iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85383

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

KavachAI: A Privacy-Preserving Desktop System for Multimodal Sensitive Data Detection and Redaction

Dr. Sreenivasa Murthy V.[1], Charan S. N.[2], Madan Gowda R.[3], Venu B. S.[4], Vinay V.Y.[5]
[1]*Professor and Head, Department of Information Science and Engineering, Rajarajeswari College of Engineering, Bengaluru, Karnataka, India*
[2,3,4,5]*Department of Information Science and Engineering, Rajarajeswari College of Engineering, Bengaluru, Karnataka, India.*

Abstract: ***Sensitive information can appear in text documents, images, and audio recordings, making it difficult for users to protect personal data before sharing digital content. This paper presents KavachAI, a Windows desktop application designed to integrate multimodal sensitive-data detection, redaction, and protected local storage within a unified workflow. The system combines regular-expression matching and named-entity recognition (NER) for text analysis, optical character recognition (OCR) for extracting text from images, and locally executed speech-to-text transcription for processing audio recordings. Extracted text from images and audio is passed through a shared detection pipeline to identify configured sensitive-data patterns and supported named entities. The application provides text masking, visual image redaction, and protection of audio-derived transcripts. Selected retained outputs are encrypted using AES-256 in Galois/Counter Mode (AES-256-GCM), while the Windows Data Protection API (DPAPI) protects the associated encryption key. SQLite supports local application data and scan-history management. The system is implemented using Electron, React, and a locally hosted FastAPI backend with Python-based processing components. This paper describes the system architecture, implementation, privacy boundaries, and functional verification approach. Quantitative evaluation of detection accuracy, processing latency, and redaction completeness requires a separately documented and controlled assessment.***

Keywords: ***Sensitive-data protection, multimodal redaction, named-entity recognition, optical character recognition, speech-to-text, local data encryption.***

I. INTRODUCTION

The increasing use of digital communication, electronic documents, photographs, and audio recordings has increased the amount of personal and sensitive information handled by individuals and organizations. Such information may include names, telephone numbers, email addresses, residential details, government identification numbers, financial identifiers, and other details that could expose an individual's identity. Before documents or media are shared for academic, administrative, professional, or personal purposes, sensitive information may need to be identified and removed. Manual inspection can be time-consuming, particularly when information is distributed across different file formats or embedded in images and recorded speech.

Automated sensitive-data detection can assist this process. Regular expressions can identify structured patterns, such as email addresses and telephone numbers, while named-entity recognition (NER) can identify contextual entities, including person names, organizations, locations, and dates. However, neither technique alone covers every type of sensitive information. Pattern-based methods depend on defined rules, whereas NER performance depends on language, context, and the selected model. A combined approach provides complementary detection mechanisms, although the resulting output still requires validation.

The challenge becomes more complex when sensitive information appears in different modalities. Text documents can be analysed directly, but images generally require optical character recognition (OCR) before their text can be inspected. Audio recordings require speech-to-text transcription before sensitive words or identifiers can be detected. These extraction stages introduce additional challenges: OCR may misrecognize characters or positions, and speech recognition may omit or incorrectly transcribe words. A detection system must therefore consider the limitations of both extraction and identification. Another consideration is where processing takes place and how its outputs are retained. Cloud-based services can be convenient, but sending unprocessed documents, images, or recordings to remote services may be unsuitable for users who prefer local handling of sensitive material. Local processing can reduce dependence on external processing services; however, it does not automatically guarantee privacy. Temporary files, application logs, exported results, authentication mechanisms, and stored encryption keys must also be considered.

©IJRASET: All Rights are Reserved

This paper presents KavachAI, a Windows desktop application for multimodal sensitive-data detection and redaction. The application integrates an Electron-based desktop interface, React and Vite for the user interface, and a locally hosted FastAPI backend. Its text-processing pipeline combines regular-expression matching with NER. For images, OpenCV and Tesseract OCR extract text and associated spatial information. For audio, locally executed Whisper transcription converts speech into text that can be passed through the shared detection pipeline.

The principal contribution is architectural and implementation-oriented. KavachAI integrates modality-specific extraction processes with a common sensitive-data detection and protection workflow in a desktop application. The design reuses the text-detection stage after extracting textual content from each supported modality. It also supports corresponding output-protection workflows, including text masking, visual image redaction, and protection of audio-derived transcripts. Selected retained outputs are protected using AES-256-GCM, with Windows DPAPI used to protect the encryption key. SQLite supports local application data and scan-history management.

This work does not propose a new NER model, OCR algorithm, speech-recognition model, or cryptographic primitive. It documents how these components are integrated and identifies the boundaries of the supported workflows. Quantitative claims about detection accuracy, processing speed, redaction effectiveness, or resistance to security attacks require separate, controlled evaluation.

II. LITERATURE REVIEW

Sensitive-data protection involves identifying confidential information, determining which content should be protected, and applying an appropriate transformation before the content is retained or shared. Research and software development in this area draw on several complementary technologies, including pattern matching, named-entity recognition, optical character recognition, speech recognition, and cryptographic protection. KavachAI combines these techniques in a desktop workflow for processing text, images, and audio-derived transcripts.

A. Rule-Based Detection and Named-Entity Recognition

Rule-based detection is commonly used to identify structured information whose format can be described through predefined patterns. Regular expressions can be configured to recognize email addresses, telephone numbers, and other identifier formats. Their main advantages are that the rules are explicit and can be adjusted to match application requirements. However, rule-based detection depends on the completeness of the configured patterns and may produce false positives or miss values that do not match the expected format.

Named-entity recognition (NER) complements pattern matching by identifying entities from linguistic context. The spaCy natural-language-processing framework provides named-entity recognition capabilities that can be integrated into a Python application [5]. Depending on the selected model and its supported entity categories, NER can identify entities such as persons, organizations, locations, dates, and monetary expressions. However, model predictions can be affected by sentence structure, language, context, and input quality.

A combined approach uses regular expressions for configured structured patterns and NER for contextual entities. In KavachAI, these complementary methods form the shared text-detection stage. Where implemented, additional validation, such as Luhn validation for payment-card candidates or Verhoeff validation for eligible identifier candidates, can help reject certain invalid numeric strings. Such checks validate particular number structures; they do not establish that an identifier belongs to a real person or that it is necessarily sensitive.

B. Optical Character Recognition and Image Redaction

Images may contain sensitive information in identity documents, forms, receipts, screenshots, and other visual materials. Unlike ordinary text input, image content must first be converted into a textual representation before conventional text-detection methods can inspect it. Optical character recognition (OCR) performs this conversion by identifying characters and words in an image.

R. Smith describes the Tesseract OCR engine and its approach to recognizing text in images [4]. Tesseract can be used to extract text and, through its interfaces, provide information about the location of recognized text. OpenCV can support image preprocessing, while the pytesseract interface allows Python applications to invoke Tesseract OCR.

OCR output can be passed to a sensitive-data detector, but recognition errors may affect the result. Incorrectly recognized characters can prevent an identifier from matching a configured pattern. Furthermore, detecting a sensitive text span is not sufficient to redact an image: the application must correctly associate that span with the corresponding image region.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

KavachAI uses OCR-based extraction and spatial information to support visual redaction. The intended workflow identifies sensitive text, maps the detected content to its image location, and applies a visual transformation such as blurring, pixelation, or an opaque mask, according to the supported application options. Redaction quality depends on the correctness of OCR, coordinate mapping, and the exported image. The presence of a black box in a preview alone does not demonstrate that sensitive information has been completely removed from the saved output.

C. Speech Recognition and Audio-Derived Text Protection

Audio recordings may contain spoken names, contact details, identification numbers, and other sensitive information. To apply text-based detection methods to speech, the recording must first be transcribed. Automatic speech recognition converts speech signals into a sequence of words that can subsequently be analysed by text-processing techniques.

Radford et al. describe Whisper, a speech-recognition system trained using large-scale weak supervision [3]. A locally executed Whisper model can produce a transcript without requiring the recording to be sent to a remote transcription service, provided that the application is configured to run the model locally and its dependencies are available.

Transcription introduces a separate source of uncertainty. Speech recognition may omit words, misrecognize names or numbers, or produce text that differs from the recording. Consequently, sensitive information that is absent or incorrectly represented in the transcript may not be detected by a downstream text detector.

KavachAI processes audio through a transcription stage and passes the resulting text to its shared sensitive-data detection workflow. The protected result is an audio-derived transcript. This implementation does not directly redact or modify the original audio waveform. Therefore, transcript protection should not be interpreted as removing sensitive speech from the original recording.

D. Encryption and Local Key Protection

Detection and redaction address sensitive content, while encryption protects retained data against unauthorized access under the applicable key-management and access-control assumptions. The Advanced Encryption Standard (AES) is specified by the National Institute of Standards and Technology (NIST) [1]. Galois/Counter Mode (GCM) provides authenticated encryption and is specified in NIST Special Publication 800-38D [2]. KavachAI uses AES-256-GCM for selected retained outputs. Authentication provided by an authenticated-encryption mode can help detect unauthorized modification of ciphertext when verification is performed correctly. The security of the complete implementation also depends on factors such as secure key generation, nonce management, key protection, error handling, and access control. On Windows, the Data Protection API (DPAPI) provides operating-system facilities for protecting data within a user or machine security context, depending on the configuration [6], [7]. KavachAI uses DPAPI to protect the associated AES key. This approach separates the retained content's encryption from the operating-system mechanism used to protect the key. It does not, by itself, guarantee protection against every threat, including compromise of an already authenticated user session or exposure of plaintext while it is being processed.

