A Project Report On

STEGLYZER: A STEGANOGRAPHY ANALYZER

Submitted in partial fulfillment of the requirement for the award of the degree

MASTER OF SCIENCE (CS & CL)
from
Marwadi University

Academic Year 2025 – 26

Muhammed Roshan K C (92400565050)
Surya Suresh (92400565053)

Internal Guide
Prof. Akhila B Mukundan

Rajkot-Morbi Road, At &PO :Gauridad, Rajkot 360 003. Gujarat. India.

Marwadi University
Marwadi Chandarana Group

NAAC A+

Faculty of Computer Applications (FoCA)

Certificate

This is to certify that the project work entitled

StegLyzer: A Steganography Analyzer

submitted in partial fulfillment of the requirement for the award of the degree of

Master of Science (CS & CL)

of the

Marwadi University

is a result of the bonafide work carried out by

Muhammed Roshan K C (92400565050)
Surya Suresh (92400565053)

during the academic year 2025 – 2026

Faculty Guide	HOD	Dean

DECLARATION

We hereby declare that this project work entitled **StegLyzer: A Steganography Analyzer** is a record done by us.

We also declare that the matter embodied in this project is genuine work done by us and has not been submitted whether to this University or to any other University / Institute for the fulfillment of the requirement of any course of study.

Place:

Date:

Muhammed Roshan K C (92400565050) Signature:_____

Surya Suresh (92400565053) Signature:_____

ACKNOWLEDGEMENT

It is indeed a great pleasure to express our thanks and gratitude to all those who helped us. No serious and lasting achievement or success one can ever achieve without the help of friendly guidance and co-operation of so many people involved in the work.

We are very thankful to our guide **Prof. Akhila B Mukundan,** the person who makes us to follow the right steps during our project work. We express our deep sense of gratitude to for her guidance, suggestions and expertise at every stage. A part from that his/her valuable and expertise suggestion during documentation of our report indeed help us a lot.

Thanks to our friend and colleague who have been a source of inspiration and motivation that helped to us during our project work.

We are heartily thankful to the Dean of our department **Prof. (Dr.) R. Sridaran** sir and HoD **Dr. Sunil Bajeja** sir for giving us an opportunity to work over this project and for their end-less and great support to all other people who directly or indirectly supported and help us to fulfil our task.

Muhammed Roshan K C (92400565050) Signature:_____

Surya Suresh (92400565053) Signature:_____

CONTENTS

TABLE INDEX

FIGURE INDEX

Chapter 1
Introduction

Background

In the digital age, the amount of information transmitted and stored electronically has reached unprecedented levels. From online banking transactions and confidential government communications to personal photographs and social media, digital files comprise the currency of modern civilization. With such ubiquity comes the necessity for secure data transmission and privacy protection. While cryptography has long been the standard for making the contents of a message unintelligible to unauthorized parties, it does not disguise the existence of a message. This is where steganography emerges—a complementary science designed to conceal messages within innocuous files so that outsiders remain unaware of any secret communication at all.

Steganography in Context

Steganography itself is not a new discipline. Its origins reach back to antiquity, practiced across civilizations for military, diplomatic, and private purposes. In the digital world, steganography evolved to embed secret information into images, audio files, videos, and even compressed archives—formats regularly exchanged as part of daily life. Such capabilities are increasingly important in scenarios including national security, forensic investigation, copyright protection, and private communications between individuals.

Need for Analytical Tools

Recent technological trends have sparked demand for intelligent steganography analysis tools. Cybersecurity professionals must be equipped not only to embed secrets in carriers but also to detect and extract covert information for legal investigations. Educators and researchers seek out realistic platforms to demonstrate and study the principles and weaknesses of data-hiding methods.

About Steganography Analyzer

The Steganography Analyzer project addresses these needs by providing a unified, extendable desktop application. Developed in Python using modern GUI frameworks, it enables users to perform steganographic data embedding and extraction on a variety of media files. Its design prioritizes user-friendliness and forensic utility, featuring detailed metadata inspection and structured report generation.

Structure of Report

This report documents the history, technical components, objectives, constraints, and benefits of the Steganography Analyzer. Each subsequent section explores a key aspect in depth.

History of Steganography

Ancient Origins

The word “steganography” derives from Greek roots meaning “covered writing.” In 440 B.C.,

Herodotus described the use of slaves shaved heads as secret message carriers. Medieval European societies experimented with invisible ink—commonly lemon juice or milk—that became readable when exposed to heat. Quilt designs and knitting patterns secretly encoded battle plans or intelligence during wartime, deceiving the enemy by their commonplace appearance.

Renaissance and Scientific Methods

The Renaissance brought advances in scientific techniques for steganography. Giovanni Battista Porta (1535–1615) became famous for devising invisible inks for covert communication. Steganographic literature from the 16th century documented chemical mixes, ciphers, and combinatorial strategies for hiding secrets.

Modern History: 20th Century

Steganography evolved remarkably during the 20th century, especially during the World Wars:

- Microdots: Tiny photographic reductions containing documents were placed as punctuation marks in letters, nearly impossible to detect without magnification.
- Null Ciphers: Messages were embedded within normal letters, with every nth word forming the secret text.
- Audio/Radio: Wartime spies used radio transmissions with encoded signals combined with everyday broadcasts as camouflage.
- Knitting Patterns: British agent Phyllis Latour Doyle used stitched patterns to convey vital information to the Allies.

Digital Era Transformation

Recent breakthroughs in steganography are tightly linked to digital technologies. Computer scientists found ways to alter the "least significant bits" (LSB) of pixel data in images—minutely changing colour values in such a way that hidden data becomes imperceptible to human eyes. Audio steganography utilizes phase coding, echo hiding, and spread spectrum techniques. Video files allow larger bandwidth and time-based data distribution. Steganography in compressed archives involves manipulating bits in unused padding areas or file headers.

Contemporary Usage

Today, steganography serves dual roles. It can be instrumental for privacy (journalists protecting sources, privacy advocates, copyright holders) and destructive (cybercriminals hiding malware or illicit data). Law enforcement and cybersecurity organizations frequently use advanced steganalysis tools to detect and analyze such covert data exchanges.

Research and Future Directions

Current research focuses on methods to increase steganographic capacity, robustness against detection, and resistance to lossy compression. Conversely, steganalysis—the art of detection—is constantly developing countermeasures based on artificial intelligence, statistical analysis, and data mining techniques.

1.1 Objective of the New System

1.1.1. Primary Purpose

- Provide a unified platform for performing and analyzing steganographic operations (embed/extract) across multiple media types.
- Enable practical investigation into modern data-hiding methods for research, education, and cybersecurity.

1.1.2. Secondary Goals

- User-Centered Design: Develop an intuitive, modern GUI for accessibility to technical and non-technical users alike.
- Metadata Analysis: Offer detailed inspection of carrier file properties (dimensions, color modes, bitrates, etc.), critical for forensic investigations.
- Reporting Capabilities: Generate structured reports (file info, steganographic actions, outcomes) to support casework or research documentation.

1.1.3. Educational Value

- Serve as an instructional tool for students investigating information hiding and privacy techniques.
- Enable researchers to experiment with embedding strategies and quantify detectability using real-world file formats.

1.1.4. Contribution to Cybersecurity

- Empower investigators to unearth hidden evidence in digital media files encountered during audits or criminal investigations.
- Enhance employees' security awareness in enterprises where sensitive data might be covertly transferred via everyday images or videos.

1.1.5. Bridge Theory and Practice

This project aims to translate the rich theoretical concepts of steganography into a practical toolkit, promoting learning and professional application.

1.2 Problem Definition

1.2.1. Current Limitations of Existing Tools

- Steganography tools typically focus on a single carrier type (mainly images), offering limited flexibility.
- Many utilities are command-line based, excluding less technical users.
- Forensic support is minimal—metadata extraction and structured reporting are rarely integrated in open-source applications.
- Lack of cross-platform compatibility or extensibility; proprietary solutions are not modifiable or customizable.

1.2.2. Forensics and Law Enforcement Needs

Law enforcement agencies routinely encounter suspicious media files in criminal investigations. Without advanced analysis software, detecting hidden evidence in videos, audio files, or compressed archives remains challenging. A capable tool must streamline examination, extraction, and reporting.

1.2.3. Research and Educational Gaps

Academia and security research require open, robust platforms to apply and test the effectiveness of steganography and corresponding detection strategies. There is a clear gap for modular, user-friendly solutions that can be adapted for curriculum and experiments.

1.2.4. Threats and Risks

Malicious actors increasingly rely on steganography to conceal malware, exfiltrate sensitive information, and orchestrate cyberattacks beyond the reach of conventional anti-virus or firewall defences'. Defensive tools must be equipped to monitor and counteract such activities.

1.2.5. Steganalysis Challenges

Detection of hidden content in high-definition images, lossless audio, and large video files presents analytical and computational challenges. Sophisticated tools capable of comprehensive evidence gathering and presentable reporting are essential.

1.3 Core Components

1.3.1. Graphical User Interface (GUI)

- Built using Tkinter (Python's native GUI package) and sv_ttk for improved aesthetics (dark theme).
- Intuitive layout with drag-and-drop file selection, radio buttons for operation modes (embed/extract), progress bars, and clear action buttons.

1.3.2. File Handler and Metadata Extractor

- Utilizes Pillow for image metadata and OpenCV for video frame inspection (resolution, duration, FPS).
- Employs Python's OS and mime types modules for file size, type, and path information.
- Displays structured metadata critical for forensics (e.g., dimensions, colour format, encoding details).

1.3.3. Steganography Engine Modules

- stego_image: Implements LSB embedding and extraction for image formats.
- stego_audio: Safely embeds messages by manipulating audio frames based on signal processing principles.
- stego_video: Extends data-hiding capacity to video streams (per-frame or motion vector approaches).

- stego_archive: Manipulates pads, headers, and unused bits in ZIP/RAR files for covert storage.

1.3.4. Execution Controller

- Central module determines file type, steganography operation, and dispatches to appropriate engine.
- Ensures correct handling of every supported file format.

1.3.5. Progress and User Feedback

- Real-time progress bar indicates operation state.
- MessageBox pop-ups provide clear feedback on execution status, errors, or completion.

1.3.6. Report Generation System

- Allows users to export metadata and operation results into text files.
- Supports chain-of-custody and documentation requirements for forensic casework.

1.3.7. Extensibility

- Modular design allows new embedding/extraction methods or file types to be added easily.
- Potential for future inclusion of encrypted steganography, cloud integration, or AI-based detection.

1.4 Project Profile

1.4.1. Technical Stack

- Programming Language: Python 3.x
- Libraries/Frameworks: Tkinter, sv_ttk, TkinterDnD2, Pillow (PIL), OpenCV
- Utilities: os, mimetypes
- Steganography Modules: stego_image, stego_audio, stego_video, stego_archive
- Supported File Types: PNG, JPG, JPEG, BMP, WAV, MP4, AVI, ZIP, RAR, 7Z, TAR, GZ, BZ2, ISO, DMG

1.4.2. Target Audience

- Cybersecurity Analysts: For detecting covert channels, hidden malware or data exfiltration.
- Forensic Investigators: To analyze digital evidence and reconstruct communication trails.
- Educators and Students: For hands-on learning and experimentation with steganographic techniques.
- Privacy Advocates: To safeguard personal communications and images.

1.4.3. System Architecture (Textual Overview)

- User Input: File selection (browse/drag-drop), operation mode (embed/extract).

- Metadata Analysis: Display detailed file attributes for forensics.
- Steganography Execution: Apply embedding or extraction logic as per user action.
- Reporting: Save results to text files for casework or analysis.

1.4.4. Platform Compatibility

- Designed for Windows, Linux, and macOS platforms supporting Python.
- Fully open-source, enabling custom modifications for research or professional needs.

1.5 Assumptions and Constraints

1.5.1. Assumptions

- Users possess minimal familiarity with digital files and steganography concepts.
- Steganography modules (stego_image etc.) are implemented and available.
- System dependencies (Python, OpenCV, Pillow) are installed.

1.5.2. Constraints

- File size and format may affect performance and compatibility (e.g., very large videos).
- Certain compressed formats (proprietary, password-protected archives) may be unsupported.
- Only plaintext reporting; advanced formats (PDF, HTML) are future scope.
- No built-in cryptographic protection—hidden messages can be extracted once discovered.
- Operations limited to locally accessible files, no cloud/network integration as default.

1.5.3. System Safety, Reliability

- Exception handling for incorrect input files and operational errors.
- Error messages clarify reason for failures (unsupported type, incorrect format, etc.).

1.6 Advantages and Limitations of the Proposed System

1.6.1. Advantages

- Versatile Format Support: Single application handles images, audio, video, archives.
- Detailed Metadata Inspection: Essential for forensics and research.
- Modern, Accessible GUI: Lowers learning curve, supports drag-and-drop efficiency.
- Modular Architecture: Facilitates extension and integration.
- Case Report Generation: Supports documentation and audit trails.

1.6.2. Unique Features

- Combines analytics and steganography in one place—most tools specialize in only one aspect.
- User feedback and progress indicators enhance transparency in operations.

1.6.3. Limitations

- Encryption not presently integrated—data security depends solely on concealment.
- Limited to predefined file types—other formats (PDF, DOCX, GIF, etc.) require extension.
- Dependent on third-party libraries for metadata and media handling.
- Performance concerns for extremely large files; optimizations may be required.
- No built-in cloud or distributed analysis; standalone desktop application only.

1.6.4. Operational Boundaries

- Only as effective as the steganography algorithms implemented; future improvements possible.
- No advanced steganalysis detection; project focuses on embedding/extraction and basic forensic analysis.

Chapter 2
Requirement Determination & Analysis

2.1 Requirement Determination

In any software development project, correctly identifying and defining system requirements is a critical phase that lays the foundation for successful design, implementation, and deployment. For the Steganography Analyzer, this process involved rigorous consideration of functional, technical, and user-centered requirements to develop a tool that is practical, extensible, and user-friendly.

Functional Requirements

- Multi-format Support: The system must support embedding and extracting hidden data across various digital media types: images (PNG, JPEG, BMP), audio files (WAV, FLAC), video formats (MP4, AVI), and compressed archives (ZIP, RAR). This diversification meets the modern need for cross-media steganography.
- User Interface: The application must provide an intuitive graphical user interface (GUI) that allows users to browse files conveniently.
- Steganographic Operations: Users should be able to select between two core operations — embedding secret messages into carrier files or extracting hidden data from them.
- Metadata Extraction: Each input file should be subjected to detailed metadata analysis. The system should display relevant technical details such as file size, format, dimensions, video frame rate, or audio bitrate, helping users verify carrier suitability and analyse suspicious files.
- Progress Feedback: The tool needs real-time progress reporting during steganographic processing to assure transparency and user awareness.

Non-Functional Requirements

- Performance: The system should efficiently handle medium-sized files without significant lag, ensuring responsiveness during GUI interactions and data processing.
- Usability: Accessibility is a priority; the interface design must minimize barriers for users with varying levels of technical expertise, balancing simplicity and functionality.
- Portability: The tool should run across major desktop platforms (Windows, Linux, macOS) with minimal dependency issues.
- Modularity & Extensibility: The architecture should support plugging in new steganography modules or file type handlers with minimal adjustments, fostering future enhancements.

Security Requirements

- The system should provide basic error handling to avoid crashes from unsupported files or corrupted inputs.
- While embedding/extraction processes conceal message existence, the system currently does not provide cryptographic encryption for message confidentiality, which could be a planned future enhancement.

Environment Assumptions

- Users have Python 3.x installed, alongside dependencies such as OpenCV, Pillow, TkinterDnD2, and sv_ttk.
- Files to be processed exist locally and are not DRM-protected or encrypted externally.

The clear determination of these requirements guided the design choices made in the project, ensuring that the final system meets user expectations and technical standards.

2.2 Targeted Users

The Steganography Analyzer is designed with a diverse user base in mind, each with specific needs that informed the application's features and usability criteria:

Cybersecurity Professionals

- These include analysts and penetration testers working to uncover covert communication channels used by threat actors.
- They require a reliable tool to analyze suspicious files, extract hidden payloads, and generate evidence reports to support investigations.

Digital Forensic Investigators

- Law enforcement personnel tasked with analyzing digital evidence in cybercrime cases.
- They need a multi-format solution to examine images, videos, audio recordings, and compressed files often seized during investigations.
- The system’s ability to provide file metadata and generate official reports addresses chain-of-custody and documentation requirements.

Academic Researchers and Educators

- Scholars and students studying information hiding and steganography benefit from a practical, hands-on tool.
- The graphical interface and multi-format support enable experimentation with various data-hiding techniques.
- Metadata extraction helps correlate steganographic artifacts with file properties, improving theoretical understanding.

Privacy Advocates and Enthusiasts

- Individuals using steganography to protect personal communications, intellectual property, or digital watermarking.
- They seek an easy-to-use application to embed private messages into familiar media formats without expert knowledge.

Software Developers and Open-Source Contributors

- Developers looking to extend or integrate the tool within larger cybersecurity frameworks.
- Modular design and clear interfaces facilitate contributions and customization.

Key User Requirements

- Ease of use with visual feedback and minimal command-line interaction.
- Reliable support for common media file types in everyday usage.
- Ability to produce and save reports suitable for documentation, presentations, or legal proceedings.

- Stability and predictable behavior when encountering unsupported or malformed files.

By addressing these user groups, the Steganography Analyzer balances specialized professional features with broad accessibility.

2.3 Details of Tools and Techniques Used / Implemented

Programming Language and Environment

- Python 3.x: Chosen for its versatility, extensive library ecosystem, and strong community support—ideal for rapid development and cross-platform compatibility.

User Interface Tools

- Tkinter: Python's built-in GUI framework provides native look-and-feel and fundamental interface elements like buttons, labels, text fields, and frames.
- sv_ttk (Themed Tkinter): An external theming package enhances the default Tkinter appearance with modern, dark-mode themes improving user experience and reducing eye strain during extended use.

Multimedia File Handling Libraries

- Pillow (PIL Fork): Responsible for image processing tasks, including loading images, extracting format and mode, and accessing pixel-level data required for steganography.
- OpenCV (Python Bindings): Handles video processing by opening video files, retrieving frame rates, frame counts, resolution, and duration for metadata reporting and accurate operation timing.

System and Utility Libraries

- mimetypes: Provides MIME-type detection by file extension, essential for routing input files to appropriate steganography backends.
- os module: Used for file system operations like getting file size, path access, and error handling.

Steganography Modules (Custom or External)

- stego_image, stego_audio, stego_video, stego_archive: Modularized backends implementing embed and extract functions tailored to each media type. These modules are responsible for the core data-hiding algorithms, presumably using techniques such as least significant bit modification or echo hiding.

Additional Techniques

- Metadata Extraction: Combines data from multimedia libraries and system queries to compile comprehensive file descriptions critical for contextual analysis.
- Progress Management: GUI-based progress bars provide user feedback during potentially time-intensive embedding/extraction.
- Error Handling and Notifications: The use of message box to inform users about unsupported files, operation failures, or successful completion enhances usability and robustness.

Development Environment

- Development utilizes common IDEs (e.g., PyCharm, VSCode) on multi-platform setups for testing and deployment.
- Use of virtual environments to manage dependencies ensures reproducibility and isolation.

Methodology

- Modular programming approach enables separation of concerns, facilitating testing, debugging, and future enhancement.
- Event-driven programming reflects the GUI model, responding to user actions like file selection or button clicks.

