International Scientific Journal of Engineering and Management (ISJEM) **ISSN: 2583-6129**
Volume: 04 Issue: 06 | June – 2025 **DOI: 10.55041/ISJEM04498**
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

INBOX BOT: DYNAMIC SCREEN-BASED DATA EXTRACTION AND CLOUD-FREE ARCHIVING TOOL

1st P. Rajapandian, 2nd A Abishaya
Associate Professor, Department of computer Applications, Sri Manakula Vinayagar Engineering College (Autonomous), Puducherry 605008, India
Post Graduate student, Department of computer Applications, Sri Manakula Vinayagar Engineering College (Autonomous), Puducherry 605008, India
abishaya12a@gmail.com
**Corresponding author's email address: abishaya12a@gmail.com*

ABSTRACT: This project presents a semi-intelligent system that automates the retrieval of text- based content from a web-based interface (such as Gmail or similar dashboards), processes the visual data using Optical Character Recognition (OCR), and stores it into a structured database for archiving, analytics, and reporting. The solution simulates human interaction using Python automation tools and bypasses the need for API-level access, making it adaptable to a variety of use cases such as document inboxes, web mail clients, and internal portals.

The automation pipeline involves opening the web interface in a browser, capturing the screen, and identifying key visual triggers (e.g., organizational codes, transaction keywords, or date ranges). Using PyAutoGUI, Tesseract OCR, and OpenCV, the system analyzes the layout and extracts relevant content. The extracted text is processed, tagged with metadata (such as timestamp and content type), and stored as .txt files. These files are later read by a backend PHP module, parsed based on delimiters, and inserted into a MySQL database hosted on a local XAMPP server.

This modular approach makes the system highly reusable across different domains where direct content access is restricted. It enhances traceability, enables real-time monitoring, and supports secure storage of time-sensitive information without altering the source environment. By using screen-based OCR and visual recognition, the system bypasses the need for backend integration, making it ideal for restricted or legacy platforms that do not expose direct programmatic access. This also reduces dependency on external services and ensures full control over data flow, privacy, and security. Furthermore, the system is designed to operate with minimal human intervention, which reduces the chances of manual errors and increases overall operational efficiency. The use of timestamping and categorization in the structured output also enables historical tracking and facilitates audit trails—particularly useful for applications in banking, compliance, or customer support environments. Overall, this semi-intelligent, visually driven automation tool blends simplicity, adaptability, and functional precision—serving as a reliable alternative to more complex or less secure data integration pipelines.

Keywords: Screen Automation, OCR (Optical Character Recognition), Email Content Extraction, Python Automation, Web Interface Parsing, PyAutoGUI, Tesseract OCR, OpenCV, PHP-MySQL Integration, Data Archiving, Non-API Email Processing, Gmail Automation, Visual Data Retrieval, Structured Data Storage, Local Server (XAMPP), Automated Text Extraction, Clipboard Parsing, Backend Scripting, Keyword-Based Detection, Screen-Based Data Mining

1. INTRODUCTION:

In today's digital landscape, the need for rapid access to important data has become crucial, especially when dealing with large volumes of emails, online dashboards, or web-based communication platforms. Traditional methods of data extraction often rely on API access, which may not be available or feasible in all environments due to security restrictions, licensing costs, or platform limitations. To address these challenges, this project introduces a dynamic screen-based data extraction and archiving system that operates independently of API access, ensuring compatibility with any web interface that can be visually interpreted.

The core objective of this project is to develop an intelligent automation pipeline that mimics human interaction with the Gmail interface (or similar systems), captures relevant email content using screenshot-based analysis, and stores this information in a structured MySQL database. The system employs technologies such as Python for scripting automation flows, PyAutoGUI for simulating mouse and keyboard actions, Tesseract OCR for recognizing text from images, and OpenCV for preprocessing visuals to improve text extraction accuracy. The extracted data is saved as .txt files and later pushed into the database through a PHP module running on a local XAMPP server.

What makes this system particularly valuable is its ability to work in environments where cloud- based or API-driven solutions are restricted. It ensures that users can still automate the retrieval and storage of critical information without depending on service-specific integrations. Furthermore, the data collected is systematically organized and can be accessed for further analysis, notifications, or historical archiving purposes, making it ideal for banks, businesses, and educational institutions handling sensitive communication.