E. Desktop Integration and Local Processing

A multimodal protection application must coordinate its user interface, processing services, and local data storage. Electron provides a framework for desktop applications, while FastAPI provides facilities for implementing Python-based API services [8], [9]. These technologies support an architecture in which the desktop interface communicates with a local processing backend.

KavachAI uses Electron with React and Vite for its desktop interface and a locally hosted FastAPI backend for Python-based processing. SQLite is used for local application data and scan-history management. The architecture connects modality-specific extraction stages to the shared detection workflow and then to the relevant protection operation.

Local execution reduces the need to send input files to an external processing service when the complete workflow operates as intended. However, the use of a local architecture is not sufficient evidence that the application makes no network connections. Dependencies, model downloads, updates, temporary files, logs, exported results, and other data-handling paths must be considered separately when assessing the application's privacy properties.

F. Research Gap and Motivation for KavachAI

The technologies discussed above address different parts of the sensitive-data protection process. Regular expressions and NER identify candidate sensitive content in text; OCR makes image text available for analysis; speech recognition produces transcripts for audio analysis; and cryptographic mechanisms protect selected retained outputs. These capabilities are complementary, but their combination creates integration challenges.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

First, a user may need separate tools or workflows to inspect typed text, images, and recorded speech. Second, independently implemented detectors can produce inconsistent results across modalities. Third, identifying sensitive content does not by itself ensure that the corresponding text, image region, or transcript is transformed and stored appropriately. Finally, local processing and encrypted retention must be considered alongside temporary data, key management, and the limits of each modality's extraction process.

KavachAI addresses these integration challenges through a desktop workflow that connects modality-specific extraction with a shared sensitive-data detection stage. Text is analysed directly, image text is extracted through OCR, and audio is converted into a transcript before detection. The resulting content can then be protected through text masking, visual image redaction, or transcript protection, as appropriate to the modality. Selected retained outputs use AES-256-GCM, with DPAPI protecting the associated AES key.

The contribution of this work is the design and implementation of an integrated multimodal desktop workflow rather than a new detection algorithm or cryptographic primitive. The system's practical effectiveness must be established through reproducible tests covering detection quality, image-coordinate mapping, transcription errors, redaction completeness, processing behaviour, and local data protection. These evaluation requirements distinguish the implemented architecture from claims that would require further experimental evidence.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

A. System Overview

KavachAI is a Windows desktop application designed to identify and protect sensitive information in text, images, and audio recordings through a unified processing workflow. The application integrates an Electron and React-based user interface with a locally hosted FastAPI backend and Python-based processing components. SQLite supports local application data and scan-history management, while cryptographic services protect selected retained outputs.

The system is organized into five principal stages: input acquisition, modality-specific preprocessing, sensitive-information detection, redaction and protection, and local output management. Text is processed directly by the text-detection module. Images are processed using optical character recognition (OCR) to extract text and its associated locations. Audio recordings are transcribed into text using a locally executed speech-recognition model. The extracted text from image and audio inputs is then passed through the shared text-detection pipeline.

This shared pipeline reduces the need to implement separate sensitive-information detection rules for each modality. However, detection performance depends on input quality, OCR or transcription quality, configured patterns, and the capabilities of the named-entity recognition model.

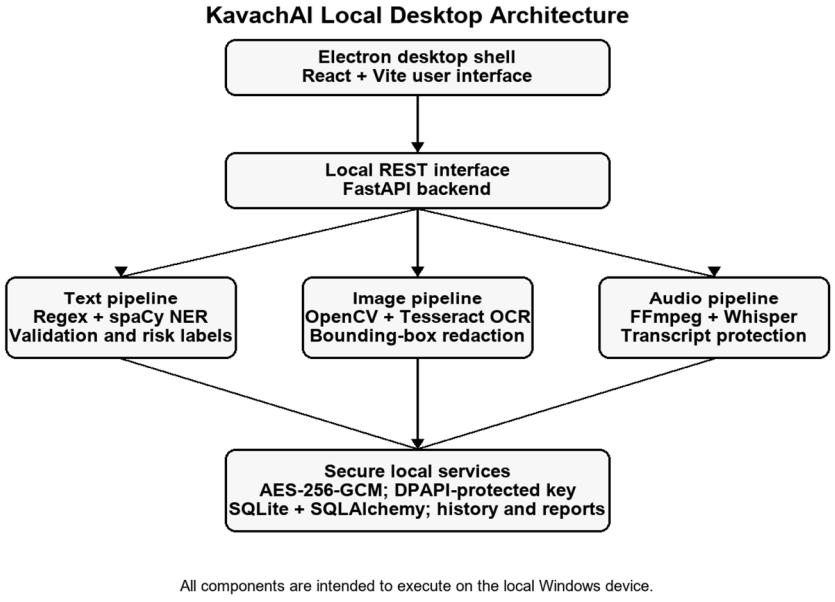


Fig. 1. Overall architecture of the KavachAI multimodal sensitive-data protection system.

B. System Architecture

The architecture consists of the following logical components.

1) User Interface Layer: The desktop interface is implemented using Electron and React. It provides access to the text scanner, image scanner, audio scanner, and relevant application functions. Users submit content for processing and review the detected information and generated protected output.

2) Application Communication Layer: Electron provides the desktop application environment, while the React interface communicates with the locally hosted backend through application API requests. The backend is implemented using FastAPI. This separation allows the user interface and Python-based detection modules to operate as distinct components.

3) Input and Preprocessing Layer: This layer accepts text, image files, and audio recordings. Each input follows a suitable preprocessing procedure before detection. Text is normalized for analysis, images undergo image-processing operations where required, and audio recordings are prepared for transcription.

4) Sensitive-Information Detection Layer: The detection engine combines regular-expression matching with named-entity recognition (NER). OCR-derived and transcription-derived text can use the same detection pipeline as directly entered text. This approach supports a common detection policy across the three input modalities.

5) Redaction and Protection Layer: Based on detected information, the application generates masked text, visually redacted images, or protected transcripts derived from audio. Selected retained outputs can be encrypted using AES-256-GCM, with Windows Data Protection API (DPAPI) protection for the associated encryption key.

6) Local Data Management Layer: SQLite supports local application records and scan-history management. The encryption and storage workflow is intended to reduce unnecessary exposure of sensitive retained outputs. The exact protection of each stored field depends on the implemented storage path and must be described according to the actual application code.

The architecture is designed around local processing. Nevertheless, a claim that the application operates completely offline should be supported by verification that model loading, dependencies, and all processing functions work without external network access.

C. Text-Based Sensitive-Information Detection

The text-processing module identifies configured sensitive patterns and supported named entities. It combines regular expressions with spaCy-based named-entity recognition to handle different types of information.

1) Regular-Expression Matching: Regular expressions identify structured patterns such as email addresses, phone numbers, Aadhaar-like identifiers, PAN-like identifiers, payment-card numbers, and other configured formats. These rules are useful when sensitive information follows a recognizable character pattern.

2) Named-Entity Recognition: The spaCy NER model identifies contextual entities such as person names, organizations, locations, dates, and monetary expressions, depending on the model's supported entity categories. NER complements pattern matching because many sensitive entities do not follow a single fixed format.

3) Candidate Validation: Where implemented, additional validation checks can be applied to candidate identifiers. The Luhn algorithm can validate supported payment-card number candidates, while the Verhoeff algorithm can be used for compatible identifier formats. These checks help reject some invalid candidates but do not establish that an identifier is genuine, belongs to a particular person, or is legally valid.

4) Detection Consolidation: Candidates from different detectors are collected for subsequent redaction. If candidates overlap, the application needs a defined conflict-resolution policy so that overlapping matches do not produce inconsistent or malformed output.

5) Text Masking: After detection, sensitive spans are replaced with configured masking labels or characters. The protected text is then presented for user review or retained through the relevant application workflow. Masking is intended to preserve the readability of non-sensitive text while reducing exposure of detected sensitive values.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

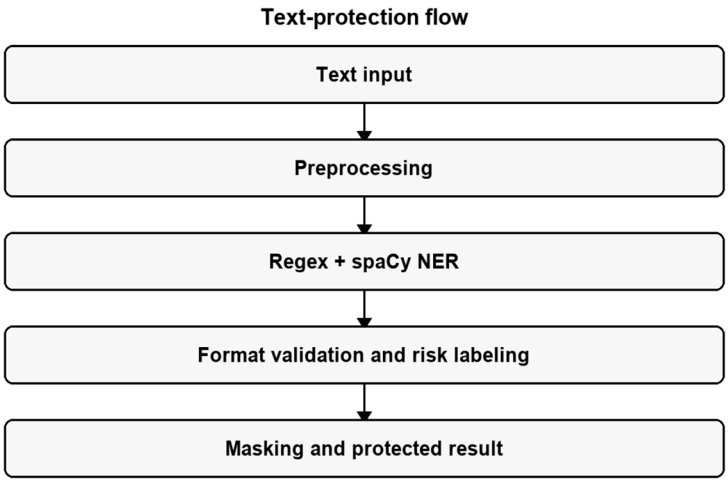


Fig. 2. Text-processing workflow using regular-expression matching, named-entity recognition, candidate consolidation, and masking.