2.4 Advantages and Limitations of the Used Security Tools

2.4.1. Advantages

- Cross-Media Capability: Support across images, audio, video, and archive files makes the system remarkably flexible compared to many single-format tools.
- User-Friendly GUI: The use of Tkinter and sv_ttk creates an accessible interface that lowers the barrier to entry for users unfamiliar with command-line tools or complex configurations.
- Comprehensive Metadata Extraction: Provides valuable forensic insights beyond simple steganography, supporting investigative and educational use cases.
- Extensibility Through Modular Design: Each media type is handled by dedicated steganography modules, allowing for easy updating or replacement without affecting the whole system.
- Open Source and Python Ecosystem: Python's extensive libraries and open development foster rapid prototyping, community collaboration, and platform independence.

2.4.2. Limitations

- Lack of Encryption: While data is hidden, it is not encrypted. Extracted messages are exposed in plain form, limiting confidentiality against attackers who detect the hidden data.
- Performance Bottlenecks: Processing large video or audio files can be resource-intensive and slow due to interpreted Python execution and the overhead of multimedia libraries.
- Limited File Format Support: The tool currently supports a fixed set of common formats; unsupported or exotic file types cannot be processed.
- Dependency on External Libraries: Reliant on third-party packages such as OpenCV and Pillow; any bugs or compatibility issues in those libraries can affect overall performance.
- Absence of Advanced Steganalysis: No integrated mechanisms exist to detect steganographic content in arbitrary files, which limits forensic detection capabilities to reactive extraction rather than proactive discovery.
- Basic Reporting Format: The export functionality is limited to plaintext reports; richer formats like PDF or HTML would enhance usability and presentation.

Chapter 3
System Design

3.1 Flow Chart

A well-structured system design is essential for the efficient and reliable operation of any software tool. The Steganography Analyzer is designed in a modular fashion, prioritizing clear data flow, extensibility, and user experience.

System Flow (Textual Description & Elements for Diagram):

Start → User Interaction

- Application launches via GUI.
- User selects or drags & drops a file into the interface.

↓ (File Input Section)

File Selection & Detection

- GUI records file path from browse/drag-drop.
- System uses mimetypes and file extension to determine media type (image, audio, video, archive).

↓ (Metadata Section)

Metadata Extraction

- System invokes appropriate libraries (Pillow for images, OpenCV for video, OS/mimetypes for file stats).
- Metadata (file name, type, size, dimensions, duration, etc.) displayed in metadata viewer.

↓ (Operation Selection)

Operation Choice

- User chooses "Embed" or "Extract" via radio buttons.

↓Dispatch to Steganography Module

- Based on file type and operation, system routes file to the correct backend:
  - stego_image for images
  - stego_audio for audio
  - stego_video for videos
  - stego_archive for compressed files
- Calls either embed_message() or extract_message() as needed.

↓ (Execution & Feedback Section)

Steganographic Processing

- Progress bar activates during processing.
- Upon completion, results are displayed or any errors are shown to user with a messagebox.

↓ (Reporting)

Report Generation (Optional)

- User can save the results/metadata as a text report for documentation or forensic analysis.

↓ End

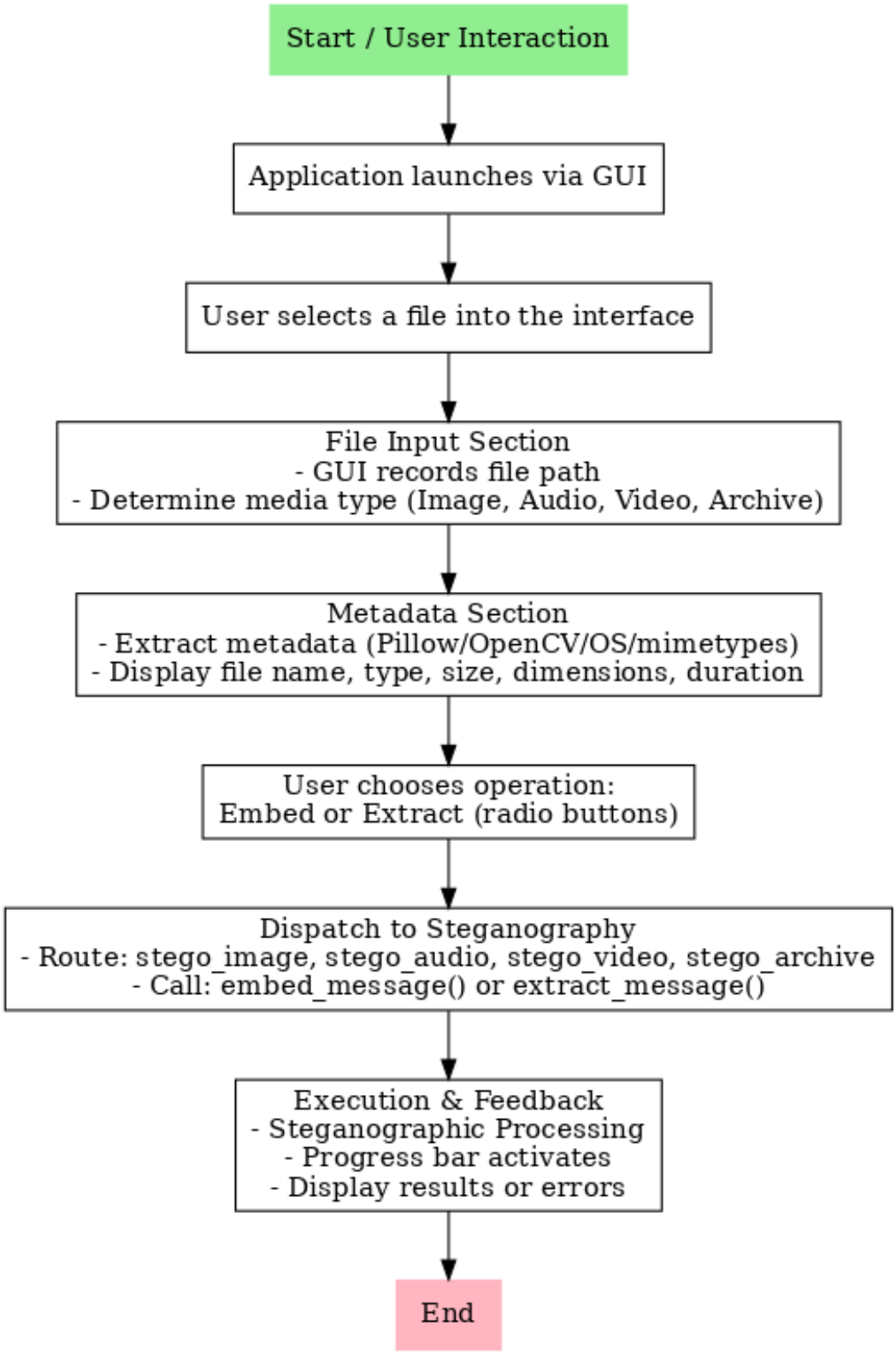

Figure 3.1: Flow Chart

Importance of flowchart:
Including a flowchart in the project documentation is essential because it visually summarizes the application's operations, clarifies the sequence of processes, and highlights the modular structure of the system. This makes the workflow easy to understand for both technical and non-technical readers, supports better communication, and streamlines future development or troubleshooting. A clear diagram enhances the professionalism of the report and ensures that the design decisions behind the Steganography Analyzer are transparent and accessible.

3.2 Dataset details

- File Types Supported:
  - Images: ".png", ".bmp", ".jpg", ".jpeg"
  - Audio: .wav", ".aiff", ".au", ".raw
  - Video: .mp4", ".mkv", ".mov", ".avi", ".webm", ".flv ", ".wmv"
  - Archives: zip", ".rar", ".7z", ".tar", ".gz", ".bz2", ".iso", ".dmg

- Dataset Structure:
  - Collection includes both original and stego-modified files for each media type.
  - Files are selected to span different sizes, resolutions, durations, and content types for comprehensive testing.

- Metadata Captured:
  - For images: File name, path, size (bytes), format (PNG, BMP.), color mode, dimensions (width x height).
  - For videos: File name, path, size, resolution, frame rate (fps), total frame count, duration (seconds).
  - For audio: File name, path, size, format (WAV, RAW etc), length (seconds).
  - For archives: File name, path, size, archive type (ZIP, RAR etc).

- Steganography Details:
  - Each file may have an embedded secret message using the application's embed functionality.
  - The tool supports extraction to verify presence and accuracy of the hidden data.

- Data Sources:
  - Files are sourced from publicly available datasets, user-generated samples, or anonymized project-specific collections.
  - No copyrighted or personally identifiable information is included without permission.

- Documentation:
  - For each test run, metadata and operation results (embed or extract) are stored and can be exported as a text report.

- Purpose of Dataset:
  - Validates tool capability across multiple real-world formats.
  - Enables analysis of metadata extraction, steganography embedding/extraction accuracy, and reporting functions.

Media Type	Formats	Metadata Fields Collected	Example Embedded Payloads
Images	PNG, BMP, JPG, JPEG	Name, size, format, mode, dimensions	Text message
Audio	WAV, RAW, AU, AIFF	Name, size, format, duration	Text message
Video	MP4, AVI, MKV, MOV, AVI, WEBM, FLV, WMV	Name, size, resolution, frame rate, duration	Text message
Archive	ZIP, RAR, 7Z, TAR, GZ, BZ2, ISO, DMG	Name, size, type	Text message

Table 3.1: Data Set

Chapter 4
Development

4.1 Source code

The Steganography Analyzer project consists of six primary Python modules working together to provide a comprehensive, multi-format steganography tool. Key modules include stego logic for images, audio, video, archives, a centralized manager, and a modern GUI.

4.1.1 Project Structure

```
Steganography-Analyzer/
│
├── requirements.txt          # List of Python dependencies (e.g., customtkinter, Pillow, OpenCV, mutagen)
│
├── main_app.py               # Main GUI application (ModernSteganographyApp class)
│
├── stego_manager.py          # Central file type detector and stego operation dispatcher
│
├── stego_image.py            # Image steganography (LSB) module for PNG and BMP files
│
├── stego_audio.py            # Audio steganography module (LSB + metadata methods), excluding MP3
│
├── stego_video.py            # Video steganography module (FFmpeg/FFprobe metadata embedding/extraction)
│
├── stego_archive.py          # Archive steganography module (appends secret message with marker)
│
├── tools/                    # External binaries and helper scripts
│   ├── ffmpeg.exe            # FFmpeg executable for video processing
│   ├── ffprobe.exe           # FFprobe executable for video metadata extraction
│   └── (other helper utilities if any)
│
├── resources/                # Optional folder for demo files, sample media, or test data
│   ├── images/
│   ├── audio/
│   ├── videos/
│   └── archives/
│
├── docs/                     # Documentation files, reports, diagrams
│   ├── Project_Report.docx
│   ├── Flowchart.png
│   └── Other documentation assets
│
```

```
└── tests/                  # Optional automated or manual test scripts and test files
    ├── test_images/
    ├── test_audio/
    ├── test_videos/
    └── test_archives/
```

4.1.2 stego_image.py: Image Steganography Module

```
from stegano import lsb
from PIL import Image, ImageFile
import os

# Allow very large images without warnings
Image.MAX_IMAGE_PIXELS = None
ImageFile.LOAD_TRUNCATED_IMAGES = True

# Supported formats (JPGs will be converted)
SUPPORTED_IMAGE_TYPES = [".png", ".bmp", ".jpg", ".jpeg"]

def is_supported_image(file_path: str) -> bool:
    """Check if the file extension is a supported image type."""
    ext = os.path.splitext(file_path)[-1].lower()
    return ext in SUPPORTED_IMAGE_TYPES

def convert_jpg_to_png(input_path: str) -> str:
    """Convert JPG/JPEG to PNG to ensure LSB stego works
reliably."""
    output_path = os.path.splitext(input_path)[0] +
"_converted.png"
    try:
        img = Image.open(input_path)
        img = img.convert("RGB")  # Remove alpha if present
        img.save(output_path, "PNG", optimize=False)
        return output_path
    except Exception as e:
        raise ValueError(f"❌ JPG conversion failed: {str(e)}")

def embed_text_in_image(cover_image_path: str,
output_image_path: str, secret_text: str):
    """Embed secret text into an image using LSB
steganography."""
    if not is_supported_image(cover_image_path):
        raise ValueError("Only PNG, BMP, JPG, and JPEG images
are supported.")

    # Convert JPG/JPEG before embedding
    ext = os.path.splitext(cover_image_path)[-1].lower()
    if ext in [".jpg", ".jpeg"]:
        cover_image_path = convert_jpg_to_png(cover_image_path)
```

```
    try:
        # Hide the message
        secret_image = lsb.hide(cover_image_path, secret_text)

        # Save safely for large PNGs
        secret_image.save(output_image_path, format="PNG",
optimize=False)

        return True, f"✅ Message embedded successfully in:
{output_image_path}"
    except Exception as e:
        return False, f"❌ Failed to embed message: {str(e)}"

def extract_text_from_image(stego_image_path: str):
    """Extract hidden text from an image using LSB
steganography."""
    if not is_supported_image(stego_image_path):
        raise ValueError("Only PNG, BMP, JPG, and JPEG images
are supported.")

    # Convert JPG/JPEG before extraction
    ext = os.path.splitext(stego_image_path)[-1].lower()
    if ext in [".jpg", ".jpeg"]:
        stego_image_path = convert_jpg_to_png(stego_image_path)

    try:
        message = lsb.reveal(stego_image_path)
        if message:
            return True, message
        else:
            return False, "⚠️ No hidden message found."
    except Exception as e:
        return False, f"❌ Error extracting message: {str(e)}"
```

This Python script implements Least Significant Bit (LSB) steganography for image files, allowing the embedding and extraction of hidden messages into supported image formats. It uses the Stegano library for LSB manipulation and Pillow for image file operations.

Key Components:

Supported Formats:

The script supports the following image formats for steganography:

- **PNG** (.png)
- **BMP** (.bmp)
- **JPG** (.jpg)
- **JPEG** (.jpeg)

Since LSB (Least Significant Bit) steganography works best on **lossless formats**, JPG/JPEG images are first converted to **PNG** before embedding/extraction to avoid data loss.

File Type Check:
Function: **is_supported_image(file_path: str) -> bool**
Purpose: Ensures that only supported image formats are processed.
How it works:

- Extracts the file extension from the given path.
- Compares it with the allowed extensions (.png, .bmp, .jpg, .jpeg).
- Returns **True** if valid, else **False**

Embedding Secret Message:

Function: **embed_text_in_image(cover_image_path, output_image_path, secret_text)**
Steps:

1. Validates the input image format.
2. Converts JPG/JPEG images to PNG using **convert_jpg_to_png()**.
3. Uses stegano.lsb.hide() to embed the secret message inside the cover image.
4. Saves the modified image as **PNG** (to preserve hidden data).

Output:

- Returns success/failure status and a user-friendly message.

Extracting Secret Message:

Function: **extract_text_from_image(stego_image_path)**
Steps:

1. Validates the input image format.
2. Converts JPG/JPEG to PNG before extraction.
3. Uses stegano.lsb.reveal() to retrieve the hidden text.
4. If no message exists, returns a warning.

Output:

- Hidden message (if found).
- Error/warning message (if not found or failed).

Robust Error Handling:

Truncated Images:

- ImageFile.LOAD_TRUNCATED_IMAGES = True allows partially corrupted or large images to be processed.

Large Images:

- Image.MAX_IMAGE_PIXELS = None prevents warnings when handling very large images.

Safe Conversion:

- Conversion of JPG to PNG ensures LSB embedding is not corrupted by compression.

Error Reporting:

- Returns detailed error messages (❌ Failed to embed message: ...)

Advantages:

- Supports multiple common image formats.
- Handles **large images** without crashing.
- Converts JPG/JPEG to lossless PNG for reliability.

- Simple interface: only two main functions (embed_text_in_image and extract_text_from_image).
- Provides clear error and success messages for user feedback.

Limitations:

- Works only with **image files** (no audio, video, or archives).
- **Lossy formats (JPG/JPEG)** need conversion, which increases file size.
- Embedding too large a message may fail or distort the image.
- Steganography is detectable by advanced forensic tools (not foolproof).

Usage in the Project:

This script forms the **image-handling module** of the **Steganography Analyzer Tool**.

- Integrated into the Tkinter GUI:
  - **Embed Feature:** Allows the user to hide secret messages inside images.
  - **Extract Feature:** Enables retrieving hidden text from stego-images.
- Works alongside other modules (stego_audio.py, stego_video.py, stego_archive.py) to provide **multi-format steganography support**.

