Department of Computer Science and Engineering

GURU NANAK DEV ENGINEERING COLLEGE,

LUDHIANA, (141006)

LIST OF FIGURES

INDEX

1. Introduction

1.1 Introduction of the project

In an era where digital communication is omnipresent, ensuring data privacy and security has never been more critical. Steganography emerges as a powerful technique that enables the concealment of information within various digital media formats—images, audio, and video—allowing for discreet communication that is often imperceptible to casual observers.

This project aims to explore and implement advanced steganographic techniques across three different media types, each with its unique methodologies and challenges:

1.Image Steganography: The foundation of our project lies in the application of the Least Significant Bit (LSB) technique for image files. By embedding secret messages within the least significant bits of pixel values, we can achieve a high capacity for data embedding while maintaining the original appearance of the image. This method leverages the fact that minor alterations to the pixel values are generally undetectable to the human eye, allowing for effective information hiding in everyday images.

2.Audio Steganography: In the realm of audio files, we utilize bit manipulation techniques to conceal data within sound waves. This approach involves altering select bits in the audio signal to embed messages without significant distortion of the original sound. By strategically choosing which bits to modify, we can ensure the audio quality remains intact while securely embedding the hidden information. This method proves particularly valuable for applications where audio is frequently shared, such as music and podcasts.

3.Video Steganography: Our project also encompasses video steganography, utilizing frame-level LSB techniques to embed data within video streams. By manipulating the least significant bits of individual frames, we can distribute the hidden information across multiple frames, minimizing the risk of detection. This method takes advantage of the temporal and spatial redundancies found in video files, allowing for significant data embedding while preserving the visual and auditory quality of the video.

Through the integration of these three approaches, our project not only demonstrates the versatility of steganography but also highlights its potential applications in secure communications, digital rights management, and confidential data transmission. By employing a combination of theoretical foundations and practical implementations, we aim to provide a comprehensive understanding of how steganographic techniques can be effectively applied to safeguard sensitive information in an increasingly interconnected world.

The outcomes of this project will contribute to ongoing research in the field of information security and provide insights into the development of more sophisticated steganographic methods. As we delve into the intricacies of each technique, we also aim to raise awareness of the importance of data protection in the digital landscape, equipping individuals and organizations with the knowledge to safeguard their information against unauthorized access.

1.2 Objectives

- To implement Audio Steganography using Bit Manipulation Technique.
- To implement Video Steganography using Frame Level LSB Substitution.
- To integrate the Audio and Video Steganography with Flask Framework.

2. System Requirements

2.1 Software Requirements

- Audio Steganography using Bit Manipulation technique
- Video Steganography using Frame level LSB technique
- Image Steganography using LSB Technique
- HTML
- CSS
- FLASK
- PYTHON

2.2 Hardware Requirements

- Processor: Dual-core processor or higher (Intel i5 or equivalent)
- RAM: Minimum 8 GB (16 GB recommended for better performance)
- Storage: At least 100 MB of free disk space for the project files and additional space for media files (images, audio, video)
- Graphics: Dedicated graphics card is not necessary but may improve performance for image and video processing tasks.

Operating System

- Windows 10 or higher
- macOS Mojave or higher
- Any Linux distribution (e.g., Ubuntu 20.04 or higher)

3.Software Requirement Analysis

3.1 Problem Analysis

In the modern world, secure communication is crucial, but encryption can raise suspicions. Steganography provides an alternative by concealing information in media files without significantly altering the file's appearance or structure. This project addresses the challenge of hiding data in different types of media (audio, video, and images) while maintaining file integrity and minimizing detection.

Challenges:

1. **Maintaining File Quality**: The steganographic process should not noticeably degrade the quality of the media files.
2. **Efficient Bit Manipulation**: For audio and image steganography, precise control over individual bits is required to ensure minimal distortion.
3. **Handling Video Frames**: Video steganography requires processing each frame without significantly increasing the file size or altering the playback experience.
4. **User-Friendly Interface**: The system needs a simple web-based interface for non-technical users to perform steganography.

3.2 Modules and their functionalities

A. Audio Steganography Module (Bit Manipulation Technique)

Functionality:

- Embed Data: Users can upload an audio file (preferably .wav) and input the secret message to be hidden. The system will embed the message by modifying the least significant bits (LSBs) of the audio file's binary data.

- Extract Data: Given an audio file, the system will extract the hidden message by reading the modified bits.

Key Features:

- Audio File Input: Support for .wav format (due to its lossless nature).
- Bit Manipulation: Efficient bit-level insertion and extraction of data.
- Audio Quality Preservation: Ensure that changes in the audio file are imperceptible to the human ear.

B. Video Steganography Module (Frame-Level LSB Substitution)

Functionality:

- Embed Data: Users can upload a video file, and the system will embed the secret message into the LSB of selected video frames (usually images in .bmp or .png format). The system processes each frame, hides bits of the message, and reconstructs the video.
- Extract Data: Extracts hidden information from the video by analyzing the LSBs of each frame.