D. Image-Based Detection and Visual Redaction

The image-processing module detects sensitive information embedded in images by combining OCR with the shared text-detection pipeline. The application uses OpenCV for image-processing operations and Tesseract OCR to extract textual content and corresponding location information.

First, the image is loaded and prepared for OCR. Depending on the input and implementation, preprocessing may be applied to improve text extraction. Tesseract then produces recognized text and associated bounding-box coordinates. The extracted text is passed to the shared detection pipeline to identify configured sensitive patterns and supported named entities.

The detected text spans must then be associated with the relevant OCR regions. Because a detected text span may cover only part of an OCR line or may span multiple OCR tokens, the mapping process must account for the relationship between detected text and OCR bounding boxes. If image preprocessing changes the image dimensions or coordinate system, the resulting coordinates must be mapped back to the original image before redaction.

The application generates a visually protected image by applying the configured redaction operation to the relevant regions. Depending on the implementation, this may involve blurring, pixelation, or an opaque mask. An opaque mask can provide a clearer visual barrier than blur or pixelation, but the exported image must still be inspected to confirm that sensitive text is not readable elsewhere in the output.

OCR-based protection has limitations. Incorrect recognition, missed text, low image resolution, unusual fonts, rotated text, and complex layouts can affect detection and localization. Consequently, a successful redaction operation does not automatically prove that every sensitive value in the image has been removed.

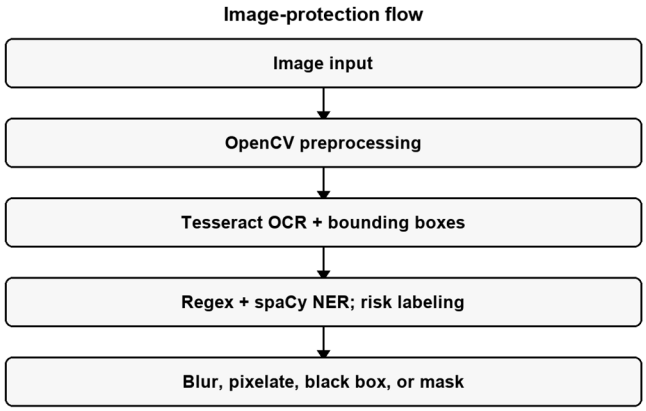


Fig. 3. Image-processing workflow for extracting, identifying, localizing, and visually redacting sensitive information.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

E. Audio-Based Detection and Transcript Protection

The audio module processes recordings using a locally executed Whisper speech-recognition model. Where required, FFmpeg supports audio preparation or conversion. The speech-recognition component generates a text transcript, which is subsequently processed by the shared sensitive-information detection pipeline.

The transcript can contain structured identifiers and contextual entities that are detectable through regular expressions and NER. Identified sensitive spans are then masked to produce a protected transcript. The resulting text can be presented to the user or handled by the application's retained-output workflow.

An important distinction is that the current audio workflow protects the transcript derived from the recording, not necessarily the original audio waveform. Unless an additional audio-redaction mechanism has been implemented and verified, the original recording may still contain the spoken sensitive information. The paper therefore does not claim that the system removes or masks sensitive speech directly from the original audio.

Transcription quality also affects the protection workflow. If speech recognition omits, substitutes, or incorrectly transcribes a sensitive value, the downstream detector may fail to identify it. Audio quality, background noise, accents, pronunciation, and language support can influence the generated transcript. The protected transcript should therefore be reviewed where complete removal of sensitive information is important.

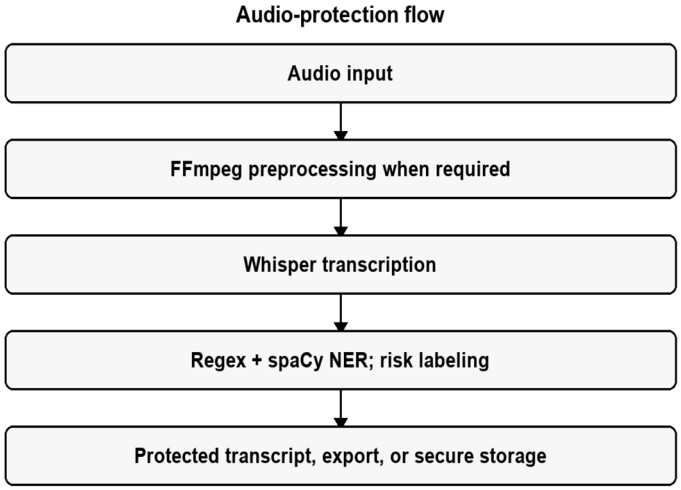


Fig. 4. Audio processing through local speech recognition, transcript analysis, and text-level masking.

F. Encryption and Local Storage

KavachAI includes a local data-management workflow for application records, scan history, and selected protected outputs. SQLite is used for local application data, while AES-256-GCM provides authenticated encryption for outputs processed through the encryption workflow.

AES-GCM combines confidentiality with integrity protection, allowing the application to detect certain forms of unauthorized modification to encrypted data when authentication is correctly verified [1], [2]. The application must manage encryption keys and authentication failures carefully. Sensitive plaintext should not be unintentionally retained in logs, temporary files, error messages, or unprotected database fields.

The Windows Data Protection API (DPAPI) is used to protect the associated AES key. DPAPI provides Windows-based data-protection functionality, with protection behavior depending on the selected configuration and Windows security context [6], [7]. The key-protection mechanism should not be described as eliminating all risks: access by an appropriately privileged or compromised user context may still affect the security of locally available data.

SQLite is used for application data and scan-history management. Its use does not, by itself, encrypt every stored field. In KavachAI, AES-256-GCM is used for retained outputs and Windows DPAPI is used to protect the encryption key. Database metadata and other application records should not be described as encrypted unless encryption is applied to those specific fields. The handling and retention of original inputs, intermediate files, transcripts, and detection records should be documented according to the actual data path.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

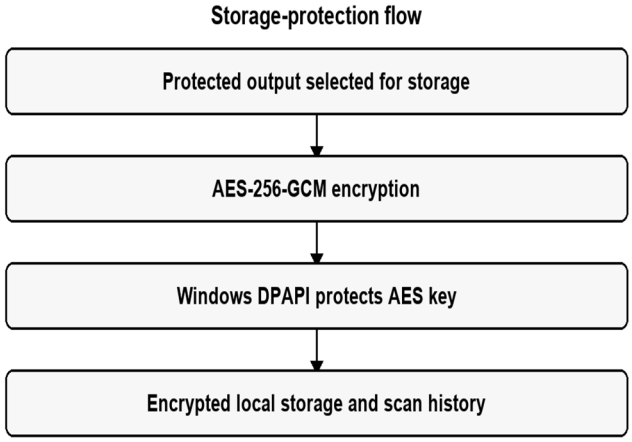


Fig. 5. Local output-protection workflow using AES-256-GCM, Windows DPAPI key protection, and SQLite-based application data management.

G. End-to-End Processing Workflow

The system follows a common high-level workflow across its supported input modalities:

1) The user selects a text, image, or audio input through the desktop interface.
2) The application forwards the input to the relevant backend processing module.
3) The selected module prepares the input for analysis.
4) For text inputs, the text is analyzed directly. For images, OCR extracts text and associated locations. For audio, speech recognition produces a transcript.
5) The resulting text is analyzed using configured regular expressions and named-entity recognition.
6) The application consolidates detected candidates and applies the corresponding masking or visual-redaction operation.
7) The protected result is returned to the interface for review.
8) Where the user or application workflow retains an output, the applicable encryption and local-storage mechanisms are used.
9) Application records and scan history are managed locally according to the implemented retention behavior.

This workflow unifies sensitive-information detection across different input types while retaining modality-specific preprocessing and redaction. Its effectiveness depends on the quality of extraction, the detection rules and model, the mapping from detected content to its source, and the correctness of the protection operation.

H. Scope of the Methodology

The methodology focuses on the integration of text detection, OCR-based image processing, speech-to-text transcription, redaction, and local data protection in one desktop application. It does not, by itself, establish a measured detection accuracy, a guaranteed absence of network traffic, or complete removal of every sensitive value from every supported file.

These properties require separate verification. In particular, reproducible evaluation should document the test inputs, expected sensitive spans, observed detections, redaction results, storage behavior, and offline-operation checks. The following implementation and evaluation sections should report only results supported by actual test records.

IV. IMPLEMENTATION AND RESULTS

A. Desktop Interface and Backend Communication

KavachAI is implemented as a Windows-oriented desktop application using Electron, React, and Vite for the user interface and FastAPI for backend services. Python-based modules perform sensitive-information detection, image processing, and audio transcription. SQLite, accessed through SQLAlchemy, supports local application data and scan-history management.

The desktop interface provides access to the text, image, and audio processing workflows. When a user submits an input, the frontend sends the relevant request to the local backend. The backend invokes the corresponding processing module and returns the resulting detections or protected output for presentation in the interface.