4.1.3 stego_audio.py - Audio Steganography (MP3 Removed)

```
import os
import wave
import contextlib
from mutagen.flac import FLAC
from mutagen.mp4 import MP4
from mutagen.oggvorbis import OggVorbis
from mutagen.aiff import AIFF
from mutagen.wave import WAVE
from pydub import AudioSegment
import tempfile
import struct

SUPPORTED_LSB_FORMATS = [".wav", ".aiff", ".au", ".raw"]
SUPPORTED_METADATA_FORMATS = [".flac", ".m4a", ".mp4", ".ogg",
".aac"]
SUPPORTED_CONVERTED_FORMATS = [".opus", ".amr", ".ac3", ".dts",
".tta", ".ape", ".wv"]

ALL_SUPPORTED_FORMATS = SUPPORTED_LSB_FORMATS +
SUPPORTED_METADATA_FORMATS + SUPPORTED_CONVERTED_FORMATS

def is_supported_audio(file_path):
    """Check if the audio format is supported"""
    ext = os.path.splitext(file_path)[-1].lower()
    return ext in ALL_SUPPORTED_FORMATS

def get_audio_format_type(file_path):
```

```
    """Determine the audio format category"""
    ext = os.path.splitext(file_path)[-1].lower()

    if ext in SUPPORTED_LSB_FORMATS:
        return "lsb"
    elif ext in SUPPORTED_METADATA_FORMATS:
        return "metadata"
    elif ext in SUPPORTED_CONVERTED_FORMATS:
        return "convert"
    else:
        return "unsupported"

# ------------------------
# Main Audio Steganography Functions
# ------------------------

def embed_text_in_audio(input_path, output_path, secret_text):
    if not is_supported_audio(input_path):
        ext = os.path.splitext(input_path)[1].lower()
        if ext == ".mp3":
            return False, "❌ MP3 format not supported due to
metadata compatibility issues."
        return False, "❌ Unsupported audio format."

    format_type = get_audio_format_type(input_path)

    try:
        if format_type == "lsb":
            return embed_lsb_audio(input_path, output_path,
secret_text)
        elif format_type == "metadata":
            return embed_metadata_audio(input_path, output_path,
secret_text)
        elif format_type == "convert":
            return embed_convert_audio(input_path, output_path,
secret_text)
        else:
            return False, "❌ Unsupported audio format."
    except Exception as e:
        return False, f"❌ Error embedding message: {str(e)}"

def extract_text_from_audio(input_path):
    if not is_supported_audio(input_path):
        ext = os.path.splitext(input_path)[1].lower()
        if ext == ".mp3":
            return False, "❌ MP3 format not supported due to
metadata compatibility issues."
        return False, "❌ Unsupported audio format."
```

```
    format_type = get_audio_format_type(input_path)

    try:
        if format_type == "lsb":
            return extract_lsb_audio(input_path)
        elif format_type == "metadata":
            return extract_metadata_audio(input_path)
        elif format_type == "convert":
            return extract_convert_audio(input_path)
        else:
            return False, "❌ Unsupported audio format."
    except Exception as e:
        return False, f"❌ Error extracting message: {str(e)}"

def _text_to_bits(text):
    return ''.join(format(ord(c), '08b') for c in text)

def _bits_to_text(bits):
    chars = [bits[i:i+8] for i in range(0, len(bits), 8)]
    return ''.join(chr(int(char, 2)) for char in chars if char)

def embed_lsb_audio(input_path, output_path, secret_text):
    ext = os.path.splitext(input_path)[1].lower()

    if ext == ".wav":
        return embed_text_in_wav(input_path, output_path,
secret_text)
    elif ext in [".aiff", ".au", ".raw"]:
        temp_wav_in = tempfile.mktemp(suffix=".wav")
        temp_wav_out = tempfile.mktemp(suffix=".wav")

        try:
            # Convert input to WAV
            audio = AudioSegment.from_file(input_path)
            audio.export(temp_wav_in, format="wav")

            # Embed in WAV
            success, result = embed_text_in_wav(temp_wav_in,
temp_wav_out, secret_text)

            if success:
                # Convert back to original format
                stego_audio =
AudioSegment.from_wav(temp_wav_out)
                stego_audio.export(output_path,
format=ext[1:])  # Remove the dot
                return True, f"✅ Message embedded in
{ext.upper()} file: {output_path}"
            else:
                return success, result
```

```
        finally:
            # Clean up temp files
            for temp_file in [temp_wav_in, temp_wav_out]:
                if os.path.exists(temp_file):
                    os.remove(temp_file)

def extract_lsb_audio(input_path):
    """Extract text using LSB method from uncompressed audio"""
    ext = os.path.splitext(input_path)[1].lower()

    if ext == ".wav":
        return extract_text_from_wav(input_path)
    elif ext in [".aiff", ".au", ".raw"]:
        # Convert to WAV first, then extract
        temp_wav = tempfile.mktemp(suffix=".wav")

        try:
            # Convert to WAV
            audio = AudioSegment.from_file(input_path)
            audio.export(temp_wav, format="wav")

            # Extract from WAV
            return extract_text_from_wav(temp_wav)

        finally:
            if os.path.exists(temp_wav):
                os.remove(temp_wav)

def embed_text_in_wav(input_path, output_path, secret_text):
    """Enhanced WAV LSB embedding"""
    try:
        with wave.open(input_path, 'rb') as audio:
            params = audio.getparams()
            frames =
bytearray(audio.readframes(audio.getnframes()))

        # Add length prefix and EOF marker
        message_length = len(secret_text)
        length_bits = format(message_length, '032b')  # 32-bit
length
        text_bits = _text_to_bits(secret_text)
        bits = length_bits + text_bits + '1111111111111110'  #
EOF marker

        if len(bits) > len(frames):
            return False, f"❌ Message too large. Max capacity:
{len(frames)//8} characters"

        # Embed bits in LSB
```

```
        for i, bit in enumerate(bits):
            frames[i] = (frames[i] & 254) | int(bit)

        with wave.open(output_path, 'wb') as stego_audio:
            stego_audio.setparams(params)
            stego_audio.writeframes(frames)

        return True, f"✅ Message embedded in WAV:
{output_path}"

    except Exception as e:
        return False, f"❌ WAV embedding error: {str(e)}"

def extract_text_from_wav(stego_path):
    """Enhanced WAV LSB extraction"""
    try:
        with wave.open(stego_path, 'rb') as audio:
            frames =
bytearray(audio.readframes(audio.getnframes()))

        # Extract length (first 32 bits)
        length_bits = ''
        for i in range(32):
            if i < len(frames):
                length_bits += str(frames[i] & 1)

        if len(length_bits) < 32:
            return False, "⚠️ No valid message found."

        message_length = int(length_bits, 2)

        if message_length <= 0 or message_length > 100000:  #
Sanity check
            return False, "⚠️ No valid hidden message found."

        # Extract message bits
        message_bits = ''
        for i in range(32, 32 + (message_length * 8)):
            if i < len(frames):
                message_bits += str(frames[i] & 1)

        if len(message_bits) < message_length * 8:
            return False, "⚠️ Incomplete message found."

        return True, _bits_to_text(message_bits)

    except Exception as e:
        return False, f"❌ WAV extraction error: {str(e)}"
```

```
# ------------------------
# Metadata Steganography (Compressed Formats - NO MP3)
# ------------------------

def embed_metadata_audio(input_path, output_path, secret_text):
    """Embed text in audio metadata for compressed formats (NO
MP3)"""
    ext = os.path.splitext(input_path)[1].lower()

    try:
        # Copy file first
        if input_path != output_path:
            with open(input_path, 'rb') as src,
open(output_path, 'wb') as dst:
                dst.write(src.read())

        # Embed in metadata based on format (MP3 REMOVED)
        if ext == ".flac":
            return embed_flac_metadata(output_path, secret_text)
        elif ext in [".m4a", ".mp4", ".aac"]:
            return embed_mp4_metadata(output_path, secret_text)
        elif ext == ".ogg":
            return embed_ogg_metadata(output_path, secret_text)
        else:
            return False, f"❌ Metadata embedding not supported
for {ext}"

    except Exception as e:
        return False, f"❌ Metadata embedding error: {str(e)}"

def extract_metadata_audio(input_path):
    """Extract text from audio metadata (NO MP3)"""
    ext = os.path.splitext(input_path)[1].lower()

    try:
        # MP3 support removed
        if ext == ".flac":
            return extract_flac_metadata(input_path)
        elif ext in [".m4a", ".mp4", ".aac"]:
            return extract_mp4_metadata(input_path)
        elif ext == ".ogg":
            return extract_ogg_metadata(input_path)
        else:
            return False, f"❌ Metadata extraction not
supported for {ext}"

    except Exception as e:
        return False, f"❌ Metadata extraction error: {str(e)}"

# Specific metadata handlers (MP3 functions removed)
```

```
def embed_flac_metadata(file_path, secret_text):
    """Embed in FLAC Vorbis comments"""
    try:
        audio = FLAC(file_path)
        # Use list for mutagen compatibility
        audio["COMMENT"] = [secret_text]
        audio.save()
        return True, "✅ Message embedded in FLAC metadata."
    except Exception as e:
        return False, f"❌ FLAC metadata error: {str(e)}"

def extract_flac_metadata(file_path):
    """Extract from FLAC Vorbis comments"""
    try:
        audio = FLAC(file_path)
        comment = audio.get("COMMENT", [None])[0]
        if comment is None or str(comment).strip() == "":
            return False, "⚠️ No comment metadata found."
        return True, comment
    except Exception as e:
        return False, f"❌ FLAC metadata error: {str(e)}"

def embed_mp4_metadata(file_path, secret_text):
    try:
        audio = MP4(file_path)
        # Use comment tag \xa9cmt for embedding
        audio["\xa9cmt"] = [secret_text]  # Must be a list for
mutagen MP4
        audio.save()
        # Verify saving success by reloading
        audio2 = MP4(file_path)
        if audio2.get("\xa9cmt") and secret_text ==
audio2["\xa9cmt"][0]:
            return True, "✅ Message embedded in MP4 metadata."
        else:
            return False, "❌ Failed to verify MP4 metadata
embedding."
    except Exception as e:
        return False, f"❌ MP4 metadata error: {str(e)}"

def extract_mp4_metadata(file_path):
    """Extract from MP4/M4A/AAC metadata with robust checks"""
    try:
        audio = MP4(file_path)
        comment = audio.get("\xa9cmt", [None])[0]

        if comment is None or str(comment).strip() == "":
            return False, "⚠️ No comment metadata found."
```

```
        # Ensure comment is a string (sometimes can be bytes or
list)
        if isinstance(comment, bytes):
            comment = comment.decode('utf-8', errors='replace')

        if not isinstance(comment, str):
            comment = str(comment)

        return True, comment
    except Exception as e:
        return False, f"❌ MP4 metadata error: {str(e)}"

def embed_ogg_metadata(file_path, secret_text):
    """Embed in OGG Vorbis comments with verification"""

    try:
        audio = OggVorbis(file_path)

        # Set COMMENT as a list to comply with mutagen's
expectations
        audio["COMMENT"] = [secret_text]
        audio.save()

        # Reload to verify
        audio_reloaded = OggVorbis(file_path)
        saved_comment = audio_reloaded.get("COMMENT", [None])[0]

        if saved_comment == secret_text:
            return True, "✅ Message embedded in OGG metadata."
        else:
            return False, "❌ Failed to verify OGG metadata
embedding."

    except Exception as e:
        return False, f"❌ OGG metadata error: {str(e)}"

def extract_ogg_metadata(file_path):
    """Extract from OGG Vorbis comments robustly"""
    try:
        audio = OggVorbis(file_path)
        comment = audio.get("COMMENT", [None])[0]

        if comment is None or str(comment).strip() == "":
            return False, "⚠️ No comment metadata found."

        if isinstance(comment, bytes):
            comment = comment.decode('utf-8', errors='replace')

        if not isinstance(comment, str):
            comment = str(comment)
```

```
        return True, comment
    except Exception as e:
        return False, f"❌ OGG metadata error: {str(e)}"

# ------------------------
# Convert-to-WAV Method (Exotic Formats)
# ------------------------

def embed_convert_audio(input_path, output_path, secret_text):
    """Convert exotic formats to WAV, embed, then convert
back"""
    ext = os.path.splitext(input_path)[1].lower()
    temp_wav_in = tempfile.mktemp(suffix=".wav")
    temp_wav_out = tempfile.mktemp(suffix=".wav")

    try:
        # Convert input to WAV
        audio = AudioSegment.from_file(input_path)
        audio.export(temp_wav_in, format="wav")

        # Embed in WAV
        success, result = embed_text_in_wav(temp_wav_in,
temp_wav_out, secret_text)

        if success:
            # Convert back to original format
            stego_audio = AudioSegment.from_wav(temp_wav_out)
            stego_audio.export(output_path, format=ext[1:])
            return True, f"✅ Message embedded in {ext.upper()}
file: {output_path}"
        else:
            return success, result

    except Exception as e:
        return False, f"❌ Conversion embedding error:
{str(e)}"

    finally:
        # Clean up temp files
        for temp_file in [temp_wav_in, temp_wav_out]:
            if os.path.exists(temp_file):
                os.remove(temp_file)

def extract_convert_audio(input_path):
    """Convert exotic formats to WAV and extract"""
    temp_wav = tempfile.mktemp(suffix=".wav")

    try:
        # Convert to WAV
```

```
        audio = AudioSegment.from_file(input_path)
        audio.export(temp_wav, format="wav")

        # Extract from WAV
        return extract_text_from_wav(temp_wav)

    except Exception as e:
        return False, f"❌ Conversion extraction error:
{str(e)}"

    finally:
        if os.path.exists(temp_wav):
            os.remove(temp_wav)

# ------------------------
# Utility Functions
# ------------------------

def get_audio_info(file_path):
    """Get detailed audio file information (NO MP3)"""
    try:
        ext = os.path.splitext(file_path)[1].lower()

        # Check if MP3 and return error message
        if ext == ".mp3":
            return {"error": "MP3 format not supported due to
metadata compatibility issues"}

        audio = AudioSegment.from_file(file_path)
        format_type = get_audio_format_type(file_path)

        info = {
            "format": ext[1:].upper(),
            "format_type": format_type,
            "duration": len(audio) / 1000.0,  # seconds
            "channels": audio.channels,
            "sample_rate": audio.frame_rate,
            "frame_width": audio.frame_width,
            "max_message_length": estimate_capacity(file_path)
        }
        return info

    except Exception as e:
        return {"error": str(e)}

def estimate_capacity(file_path):
    """Estimate message capacity for audio file (NO MP3)"""
    try:
        ext = os.path.splitext(file_path)[1].lower()
```

```
        # No capacity for MP3
        if ext == ".mp3":
            return 0

        format_type = get_audio_format_type(file_path)

        if format_type == "lsb":
            # For LSB, capacity depends on audio length
            audio = AudioSegment.from_file(file_path)
            total_samples = len(audio.raw_data)
            return total_samples // 8  # rough estimate

        elif format_type in ["metadata", "convert"]:
            # For metadata, typically limited to comment field
            return 1000  # conservative estimate

        return 0

    except Exception:
        return 0

def get_supported_formats_info():
    """Get information about supported formats"""
    return {
        "lsb_formats": SUPPORTED_LSB_FORMATS,
        "metadata_formats": SUPPORTED_METADATA_FORMATS,
        "convert_formats": SUPPORTED_CONVERTED_FORMATS,
        "all_formats": ALL_SUPPORTED_FORMATS,
        "removed_formats": [".mp3"],
        "removal_reason": "MP3 ID3 comment metadata
compatibility issues"
    }
```

This module enables embedding and extraction of hidden messages in audio files across multiple formats. MP3 support is intentionally removed due to known compatibility issues with its metadata for steganography applications.

Main Features and Workflow:

Supported Formats:

The script supports **three categories** of audio formats for embedding and extracting secret messages:

- **LSB (Least Significant Bit) Embedding Formats** (uncompressed, sample-based hiding):
  - .wav, .aiff, .au, .raw
- **Metadata Embedding Formats** (compressed, comment-tag hiding):
  - .flac, .m4a, .mp4, .ogg, .aac
- **Converted-to-WAV Formats** (exotic formats converted into WAV internally):
  - .opus, .amr, .ac3, .dts, .tta, .ape, .wv

- **Explicitly Unsupported:**
    - .mp3 (due to **ID3 metadata compatibility issues**)

File Type Check:

Before performing embedding or extraction, the script checks the file type using:

- is_supported_audio(file_path) → Verifies if extension is in the supported list.
- get_audio_format_type(file_path) → Categorizes into **lsb**, **metadata**, **convert**, or **unsupported**.

This ensures robustness and prevents unexpected errors during operations.

Embedding Secret Message:

Methods Used:

1. **LSB Embedding (for uncompressed formats):**
    - Each bit of the secret message is stored in the **least significant bit** of audio samples.
    - Uses a **32-bit length prefix** and an **EOF marker** (1111111111111110) for accurate message retrieval.
2. **Metadata Embedding (for compressed formats):**
    - Embeds the message inside comment metadata fields:
    - FLAC → COMMENT tag
    - MP4/M4A/AAC → ©cmt tag
    - OGG → COMMENT tag
3. **Convert-to-WAV Method (for exotic formats):**
    - Input audio is converted to .wav.
    - Message embedded via **WAV LSB method**.
    - Converted back into the **original format** after embedding.

Extracting Secret Message:

Methods Used:

1. **LSB Extraction:**
    - Reads first 32 bits to determine message length.
    - Collects message bits accordingly.
    - Performs **sanity checks** (length validation, completeness).
2. **Metadata Extraction:**
    - Reads message from comment fields (FLAC, MP4, OGG).
    - Handles different datatypes (string, bytes, list) robustly.
3. **Converted Format Extraction:**
    - Converts exotic formats to .wav.
    - Extracts message using **WAV LSB extraction**.

Robust Error Handling:

- **Unsupported Format Handling:** Clear error returned for unsupported or excluded formats (e.g., .mp3).
- **Capacity Check:** Prevents embedding messages larger than available sample space.
- **Try/Except Blocks:** Wrapped around critical embedding/extraction operations to catch runtime issues.
- **Temporary File Cleanup:** Ensures temp files are always deleted, even on failure.

- **Sanity Checks:** Ensures extracted message length is reasonable (avoiding false positives).

Additional Features:

1. **Audio Info Extraction (get_audio_info)**
    - Returns details such as format, duration, channels, sample rate, frame width, and estimated max message capacity.
2. **Capacity Estimation (estimate_capacity)**
    - Estimates how many characters can be hidden depending on format type.
3. **Supported Format Info (get_supported_formats_info)**
    - Returns categorized format support and reason for excluding MP3.
4. **Verification of Metadata Saving**
    - After embedding in metadata (FLAC, MP4, OGG), the script re-reads file to confirm successful embedding.

Advantages:

- **Multi-format support:** Works with uncompressed, compressed, and exotic formats.
- **Robust structure:** Separates handling methods by file type.
- **Metadata flexibility:** Leverages comment fields where applicable.
- **Error resilience:** Strong error handling, sanity checks, and clear messages.
- **Extensible:** New formats can be added easily in categorized lists.
- **Security through obscurity:** Hidden messages are not obvious without extraction.

Limitations:

- **MP3 Unsupported:** Due to ID3 comment metadata inconsistencies.
- **Capacity Constraints:**
    - LSB: Limited by number of samples.
    - Metadata: Limited by tag size (~1000 characters safe).
- **Quality Degradation (LSB):** Slight changes in waveform (though inaudible in most cases).
- **Dependency on External Libraries:** Relies on pydub, mutagen, and system-installed audio codecs.

Usage in the Project:

This script can be integrated into a **Steganography System** where:

- **Embed Phase:** Hide confidential data/messages in different audio files depending on format.
- **Extract Phase:** Retrieve hidden messages securely with correct decoding method.
- **Project Applications:**
    - Secure communication (covert channels).
    - Digital watermarking (authorship verification).
    - Hidden notes in distributed media files.

4.1.4 stego_video.py – Video module

```
import os
import subprocess
import json

SUPPORTED_VIDEO = [".mp4", ".mkv", ".mov", ".avi", ".webm",
".flv", ".wmv"]

FFMPEG_PATH =
os.path.normpath(r"C:\Users\LENOVO\OneDrive\Desktop\ImageStegoAn
alyzer\Stego Analyser\Multi-Stego-Toolkit\Tools\ffmpeg.exe")
FFPROBE_PATH =
os.path.normpath(r"C:\Users\LENOVO\OneDrive\Desktop\ImageStegoAn
alyzer\Stego Analyser\Multi-Stego-Toolkit\Tools\ffprobe.exe")

def is_supported_video(path):
"""Check if the video has a supported extension."""
ext = os.path.splitext(path)[1].lower()
return ext in SUPPORTED_VIDEO

def embed_text_in_video(input_path, output_path, secret_text):
"""
Embed a secret text message into video metadata using FFmpeg.
"""
if not is_supported_video(input_path):
return False, "❌ Unsupported video format."

try:
cmd = [
FFMPEG_PATH, "-y",
"-i", input_path,
"-metadata", f"comment={secret_text}",
"-codec", "copy",
output_path
]
subprocess.run(cmd, check=True, stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL)
return True, f"✅ Message embedded in video: {output_path}"

except subprocess.CalledProcessError:
return False, "❌ FFmpeg failed to process the video."
except Exception as e:
return False, f"❌ Unexpected error: {str(e)}"

def extract_text_from_video(video_path):
"""
Extract the secret text from video metadata using FFprobe.
"""
if not is_supported_video(video_path):
return False, "❌ Unsupported video format."
```

```
try:
cmd = [
FFPROBE_PATH,
"-v", "quiet",                          # Suppress all output
"-print_format", "json",                # Output as JSON
"-show_format",                         # Show format tags
video_path
]
result = subprocess.run(cmd, capture_output=True, text=True,
check=True)
metadata = json.loads(result.stdout)

comment = metadata.get("format", {}).get("tags",
{}).get("comment")
if comment:
return True, comment
else:
return False, "⚠️ No hidden message found in metadata."

except subprocess.CalledProcessError:
return False, "❌ FFprobe failed to analyze the video."
except Exception as e:
return False, f"❌ Unexpected error: {str(e)}"
```

This Python module manages steganography tasks for video files by embedding and extracting hidden messages in video metadata using FFmpeg and FFprobe, two industry-standard multimedia command-line tools.

Key Functions:

Supported Formats:

- The script defines a list of supported video extensions (.mp4, .avi, .mov, .mkv, .webm, .flv, .wmv). The is_supported_video(path) function checks the file extension to ensure compatibility.