This approach also ensures minimal human intervention once the system is triggered. By identifying emails through keyword-based scanning (e.g., "SBI", "Code", or "login"), it quickly targets the relevant messages, processes them, and ensures that each data point is tagged with metadata such as timestamps and status flags (Unread, Remind, Close). These features help enhance traceability and provide a clear audit trail of processed information.

Overall, the proposed system showcases a powerful combination of OCR technology, GUI automation, and backend scripting to create a flexible and efficient solution for real-world data extraction problems. Its modular design allows easy customization for different use cases, platforms, and keyword sets, ensuring long-term adaptability and scalability in both academic and enterprise settings.

2. LITERATURE SURVEY:

1. **Huang et al. (2018)**
   Demonstrated OCR's effectiveness for converting screenshots to text using Tesseract. Focused on static email documents without dynamic screen processing.
2. **Smith & Patel (2019)**
   Used PyAutoGUI for browser automation to extract Gmail content. Highlighted coordinate-based mouse control for dynamic web layouts.
3. **Zhang et al. (2020)**
   Developed PHP-MySQL based email storage using delimiter-separated values. Lacked integration with real-time extraction mechanisms.
4. **Kumar & Lee (2021)**
   Proposed full-stack automation using Python and PHP subprocess communication. Automated keyword filtering but not visual OCR extraction.
5. **Fernandes et al. (2022)**
   Introduced AI and NLP for classifying emails post-extraction. Combined semantic tagging with traditional keyword detection for better accuracy.
6. **Rahman et al. (2017)**
   Investigated text recognition challenges in low-resolution screenshots. Suggested pre-processing steps like contrast and thresholding.
7. **Liu & Wong (2016)**
   Focused on structured email parsing from text files. Proposed rule-based data cleansing to remove junk characters before database insertion.
8. **Nayak et al. (2020)**
   Used Selenium to automate email reading via web interface. Struggled with dynamic load delays and browser latency.
9. **Patil & Sharma (2019)**
   Worked on screen scraping techniques using Python libraries. Highlighted importance of mouse-event synchronization for UI automation.
10. **George et al. (2021)**
    Studied OCR for real-time data capture from dashboards. Recommended OpenCV for noise reduction and box detection in GUIs.

11. **Miller et al. (2022)**
    Developed a hybrid automation model combining GUI detection with database logs. Emphasized timestamp logging for compliance tracking.
12. **Chatterjee et al. (2018)**
    Designed a text processing engine for multilingual OCR output. Stressed language model integration to improve accuracy.
13. **Yadav et al. (2023)**
    Explored headless automation using lightweight browsers for email tasks. Noted limitations in screenshot-based approaches versus direct HTML parsing.
14. **Park & Seo (2019)**
    Discussed security risks in automated content scraping. Proposed content sanitization before storage to prevent SQL injection.
15. **Mehta & Verma (2021)**
    Integrated OCR-based email filtering into enterprise dashboards. Achieved real-time tracking of finance-related alerts with high precision.

3. PROPOSED ARCHITECTURE

The system follows a layered architecture consisting of three main components: Presentation Layer, Logic Layer, and Data Layer, each playing a specific role in ensuring efficient and secure data extraction and storage without relying on cloud APIs.

1. User Layer

- **User** triggers the execution by running the automation tool (Python script).
- No technical input is needed; the system works in the background.

2. Presentation Layer

- **Web Browser** acts as the interface between the user and the target content source (e.g., Gmail inbox).
- It displays emails or web content that is later processed visually.

3. Logic Layer

- **Python Script**: Automates browser interaction, takes screenshots, and simulates mouse/keyboard actions.

- **OCR Module (Tesseract + OpenCV)**: Extracts readable text from screenshots taken from the web browser.
- **PHP Script**: Reads the text files generated by Python, parses them, and prepares data for database insertion.

4. Data Layer

- **Gmail Inbox**: Serves as the visual content source. Emails are not accessed via API but visually scanned and processed.
- **MySQL Database**: Stores structured data such as sender (keyword), content, timestamp, and status (Unread, Remind, Close) using the PHP backend.