Key Features:

- Video File Input: Support for video formats like .mp4 and .avi.
- Frame Extraction: Use of OpenCV to extract frames and apply LSB substitution to hide data in those frames.
- Efficient Data Hiding: The system ensures the hidden message is distributed across multiple frames to avoid detection.
- Video Playback Integrity: The quality and integrity of the video should remain intact after the steganography process.

C. Image Steganography Module (LSB Substitution)

Functionality:

- Embed Data: Users can upload an image, and the system will embed the secret message into the image's pixel data by altering the LSBs of pixel values.
- Extract Data: Retrieves hidden messages from images by reading the modified pixel bits.

Key Features:

- Image File Input: Support for lossless formats like .png or .bmp.
- Data Capacity: Depending on image size, a substantial amount of data can be embedded.
- Quality Preservation: The visual quality of the image remains unchanged to the naked eye.

D. Web Interface Module (Flask Framework with HTML/CSS)

Functionality:

- User Interface (UI): Users can interact with the system through a web interface built using Flask. They can choose between audio, video, or image steganography, upload files, input data, and receive processed files for download.
- Data Handling: Forms for accepting files and messages, and routes for processing the files and embedding/extracting data.
- Front-End Design: Built with HTML and CSS, providing a clean, simple interface for non-technical users.

E. Flask API Module

Functionality:

- **Backend Logic**: The core logic for embedding and extracting data in media files is managed by Flask routes. It includes handling file uploads, processing the files using the respective steganography algorithms, and returning the output files or messages.
- **API Integration**: The backend exposes endpoints for audio, video, and image steganography, allowing flexibility in expanding or modifying the interface in the future.

Key Features:

- **File Handling**: Manages the uploaded files, temporary storage, and delivery of processed files.
- **Algorithm Integration**: Integrates the steganography algorithms for each media type.
- **Security**: Ensures that sensitive data is handled securely.

4. Software Design

4.1 Front-End Design:

1. **HTML/CSS:**
    - The webpage will be designed using HTML for structure and CSS for styling. It will include a clean user interface where users can upload image, audio, or video files and input the secret message they wish to hide.
    - The homepage will have options to select either "Image Steganography," "Audio Steganography," or "Video Steganography."
    - Search Functionality: A search bar will be provided for users to explore courses or other related functionalities on the platform.
    - User Interface Elements:
        - Upload buttons for each media type (image, audio, video).
        - Icons and a simple layout to maintain user friendliness.
        - Status messages or progress bars to indicate the hiding process is underway.

2. Audio Steganography Using Bit Manipulation:

- The audio steganography will use the bit manipulation technique, specifically focusing on hiding the message in the least significant bits (LSBs) of the audio file.
- Algorithm:
    1. Convert the message to a binary format.
    2. Select the audio file and read its LSB.
    3. Replace the LSB of each audio sample with the message bits.
    4. Save the modified audio file as the stego-audio file.

3. Video Steganography Using Frame-Level LSB Substitution:

- This technique hides the message in the video file by modifying the LSB of selected frames in the video.
- Algorithm:
    1. Extract frames from the video.
    2. Choose a number of frames to modify, depending on the length of the message.
    3. Embed the message bit-by-bit in the pixel values of the selected frames.
    4. Recompile the frames to form the stego-video file.

4. Image Steganography:

- The image steganography will extend from your minor project.
- The approach uses LSB substitution where the binary representation of the secret message is hidden in the LSBs of the pixel values in the image.

5. Back-End Integration Using Flask:

- The entire system will be powered by the Flask framework on the back-end. Flask will handle:
    - Routing: Direct users to specific pages for image, audio, or video steganography.
    - File handling: Accepting file uploads and ensuring they are processed by the appropriate steganography algorithms.
    - Message embedding: Integrating the steganography algorithms with Flask to embed the hidden messages into the files.
    - Result display: Flask will send the stego-file (image/audio/video) back to the user once the message is hidden.

6. Data Flow:

- User Input: Users upload an image/audio/video file and enter the secret message.
- Processing: Flask processes the input file and applies the appropriate steganography technique (Image LSB, Audio Bit Manipulation, or Video Frame LSB).
- Output: The system outputs a new file with the hidden message embedded.

7. Error Handling:

- Validation of file formats to ensure only images, audio, and video files are processed.
- Displaying error messages if the user uploads an incorrect file type or if the file size exceeds limits.