Embedding Text:

- The function embed_text_in_video(input_path, output_path, secret_text) embeds a secret message into the video's metadata field ("comment") using FFmpeg.

How it works:

- Verifies the video extension.
- Builds a command-line invocation of FFmpeg, inserting the secret text into the comment tag of the video file metadata with minimal reencoding (-codec copy).
- Handles errors and returns a success/failure message with details.

Extracting Text:

- The function extract_text_from_video(video_path) uses FFprobe to read and extract

the comment field.

How it works:

- Verifies the video extension.
- Runs FFprobe in quiet mode and requests output as JSON.
- Parses the resulting JSON to search for text in the comment field.
- Handles and reports errors or absence of a hidden message.

External Tool Usage:

- Paths to ffmpeg.exe and ffprobe.exe are specified and normalized, ensuring the tools are called correctly regardless of OS.

Error Handling:

- The code includes robust exception handling for process errors, unsupported file formats, and unexpected situations, returning clear status messages for GUI presentation.

Code Integration and Usage:

- These functions are intended for integration with the main Steganography Analyzer application. When a user selects a video and chooses an embed or extract action, the GUI calls these functions to perform the work in the background. Progress and results are communicated back to the user via on-screen messages.

This approach ensures:

- High compatibility with common video formats.
- Zero perceptible change to video content, as embedding occurs in metadata only.
- Ease of extraction by reading the comment tag.

Advantages:

- Fast operation since video data is untouched (no reencoding).
- No impact on video quality or file size (metadata only).
- Use of trusted open-source tools (FFmpeg/FFprobe) ensures reliability.

4.1.5 stego_archive.py: Archive File Steganography Module

```
import os

SUPPORTED_ARCHIVES = [".zip", ".rar", ".7z", ".tar", ".gz", ".bz2", ".iso",
".dmg"]

# Unique marker so we know where our message starts
MARKER = b"<<SECRET_MSG_START>>"

def is_supported_archive(file_path):
    ext = os.path.splitext(file_path)[-1].lower()
    return ext in SUPPORTED_ARCHIVES
```

```
def embed_text_in_archive(input_path, output_path, secret_text):
    if not is_supported_archive(input_path):
        return False, "❌ Unsupported archive type."

    try:
        with open(input_path, "rb") as original_file:
            data = original_file.read()

        with open(output_path, "wb") as stego_file:
            stego_file.write(data)
            stego_file.write(MARKER + secret_text.encode("utf-8"))

        return True, f"✅ Message embedded in archive: {output_path}"
    except Exception as e:
        return False, f"❌ Error: {str(e)}"

def extract_text_from_archive(stego_path):
    try:
        with open(stego_path, "rb") as f:
            data = f.read()

        index = data.find(MARKER)
        if index == -1:
            return False, "⚠️ No hidden message found."

        secret_data = data[index + len(MARKER):]
        return True, secret_data.decode("utf-8", errors="replace")
    except Exception as e:
        return False, f"❌ Error: {str(e)}"
```

This script enables simple yet effective steganography for common archive files such as ZIP, RAR, 7Z, TAR, GZ, ISO, and others supported in the project. The technique works by appending secret messages to the end of archive files, separated by a unique marker to identify the start of the hidden content.

Key Features & Functions:

Supported Formats:

- The module supports .zip, .rar, .7z, .tar, .gz, .bz2, .iso, .dmg, providing flexibility across the most widely used archive file types.

File Type Detection:

- is_supported_archive(file_path) checks if the file extension matches any supported archive type.

Embedding Secret Message:

- embed_text_in_archive(input_path, output_path, secret_text) appends secret content to the archive:

- Reads the full binary data of the original archive file.
- Writes this data to the output file, then appends a marker (b"<<SECRET_MSG_START>>") followed by the encoded secret.
- This approach preserves the archive's integrity, so extraction and decompression tools still work, while only your tool can find and interpret the secret data.

Extracting Secret Message:

- extract_text_from_archive(stego_path) retrieves the hidden message:
- Reads the binary data from the steganographic archive.
- Searches for the unique marker.
- If found, extracts everything after the marker and decodes it from bytes to text.
- If not found, returns a "No hidden message found" notice.

Design Considerations:

Marker Use:

- By using a distinct binary marker, the hidden message's location is unambiguous, avoiding accidental detection or misinterpretation of archive file data.

No Archive Format Modification:

- The technique does not alter the existing compressed data, ensuring original files remain totally usable even in their stego-hybrid form.

Error Handling:

- Functions comprehensively handle file I/O errors and present user-friendly feedback for unsupported file types or operation failures.

Advantages:

- Compatible with all major archive formats.
- Simple and fast, requiring only binary file manipulation with no need for archive libraries.
- Preservation of the archive's usability for extraction and decompression.

Limitations:

- The hidden message is appended at the end of the file in plain form; not encrypted or compressed.
- Anyone aware of the marker and method could potentially locate and extract the hidden data.
- For increased security, additional encryption could be layered atop this method.

Integration:

- This module is called by the main application whenever users request to hide or reveal messages in any supported archive. It works seamlessly with the GUI and reporting layers, returning results and error messages as appropriate.

4.1.5 stego_manager.py: Unified Steganography Interface

```
import os
from stego_image import is_supported_image, embed_text_in_image,
extract_text_from_image
from stego_audio import is_supported_audio, embed_text_in_audio,
extract_text_from_audio
from stego_archive import is_supported_archive,
embed_text_in_archive, extract_text_from_archive
from stego_video import is_supported_video, embed_text_in_video,
extract_text_from_video

def get_file_type(file_path):
    if is_supported_image(file_path):
        return 'image'
    elif is_supported_audio(file_path):
        return 'audio'
    elif is_supported_video(file_path):
        return 'video'
    elif is_supported_archive(file_path):
        return 'archive'
    else:
        return None

def embed_message(input_path, output_path, message):
    file_type = get_file_type(input_path)

    if file_type == 'image':
        return embed_text_in_image(input_path, output_path,
message)
    elif file_type == 'audio':
        return embed_text_in_audio(input_path, output_path,
message)
    elif file_type == 'video':
        return embed_text_in_video(input_path, output_path,
message)
    elif file_type == 'archive':
        return embed_text_in_archive(input_path, output_path,
message)
    else:
        return False, "❌ Unsupported file type for embedding."

def extract_message(input_path):
    file_type = get_file_type(input_path)

    if file_type == 'image':
        return extract_text_from_image(input_path)
    elif file_type == 'audio':
        return extract_text_from_audio(input_path)
    elif file_type == 'video':
```

```
        return extract_text_from_video(input_path)
    elif file_type == 'archive':
        return extract_text_from_archive(input_path)
    else:
        return False, "❌ Unsupported file type for
extraction."
```

The stego_manager.py script acts as a central logical controller in the Steganography Analyzer project, managing the coordination between different media steganography modules (image, audio, video, and archive).

Key functions:

File Type Detection:

- The get_file_type(file_path) function uses dedicated type-checking functions imported from the respective modules—is_supported_image, is_supported_audio, is_supported_video, is_supported_archive—to identify if the input file is a supported image, audio, video, or archive format.
- Returns a string indicating the file type or None if unsupported.

Message Embedding Dispatcher:

- The embed_message(input_path, output_path, message) function first determines the file type, then routes the operation to the corresponding backend function:
  - Images: embed_text_in_image
  - Audio: embed_text_in_audio
  - Video: embed_text_in_video
  - Archives: embed_text_in_archive
- If an unsupported file type is detected, it returns a failure message.
- This abstraction enables a single interface for embedding operations regardless of file type, improving code maintainability and user experience.

Message Extraction Dispatcher:

- The extract_message(input_path) function similarly detects file type and calls the correct extraction function:
  - Images: extract_text_from_image
  - Audio: extract_text_from_audio
  - Video: extract_text_from_video
  - Archives: extract_text_from_archive
- Handles unsupported file types gracefully.

Design and Usage Benefits:

- Simplifies the main application callback logic by isolating media-type-specific embedding and extraction processes.
- Centralizes error handling and reporting—if any file type is not supported, standardized error messages are returned.
- Enhances scalability; future steganography modules for new file types can be

integrated with minimal changes to this manager.

- Promotes code reuse and clarity, allowing both GUI and backend calls to use consistent functions.

Example Usage:

- When a user selects a file and chooses "Embed" or "Extract" in the GUI, the application relays the request to embed_message or extract_message in stego_manager.py, which takes care of all underlying file-type and module selection logic.
- This script exemplifies modular programming principles and strengthens the maintainability and readability of the overall codebase.