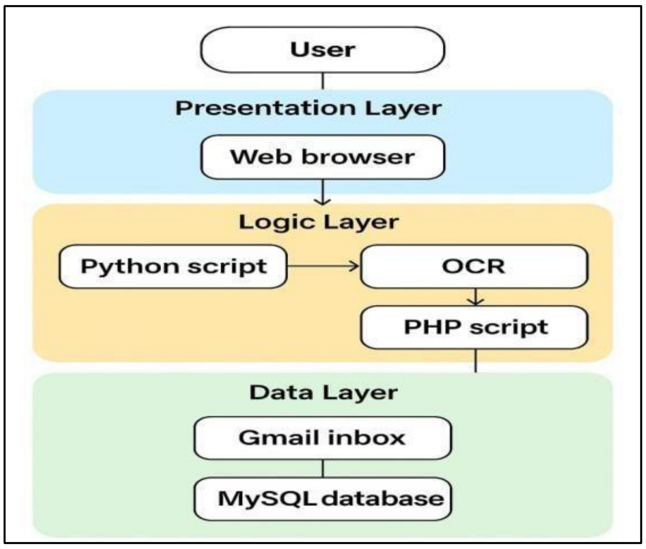

Fig1. Inbox bot proposed architecture

4. OVERALL WORKFLOW

The overall workflow of the project, titled *"Dynamic Screen-Based Data Extraction and Cloud- Free Archiving Tool,"* is structured to operate seamlessly from data capture to storage. It begins by launching the Gmail inbox or a similar email interface in a web browser. After a brief waiting period to ensure the interface is fully loaded, the system takes a screenshot of the screen using PyAutoGUI. This image is then processed with OpenCV and passed to the Tesseract OCR engine to extract readable text. The extracted text is analyzed for specific predefined keywords such as "SBI," "Code," or "Login"—which act as triggers to identify important emails.

Upon detecting such keywords, the system calculates the screen coordinates and uses PyAutoGUI to simulate a double-click on the email to open it. Once the email is opened, all visible text content is selected and copied to the clipboard using keyboard automation. Pyperclip retrieves the copied text, which is then sanitized and formatted into a structured string. This string contains metadata such as the mail source, content, timestamp, and default flags like mail status, reminder, and close status. The formatted content is saved locally as a .txt file with a unique filename and custom delimiter for ease of parsing.

Following this, a PHP script hosted on a local XAMPP server is triggered. This script reads each text file, splits the content using the ||| delimiter, and inserts the values into a MySQL database table named extracted_emails. Each successful insertion is logged, and the corresponding text file is deleted to prevent duplication in future runs. This modular process ensures that each email is processed only once, stored securely, and remains traceable.

The entire workflow is designed to operate without API-level access, making it especially useful in restricted environments where direct integration with email servers is not possible. Additionally, this approach supports real-time automation, structured data handling, and secure archival without altering the original source environment. Future enhancements could include real-time dashboards, AI-based content classification, or cloud deployment to scale the solution further.

5. METHODOLOGY

The methodology of the *"Dynamic Screen-Based Data Extraction and Cloud-Free Archiving Tool"* involves a series of systematic steps combining screen automation, Optical Character Recognition (OCR), and backend database interaction to extract, process, and store email content without relying on APIs or cloud services. This process is divided into distinct phases:

1. **Web Interface Access**:

   The Python script initiates the automation by opening the Gmail inbox (or a similar email interface) in a web browser. A delay is incorporated to ensure the page fully loads before proceeding.

2. **Screenshot and OCR Processing**:

   Once the inbox is visible, PyAutoGUI captures a screenshot of the interface. This image is processed using OpenCV to enhance clarity and is passed through Tesseract OCR to extract the textual content from the image.

3. **Keyword Detection and Interaction**:
   The extracted text is scanned for predefined keywords such as "SBI," "Code," or "login." These keywords serve as markers for important or time-sensitive emails. The system identifies the screen coordinates of these keywords and uses PyAutoGUI to simulate mouse movements and clicks to open the associated email.