The frontend and backend are separated so that the desktop interface remains distinct from the processing logic. The application was tested using the configured local frontend-to-backend workflow. For reproducibility, the exact dependency and model versions, startup commands, API endpoint configuration, and error-handling behavior should be recorded in the project's release documentation; these details are not fully enumerated in this paper.

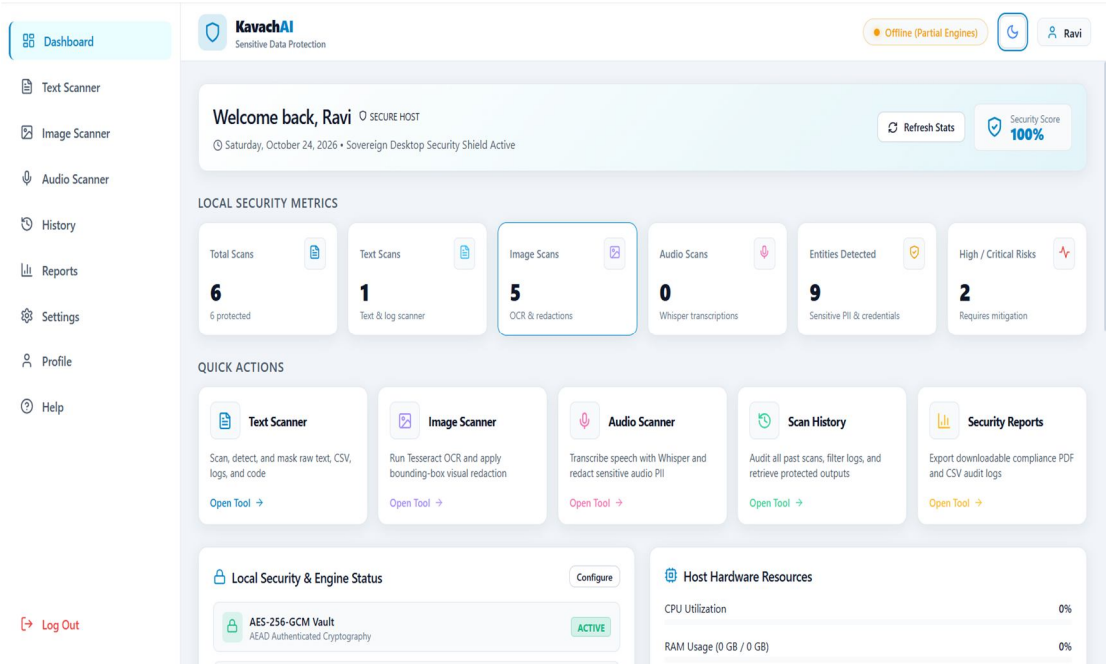


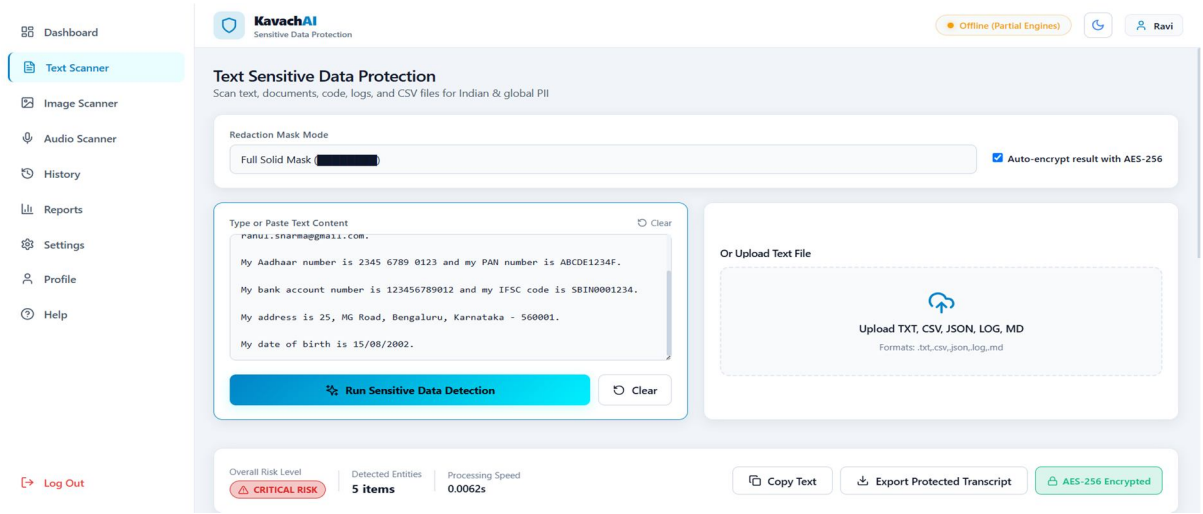
Fig. 6. KavachAI desktop dashboard showing the application's main processing options.

B. Text Detection and Masking Implementation

The text-processing module combines regular-expression matching with spaCy-based named-entity recognition. Regular expressions identify configured structured patterns, while NER identifies supported contextual entities. Where implemented, additional validation checks can be applied to compatible identifier candidates.

After detection, the application generates a masked version of the text. The output should retain non-sensitive content while replacing identified sensitive spans according to the configured masking policy. The resulting text can then be reviewed by the user or passed to the relevant output-retention workflow.

The effectiveness of this module depends on the detection rules, entity-recognition model, text normalization, and handling of overlapping candidate spans. A complete verification should compare the expected sensitive spans against the observed detections and inspect the masked text for any remaining readable sensitive values.