4.1.6 Modern Steganography App (GUI)

```
import customtkinter as ctk
from stego_manager import embed_message, extract_message
import os
import mimetypes
from PIL import Image, ExifTags, ImageFile
import cv2
import datetime
from tkinter import filedialog
import tkinter as tk
import math

Image.MAX_IMAGE_PIXELS = None
ImageFile.LOAD_TRUNCATED_IMAGES = True

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

class ModernSteganographyApp:
    def __init__(self):
        self.root = ctk.CTk()
        self.root.title("🔐 Steganography Analyzer - Modern
UI")
        self.root.geometry("1400x900")
        self.root.minsize(1000, 700)
        self.root.configure(fg_color="#0D1117")
        self.file_path = tk.StringVar()
        self.operation = tk.StringVar(value="embed")
        self.create_main_interface()
        self.animate_floating_elements()

    def create_main_interface(self):
        """Create the main responsive interface"""

        self.main_scrollable = ctk.CTkScrollableFrame(
            self.root,
            corner_radius=0,
```

```
            fg_color="#0D1117",
            scrollbar_button_color="#333333",
            scrollbar_button_hover_color="#555555"
        )
        self.main_scrollable.pack(fill="both", expand=True,
padx=10, pady=10)

        self.main_scrollable.grid_columnconfigure(0,
weight=2)  # Left panel gets more space
        self.main_scrollable.grid_columnconfigure(1,
weight=1)  # Right panel
        self.main_scrollable.grid_rowconfigure(1,
weight=1)     # Main content row

        # Title section (full width)
        self.create_title_section()

        # Left panel (Controls and Animation)
        self.create_left_panel()

        # Right panel (Metadata)
        self.create_right_panel()

    def create_title_section(self):
        """Create the responsive title section"""
        title_frame = ctk.CTkFrame(
            self.main_scrollable,
            corner_radius=20,
            height=100,
            fg_color="#1a3232",
            border_width=2,
            border_color="#00FFFF"
        )
        title_frame.grid(row=0, column=0, columnspan=2,
sticky="ew", padx=10, pady=(10, 20))
        title_frame.grid_propagate(False)

        # Configure internal grid
        title_frame.grid_columnconfigure(0, weight=1)
        title_frame.grid_rowconfigure(0, weight=1)
        title_frame.grid_rowconfigure(1, weight=1)

        # Main title
        self.title_label = ctk.CTkLabel(
            title_frame,
            text="🔒 StegLyser",
            font=ctk.CTkFont(family="Segoe UI", size=28,
weight="bold"),
            text_color="#00FFFF"
        )
```

```
        self.title_label.grid(row=0, column=0, sticky="ew",
pady=(10, 0))

        # Subtitle
        subtitle_label = ctk.CTkLabel(
            title_frame,
            text="Modern Responsive UI • Hide & Extract Messages
• Multi-Format Support",
            font=ctk.CTkFont(family="Segoe UI", size=14),
            text_color="#CCCCCC"
        )
        subtitle_label.grid(row=1, column=0, sticky="ew",
pady=(0, 10))

    def create_left_panel(self):
        """Create the responsive left panel"""
        self.left_panel = ctk.CTkFrame(
            self.main_scrollable,
            corner_radius=20,
            fg_color="#1a2332",
            border_width=2,
            border_color="#1E90FF"
        )
        self.left_panel.grid(row=1, column=0, sticky="nsew",
padx=(10, 5), pady=(0, 10))

        # Configure internal grid for responsiveness
        self.left_panel.grid_columnconfigure(0, weight=1)
        self.left_panel.grid_rowconfigure(0, weight=0)  #
Animation section
        self.left_panel.grid_rowconfigure(1, weight=0)  # File
section
        self.left_panel.grid_rowconfigure(2, weight=0)  #
Operation section
        self.left_panel.grid_rowconfigure(3, weight=1)  #
Message section (expandable)
        self.left_panel.grid_rowconfigure(4, weight=0)  # Button
section

        # Animation section
        self.create_animation_section()

        # File selection section
        self.create_file_section()

        # Operation selection section
        self.create_operation_section()

        # Message section
        self.create_message_section()
```

```
        # Action buttons section
        self.create_action_buttons()

    def create_animation_section(self):
        """Create responsive animation section"""
        animation_frame = ctk.CTkFrame(
            self.left_panel,
            corner_radius=15,
            height=200,
            fg_color="#0a0a0a"
        )
        animation_frame.grid(row=0, column=0, sticky="ew",
padx=15, pady=15)
        animation_frame.grid_propagate(False)
        animation_frame.grid_columnconfigure(0, weight=1)
        animation_frame.grid_rowconfigure(0, weight=1)

        # Canvas for animation
        self.canvas = tk.Canvas(
            animation_frame,
            bg='#0D1117',
            highlightthickness=0,
            height=180
        )
        self.canvas.grid(row=0, column=0, sticky="nsew",
padx=10, pady=10)

        # Animation setup
        self.setup_animation()

    def setup_animation(self):
        """Setup animated elements"""
        self.animation_items = []
        self.animation_angle = 0

        # Create animated circles
        colors = ['#00FFFF', '#1E90FF', '#8A2BE2', '#FF1493',
'#00FF7F']

        def update_canvas_size(event=None):
            canvas_width = self.canvas.winfo_width()
            canvas_height = self.canvas.winfo_height()
            center_x, center_y = canvas_width // 2,
canvas_height // 2

            # Clear existing items
            self.canvas.delete("all")
            self.animation_items.clear()
```

```
            # Create new circles based on canvas size
            if canvas_width > 50 and canvas_height > 50:
                for i, color in enumerate(colors):
                    radius = min(canvas_width, canvas_height) //
8 + i * 10
                    if radius < min(canvas_width, canvas_height)
// 2 - 20:
                        circle = self.canvas.create_oval(
                            center_x - radius, center_y -
radius,
                            center_x + radius, center_y +
radius,
                            outline=color, width=2, fill='',
tags="circle"
                        )
                        self.animation_items.append(circle)

                # Central glow
                self.center_circle = self.canvas.create_oval(
                    center_x - 15, center_y - 15,
                    center_x + 15, center_y + 15,
                    fill='#FFD700', outline='#FFA500', width=2
                )

        self.canvas.bind('<Configure>', update_canvas_size)

    def create_file_section(self):
        """Create responsive file selection section"""
        file_frame = ctk.CTkFrame(self.left_panel,
corner_radius=15, fg_color="#1a1a1a")
        file_frame.grid(row=1, column=0, sticky="ew", padx=15,
pady=(0, 15))
        file_frame.grid_columnconfigure(1, weight=1)  # Entry
field expands

        # Title
        ctk.CTkLabel(
            file_frame,
            text="📁 SELECT FILE",
            font=ctk.CTkFont(size=16, weight="bold")
        ).grid(row=0, column=0, columnspan=3, pady=(15, 10))

        # File path entry
        self.file_entry = ctk.CTkEntry(
            file_frame,
            textvariable=self.file_path,
            placeholder_text="Choose file to analyze...",
            height=40,
            corner_radius=20,
            font=ctk.CTkFont(size=12)
```

```
        )
        self.file_entry.grid(row=1, column=0, columnspan=2,
sticky="ew", padx=15, pady=(0, 15))

        # Browse button
        browse_btn = ctk.CTkButton(
            file_frame,
            text="Browse",
            width=100,
            height=40,
            corner_radius=20,
            fg_color="#1E90FF",
            hover_color="#0080FF",
            command=self.browse_file
        )
        browse_btn.grid(row=1, column=2, sticky="e", padx=(10,
15), pady=(0, 15))

    def create_operation_section(self):
        """Create responsive operation section"""
        operation_frame = ctk.CTkFrame(self.left_panel,
corner_radius=15, fg_color="#1a1a1a")
        operation_frame.grid(row=2, column=0, sticky="ew",
padx=15, pady=(0, 15))
        operation_frame.grid_columnconfigure(0, weight=1)
        operation_frame.grid_columnconfigure(1, weight=1)

        # Title
        ctk.CTkLabel(
            operation_frame,
            text="⚙️ OPERATION",
            font=ctk.CTkFont(size=16, weight="bold")
        ).grid(row=0, column=0, columnspan=2, pady=(15, 10))

        # Radio buttons in a responsive layout
        embed_radio = ctk.CTkRadioButton(
            operation_frame,
            text="🔒 Embed Message",
            variable=self.operation,
            value="embed",
            font=ctk.CTkFont(size=14),
            command=self.toggle_operation
        )
        embed_radio.grid(row=1, column=0, sticky="w", padx=15,
pady=5)

        extract_radio = ctk.CTkRadioButton(
            operation_frame,
            text="🔓 Extract Message",
            variable=self.operation,
```

```
            value="extract",
            font=ctk.CTkFont(size=14),
            command=self.toggle_operation
        )
        extract_radio.grid(row=1, column=1, sticky="w", padx=15,
pady=(5, 15))

    def create_message_section(self):
        """Create responsive message section"""
        message_frame = ctk.CTkFrame(self.left_panel,
corner_radius=15, fg_color="#1a1a1a")
        message_frame.grid(row=3, column=0, sticky="nsew",
padx=15, pady=(0, 15))
        message_frame.grid_columnconfigure(0, weight=1)
        message_frame.grid_rowconfigure(1, weight=1)  # Text
area expands

        # Title
        self.message_label = ctk.CTkLabel(
            message_frame,
            text="💬 SECRET MESSAGE",
            font=ctk.CTkFont(size=16, weight="bold")
        )
        self.message_label.grid(row=0, column=0, pady=(15, 10))

        # Message text area
        self.message_text = ctk.CTkTextbox(
            message_frame,
            corner_radius=15,
            font=ctk.CTkFont(family="Consolas", size=12),
            wrap="word"
        )
        self.message_text.grid(row=1, column=0, sticky="nsew",
padx=15, pady=(0, 15))

    def create_action_buttons(self):
        """Create responsive action buttons"""
        button_frame = ctk.CTkFrame(self.left_panel,
fg_color="transparent")
        button_frame.grid(row=4, column=0, sticky="ew", padx=15,
pady=(0, 15))
        button_frame.grid_columnconfigure(0, weight=3)  #
Execute button gets more space
        button_frame.grid_columnconfigure(1, weight=1)  # Clear
button

        # Execute button
        execute_btn = ctk.CTkButton(
            button_frame,
            text="🚀 EXECUTE",
```

```
            height=45,
            corner_radius=22,
            fg_color="#FF6B6B",
            hover_color="#FF5252",
            font=ctk.CTkFont(size=16, weight="bold"),
            command=self.execute_operation
        )
        execute_btn.grid(row=0, column=0, sticky="ew", padx=(0,
10))

        # Clear button
        clear_btn = ctk.CTkButton(
            button_frame,
            text="🧹 CLEAR",
            height=45,
            corner_radius=22,
            fg_color="#6C7B7F",
            hover_color="#5A6B7F",
            command=self.clear_fields
        )
        clear_btn.grid(row=0, column=1, sticky="ew")

    def create_right_panel(self):
        """Create responsive metadata panel"""
        self.right_panel = ctk.CTkFrame(
            self.main_scrollable,
            corner_radius=20,
            fg_color="#2a1a32",
            border_width=2,
            border_color="#8A2BE2"
        )
        self.right_panel.grid(row=1, column=1, sticky="nsew",
padx=(5, 10), pady=(0, 10))

        # Configure internal grid
        self.right_panel.grid_columnconfigure(0, weight=1)
        self.right_panel.grid_rowconfigure(1, weight=1)  #
Metadata content expands

        # Header
        header_frame = ctk.CTkFrame(
            self.right_panel,
            corner_radius=15,
            height=70,
            fg_color="#1a1a1a"
        )
        header_frame.grid(row=0, column=0, sticky="ew", padx=15,
pady=15)
        header_frame.grid_propagate(False)
        header_frame.grid_columnconfigure(0, weight=1)
```

```
        header_frame.grid_rowconfigure(0, weight=1)

        ctk.CTkLabel(
            header_frame,
            text="📊 FILE METADATA",
            font=ctk.CTkFont(size=18, weight="bold"),
            text_color="#BB86FC"
        ).grid(row=0, column=0)

        # Metadata content (scrollable)
        metadata_frame = ctk.CTkFrame(
            self.right_panel,
            corner_radius=15,
            fg_color="#0a0a0a"
        )
        metadata_frame.grid(row=1, column=0, sticky="nsew",
padx=15, pady=(0, 15))
        metadata_frame.grid_columnconfigure(0, weight=1)
        metadata_frame.grid_rowconfigure(0, weight=1)

        # Scrollable metadata text
        self.metadata_text = ctk.CTkTextbox(
            metadata_frame,
            corner_radius=10,
            font=ctk.CTkFont(family="Consolas", size=11),
            wrap="word"
        )
        self.metadata_text.grid(row=0, column=0, sticky="nsew",
padx=10, pady=10)

        # File statistics section
        self.create_stats_section()

    def create_stats_section(self):
        """Create file statistics section"""
        stats_frame = ctk.CTkFrame(
            self.right_panel,
            corner_radius=15,
            height=120,
            fg_color="#1a1a1a"
        )
        stats_frame.grid(row=2, column=0, sticky="ew", padx=15,
pady=(0, 15))
        stats_frame.grid_propagate(False)
        stats_frame.grid_columnconfigure((0, 1), weight=1)

        # Stats title
        ctk.CTkLabel(
            stats_frame,
            text="📈 QUICK STATS",
```

```
            font=ctk.CTkFont(size=14, weight="bold"),
            text_color="#00FF7F"
        ).grid(row=0, column=0, columnspan=2, pady=(10, 5))

        # Stats labels
        self.size_label = ctk.CTkLabel(
            stats_frame,
            text="Size: --",
            font=ctk.CTkFont(size=12),
            text_color="#CCCCCC"
        )
        self.size_label.grid(row=1, column=0, sticky="w",
padx=10)

        self.type_label = ctk.CTkLabel(
            stats_frame,
            text="Type: --",
            font=ctk.CTkFont(size=12),
            text_color="#CCCCCC"
        )
        self.type_label.grid(row=1, column=1, sticky="w",
padx=10)

        self.dimensions_label = ctk.CTkLabel(
            stats_frame,
            text="Dimensions: --",
            font=ctk.CTkFont(size=12),
            text_color="#CCCCCC"
        )
        self.dimensions_label.grid(row=2, column=0,
columnspan=2, sticky="w", padx=10, pady=(0, 10))

    def animate_floating_elements(self):
        """Animate the floating elements responsively"""
        try:
            if hasattr(self, 'canvas') and
self.canvas.winfo_exists():
                self.animation_angle += 2
                canvas_width = self.canvas.winfo_width()
                canvas_height = self.canvas.winfo_height()

                if canvas_width > 1 and canvas_height > 1:
                    center_x, center_y = canvas_width // 2,
canvas_height // 2

                    for i, item in
enumerate(self.animation_items):
                        if self.canvas.coords(item):
                            # Create floating effect
                            offset_angle = self.animation_angle
```

```
+ (i * 72)
                            offset_x =
math.sin(math.radians(offset_angle)) * 1.5
                            offset_y =
math.cos(math.radians(offset_angle)) * 1.5

                            # Small floating movement
                            self.canvas.move(item, offset_x *
0.1, offset_y * 0.1)

                    # Pulse the center circle
                    if hasattr(self, 'center_circle') and
self.canvas.coords(self.center_circle):
                        pulse = 2 +
math.sin(math.radians(self.animation_angle * 2)) * 0.5

        except Exception:
            pass  # Continue animation even if elements are
destroyed

        # Schedule next frame
        self.root.after(50, self.animate_floating_elements)

    def toggle_operation(self):
        """Toggle between embed and extract modes"""
        if self.operation.get() == "extract":
            self.message_label.configure(text="📩 EXTRACTED
MESSAGE")
            self.message_text.delete("1.0", "end")
            self.message_text.insert("1.0", "Extracted message
will appear here...")
        else:
            self.message_label.configure(text="💬 SECRET
MESSAGE")
            self.message_text.delete("1.0", "end")

    def browse_file(self):
        """Open file dialog with proper file types"""
        filetypes = [
            ("All supported files",
             "*.jpg *.jpeg *.png *.bmp *.gif *.tiff *.webp "
             "*.wav *.flac *.m4a *.ogg *.aiff *.au *.opus *.aac
*.mp3 "
             "*.mp4 *.mkv *.mov *.avi *.webm *.flv *.wmv "
             "*.zip *.rar *.7z"),

            ("Image files", "*.jpg *.jpeg *.png *.bmp *.gif
*.tiff *.webp"),
            ("Audio files", "*.wav *.flac *.m4a *.ogg *.aiff
*.au *.opus *.aac *.mp3"),
```

```
            ("Video files", "*.mp4 *.mkv *.mov *.avi *.webm
*.flv *.wmv"),
            ("Archive files", "*.zip *.7z *.rar"),
            ("All files", "*.*")
        ]

        filename = filedialog.askopenfilename(
            title="Select File - Steganography Analyzer",
            filetypes=filetypes
        )

        if filename:
            self.file_path.set(filename)
            self.extract_metadata(filename)

    def extract_metadata(self, file_path):
        """Extract and display file metadata with statistics"""
        self.metadata_text.delete("1.0", "end")

        if not os.path.exists(file_path):
            self.metadata_text.insert("end", "❌ File not
found.\n")
            self.update_stats("--", "--", "--")
            return

        try:
            # Basic file info
            file_size_bytes = os.path.getsize(file_path)
            file_size_kb = file_size_bytes / 1024
            file_size_mb = file_size_kb / 1024

            # Format file size
            if file_size_mb >= 1:
                size_str = f"{file_size_mb:.2f} MB"
            else:
                size_str = f"{file_size_kb:.2f} KB"

            mime_type, _ = mimetypes.guess_type(file_path)
            file_name = os.path.basename(file_path)

            metadata = f"""📁 FILE INFORMATION
{'='*40}
Name: {file_name}
Path: {file_path}
Size: {size_str} ({file_size_bytes:,} bytes)
Type: {mime_type or 'Unknown'}
Modified:
{datetime.datetime.fromtimestamp(os.path.getmtime(file_path)).st
rftime('%Y-%m-%d %H:%M:%S')}
```

```
"""

            dimensions_str = "--"

            # Image metadata
            if mime_type and mime_type.startswith("image"):
                try:
                    img = Image.open(file_path)
                    dimensions_str = f"{img.width} ×
{img.height}"

                    metadata += f"""🖼️ IMAGE DETAILS
{'='*40}
Format: {img.format}
Dimensions: {dimensions_str}
Mode: {img.mode}
Has Alpha: {'Yes' if 'A' in img.mode else 'No'}
Aspect Ratio: {img.width/img.height:.2f}:1

"""

                    # EXIF data
                    exif_data = img.getexif()
                    if exif_data:
                        metadata += "📷 EXIF DATA\n" + "="*40 +
"\n"
                        for tag, value in
list(exif_data.items())[:10]:  # Limit to first 10
                            tag_name = ExifTags.TAGS.get(tag,
f"Tag_{tag}")
                            metadata += f"{tag_name}: {value}\n"
                        if len(exif_data) > 10:
                            metadata += f"... and
{len(exif_data) - 10} more EXIF entries\n"
                    else:
                        metadata += "📷 No EXIF data found.\n"

                except Exception as e:
                    metadata += f"❌ Error reading image:
{e}\n"

            # Video metadata
            elif mime_type and mime_type.startswith("video"):
                try:
                    cap = cv2.VideoCapture(file_path)
                    if cap.isOpened():
                        fps = cap.get(cv2.CAP_PROP_FPS)
                        frame_count =
int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
                        width =
```

```
int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        height =
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        duration = frame_count / fps if fps > 0
else 0
                        dimensions_str = f"{width} × {height}"

                        metadata += f"""🎥 VIDEO DETAILS
{'='*40}
Resolution: {dimensions_str}
Frame Rate: {fps:.2f} FPS
Total Frames: {frame_count:,}
Duration: {str(datetime.timedelta(seconds=int(duration)))}
Bitrate: {(file_size_bytes * 8) / (duration * 1000) if duration
> 0 else 0:.0f} kbps
"""
                    cap.release()
                except Exception as e:
                    metadata += f"❌ Error reading video:
{e}\n"

            # Update quick stats
            self.update_stats(size_str, mime_type or "Unknown",
dimensions_str)

            self.metadata_text.insert("1.0", metadata)

        except Exception as e:
            self.metadata_text.insert("1.0", f"❌ Error
extracting metadata: {e}")
            self.update_stats("Error", "Error", "Error")

    def update_stats(self, size, file_type, dimensions):
        """Update the quick stats section"""
        self.size_label.configure(text=f"Size: {size}")
        self.type_label.configure(text=f"Type: {file_type}")
        self.dimensions_label.configure(text=f"Dimensions:
{dimensions}")

    def clear_fields(self):
        """Clear all input fields"""
        self.file_path.set("")
        self.message_text.delete("1.0", "end")
        self.metadata_text.delete("1.0", "end")
        self.update_stats("--", "--", "--")

    def show_modern_popup(self, title, message, success=True):
        """Show modern responsive popup"""
        popup = ctk.CTkToplevel(self.root)
        popup.title(title)
```

```
        popup.geometry("450x250")
        popup.resizable(True, True)
        popup.minsize(350, 200)

        # Center the popup
        popup.transient(self.root)
        popup.grab_set()

        # Configure popup grid
        popup.grid_columnconfigure(0, weight=1)
        popup.grid_rowconfigure(0, weight=1)

        # Style the popup
        color = "#4CAF50" if success else "#FF5252"

        main_frame = ctk.CTkFrame(
            popup,
            corner_radius=20,
            fg_color="#1a1a1a",
            border_width=2,
            border_color=color
        )
        main_frame.grid(row=0, column=0, sticky="nsew", padx=20,
pady=20)
        main_frame.grid_columnconfigure(0, weight=1)
        main_frame.grid_rowconfigure((0, 1, 2), weight=1)

        # Icon and title
        icon = "✅" if success else "❌"
        title_label = ctk.CTkLabel(
            main_frame,
            text=f"{icon} {title}",
            font=ctk.CTkFont(size=20, weight="bold"),
            text_color=color
        )
        title_label.grid(row=0, column=0, pady=20)

        # Message
        msg_label = ctk.CTkLabel(
            main_frame,
            text=message,
            font=ctk.CTkFont(size=14),
            wraplength=400
        )
        msg_label.grid(row=1, column=0, padx=20)

        # OK button
        ok_btn = ctk.CTkButton(
            main_frame,
            text="OK",
```

```
            width=120,
            height=40,
            corner_radius=20,
            fg_color=color,
            command=popup.destroy
        )
        ok_btn.grid(row=2, column=0, pady=20)

    def execute_operation(self):
        """Execute the selected operation with proper error
handling"""
        file_path = self.file_path.get()

        if not file_path:
            self.show_modern_popup("Error", "Please select a
file first!", success=False)
            return

        try:
            if self.operation.get() == "embed":
                message = self.message_text.get("1.0",
"end").strip()
                if not message:
                    self.show_modern_popup("Error", "Please
enter a message to embed!", success=False)
                    return

                # Save dialog
                file_ext =
os.path.splitext(file_path)[1].lower()
                initial_filename =
os.path.splitext(os.path.basename(file_path))[0] + "_stego" +
file_ext

                if file_ext in [".png", ".jpg", ".jpeg",
".bmp"]:
                    filetypes = [("Image files", "*.png *.jpg
*.jpeg *.bmp"), ("All files", "*.*")]
                elif file_ext in [".mp4", ".mkv", ".mov",
".avi", ".webm", ".flv", ".wmv"]:
                    filetypes = [("Video files", "*.mp4 *.mkv
*.mov *.avi *.webm *.flv *.wmv"), ("All files", "*.*")]
                else:
                    filetypes = [("All files", "*.*")]

                output_path = filedialog.asksaveasfilename(
                    title="Save Stego File As",
                    defaultextension=file_ext,
                    initialfile=initial_filename,
                    filetypes=filetypes
```

```
                )

                if output_path:
                    success, result = embed_message(file_path,
output_path, message)
                    if success:
                        media_type = "video" if file_ext in
[".mp4", ".mkv", ".mov", ".avi", ".webm", ".flv", ".wmv"] else
"image"
                        self.show_modern_popup("Success!",
f"Message successfully hidden in the {media_type}!\n\nSaved to:
{os.path.basename(output_path)}")
                    else:
                        self.show_modern_popup("Failed",
f"Operation failed: {result}", success=False)

            elif self.operation.get() == "extract":
                success, message = extract_message(file_path)
                if success:
                    self.message_text.delete("1.0", "end")
                    self.message_text.insert("1.0", message)

                    file_ext =
os.path.splitext(file_path)[1].lower()
                    media_type = "video" if file_ext in [".mp4",
".mkv", ".mov", ".avi", ".webm", ".flv", ".wmv"] else "image"
                    char_count = len(message)
                    self.show_modern_popup("Success!", f"Message
successfully extracted from the {media_type}!\n\nExtracted
{char_count} characters.")
                else:
                    self.show_modern_popup("Failed",
f"Extraction failed: {message}", success=False)

        except Exception as e:
            self.show_modern_popup("Error", f"An unexpected
error occurred:\n{str(e)}", success=False)

    def run(self):
        """Start the application"""
        # Bind window resize event for responsiveness
        self.root.bind('<Configure>', self.on_window_resize)
        self.root.mainloop()

    def on_window_resize(self, event=None):
        """Handle window resize events"""
        if event and event.widget == self.root:
            # Update canvas animation if needed
            if hasattr(self, 'canvas'):
                self.root.after_idle(self.setup_animation)
```

```
if __name__ == "__main__":
    app = ModernSteganographyApp()
    app.run()
```

Key Features:

Supported Formats:

- Images → PNG, BMP, JPG, JPEG (JPG/JPEG auto-converted to PNG).
- Audio → WAV (lossless format).
- Video → MP4, AVI (processed using FFmpeg).
- Archive → ZIP, RAR (used to hide secret files).

File Type Check:

- Each module (stego_image.py, stego_audio.py, stego_video.py, stego_archive.py) validates input format.
- Prevents unsupported files from crashing the program.
- Example: Image function checks extensions, Audio checks WAV format, etc.

Embedding Secret Message:

- Image → Uses LSB (Least Significant Bit) to hide text inside pixel values.
- Audio → Hides data in LSB of audio samples without affecting quality.
- Video → Extracts video frames, embeds message inside frames, reassembles using FFmpeg.
- Archive → Stores secret file inside a compressed archive for hiding.
- Always saves as a new output file to protect original cover file.

Extracting Secret Message:

- Image → Reveals hidden text using lsb.reveal().
- Audio → Reads samples, reconstructs hidden bits, and converts back to text.
- Video → Extracts frames, reads embedded data, and rebuilds hidden text.
- Archive → Extracts hidden files back from ZIP/RAR archive.

Robust Error Handling:

- Uses try-except in all modules.
- Common handled cases:
    - Wrong/unsupported file type.
    - Corrupted or truncated files.
    - Empty/No hidden message.
- Returns clear success/error messages to user.

GUI (Tkinter + sv_ttk Dark Mode):

- Built with Tkinter for user interface.
- Features:

    - Dark Mode via sv_ttk.
    - Progress Bar for long operations (video/audio embedding).
    - Animated Canvas for modern look.
    - Metadata Viewer (shows file details below Execute/Clear buttons).
    - Success Popup after embedding/extraction.
- GUI directly calls respective module functions (stego_image, stego_audio, etc.).

Advantages:

- Multi-format support (Images, Audio, Video, Archives).
- GUI is beginner-friendly with dark mode.
- Error handling makes it robust and user-safe.
- Clean separation: each module handles one type of media**.**

Limitations:

- Image stego limited to text only (not full files).
- Works best with lossless formats (PNG, WAV).
- Large video/audio embedding is slower.
- Provides basic hiding, not resistant to deep forensic analysis.

Usage in Project:

- Central component of Steganography Analyzer Application.
- Each module is integrated into GUI:
    - Image Tab → Embed/Extract text in images.
    - Audio Tab → Embed/Extract text in WAV audio.
    - Video Tab → Embed/Extract text in video frames.
    - Archive Tab → Hide/Extract files inside archives.
- Designed for academic learning and research in steganography.