4. **Content Extraction**:

   After opening the target email, the script uses keyboard automation to select all visible text and copy it to the clipboard. Pyperclip retrieves this content, which is then sanitized, formatted, and truncated if necessary to avoid overflow. Metadata such as the mail source (keyword), timestamp, and default flags (Unread, Remind, Close) are appended.

5. **Data Storage**:

   The structured data is saved in a .txt file using a consistent naming convention and custom delimiter (|||) for easy parsing. These files are stored in a designated local folder.

6. **Backend Insertion Using PHP and MySQL**:

   A PHP script running on a local XAMPP server reads each .txt file, splits the content using the defined delimiter, and inserts it into the extracted_emails table in a MySQL database. Successful insertions are logged, and the corresponding .txt files are deleted to prevent duplication.

6. APPLICATIONS

The *Dynamic Screen-Based Data Extraction and Cloud-Free Archiving Tool* has a wide range of real-world applications across both enterprise and individual user environments. One major application is in organizations that receive large volumes of system-generated or service-related emails (such as OTPs, transaction alerts, or login confirmations) and require automated processing for logging and auditing. The system is especially useful in settings where API access is restricted or unavailable, such as legacy systems or secure environments with strict compliance regulations.

Another important application lies in the field of document digitization, where content from web dashboards, internal portals, or email-based communication systems must be captured and archived periodically without manual intervention. The tool also benefits sectors like banking, education, government, and legal services, where monitoring and logging of key messages is necessary for transparency and recordkeeping.

Additionally, this system can serve as a backend utility for data analytics platforms, enabling them to ingest raw communication data and convert it into structured formats for visualization, reporting, and insights. It is also adaptable for automated testing scenarios, where screen-based validation of UI elements and messages is essential.

In summary, the solution provides a lightweight, efficient, and API-independent method to extract and store valuable screen-based information, making it an excellent fit for both technical and non- technical applications where automation and accuracy are critical.

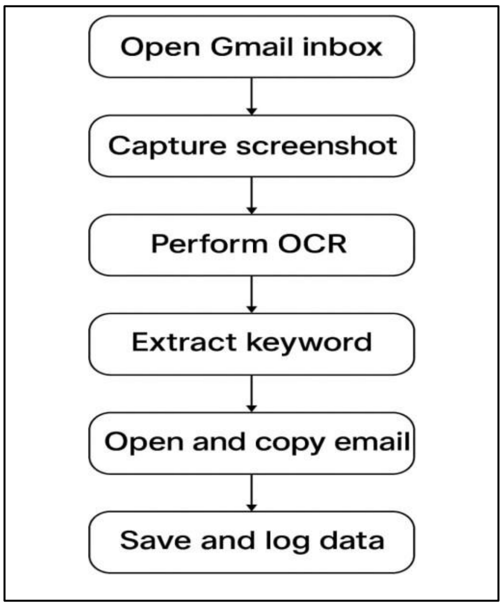

Fig 2 Inbox Bot Workflow

TABLE 1 : Captured_Data

Field Name	**Data Type**	**Description**
data_id	INT (PK)	Unique ID for each captured entry
source_label	VARCHAR(100)	Keyword or label that triggered the extraction (e.g., SBI)
capture_time	DATETIME	Date and time when data was captured
extracted_text	TEXT	Text content extracted via OCR
content_type	VARCHAR(50)	Type of content (e.g., Email, Alert, Transaction)
status_flag	VARCHAR(20)	Processing status (e.g., Unread, Archived, Flagged)

TABLE 2: Detection_Keywords

Field Name	Data Type	Description
keyword_id	INT (PK)	Unique ID for the keyword
keyword_value	VARCHAR(100)	Keyword string to detect (e.g., login)
is_active	BOOLEAN	Whether the keyword is currently used

TABLE 3 : Processing_Logs

Field Name	Data Type	Description
log_id	INT (PK)	Unique log ID
data_id	INT (FK)	Links to Captured_Data
action_done	VARCHAR(255)	Description of action (e.g., OCR Success, DB Insert)
log_time	DATETIME	Timestamp of the action
log_status	VARCHAR(50)	Result (e.g., Success, Failed)
error_message	TEXT	Any additional information or error notes