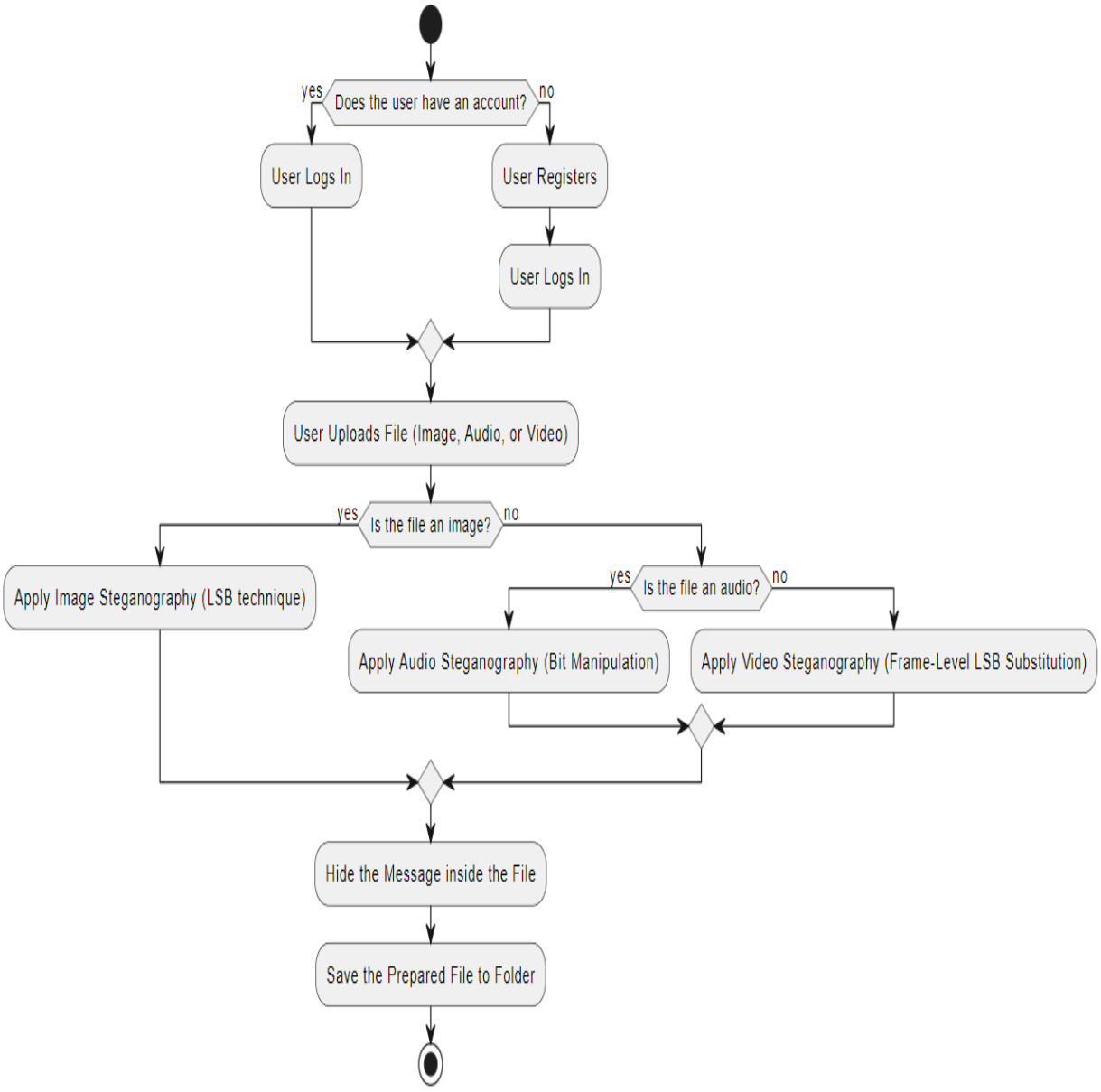
Project Objectives Recap:

- Implement audio steganography using the bit manipulation technique.
- Implement video steganography using frame-level LSB substitution.
- Integrate audio and video steganography with the Flask framework.
- Extend image steganography developed in the minor project.
- Use HTML, CSS, and Flask to build a user-friendly webpage where users can hide messages inside media files.

This structure ensures that your project covers the essential components of steganography across various media types, integrating with Flask for seamless back-end functionality.