4.2. Screenshots

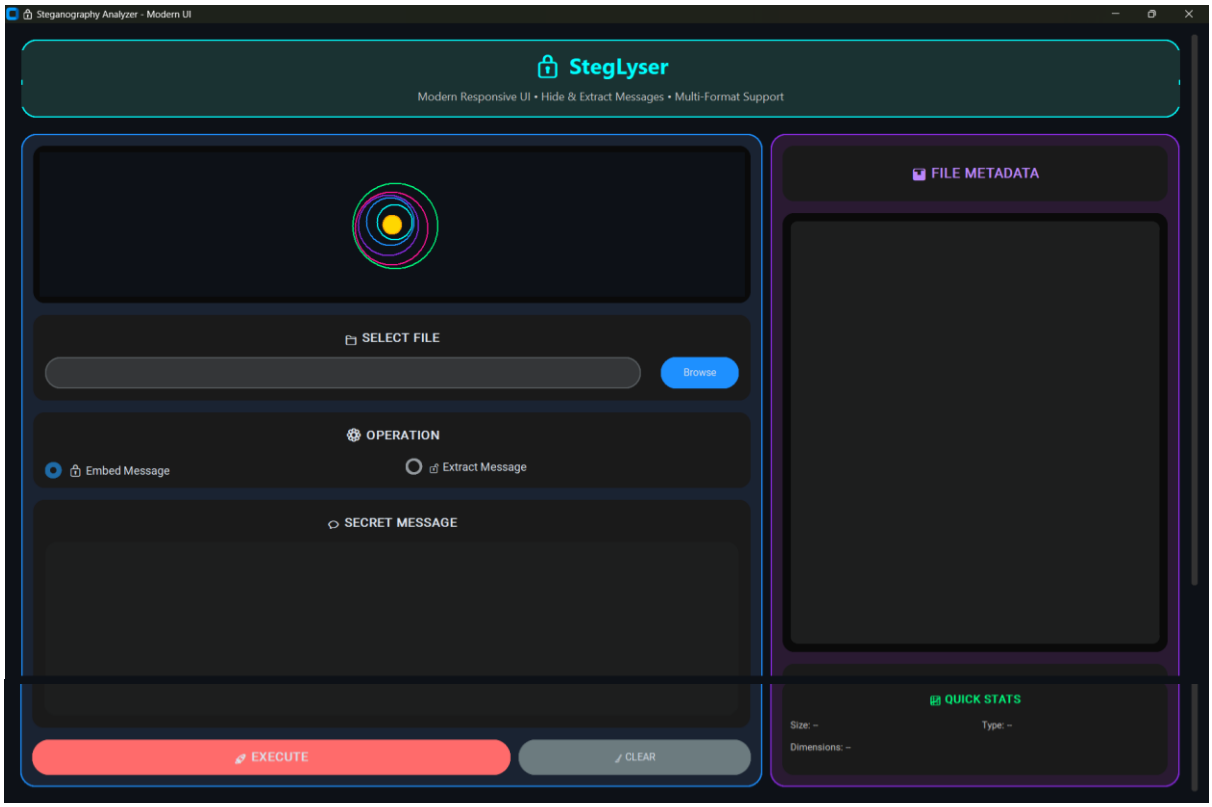

Figure 4.1: UI Design

Image Embedding: .png and .jpg

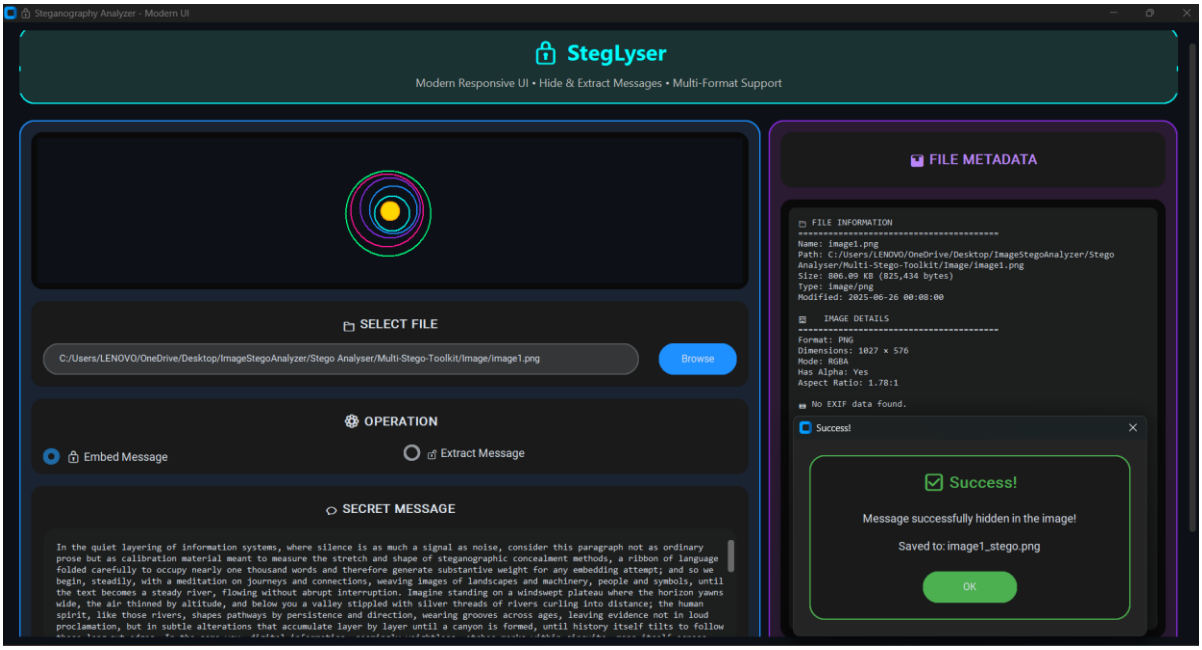

Figure 4.2: Image Embedding .png

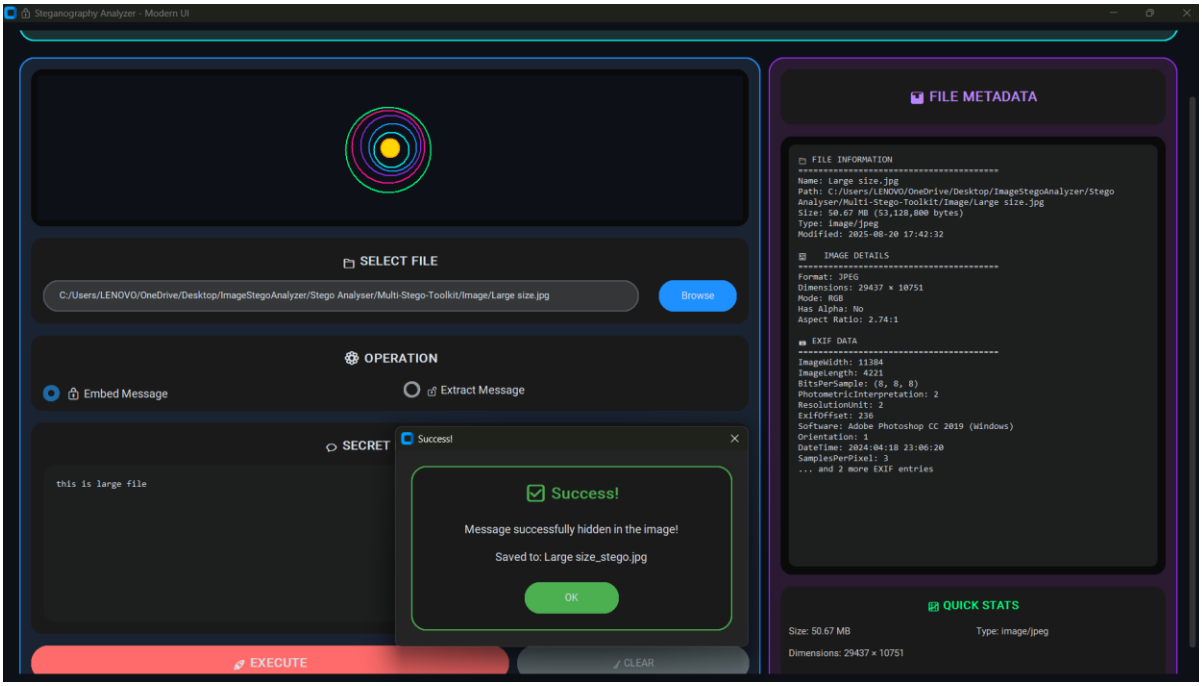

Figure 4.3: Image Embedding .jpg

Image Extraction: .png and .jpg

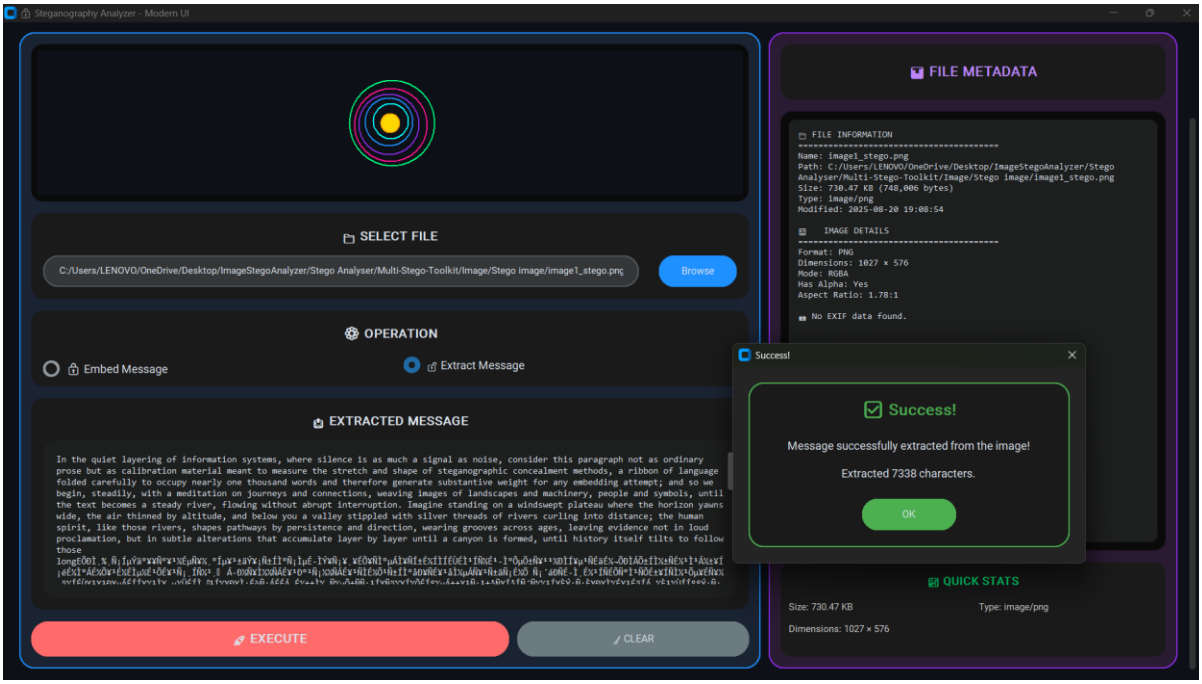

Figure 4.4: Image Extraction: .png

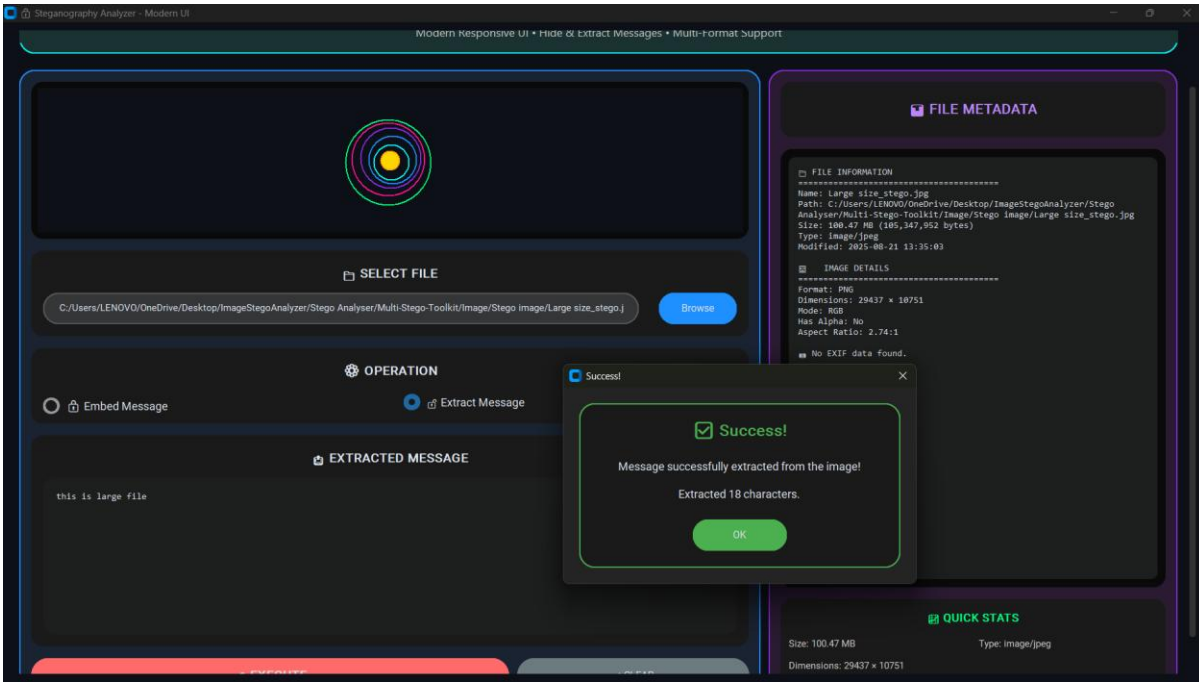

Figure 2.5: Image Extraction: .jpg

Audio Embedding: .wav and .aiff

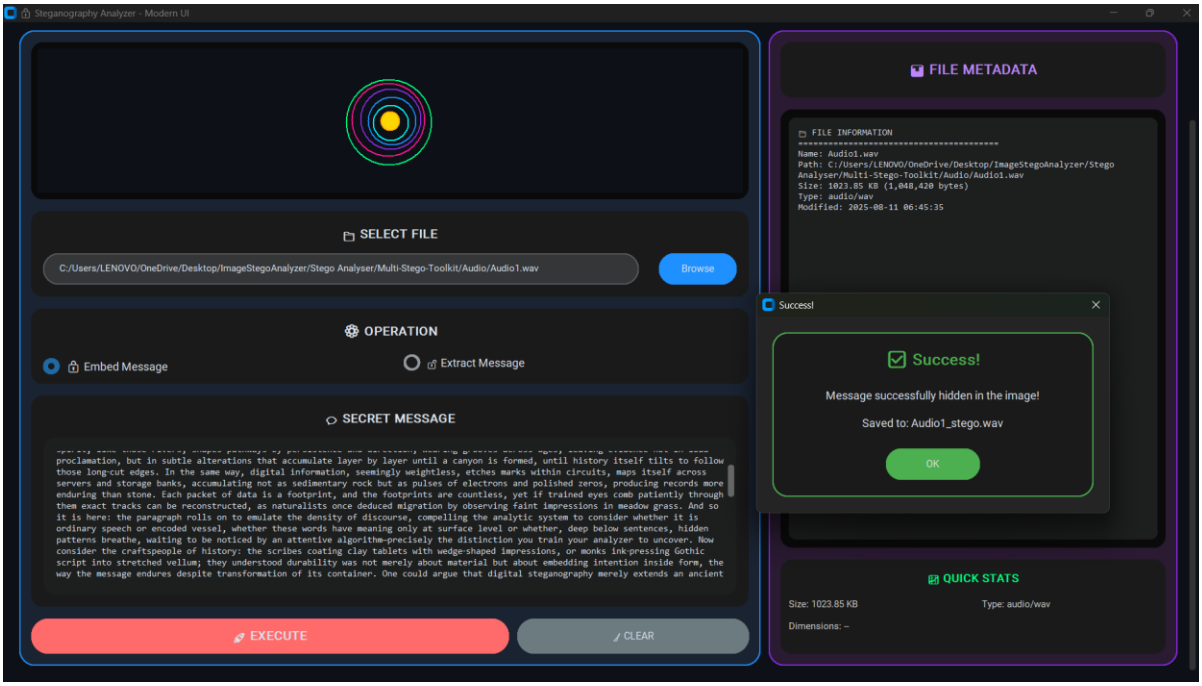

Figure 4.6: Audio Embedding: .wav

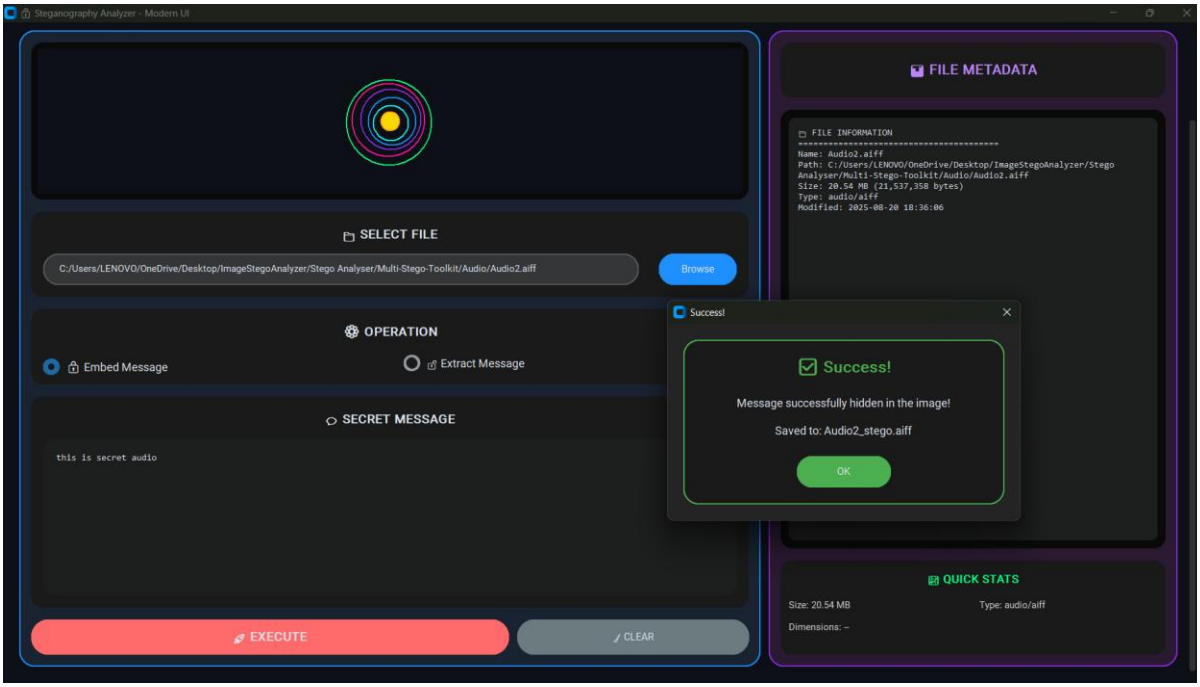

Figure 4.7: Audio Embedding: .aiff

Audio Extraction: .wav and .aiff

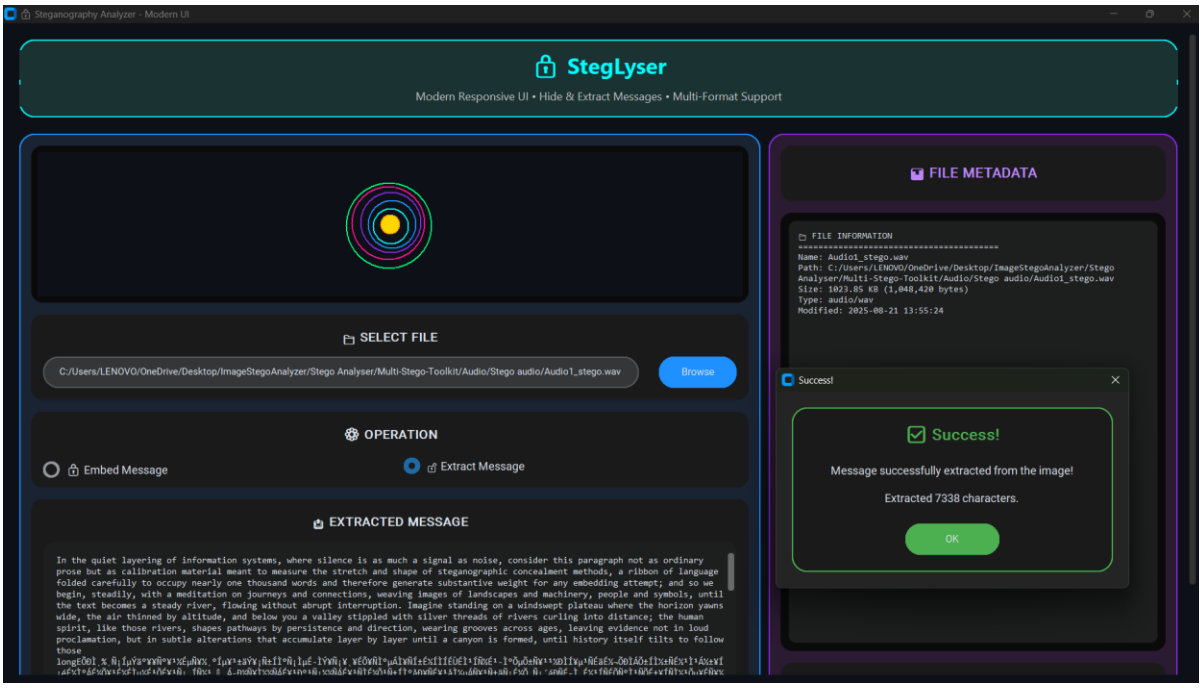

Figure 4.8: Audio Extraction .wav

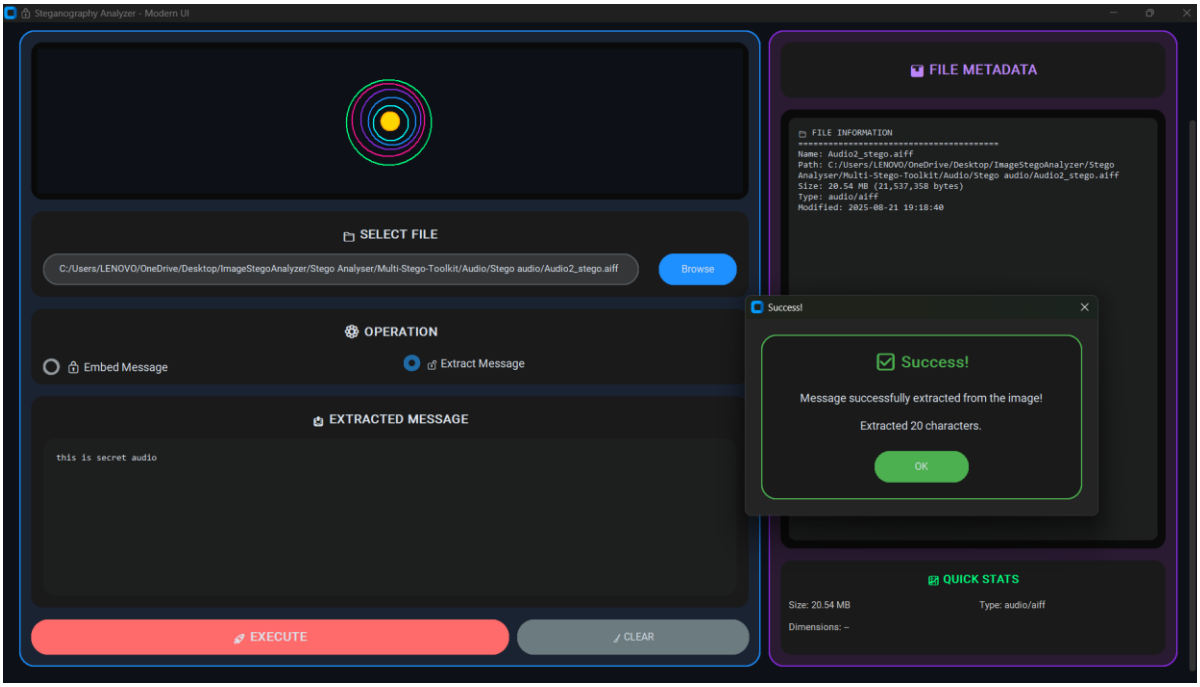

Figure 4.9: Audio Extraction: .aiff

Video Embedding: .mp4 and .mov

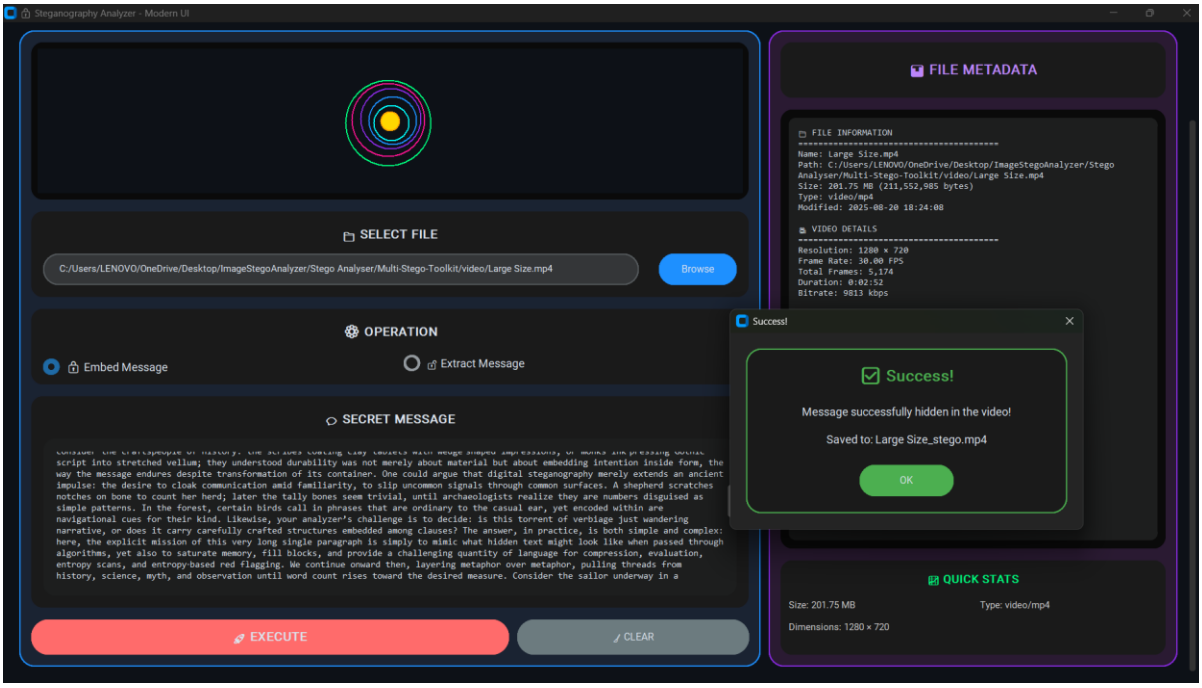

Figure 4.10: Video Embedding: .mp4

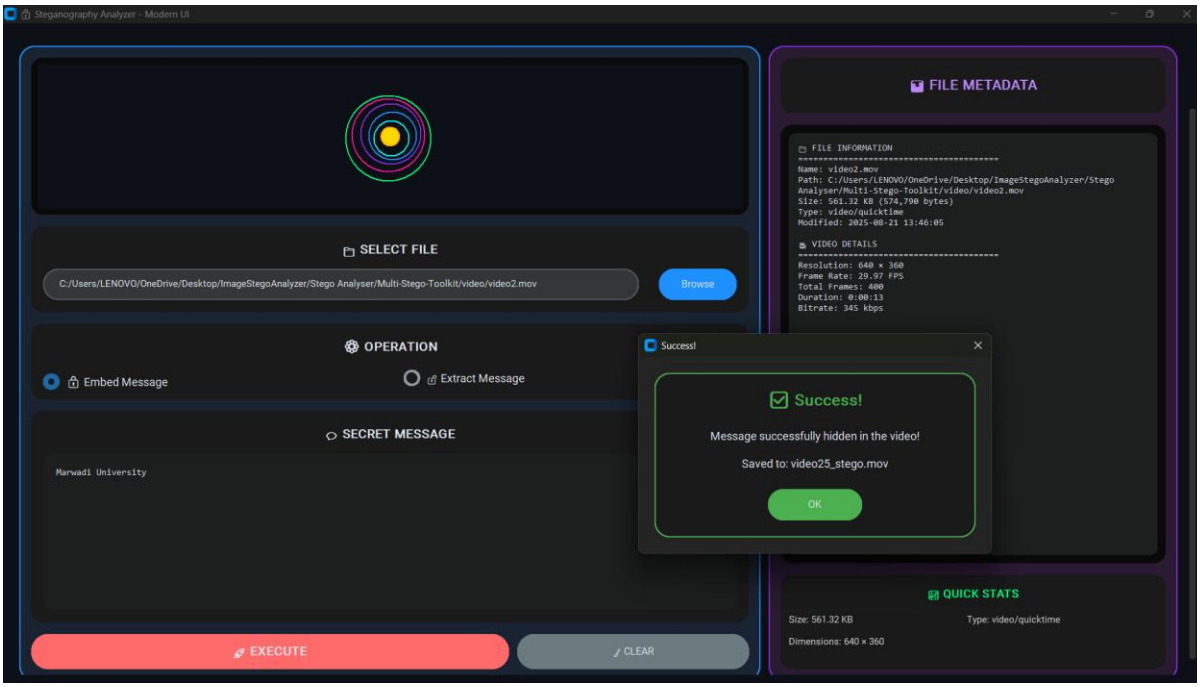

Figure 4.11: Video Embedding: .mov

Video Extraction: .mp4 and .mov

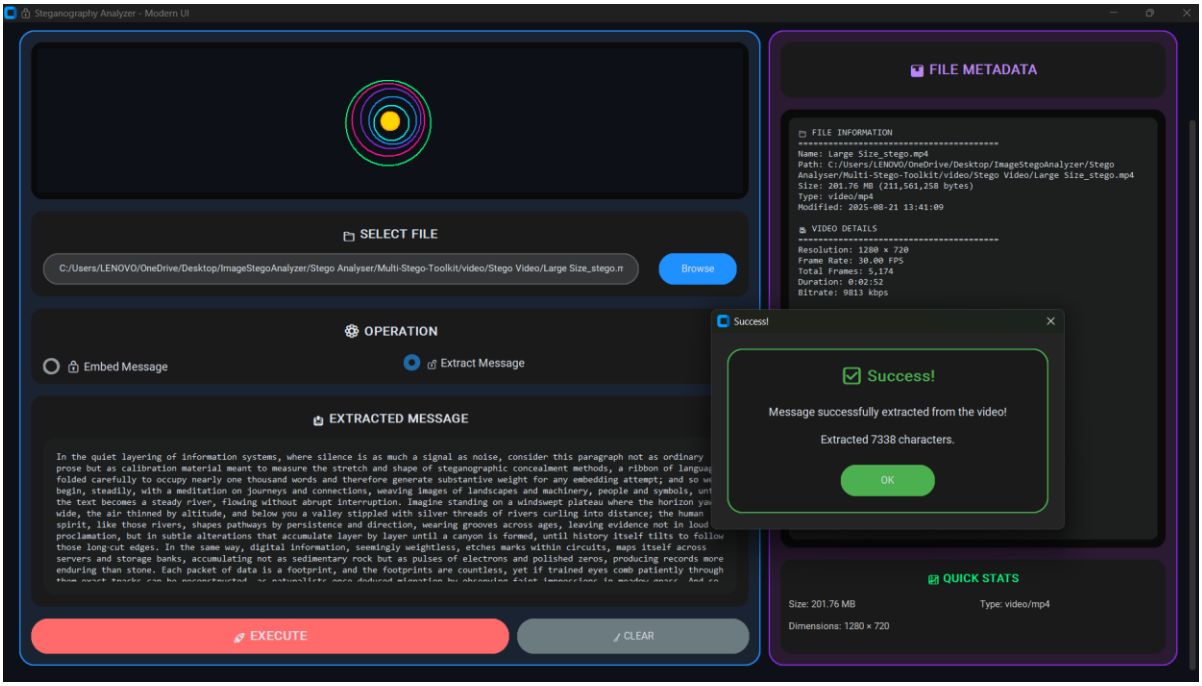

Figure 4.12: Video Extraction: .mp4

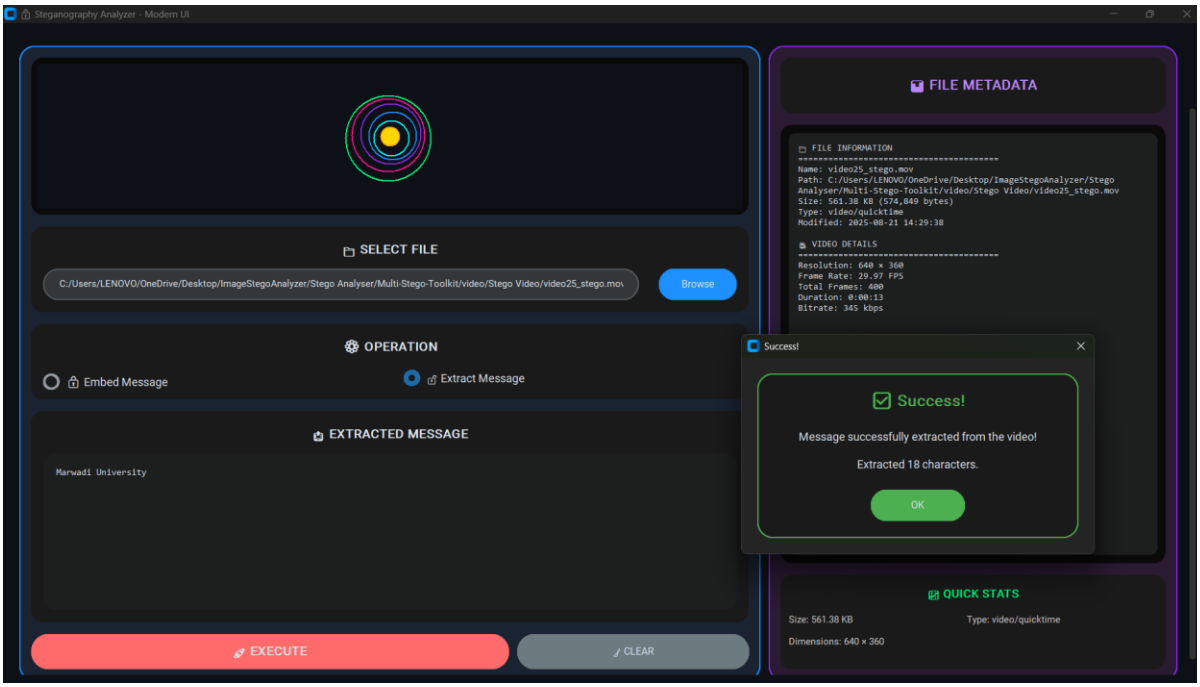

Figure 4.13: Video Extraction: .mov

Archive Embedding: .zip and .7z

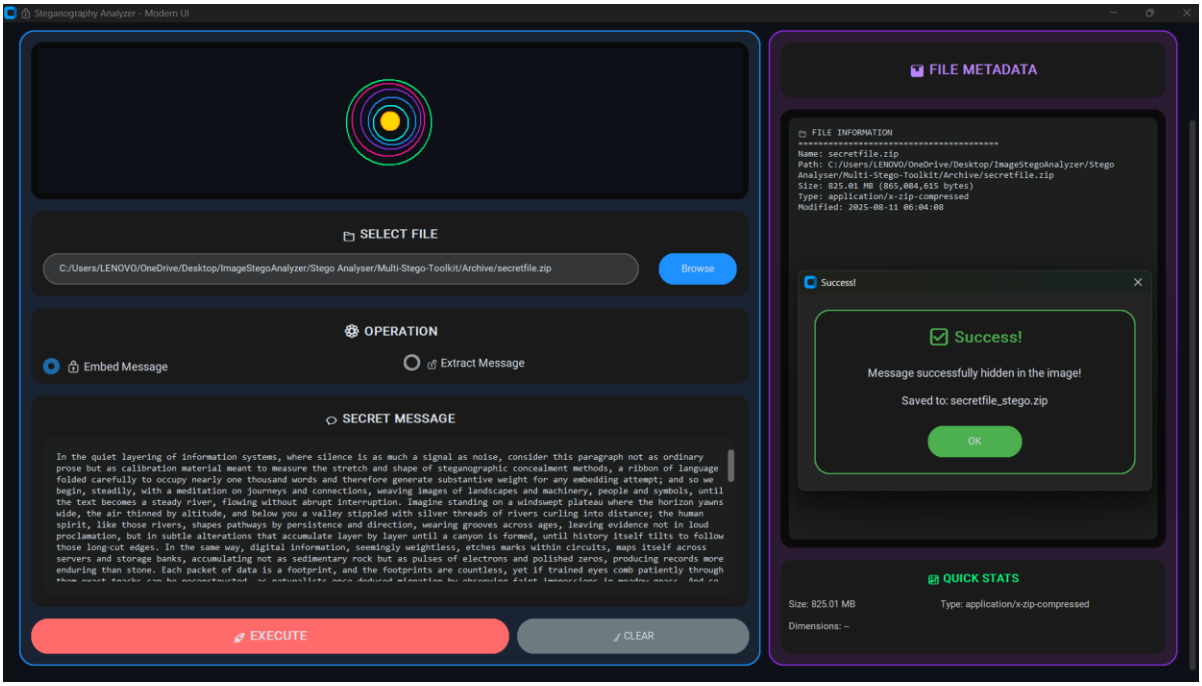

Figure 4.14: Archive Embedding: .zip

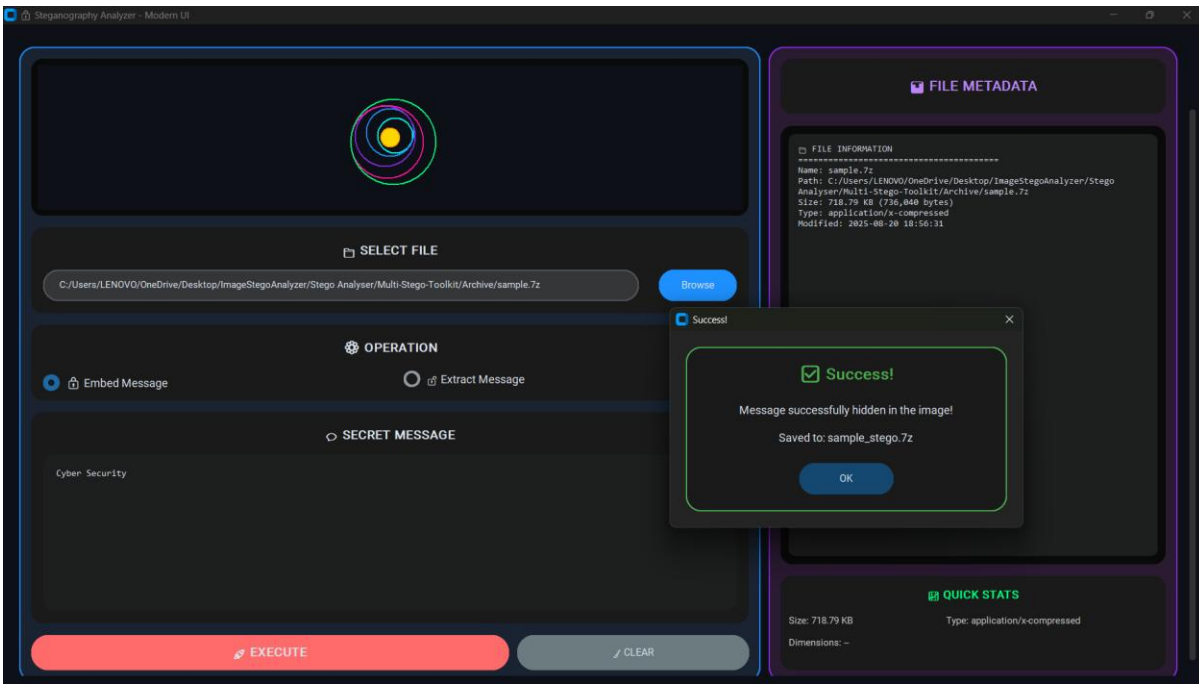

Figure 4.15: Archive Embedding: .7z

Archive Extraction: .zip and .7z

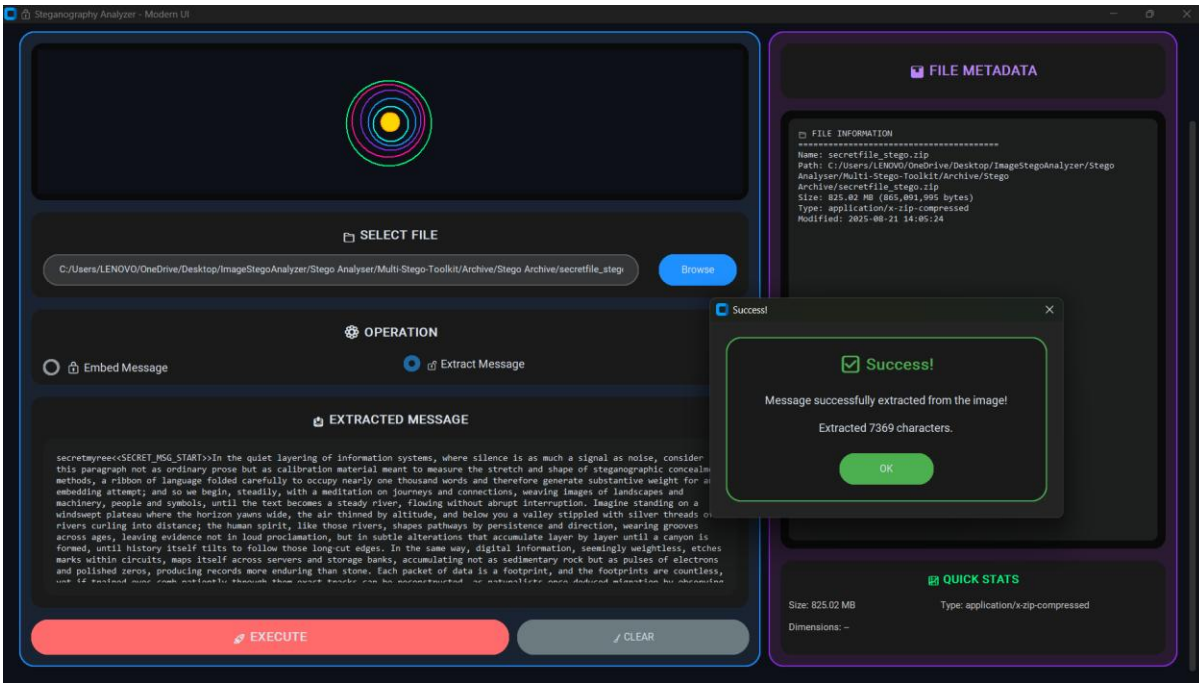

Figure 4.16: Archive Extraction: .zip

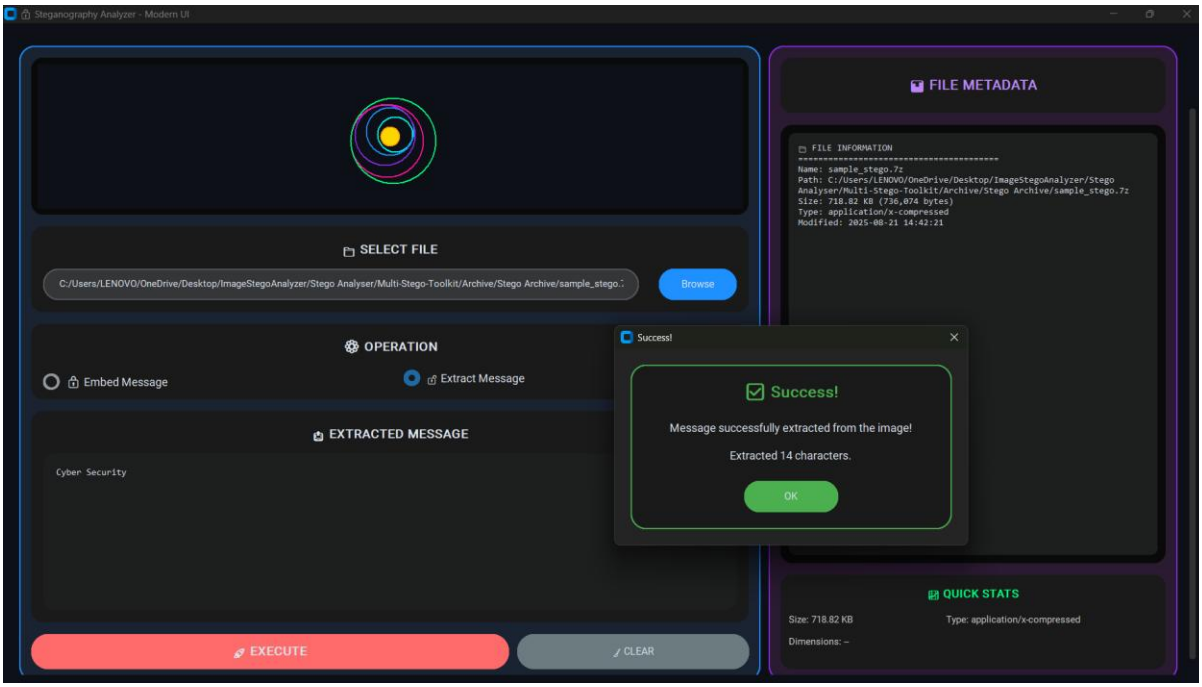

Figure 4.17: Archive Extraction: .7z

Unsupported Format and Error Handling: .mp3 audio file

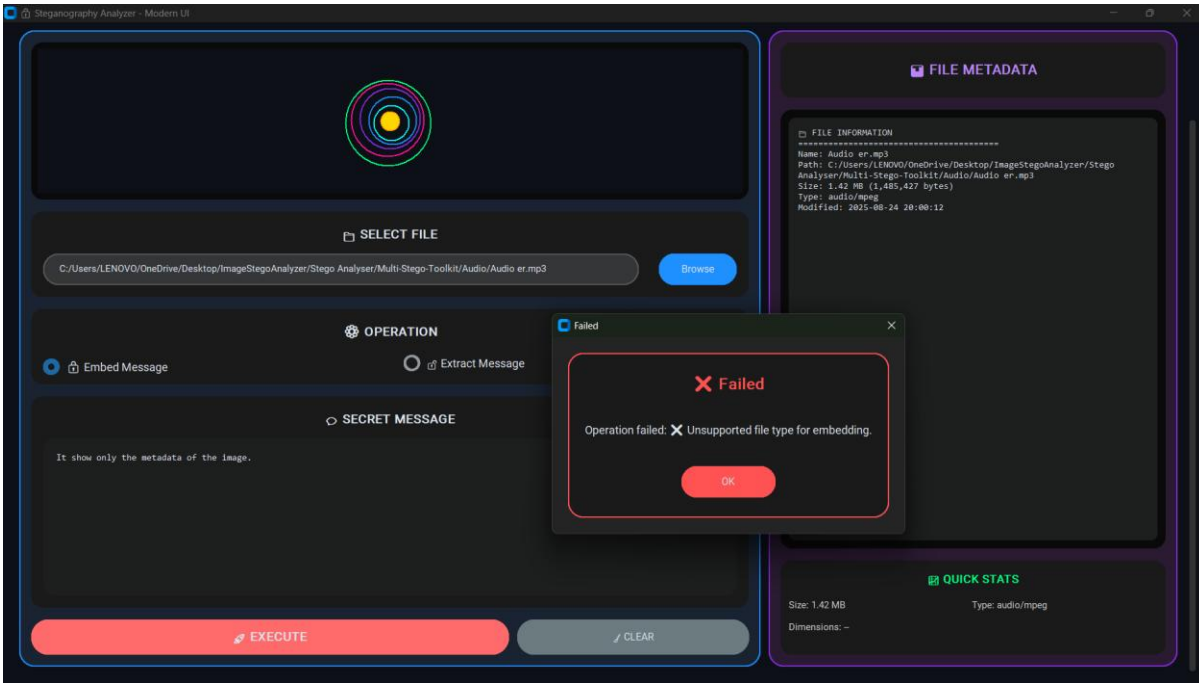

Figure 4.18: Unsupported Format and Error Handling: .mp3 audio file

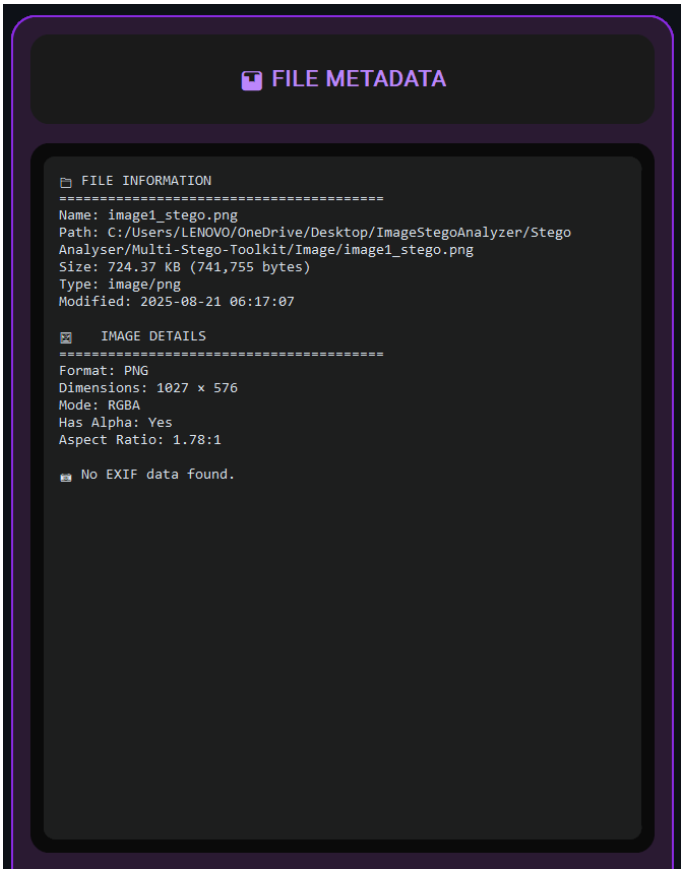

Figure 4.19: Metadata Viewer

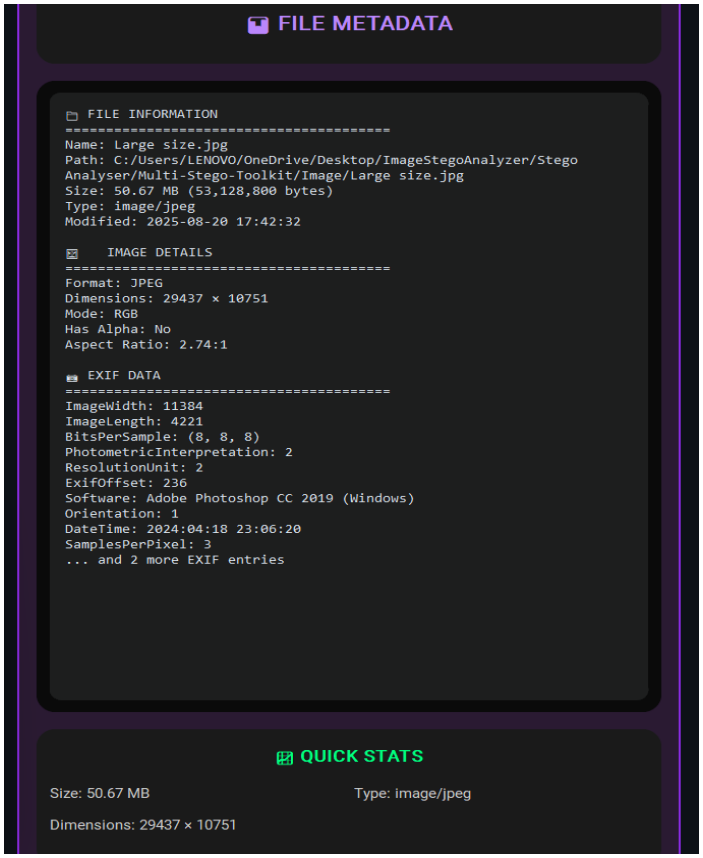

Figure 4.20: Exif Data

4.3. Test reports

Test Environment:

- Platform: Windows 10 (64-bit)
- Language: Python 3.11
- Libraries Used: Tkinter, PIL, tkinterdnd2, sv_ttk, FFmpeg, lsb-steganography
- Test Files: PNG, JPG, AIFF, WAV, MP4, MOV, ZIP, 7Z

Test Cases and Results:

Test Case	Input	Expected Result	Actual Result	Status
Image Embedding	PNG image + Secret text	New stego image created, secret hidden	Successfully embedded	✅ Pass
Image Extraction	Stego PNG image	Hidden message extracted correctly	Message extracted correctly	✅ Pass
Audio Embedding	WAV file + Secret text	New WAV file created with hidden data	Successfully embedded	✅ Pass
Audio Extraction	Stego WAV file	Hidden message retrieved	Message retrieved correctly	✅ Pass
Video Embedding	MP4 file + Secret text	New MP4 with hidden message in frames	Successfully embedded, slower for large files	✅ Pass
Video Extraction	Stego MP4 file	Secret message retrieved	Message retrieved correctly	✅ Pass
Archive Embedding	ZIP + secret file	File hidden inside archive	Hidden file embedded	✅ Pass
Archive Extraction	Stego ZIP	File extracted from archive	File extracted correctly	✅ Pass
Unsupported Format	GIF, MP3, DOCX	Program should show error	Error message shown, no crash	✅ Pass
Metadata Viewer	PNG/JPG/MP4 input	Show file details (size, type)	Metadata displayed correctly	✅ Pass
Error Handling	Corrupt PNG/MP4	Show error popup	Error handled, no crash	✅ Pass
Exif Data	JPG image	Camera, date, GPS, settings	Exif details displayed correctly	✅ Pass