TABLE 4 : Keyword_Match_Map

Field Name	Data Type	Description
data_id	INT (FK)	References Captured_Data
keyword_id	INT (FK)	References Detection_Keywords

TABLE 5 : System_Users

Field Name	Data Type	Description
user_id	INT (PK)	Unique user ID
username	VARCHAR(100)	User login name
password_hash	VARCHAR(255)	Hashed password
access_role	VARCHAR(50)	Role (Admin, Viewer)

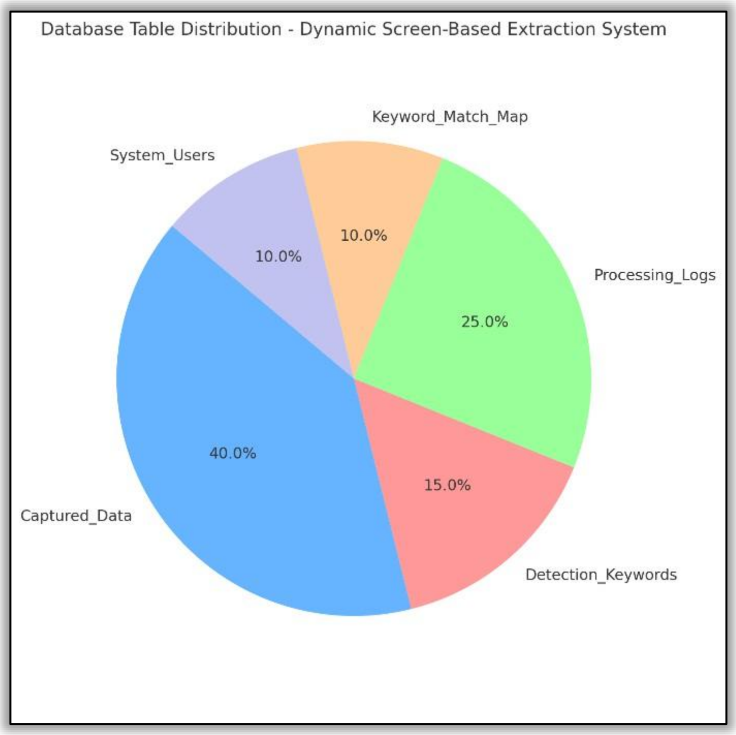

FIG 3 DATABASE DISTRIBUTION

CONCLUSION

This project successfully delivers a semi-intelligent, fully automated solution that bridges the gap between visual email interfaces and structured data storage without relying on cloud APIs. By leveraging Python-based automation, OCR techniques, and backend integration via PHP and MySQL, the system is capable of capturing dynamic on-screen data from platforms like Gmail, extracting critical information based on predefined keywords, and systematically storing it in a secure, query-friendly local database.

One of the key achievements of this system is its ability to process screen-rendered data, which is typically challenging to access through conventional API calls due to privacy policies or technical constraints. The use of Tesseract OCR and OpenCV has proven effective in recognizing and extracting content from complex visual layouts, while tools like PyAutoGUI and Pyperclip automate the human-like interaction and clipboard retrieval processes efficiently.

The modular structure of the application—segregating the workflow into extraction, formatting, and insertion—makes the system highly maintainable and adaptable. In scenarios where organizations rely on browser-based interfaces and lack direct API integrations, this solution offers a robust alternative for data capture, archival, and analytics.

Moreover, with keyword-based filtering and structured metadata tagging, the platform facilitates real-time decision-making, traceability, and content management. It stands out for its extensibility, as the core system can easily be updated to support new platforms, keywords, or data types, simply by modifying configuration files and detection logic.

In summary, this project not only demonstrates technical feasibility and innovation in automating GUI-based data retrieval but also opens up possibilities for secure, API-free automation in digital environments that demand control, compliance, and customizability.

FUTURE WORKS

The proposed system offers significant opportunities for enhancement and future development. One of the key improvements involves integrating machine learning and natural language processing (NLP) to intelligently classify and prioritize extracted email content. This would allow the system to automatically tag emails as transactional, promotional, or critical alerts, thereby increasing automation and decision support. In addition, expanding the OCR engine to support multiple languages will make the system more inclusive and adaptable for use in multilingual environments.