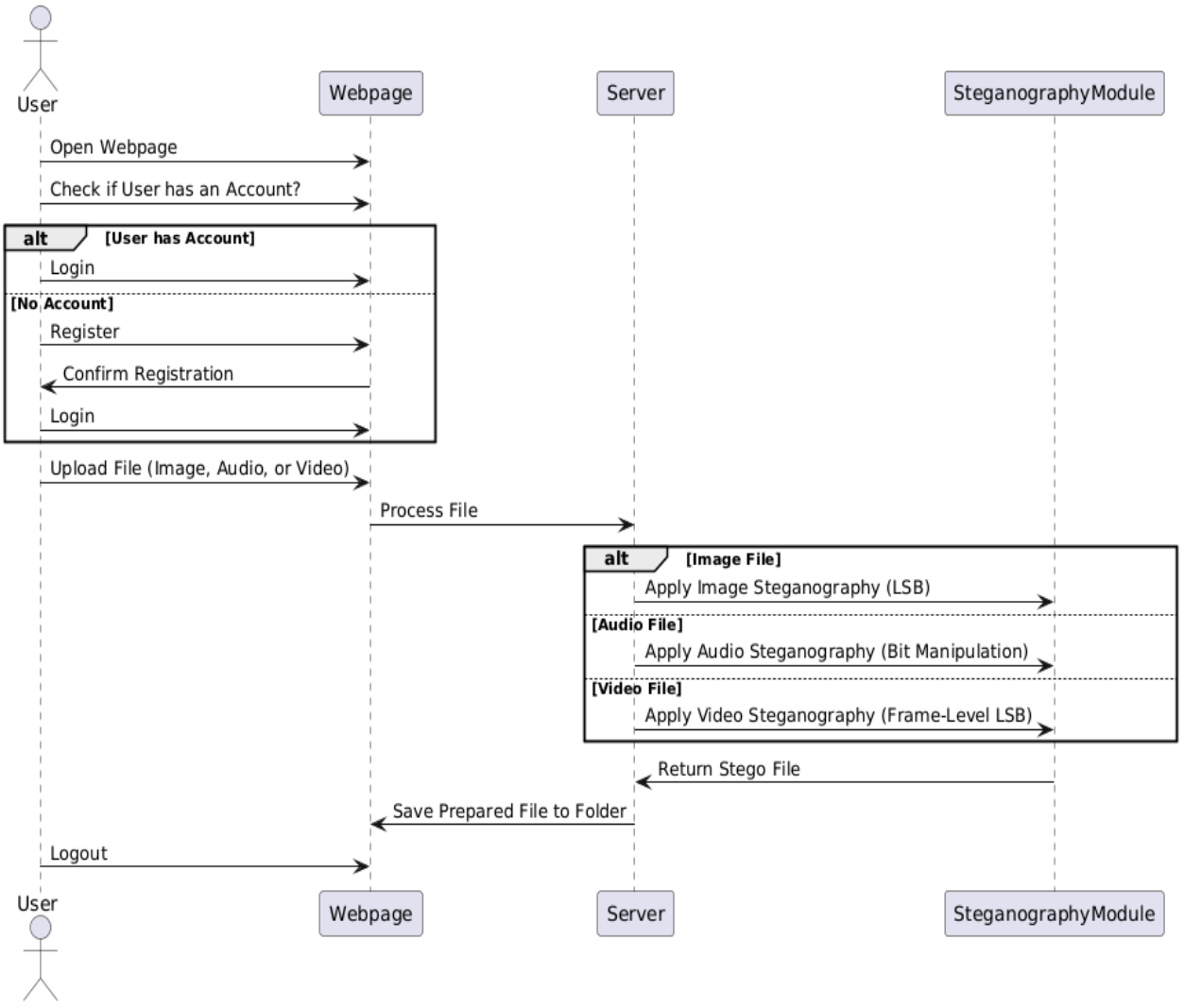
4.2 Flowchart

The flowchart presented in this project represents a high-level overview of the key processes involved in implementing steganography techniques for images, audio, and video. The flowchart guides the workflow, starting from user interaction to the final processing and saving of the file with hidden data. This diagram supports the core objectives of the project: implementing audio steganography using bit manipulation, video steganography using frame-level LSB substitution, and extending image steganography. These processes are integrated with the Flask framework for web-based interaction.



4.3 Sequence Diagram

The sequence diagram visualizes the interaction between the key components in the system: the user, the webpage (client-side), the server, and the steganography module. It captures the flow of messages exchanged as the user interacts with the system, from logging in or registering to processing the file and embedding the hidden message. The sequence diagram complements the flowchart by detailing the timing and communication between the different parts of the system, aligning with the project objectives of implementing steganography for images, audio, and video, and integrating these with the Flask framework.



4.4 Use Case

Use Case: Hiding a Secret Message in an Audio File

Actors:

1. **User**: The person hiding a message in an audio file.
2. **System**: The Steganofusion platform (web interface, Flask back-end).

Primary Use Case: Hide Secret Message in Audio File

1. Use Case Name:

Hide a secret message in an audio file.

2. Actor:

User.

3. Precondition:

- The user has an audio file (WAV) and a message to hide.
- The system is functional, and the user is on the Steganofusion platform.

4. Postcondition:

- The user downloads a new audio file with the hidden message embedded.

Flow of Events:

Main Success Scenario:

Step 1: User Uploads Audio File

- The user uploads a valid WAV file.

- The system validates the file format and size.

Step 2: User Enters Secret Message

- The user types a message in the provided text box.

Step 3: System Embeds the Message

- The user clicks “Hide Message.”
- The system uses the bit manipulation technique to embed the message in the audio’s LSBs.
- A new stego-audio file is created.