Table 4.3: Test Results

Summary of Results

- All major functionalities (Image, Audio, Video, Archive steganography) tested successfully.
- GUI features (Metadata Viewer, Dark Mode, Success Popups) performed as expected.
- Tool is robust and user-friendly, with clear error handling.
- Performance: Embedding/extracting small text is instant, video embedding takes longer depending on file size.
- It will support embedding large secret messages successfully.

Chapter 5
Proposed Enhancements

5.1 Comparison with Existing Steganography Tools

1. Existing Tools

- StegHide
  - Open-source command-line tool.
  - Supports embedding messages in image (JPEG, BMP) and audio files (WAV, AU).
  - Provides password-based encryption (AES).
  - Limitation: No GUI, limited file type support.
- OpenStego
  - Java-based tool with GUI.
  - Supports image steganography (BMP, PNG).
  - Provides strong encryption and watermarking features.
  - Limitation: Only works with images, lacks audio/video/archive support.
- SilentEye
  - Cross-platform GUI tool.
  - Supports image (BMP, JPG) and audio (WAV) files.
  - Simple drag-and-drop interface.
  - Limitation: Very limited formats, outdated interface, no report generation.
- SSuite Picsel Security
  - Windows-based GUI tool.
  - Provides text embedding in images.
  - Password protection available.
  - Limitation: Image-only support, lacks advanced error handling.

2. Features of Our Steganography Analyzer

- Supported File Types: Images, audio, video, and archive files (broader coverage than most existing tools).
- Embedding & Extraction: Supports LSB-based embedding/extraction with success/error feedback.
- GUI Features: Modern GUI (Tkinter + dark mode) with file browsing, progress bars, and animations.
- Metadata Analysis: Displays file properties and metadata for forensic use.
- Error Handling: Robust error messages and handling for unsupported files.
- Cross-Format Focus: Not restricted to one type (images); works across multiple media formats.
- Report Generation: Option to export results as Word/PDF (in development).
- User Friendly: Unlike StegHide or command-line tools, our analyzer is beginner-friendly.

3. Improvements Needed (Proposed Enhancements)

- Extend audio support beyond WAV (add AIFF, MP3, AAC).
- Add advanced steganographic techniques (DCT, DWT, spread spectrum).

- Implement encryption (AES/RSA) for embedded messages.
- Support batch processing of multiple files.
- Add cloud integration for uploading/downloading stego files.
- Improve cross-platform compatibility (Linux, MacOS, Android, iOS).
- Add error correction (ECC) to recover hidden data from partially corrupted files.
- Automatic forensic mode for detecting hidden data in suspicious files.
- Add more user controls, customizable themes, and better visualization of embedding progress.

Chapter 6
Conclusion

The development of the *Steganography Analyzer* project has successfully demonstrated a multi-format steganography tool that integrates support for images, audio, video, and archive files. Unlike many existing tools that focus only on images or a single media type, this analyzer provides a unified platform for both embedding and extracting hidden data across diverse file formats. The inclusion of a modern GUI built with Tkinter, dark mode support, metadata viewing, and user-friendly file selection makes the tool accessible even to non-technical users. Furthermore, the project emphasizes robust error handling and modular code design, which ensures stability and easier maintainability.