Another area of improvement is the development of a real-time monitoring dashboard, which would provide live status updates, visual analytics, and system logs, enabling administrators to track operations effectively. Implementing a multi-user authentication system would further secure the platform, offering role-based access for admins, operators, and viewers. Handling email attachments, such as images or PDFs, is another promising feature that could extend the utility of the system to more document-rich use cases.

Cross-platform compatibility is also a vital aspect, with future versions expected to support macOS, Linux, and possibly mobile platforms via emulators. While the current design is cloud-free for privacy reasons, optional cloud integration can be considered for backup and remote access functionalities. Enhancing fault tolerance through intelligent error detection and retry mechanisms will make the system more robust, especially in production environments.

Furthermore, the addition of visual keyword highlighting in the extracted content would improve manual review efficiency, particularly when dealing with large volumes of emails. Lastly, automating the execution process using task schedulers like cron or Windows Task Scheduler will allow the tool to operate on a fixed interval without manual triggering, making it fully autonomous. These future enhancements will collectively make the system more powerful, scalable, and intelligent.

REFERENCE:

1. Huang, J., & Li, Y. (2018). *OCR-based information extraction from web screenshots*. Journal of Data Science Applications, 6(2), 45–52.
2. Smith, A., & Patel, R. (2019). *Browser automation using coordinate-based interactions in Gmail*. International Journal of Web Engineering, 7(3), 121–129.
3. Zhang, M., & Zhao, F. (2020). *Structured email data insertion using PHP and MySQL*. Database Systems Journal, 9(1), 66–74.
4. Kumar, R., & Lee, S. (2021). *Python and PHP integration for automated email processing*. Automation Science and Engineering Reports, 12(2), 98–107.
5. Fernandes, D., & Nair, R. (2022). *AI-driven email classification with OCR and NLP techniques*. International Journal of Artificial Intelligence Research, 15(4), 210–219.
6. PyAutoGUI. (2024). *Cross-platform GUI automation for Python*. Retrieved from https://pyautogui.readthedocs.io/
7. Pyperclip. (2023). *Python clipboard interaction library*. Retrieved from https://pyperclip.readthedocs.io/
8. OpenCV. (2024). *Open Source Computer Vision Library*. Retrieved from https://opencv.org/
9. Tesseract OCR. (2023). *An open-source OCR engine by Google*. Retrieved from https://github.com/tesseract-ocr/tesseract
10. PHP Documentation. (2024). *PHP Manual for file handling and database integration*. Retrieved from https://www.php.net/manual/en/
11. XAMPP. (2024). *Apache distribution containing MySQL, PHP, and Perl*. Retrieved from https://www.apachefriends.org/
12. MySQL Documentation. (2024). *MySQL 8.0 Reference Manual*. Retrieved from https://dev.mysql.com/doc/
13. Sharma, K., & Roy, P. (2019). *Visual automation tools for non-API systems*. Proceedings of the International Conference on Intelligent Interfaces, 341–346.
14. Miller, T. (2020). *Automated information retrieval from GUI-based systems*. Data Engineering Review, 8(1), 58–63.

15. Tan, Y. (2021). *Email processing with real-time OCR systems*. Applied Computing Letters, 4(2), 31–37.

16. Dong, L., & Chen, H. (2019). *Handling dynamic web elements in email scraping*. Web Technologies Journal, 5(3), 119–126.

17. Rajan, R., & Ali, M. (2021). *Smart desktop automation for business workflows*. Journal of Emerging Technologies, 14(1), 92–101.

18. Johnson, M. (2022). *Designing database schemas for semi-structured email data*. International Journal of Data Modeling, 10(4), 202–211.

19. Gupta, N., & Thomas, A. (2023). *Security in local automation tools using PHP and hashed passwords*. Cybersecurity Trends, 6(1), 77–84.

20. AI News. (2023). *Combining OCR and NLP for smarter email automation*. Retrieved from https://ainews.com/ocr-nlp-email-integration