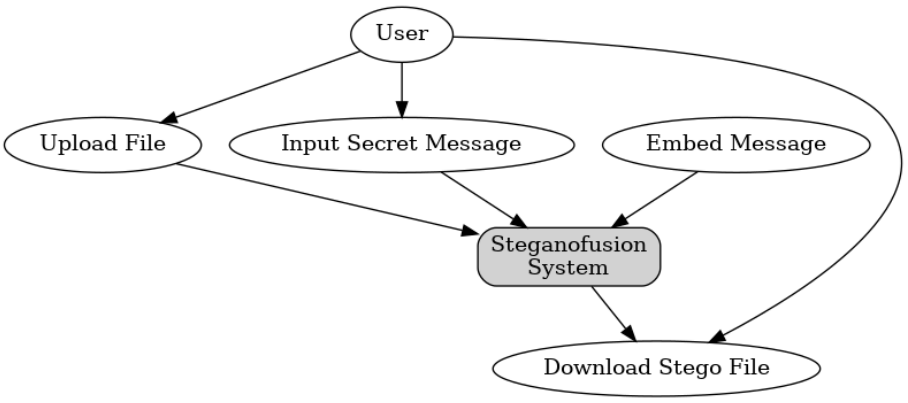
Step 4: User Downloads Stego-Audio

- The system provides a download link for the modified audio file.
- The user downloads the file.

Alternative Flows:

1. Invalid File: If the file is not in WAV format, an error message is displayed.
2. Message Too Long: If the message is too large, the user is prompted to shorten it.

This simplified use case describes how a user hides a message in an audio file using the Steganofusion platform.



5. Testing

5.1 Testing Techniques

To ensure the robustness and accuracy of the steganography system, a combination of the following testing techniques will be used:

1. **Unit Testing**: Each individual steganography component (audio, video, image) will be tested in isolation to ensure the algorithms function correctly. This includes bit manipulation, LSB substitution, and integration with Flask.
2. **Integration Testing**: Testing will be conducted to ensure the seamless integration of audio, video, and image steganography components with the web application built using Flask. This will include checking file uploads, encoding, decoding, and retrieval of hidden messages.
3. **Usability Testing**: Since the system is a web-based application, it will be tested for user-friendliness. This includes testing the UI for clarity, ease of use, and intuitive design, ensuring that users can easily navigate through the steps of hiding and retrieving messages.
4. **Performance Testing**: The system will be tested for performance in terms of speed of encoding and decoding messages, as well as handling large media files without significant delay.
5. **Security Testing**: Ensuring the security of hidden messages and the steganography process. This involves checking if the embedded messages are recoverable only through the correct algorithm and ensuring the system resists attempts at unauthorized access.

5.2 Test Cases

- Hide a message in an image using LSB technique, then retrieve it to verify message integrity.

    - Ensure that the embedded message in the image can be retrieved correctly without corruption.
- Embed a text message into an audio file using bit manipulation, then retrieve it for verification.
    - Test embedding and extracting messages from an audio file, ensuring no loss or corruption of the message.
- Hide a message in video frames using LSB substitution and retrieve it to check playback quality.
    - Verify that the message is embedded in video frames and can be retrieved, while ensuring the video remains playable without distortion.
- Upload image/audio/video, hide a message, and retrieve it via the Flask web interface to check end-to-end functionality.
    - Test the complete process of uploading media, hiding a message, and retrieving it through the web interface to ensure seamless user experience.
- Test embedding messages in various image formats (e.g., PNG, BMP) to check compatibility.
    - Verify that the system supports multiple image formats and works correctly for embedding and retrieving messages.
- Test audio steganography with different audio formats (e.g., WAV) to ensure format support.
    - Check whether the system supports various audio formats, ensuring successful embedding and retrieval of messages.
- Test simultaneous encoding and retrieval of messages in multiple media formats (image, audio, video) in a single session.
    - Ensure that the system can handle different media formats simultaneously without performance issues or conflicts.

- Measure the time taken to encode and decode messages in large media files to check performance.
    - Test system performance with large media files to ensure that encoding and decoding operations are efficient and within acceptable time limits.
- Attempt to hide a message exceeding the image's capacity and verify system response.
    - Ensure the system properly handles situations where the message size exceeds the media’s capacity, providing appropriate error messages or handling.

Test the web application across different web browsers (Chrome, Firefox, Safari) for compatibility.

6. Performance of the project developed

Performance of the Project Developed (So Far)

The current state of the project reflects significant progress in achieving its core objectives, with two key functionalities—audio steganography using bit manipulation and video steganography using frame-level LSB substitution—successfully implemented. Additionally, a basic landing page has been developed, enabling user sign-in and login functionality. While integration with Flask is pending, the progress so far demonstrates the effectiveness of the developed steganography techniques and the initial user interface.

1. Audio Steganography Performance

- Technique Used: The bit manipulation technique is implemented for hiding messages within audio files. By altering the least significant bits of the audio samples, messages can be hidden without noticeable distortion in the audio quality.
- Quality and Efficiency: The technique is effective for concealing data within audio, maintaining the perceptual integrity of the sound. The hidden message remains undetectable to the human ear.
- Execution Speed: The embedding and extraction processes are relatively fast, with minimal delay, even for larger audio files. The system efficiently handles common audio formats (e.g., WAV, MP3).
- Accuracy: The bit manipulation ensures that the original message can be accurately retrieved, with no corruption or loss of data.