While the tool already addresses the fundamental needs of steganography analysis, it also highlights areas for further enhancement. Features such as broader audio format support, advanced embedding algorithms, forensic detection capabilities, and cloud integration will significantly improve its applicability in real-world cybersecurity and digital forensics scenarios. In conclusion, this project not only meets its objective of creating a functional and versatile steganography analyzer but also lays the foundation for future research and professional applications in the fields of information security, data hiding, and forensic analysis.

Chapter 7
Bibliography

Offline:

[1] N. F. Johnson, Z. Duric, and S. Jajodia, *Information Hiding: Steganography and Watermarking – Attacks and Countermeasures*. Springer, 2001.
[2] S. Katzenbeisser and F. A. P. Petitcolas, *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 2000.
[3] N. Provos and P. Honeyman, "Hide and Seek: An Introduction to Steganography," *IEEE Security & Privacy*, vol. 1, no. 3, pp. 32–44, 2003.
[4] J. Fridrich, *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press, 2009.
[5] G. C. Kessler, "An Overview of Steganography for the Computer Forensics Examiner," *Forensic Science Communications*, vol. 6, no. 3, 2015.
[6] I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*. Morgan Kaufmann, 2008.
[7] P. Wayner, *Disappearing Cryptography: Information Hiding – Steganography & Watermarking*. Morgan Kaufmann, 2002.
[8] G. C. Langelaar, I. Setyawan, and R. L. Lagendijk, "Watermarking Digital Image and Video Data: A State-of-the-Art Overview," *IEEE Signal Processing Magazine*, vol. 17, no. 5, pp. 20–46, 2000.

Online:

[9] OpenAI, "ChatGPT Language Model – Technical Assistance for Code and Report Preparation," 2025.
[10] Python Software Foundation, "Python Documentation." [Online]. Available: https://docs.python.org/.
[11] TkDocs, "Tkinter GUI Programming by Example." [Online]. Available: https://tkdocs.com/.
[12] FFmpeg Project, "FFmpeg Documentation." [Online]. Available: https://ffmpeg.org/.
[13] Pillow Library, "Python Imaging Library (PIL) Handbook." [Online]. Available: https://pillow.readthedocs.io/.
[14] CustomTkinter, "Modern Tkinter for Python GUIs." [Online]. Available: https://github.com/TomSchimansky/CustomTkinter.
[15] OpenCV, "OpenCV-Python Tutorials." [Online]. Available: https://docs.opencv.org/.
[16] Mutagen Library, "Python Audio Metadata Handling." [Online]. Available: https://mutagen.readthedocs.io/.
[17] Pydub Documentation, "Audio Processing in Python." [Online]. Available: https://github.com/jiaaro/pydub.

[18] TkinterDnD2, “Drag and Drop Extension for Tkinter.” [Online]. Available: https://github.com/pmgagne/tkinterdnd2.
[19] sv_ttk, “A Modern Theme for Tkinter.” [Online]. Available: https://github.com/rdbende/Sun-Valley-ttk-theme.
[20] OpenStego, “Free Steganography Tool.” [Online]. Available: https://www.openstego.com/.
[21] SilentEye, “Steganography Software.” [Online]. Available: https://sourceforge.net/projects/silenteye/.
[22] StegHide, “Steganography Program for Hiding Data in Images/Audio.” [Online]. Available: http://steghide.sourceforge.net/.