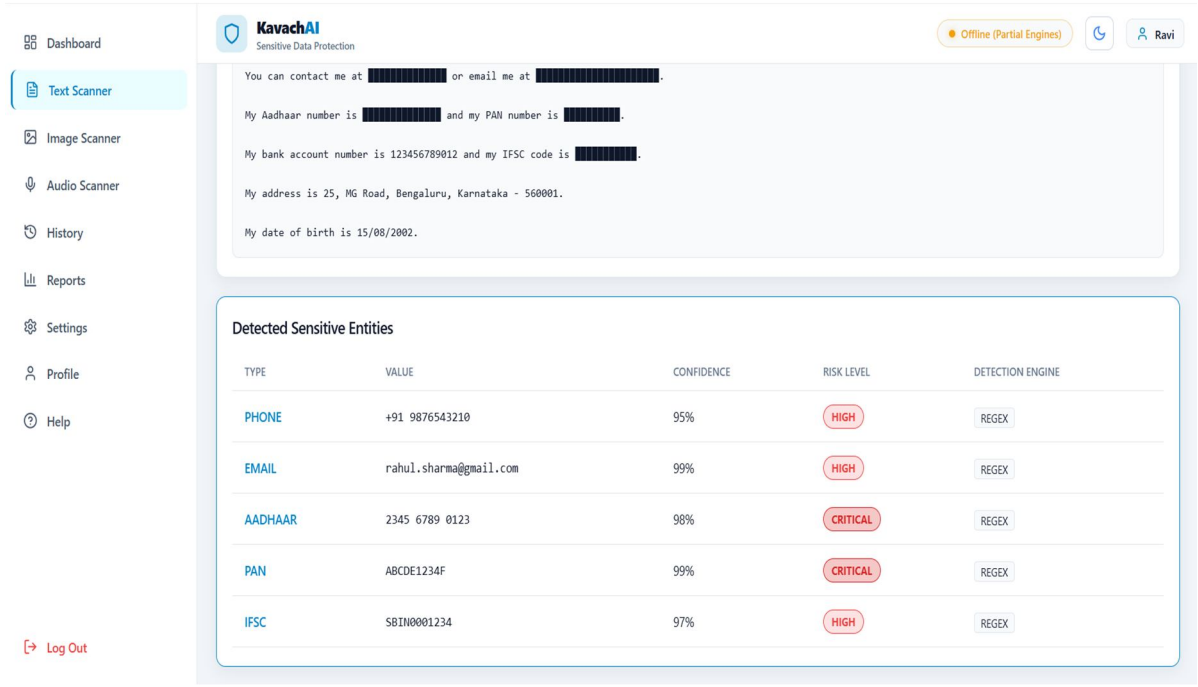


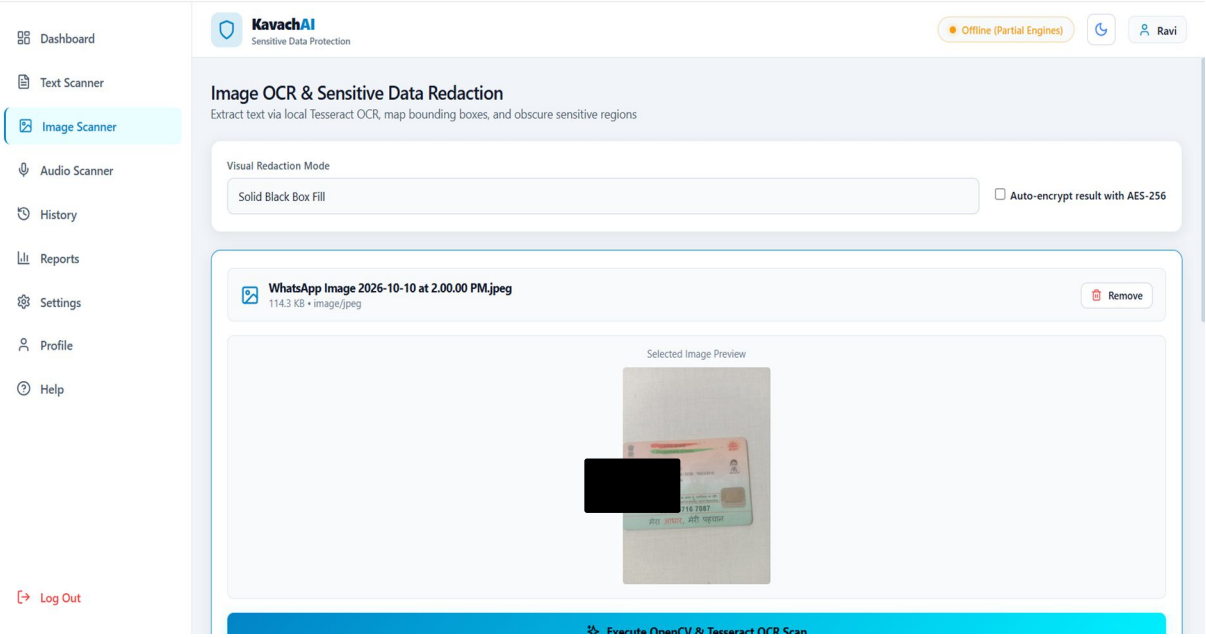
Fig. 7. Text scanner interface illustrating sensitive-information detection and text masking

C. Image OCR and Redaction Implementation

The image-processing module uses OpenCV and Tesseract OCR through the Python OCR interface. The OCR stage extracts text and spatial coordinates from an input image. The resulting text is processed by the shared sensitive-information detection pipeline, and detected spans are associated with their corresponding image regions.

The selected redaction operation is then applied to the relevant regions to produce a protected image. The supported visual operations described in the existing project paper include blurring, pixelation, black-boxing, and masking. The described visual operation should be consistent with the version of the application demonstrated in the accompanying screenshots.

Image redaction requires more than detecting sensitive text. The detected content must be mapped to the correct image coordinates, including any adjustments introduced by preprocessing. The exported image must also be checked for sensitive information that remains visible outside the selected regions.



International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

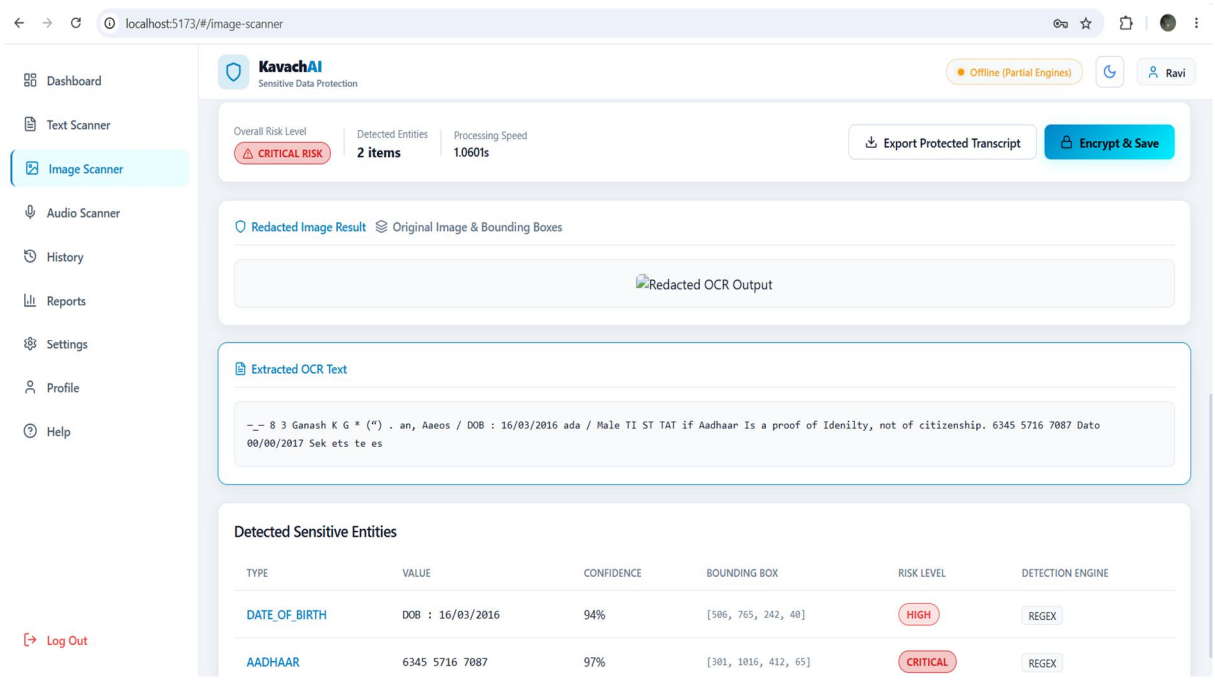


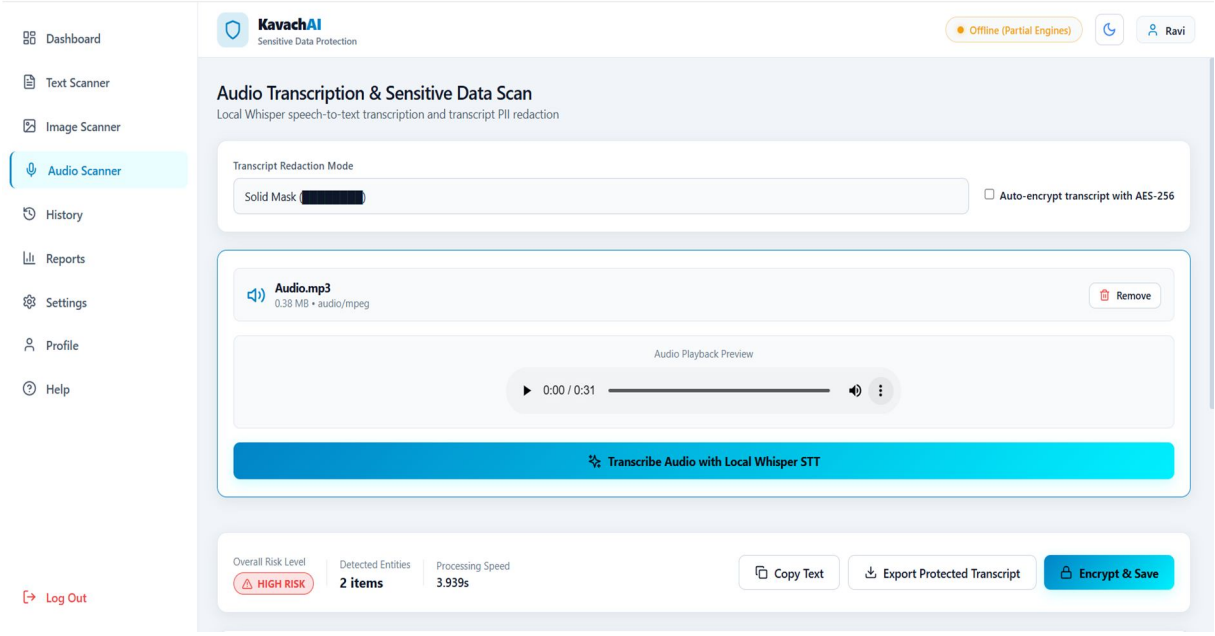
Fig. 8. Image-processing interface showing OCR-based sensitive-information localization and visual redaction.

D. Audio Transcription and Transcript Protection

The audio module uses a locally executed Whisper model to generate a transcript from a supported audio recording. FFmpeg is used when audio preparation or format conversion is required. The generated transcript is passed through the shared sensitive-information detection pipeline, after which supported sensitive spans can be masked.

The resulting protected transcript represents the text-level output of the audio workflow. It should not be interpreted as a redacted version of the original audio waveform. Unless direct waveform redaction has been separately implemented and verified, the original recording may still contain the spoken sensitive information.

Audio verification should distinguish transcription errors from detection errors. For example, if a spoken identifier is transcribed incorrectly, the failure may occur during speech recognition rather than during the downstream text-detection stage. A useful test therefore compares the intended spoken content, generated transcript, detected spans, and protected transcript.