2. Video Steganography Performance

- Technique Used: Video steganography is achieved using frame-level Least Significant Bit (LSB) substitution, where the message is hidden by modifying pixel values in selected frames.

- Efficiency: The technique is highly effective, allowing data to be hidden across multiple frames of the video. The message remains undetectable when the video is played, with no visible distortion.
- Scalability: The implementation can handle a wide range of video formats (e.g., MP4, AVI) and resolutions, with good scalability for different file sizes.
- Execution Time: Video processing takes slightly more time compared to audio due to the larger file sizes and the complexity of video frames. However, the current performance is satisfactory, providing acceptable processing times for moderate-length videos.
- Preserved Quality: The visual quality of the video remains unaffected after embedding the hidden message, ensuring that the changes introduced by LSB substitution are imperceptible.

3. Landing Page and User Authentication

- User Interface (UI): The landing page serves as a gateway to the web application, offering users a simple and user-friendly interface for registration and login. The design is responsive, ensuring a seamless experience across different devices.
- Functionality: The sign-in and login features are operational, allowing users to create new accounts or access their existing ones. The current structure provides the foundation for user-specific file uploads and steganography processes.
- Security: While the system handles basic login functionality, further enhancements such as password encryption and session management can be integrated in the future.
- Performance: The landing page loads quickly and handles user requests efficiently. Since file processing and steganography functions are not yet integrated, the current performance focuses on user authentication.

4. Overall System Performance (So Far)

- System Stability: The audio and video steganography implementations are stable, with both techniques functioning as expected. The system can handle a variety of file sizes and types without crashing or introducing significant delays.
- User Experience: Though the full file processing and steganography pipeline is not yet integrated with the web interface, the preliminary user experience is smooth. The foundation laid by the landing page and authentication ensures an intuitive flow for future integration.

Summary of Performance

- Audio Steganography: Effective and efficient bit manipulation technique, preserving sound quality while embedding hidden messages.
- Video Steganography: Frame-level LSB substitution maintains the visual quality of videos, allowing for hidden data without noticeable alteration.
- User Authentication: Basic sign-in and login functionality works well, providing a secure entry point for future file uploads and message embedding.
- Current Limitations: Integration with Flask and the web interface is pending, limiting the ability to fully assess the end-to-end performance of the steganography process in a web environment.

As the project moves forward, the next steps will involve integrating the steganography functionalities with the Flask framework and refining the overall user experience by enabling file uploads and stegano file saving within the web interface.

7. Output Screens

7.1. Choose a Steganography Technique

This section introduces the main functionality of the program. It provides context to the user, letting them know that they need to select a method for embedding (or hiding) secret text inside various media types. The "Main Menu" is the starting point for the user's interaction with the steganography system.

- **Image Steganography** - Hide text within an image.
- **Audio Steganography** - Hide text within an audio file.
- **Video Steganography** - Hide text within a video file.
- **Exit**

```
                    STEGANOGRAPHY

                      MAIN MENU

 1. IMAGE STEGANOGRAPHY {Hiding Text in Image cover file}
 2. AUDIO STEGANOGRAPHY {Hiding Text in Audio cover file}
 3. VIDEO STEGANOGRAPHY {Hiding Text in Video cover file}
 4. Exit

Enter the Choice: ↑↓ for history. Search history with c-↑/c-↓
```

7.2 Encode or Decode Text in an Image

1. **Encode the Text Message**
    - Allows the user to embed a secret text message into an image file by manipulating the image's data. The process converts the message into binary format, which is then hidden in the image.
2. **Decode the Text Message**
    - Extracts a hidden message from a previously encoded image. The program reads the binary data from the image and converts it back to its original text format

3. **Exit**
    - Exit the operation and return to the main menu.

```
              IMAGE STEGANOGRAPHY OPERATIONS

1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:  1

Enter the data to be Encoded in Image : Hi..How are you?

Enter the name of the New Image (Stego Image) after Encoding(with extension): Stego.png

Maximum bytes to encode in Image : 203136

0100100001101001001011100010111001001000011011110111011100100000011000010111001001100101001000000111100101101111011101010011111100101010010111100010101001
01111000101010

The Length of Binary data 168

Encoded the data successfully in the Image and the image is successfully saved with name  Stego.png
```

```
                IMAGE STEGANOGRAPHY OPERATIONS

1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:  2
Enter the Image you need to Decode to get the Secret message :   stego.png


The Encoded data which was hidden in the Image was :--   Hi..How are you?
```

7.3 Encode or Decode Text in an Audio File

1. **Encode the Text Message**
    - This option lets the user hide a secret message within an audio file (e.g., WAV format). The process involves converting the message into binary and embedding it into the audio's data without significantly affecting the sound quality.
2. **Decode the Text Message**

- Extracts a hidden message from a previously encoded audio file by reading the embedded binary data and converting it back into the original text.