©IJRASET: All Rights are Reserved

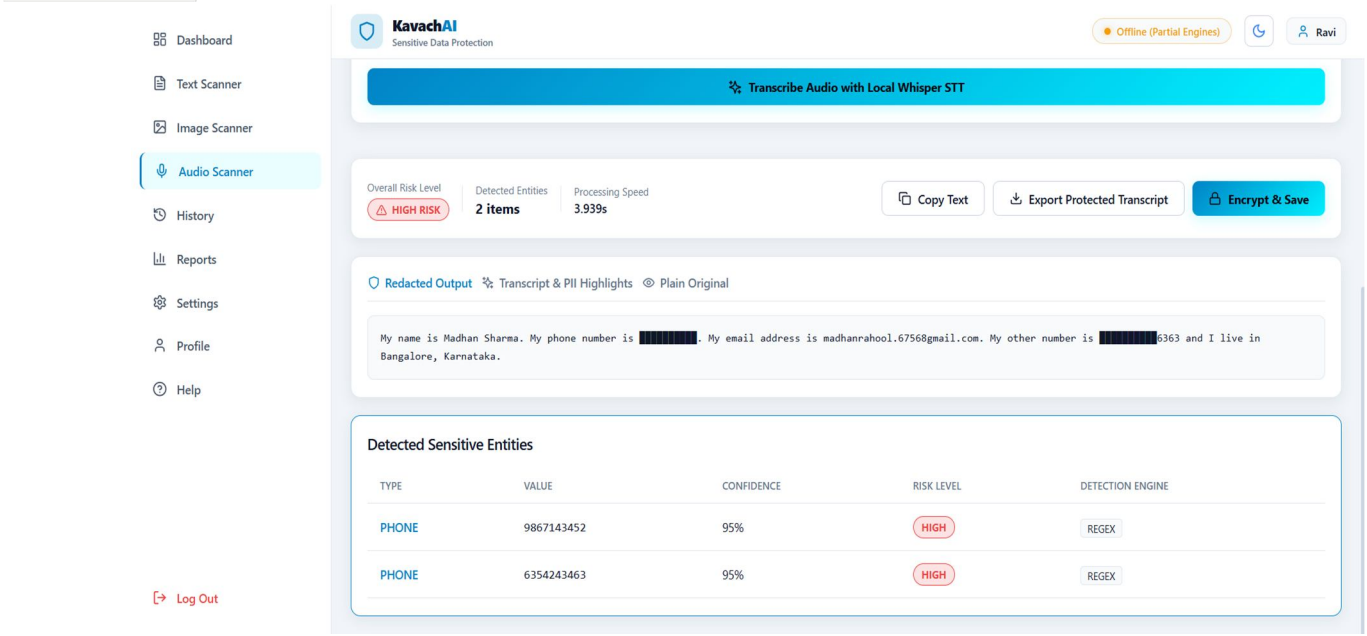


Fig. 9. Audio-processing interface showing speech transcription and text-level protection of detected sensitive information.

E. Protected Storage and Authentication

KavachAI uses SQLite and SQLAlchemy to manage local application data and scan-history information. Retained outputs processed through the encryption workflow are protected using AES-256-GCM, while Windows DPAPI protects the associated encryption key. The application also uses bcrypt-based password hashing and JSON Web Tokens (JWTs) for authentication-related operations.

These mechanisms address different security requirements. Password hashing protects stored authentication secrets, JWT validation supports authenticated requests, and AES-GCM protects the confidentiality and integrity of encrypted content when correctly implemented. DPAPI provides Windows-based protection for the associated key material.

The storage workflow distinguishes encrypted output files from ordinary database metadata. SQLite is used for application data and scan-history management, while AES-256-GCM protects retained outputs and Windows DPAPI protects the encryption key. These mechanisms do not imply that original inputs, intermediate results, transcripts, or every detection record are encrypted. Their retention and deletion behavior should be stated only where it has been confirmed in the implementation.

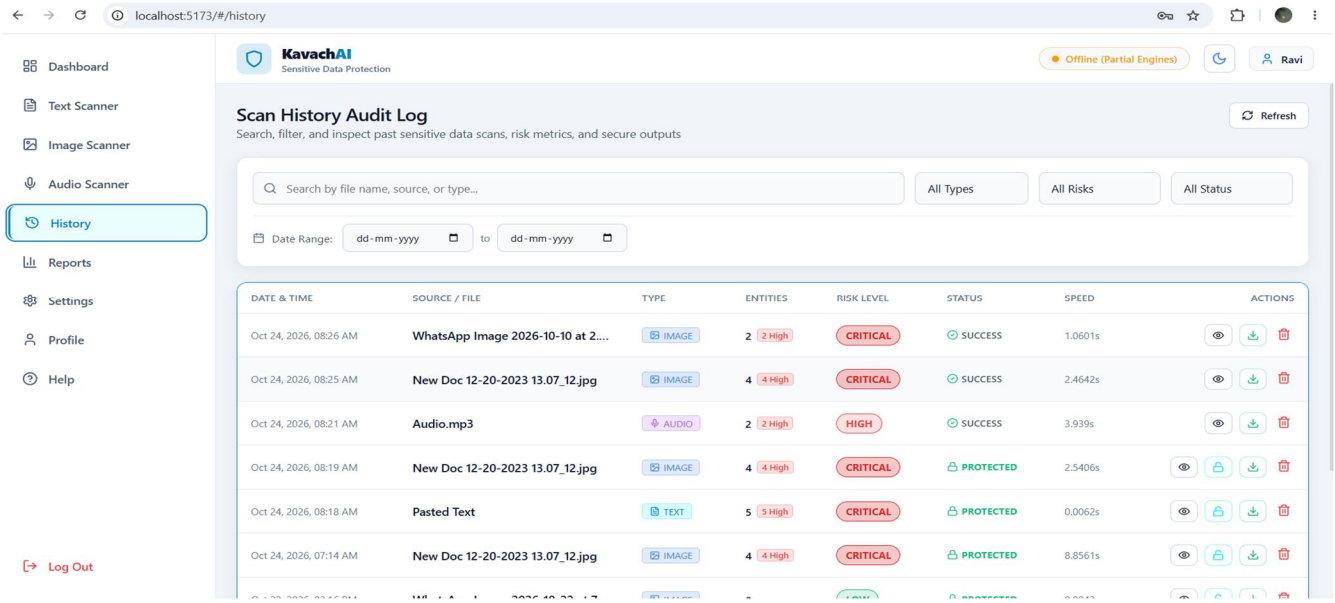


Fig. 10. KavachAI local history or protected-output interface, where available in the implemented application.

F. Implementation Components

The principal implementation components are summarized in Table I. The table describes the roles of the technologies within the documented architecture; it does not establish performance or security effectiveness.

Table I. Implementation Components Of Kavachai

Component	Technology	Role in the system
Desktop runtime	Electron	Provides the Windows desktop application environment
User interface	React and Vite	Provides the interface for input, processing, and output review
Backend service	FastAPI	Exposes local processing operations to the frontend
Text detection	Regular expressions and spaCy NER	Identifies configured patterns and supported named entities
Image processing	OpenCV	Supports image preprocessing and image operations
Optical character recognition	Tesseract OCR	Extracts text and spatial coordinates from images
Speech recognition	Whisper	Converts audio recordings into text transcripts
Audio preparation	FFmpeg, where required	Supports audio preparation or format conversion
Application data	SQLite and SQLAlchemy	Manages local application records and scan history
Output encryption	AES-256-GCM	Protects selected retained outputs
Key protection	Windows DPAPI	Protects the associated encryption key
Authentication	bcrypt and JWT	Supports password hashing and authenticated requests

G. Functional Verification Approach

Functional verification should establish whether each implemented module produces the expected type of output for representative test inputs. It is distinct from quantitative benchmarking, which requires a defined dataset and measurements.

Table II lists the functional verification activities used during development. The developer reports that the text, image, audio, encryption/decryption, and offline-operation workflows were tested. Because detailed raw test logs and quantitative measurements are not included in the available manuscript, the results are reported qualitatively and should not be interpreted as a benchmark of accuracy or completeness.

Table II. Functional Verification Plan

Test case	Verification activity	Expected outcome
Text detection	Submit synthetic text containing supported sensitive patterns and entities	Configured sensitive spans are identified
Text masking	Inspect the output generated from the test text	Detected spans are masked according to the configured policy
Image OCR	Submit a synthetic image containing readable text	OCR returns text and associated location information
Image redaction	Inspect the exported protected image	Intended regions are redacted, with no detected test value left readable in those regions
Audio transcription	Submit a supported test recording	A transcript is generated for review
Transcript protection	Process a transcript containing supported sensitive content	Detected transcript spans are masked
Protected storage	Store and retrieve a supported protected output	Retrieval follows the implemented authorized workflow
Encrypted-data integrity	If supported by the implementation, attempt to retrieve content after controlled ciphertext modification	Authentication failure is handled without accepting modified content as valid
Offline operation	Repeat core workflows with network access disabled after required dependencies and models are installed	The tested workflows complete without requiring an external service

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The functional checks reported here are qualitative. A reproducible evaluation should retain a record for each test containing a test identifier, input description, expected result, observed result, pass/fail status, and relevant software or model versions. Detailed records were not available for inclusion in this manuscript, so no numerical accuracy or performance claims are made.

Qualitative functional verification summary: the developer reports that the five principal workflows were exercised during development. The manuscript does not provide raw logs, a labeled benchmark dataset, or repeated timing measurements; therefore, no numerical accuracy, latency, or completeness values are reported.

H. Results and Reporting Boundaries

The current project documentation describes the system architecture, implementation components, and intended verification activities. However, it does not provide a complete set of recorded test cases, raw outputs, or quantitative measurements sufficient to establish detection accuracy, processing latency, OCR quality, transcription quality, or redaction completeness.

Accordingly, no numerical performance values or universal success claims are reported in this section. Such measurements should be added only after controlled testing. For example, precision, recall, and F1-score require a labeled evaluation dataset and a documented method for matching predicted detections to reference annotations. Processing latency requires repeated measurements under a specified hardware and software configuration.