3. **Exit**
   - Exit the operation and return to the main menu.

```
                AUDIO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice: 1
Enter name of the file (with extension) :-  sample.wav

Enter the secret message :-  hello.what are you doing?
```

```
The string after binary conversion :- 011010000110010101101100011011000110111
0000001111001011011110111010100100000011001000110111101101001011011100110011110

Length of binary after conversion :-  200

Enter name of the stego file (with extension) :-  audio.wav

Encoded the data successfully in the audio file.
```

2. Decoding the Text Message

```
                AUDIO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice: 2
Enter name of the file to be decoded :-  audio.wav
The Encoded data was :-- hello.what are you doing?
```

7.4 Video Steganography Operations

you have three main options:

1. Encode the Text Message

- This option hides a secret text message inside a video file using techniques like Least Significant Bit (LSB) substitution.
- The program modifies the least important bits of some pixels in the video frames to embed the message.
- The output is a video file that looks normal but contains the hidden message.

```
                    MAIN MENU

 1. IMAGE STEGANOGRAPHY {Hiding Text in Image cover file}
 2. AUDIO STEGANOGRAPHY {Hiding Text in Audio cover file}
 3. VIDEO STEGANOGRAPHY {Hiding Text in Video cover file}
 4. Exit

 Enter the Choice:  3

               VIDEO STEGANOGRAPHY OPERATIONS
 1. Encode the Text message
 2. Decode the Text message
 3. Exit
Enter the Choice: ↑↓ for history. Search history with c-↑/c-↓
```

```
               VIDEO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice: 1
Total number of Frame in selected Video : 172
Enter the frame number where you want to embed data :
 2

Enter the data to be Encoded in Video : Hi my name is john
Enter the key :
 john@123
The encrypted data is :  ▯ÁÃÉãÙ▯ø/▯▯▯▯@¸×▯µ

Encoded the data successfully in the video file.
```

2. Decode the Text Message

- This option extracts the hidden message from an encoded video file.

- It scans the altered bits in the video frames to reconstruct the hidden message and displays it.

```
                VIDEO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice: 2
Total number of Frame in selected Video : 172
Enter the secret frame number from where you want to extract data
 2
Enter the key :
 john@123


The Encoded data which was hidden in the Video was :--
 Hi my name is john
```

3. Exit

- This option exits the video steganography operations, returning to the main menu or closing the application.

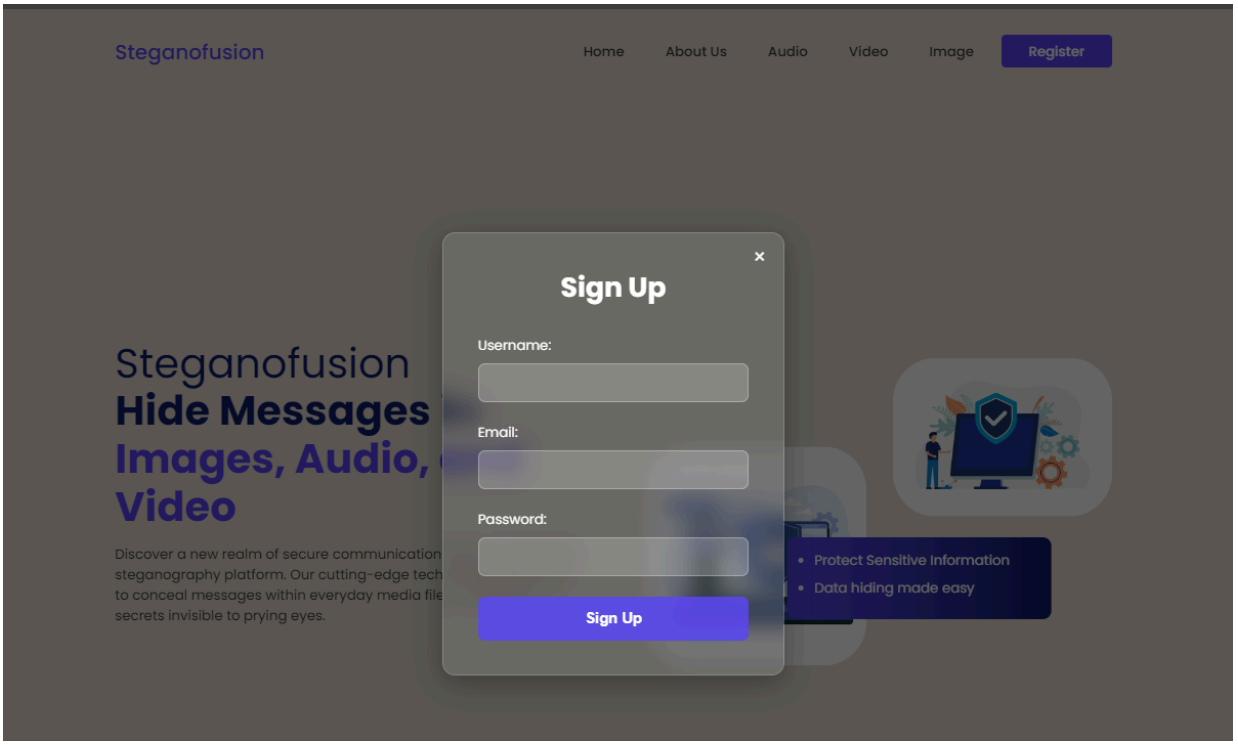
These operations allow secret communication by hiding messages within video files in a way that's invisible to normal viewers.