Similarly, a successful demonstration of a redacted image does not establish that all sensitive information in arbitrary images will be detected. A successful audio transcription does not prove that every spoken sensitive value will be recognized. Offline operation must be checked with network access disabled, and storage protection must be verified through the actual encryption and retrieval workflow.

V. SECURITY ANALYSIS AND PRIVACY CONSIDERATIONS

A. Local Processing Boundary

KavachAI is designed to perform its core sensitive-information detection and redaction operations within a Windows desktop environment. The application integrates locally executed text-processing, image OCR, and audio-transcription components with a locally hosted backend. This architecture is intended to reduce the need to transmit user content to external cloud-based processing services.

Local processing can reduce exposure to risks associated with transferring sensitive content to remote services. However, local execution alone does not guarantee complete privacy. Sensitive information may remain exposed through application logs, temporary files, unprotected database fields, cached content, error messages, or exported files.

The privacy boundary therefore includes more than the detection engine. It also includes the desktop interface, backend communication, temporary processing data, local storage, encryption-key management, authentication, and output export. These components must be considered together when evaluating the application's privacy properties.

The developer reports testing the application with internet access disabled after installing the required dependencies and models, and reports that the principal text, image, and audio workflows were exercised in that condition. This is a functional offline-operation check, not proof that every execution path is free of network activity. A stronger claim would require documented network monitoring across all relevant workflows and startup paths.

B. Encryption of Retained Outputs

KavachAI uses the Advanced Encryption Standard in Galois/Counter Mode (AES-256-GCM) to protect selected retained outputs. AES is a standardized symmetric encryption algorithm [1], while GCM provides authenticated encryption when used correctly [2].

AES-256-GCM is intended to protect the confidentiality of encrypted content and to detect unauthorized modification of ciphertext through authentication verification. Correct implementation requires careful management of encryption keys, unique nonces for each encryption under a given key, and verification of authentication tags during decryption.

The encryption mechanism protects only the content that actually passes through the implemented encryption workflow. It should not be assumed that all input files, intermediate data, transcripts, database records, logs, or exported outputs are automatically encrypted. The implementation documentation should identify which data objects are encrypted, when encryption occurs, and how decryption is authorized.

Decryption failures should be handled safely. With authenticated encryption such as AES-GCM, ciphertext authentication must be checked before plaintext is accepted. The current manuscript does not include a detailed tampering-test record, so it does not claim that modified-ciphertext rejection has been independently verified.

C. Windows DPAPI and Key Protection

Windows Data Protection API (DPAPI) is used to protect the AES encryption key associated with the retained-output workflow [6], [7]. DPAPI provides Windows-based data-protection functionality and can reduce the need to store an encryption key directly alongside protected content.

Key protection and content encryption serve different purposes. AES-256-GCM encrypts the retained output, whereas DPAPI protects the associated key material. The security of the overall design depends on correct key generation, storage, retrieval, access control, and lifecycle management.

DPAPI does not eliminate every risk associated with a compromised Windows account or system. If an attacker gains sufficient access to the relevant user context or execution environment, locally available data may remain at risk. DPAPI should not be interpreted as providing absolute protection against malware, privileged attackers, or compromise of the host operating system.

D. Authentication and Access Control

KavachAI uses bcrypt-based password hashing and JSON Web Tokens (JWTs) for authentication-related operations. Password hashing helps avoid storing user passwords directly in plaintext. JWTs can support authenticated requests between the desktop interface and the backend.

These mechanisms address different aspects of access control. Password hashing protects stored password representations, while token validation helps the backend determine whether a request carries acceptable authentication credentials. Neither mechanism independently guarantees that every sensitive operation is properly authorized.

Authentication and authorization are important controls for protected operations. Token expiration, invalid-token handling, authentication failures, and access to stored outputs should be tested and documented. The manuscript does not provide detailed results for each of these cases; therefore, it does not claim comprehensive authentication or authorization validation.

E. Protection of Text, Image, and Audio-Derived Information

KavachAI applies different redaction methods according to the input modality.

For text, detected sensitive spans are replaced with masking characters or configured labels. The protection depends on the completeness of detection and the correctness of the span replacement.

For images, OCR identifies text and associated locations, after which the application applies visual redaction to the corresponding regions. A redacted image must be inspected to determine whether sensitive information remains visible outside the selected regions. If preprocessing changes image coordinates, the mapping to the original image must also be verified.

For audio, the application generates a transcript using speech recognition and applies the text-detection pipeline to that transcript. This workflow protects the resulting textual output; it does not establish that the original audio waveform has been modified or sanitized. These distinctions are important because detection and redaction can fail independently. An extraction error may prevent the detection engine from receiving the correct text, while an incorrect coordinate mapping may cause an image redaction to cover the wrong region. Consequently, a successful processing response should not automatically be treated as proof that all sensitive information has been removed.

F. Data Retention and Local Storage Risks

SQLite supports local application data and scan-history management. Although local storage can reduce dependence on remote infrastructure, it introduces risks if sensitive records remain accessible in unprotected database fields or related files.

The privacy assessment should identify the treatment of the following data categories:

1) Original text, image, and audio inputs.
2) Intermediate OCR output and audio transcripts.
3) Detected sensitive spans and associated labels.
4) Masked text and visually redacted images.
5) Encrypted retained outputs and associated key material.
6) Scan-history metadata, application logs, and temporary files.

Data handling should be described separately for original inputs, intermediate results, transcripts, detection records, encrypted outputs, and ordinary application metadata. The manuscript identifies AES-256-GCM for retained outputs and Windows DPAPI for key protection, but does not provide a complete verified retention matrix for every data category. No blanket claim is made that all application data is encrypted or automatically deleted.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The application should minimize unnecessary retention of original inputs and intermediate plaintext where supported by its actual workflow. Temporary files and logs should be reviewed to ensure they do not retain sensitive values. This paper does not claim secure deletion unless the deletion mechanism and its behavior have been specifically verified.

G. Threats and Mitigation Measures

Table III summarizes principal risks relevant to KavachAI and the corresponding mitigation measures or verification activities. The table represents a security-analysis framework; it does not imply that every mitigation has been independently validated.

Table III. Security Risks And Mitigation Measures

Potential risk	Relevant mitigation or verification
Sensitive content transmitted to an external service	Keep core processing local and verify behavior with network access disabled
Unauthorized access to retained outputs	Apply authentication and authorization checks to protected operations
Exposure of stored output content	Encrypt applicable retained outputs using AES-256-GCM
Exposure of encryption keys	Protect associated key material using Windows DPAPI
Unauthorized modification of encrypted content	Verify AES-GCM authentication during decryption and reject invalid ciphertext
Sensitive information retained in logs or temporary files	Review logging, temporary-file handling, and retention behavior
Missed sensitive content in text	Test configured patterns and supported named entities against labeled examples
Missed or incorrectly localized image text	Verify OCR extraction, coordinate mapping, and exported redaction
Sensitive speech omitted or mistranscribed	Evaluate transcription output separately from downstream detection
Unauthorized backend requests	Test token validation, token expiry, and authorization checks
Failure of a local model or processing component	Handle errors without exposing sensitive input in logs or messages

H. Limitations of the Security Design

The security of KavachAI depends on both its implemented mechanisms and the operating environment in which it runs. The presence of encryption, key protection, authentication, and local processing does not by itself establish resistance to all attacks.

The system may fail to detect sensitive information when patterns are unsupported, text is ambiguous, OCR output is incorrect, or speech recognition omits relevant content. Visual redaction may also be incomplete if the detected text is mapped to the wrong region or if the exported file retains readable sensitive information elsewhere. Local storage introduces additional considerations, including access to the Windows user account, permissions on application files, temporary data, database contents, and exported results. Similarly, authentication mechanisms require correct configuration and authorization checks to provide their intended protection. The current evaluation does not constitute a formal penetration test or comprehensive threat-model assessment. The reported offline test is a functional check and does not establish a verified absence of all network traffic. The manuscript also does not report quantitative measures of redaction completeness.

I. Privacy and Security Assessment

KavachAI combines multimodal sensitive-information detection with redaction and local protection of selected retained outputs. Its principal architectural objective is to reduce unnecessary external transmission while providing a shared detection workflow for text, image-derived text, and audio-derived transcripts. The security assessment must consider the entire processing lifecycle rather than only the encryption algorithm. Input extraction, detection, redaction, temporary storage, key management, authentication, output export, and data retention all influence the privacy outcome. The system was exercised through functional checks of text detection and masking, image OCR and redaction, audio transcription and transcript masking, encryption/decryption of retained outputs, and operation with internet access disabled. The available evidence supports reporting these checks qualitatively; it does not establish quantitative detection accuracy, processing speed, or complete redaction under all conditions.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

VI. CONCLUSION AND FUTURE SCOPE

A. *Conclusion*

This paper presented KavachAI, a privacy-oriented desktop application designed to detect and redact sensitive information from text, images, and audio-derived transcripts. The system integrates multiple processing techniques within a unified desktop environment to support the identification and protection of potentially sensitive information before digital content is retained or shared.

For text processing, KavachAI combines regular-expression matching with named-entity recognition to identify configured patterns and supported contextual entities. For image processing, optical character recognition extracts text from images so that sensitive information can be detected and visually redacted in the corresponding image regions. For audio processing, locally executed speech-to-text transcription converts speech into text that can be analyzed using the shared detection pipeline. The resulting transcript can be masked or protected; however, the original audio waveform is not directly redacted by the current workflow.

The application uses Electron and React for the desktop interface, a locally hosted FastAPI backend for processing requests, Python-based detection components, and SQLite for local application data and scan-history management. AES-256-GCM is used to encrypt retained outputs where encryption is implemented, while the Windows Data Protection API (DPAPI) protects the associated encryption key. These mechanisms provide a foundation for local data protection without requiring a cloud-based processing service as part of the intended core workflow.

The main contribution of KavachAI is the integration of text detection, image OCR-based redaction, audio transcription, and protected local storage into a single desktop application. Rather than treating each modality as an independent workflow, the system reuses the text-detection pipeline for text extracted from images and audio recordings. This design supports a more consistent approach to identifying sensitive information across different forms of digital content.

Nevertheless, the effectiveness of the system depends on the quality of the input, the coverage of configured detection patterns, the performance of the underlying models, and the accuracy of mapping detected information to its original location. OCR and speech-recognition errors may affect subsequent detection, while named-entity recognition and rule-based matching may produce false positives or miss sensitive information. Encryption and local processing also do not, by themselves, guarantee complete protection against every possible threat.

Functional testing was carried out during development. The developer reports that the five principal workflows listed in Table II were tested; however, the available manuscript does not include a complete set of raw test records or quantitative measurements sufficient to establish detection accuracy, processing speed, or redaction completeness. These properties require a documented, reproducible evaluation and are not claimed as measured results in this paper.

In conclusion, KavachAI demonstrates an integrated engineering approach to multimodal sensitive-information detection and redaction in a desktop environment. Its architecture provides a basis for further development in detection reliability, redaction verification, security assessment, and usability.

B. *Future Scope*

Several improvements can extend the capabilities and strengthen the reliability of KavachAI.

1) Quantitative evaluation of detection performance: Future work should evaluate the system using a controlled dataset containing representative sensitive and non-sensitive examples. Precision, recall, F1-score, false-positive rate, and false-negative rate can be calculated separately for text, image-derived text, and audio-derived transcripts. The dataset, annotation procedure, test conditions, and results should be documented to support reproducibility.

2) Improved image-coordinate mapping and redaction verification: Image processing can be strengthened by validating the mapping between OCR bounding boxes and the original image coordinates after preprocessing, resizing, or other transformations. Future versions should include systematic visual inspection and automated checks to identify residual readable sensitive information in exported images. Redaction completeness should be assessed rather than assumed from the presence of a mask.

3) Enhanced audio privacy protection: The current audio workflow protects information identified in the generated transcript rather than directly modifying the original waveform. Future development could investigate timestamp-aligned redaction, audio segment replacement, muting, or tone substitution for detected sensitive speech. Such functionality would require reliable alignment between transcript segments and their corresponding audio intervals, followed by verification that the protected audio no longer reveals the targeted information.

©IJRASET: All Rights are Reserved

4) Expanded sensitive-information policies: Detection rules can be extended to support configurable categories, regional identification formats, and application-specific sensitivity policies. The system could also provide configurable masking styles and explicit handling rules for overlapping detections. Any checksum-based validation or additional identifier-specific checks should be documented according to the formats actually supported.

5) Stronger privacy and security assessment: Future work should include a systematic examination of temporary files, application logs, cached data, scan history, exported outputs, and key-handling procedures. Network activity should be examined under defined test conditions before making a verified offline-operation claim. Security testing should also assess authentication, session management, access control, and recovery behavior without treating the use of encryption as proof of overall system security.

6) Reproducible deployment and dependency documentation: The application can be improved through a documented installation procedure, explicit dependency and model versions, backend startup instructions, and a description of communication between the Electron frontend and FastAPI backend. A reproducible release package and a documented test procedure would help other developers install, inspect, and evaluate the system consistently.

7) Performance and usability evaluation: Future experiments can measure processing time, memory consumption, and resource usage for different input sizes and modalities. User-focused evaluation could examine the clarity of detection highlights, the usefulness of masking options, the accessibility of scan history, and the ease of reviewing protected outputs. Results should be reported only after measurements have been collected under defined conditions.

REFERENCES

[1] National Institute of Standards and Technology, Advanced Encryption Standard (AES), FIPS PUB 197-upd1, May 2023. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.197-upd1

[2] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D. Gaithersburg, MD, USA: National Institute of Standards and Technology, Nov. 2007. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-38D

[3] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," arXiv preprint arXiv:2212.04356, 2022. [Online]. Available: https://arxiv.org/abs/2212.04356

[4] R. Smith, "An overview of the Tesseract OCR engine," in Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR), vol. 2, 2007, pp. 629–633, doi: 10.1109/ICDAR.2007.4376991.

[5] Explosion AI, "spaCy: Industrial-strength Natural Language Processing in Python," spaCy documentation. [Online]. Available: https://spacy.io/. [Accessed: Oct. 24, 2026].

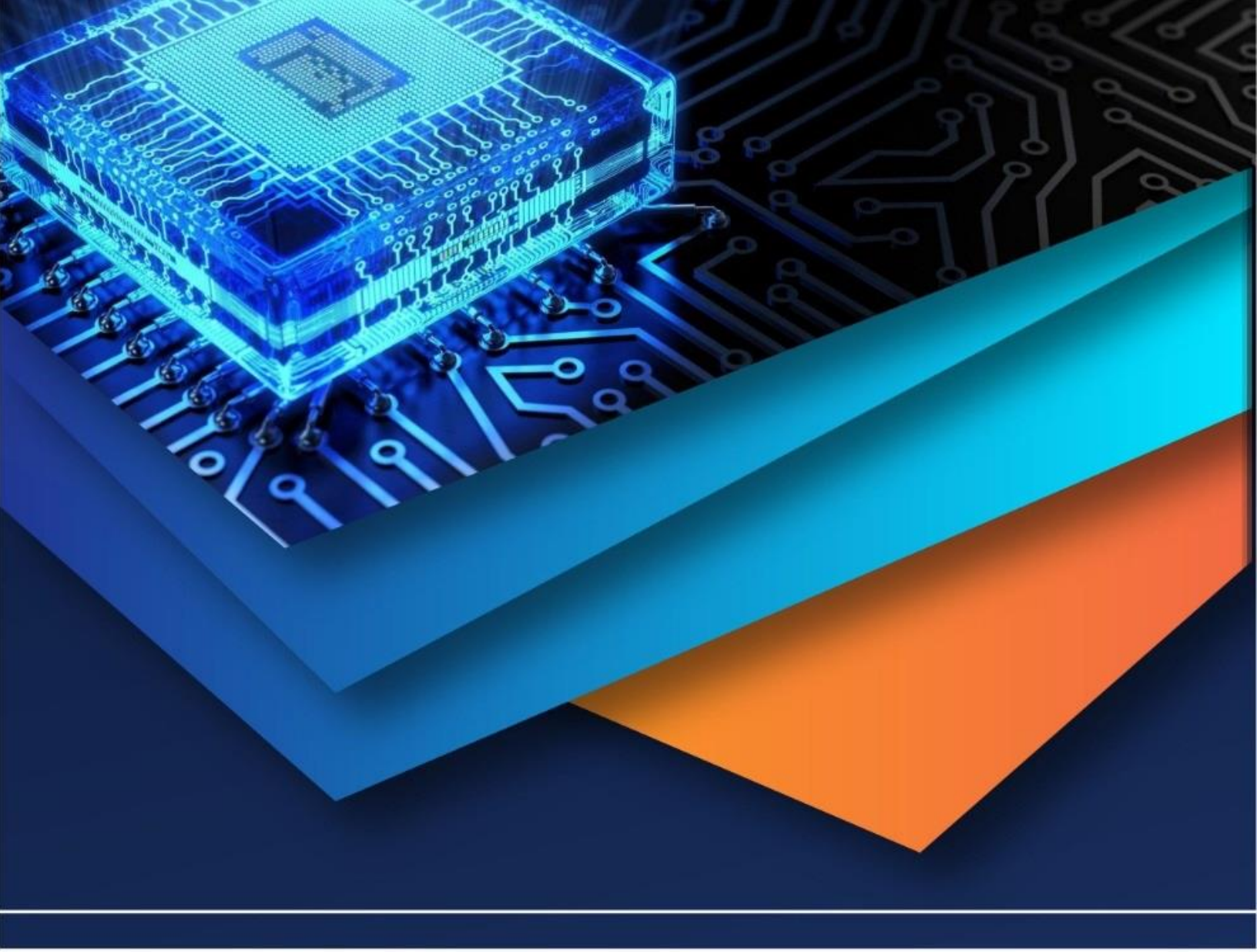
[6] Microsoft, "CryptProtectData function (dpapi.h)," Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/dpapi/nf-dpapi-cryptprotectdata

[7] Microsoft, "Data Protection API (DPAPI)," Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/secauthn/data-protection

[8] Electron, "Introduction," Electron Documentation. [Online]. Available: https://www.electronjs.org/docs/latest/

[9] FastAPI, "FastAPI documentation." [Online]. Available: https://fastapi.tiangolo.com/

[10] Tesseract OCR, "Tesseract Open Source OCR Engine." [Online]. Available: https://github.com/tesseract-ocr/tesseract



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)