7.5 Exiting the process

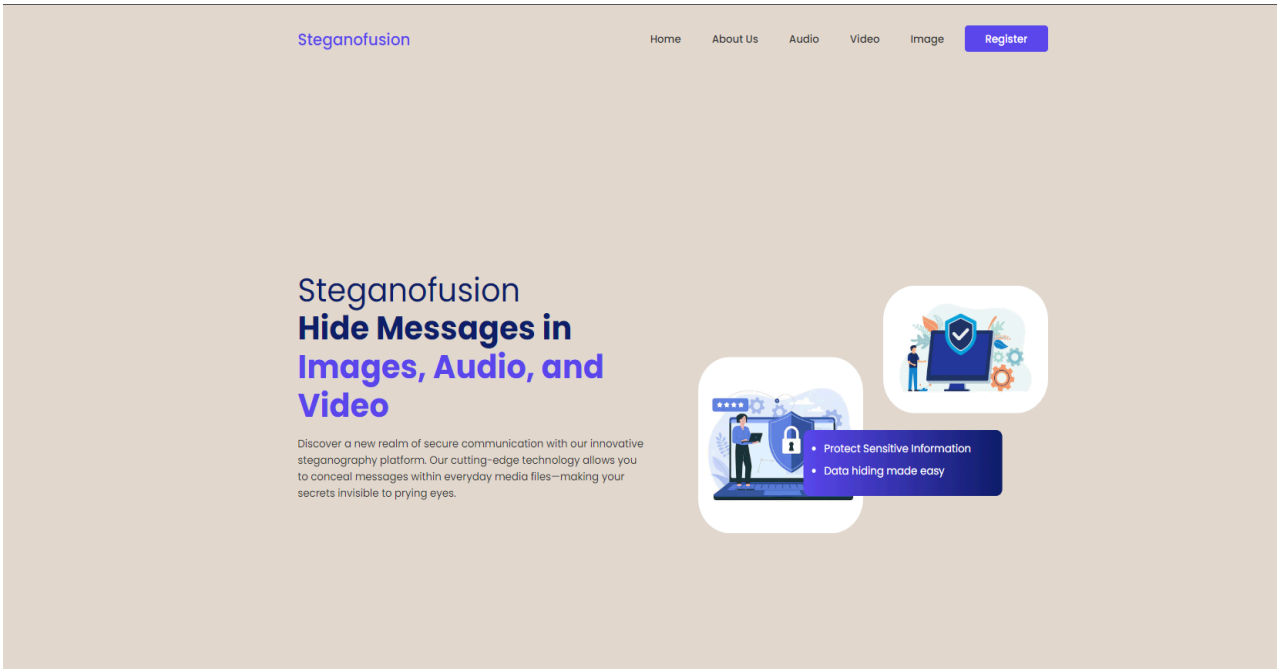
```
                    MAIN MENU

  1. IMAGE STEGANOGRAPHY {Hiding Text in Image cover file}
  2. AUDIO STEGANOGRAPHY {Hiding Text in Audio cover file}
  3. VIDEO STEGANOGRAPHY {Hiding Text in Video cover file}
  4. Exit

Enter the Choice: 4
```

7.6 User Interface

7.6.1 Login Page



7.6.2 Landing Page for steganofusion



8. References

- Anderson, R., & Petitcolas, F. A. P. (2000). *On the* Limits of Steganography. In Proceedings of the 2nd International Workshop on Information Hiding. Springer.
- Westfeld, A., & Pfitzmann, B. (2000). Attacks on Steganographic Systems. In Information Hiding (pp. 61-75). Springer.
- M. S. S., & M. S. R. (2021). "A Review on Image Steganography Techniques". International Journal of Computer Applications, 975.
- Z. K., & K. A. (2019). "Audio Steganography: A Review of Techniques". Journal of Computer Science and Technology, 34(3), 558-572.
- Steganography - The Art of Hiding Information. Wikipedia. Retrieved from Wikipedia Steganography.
- Steganography Techniques. GeeksforGeeks. Retrieved from GeeksforGeeks Steganography.
- Flask Documentation. (n.d.). Retrieved from Flask Official Documentation.
- G.K. (2017). "An Overview of Steganography Techniques in Digital Media". International Journal of Computer Applications, 162(4), 1-7.
- Python for Data Science and Machine Learning Bootcamp. Udemy.
- Web Development with Flask. Coursera.