EE769 Course Project: Lunawa Elections

Dhruv Jain,* Simran Sinha,† and Sakshi Sonawane‡
Indian Institute of Technology Bombay

Analyzing paper-based election ballots poses unique challenges due to the variability in the marks' position, size, shape, rotation, and shade. These issues are compounded by scanner noise when digitizing ballots. This project addresses these issues by integrating Django for backend processing, Streamlit for real-time visualization, and an Android application for image capture and data upload. The Android app streamlines the process of collecting and sending ballot images to the server, while the Streamlit dashboard provides real-time vote counts and insights. The combined approach ensures accurate counting of valid votes, leading to more reliable and transparent election outcomes, ultimately reducing human errors.

I. INTRODUCTION

Elections are a cornerstone of democratic societies, providing a mechanism for citizens to choose their leaders and shape their governance. The methods used to conduct elections vary significantly across countries, with evolving technology playing a crucial role in the technical implementation of voting processes. Traditional systems like voting machines with mechanical gears are becoming obsolete due to high maintenance costs and difficulties in meeting accessibility requirements. Similarly, direct recording electronic (DRE) systems face criticism due to the lack of transparency and the inability to provide voters with direct evidence of their recorded vote. Meanwhile, manual vote counting, although reliable, is often too slow for modern election demands.

To address these challenges, many electoral systems are increasingly adopting optical scan technology for paper-based elections. Optical scan ballots offer a compromise between efficiency and accuracy, allowing for the automated processing of votes while maintaining a tangible record for verification. Voters mark their choices on paper ballots, typically by filling in ovals or connecting lines, which are then scanned to count the votes. However, this method is not without its own challenges. Marks that were not intended as votes, added either by the voter or during handling, can complicate the counting process. Moreover, a non-negligible percentage of voters fail to follow explicit instructions, leading to variations in the placement and shape of marks on the ballots.

The Lunawa Elections Project aims to address these challenges by facilitating paper-based offline elections with real-time vote counts through a digital dashboard. This project, developed for the NGO Lunawa Helping Hand in Mumbai, uses image processing and machine learning techniques to automatically process scanned ballot images and determine voting outcomes. The goal is to streamline the election process and reduce human errors, thereby making the voting process more efficient and transparent.

By employing advanced algorithms, the project can identify and analyze all marks on a ballot, regardless of their position or shape. This approach not only allows for accurate vote counting but also provides a robust system for ensuring the integrity of the election. With this system, the Lunawa Elections Project contributes to a more efficient and reliable election process, supporting democratic values and enhancing voter confidence.

II. BALLOT IMAGE PROCESSING FOR MARK DETECTION

The processing of ballot images is a critical component of automated election systems, requiring accuracy, robustness, and reliability. In this section, we describe the methodology used for processing scanned ballot images to detect and analyze voting marks. This approach involves a series of image processing techniques designed to extract meaningful information from scanned ballots, even in the presence of noise or other distortions.

- **Preprocessing and Thresholding**: The first step is to read the scanned ballot image which is then converted to grayscale. This conversion simplifies the analysis by reducing the data to a single intensity channel. Grayscale images are more manageable and allow for more efficient processing.

  Next, the grayscale image undergoes thresholding to create a binary image. Thresholding converts the image into a binary format, where pixels above a certain intensity threshold are set to white, and those below the threshold are set to black. This step is essential for distinguishing marked and unmarked areas on the ballot, as it creates a clear contrast between them.

- **Contour Detection**: With the binary image in hand, contour detection is used to identify the boundaries of significant regions within the image.

* 22m0828@iitb.ac.in
† 210260051@iitb.ac.in
‡ 210100150@iitb.ac.in

Contours represent the outlines of shapes or objects, allowing the identification of the outer edges of the ballot and any internal structures.

- **Perspective Transformaion**: Once the contours are detected, the largest contour is selected as it usually represents the boundary of the ballot itself. Using this contour, a quadrilateral (four-sided shape) is derived, representing the outer corners of the ballot. This information is used to correct the perspective of the image. Perspective transformation is applied to align and orient the ballot image, removing distortions caused by scanning or camera angles. This step is crucial to identify the corners of the ballot, enabling us to correct its perspective and ensure it is properly aligned. The goal is to have a standardized view of the ballot for further processing.

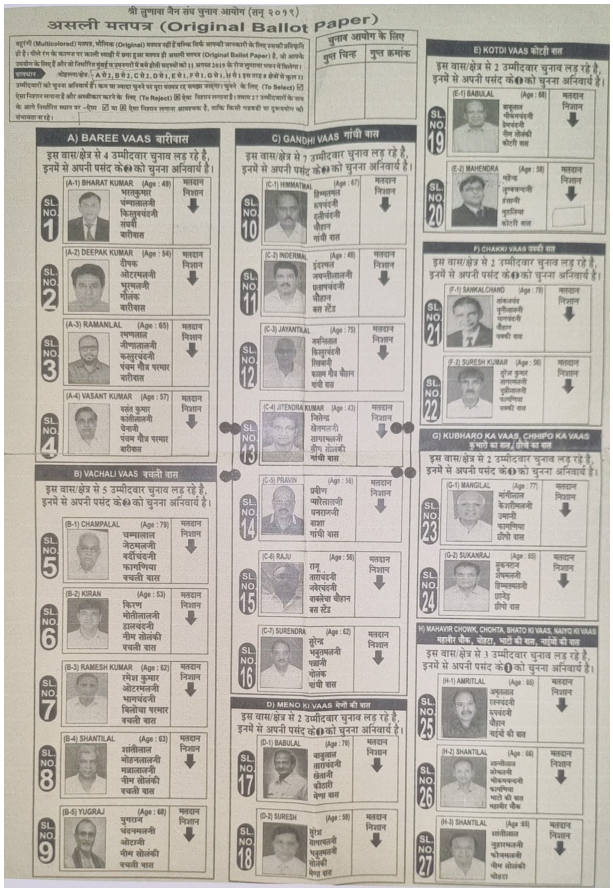

Fig1: Ballot paper

- **Structural Similarity**: After perspective correction, the transformed image is compared with a reference ballot using the Structural Similarity Index (SSIM). SSIM allows for a measure of similarity between two images, ensuring the transformed ballot matches the expected structure and orientation. If multiple possible orientations are detected, the one with the highest similarity to the reference is chosen.

- **Bounding Box and Mark Detection** Following this, predefined regions of interest (bounding boxes) are identified on the corrected ballot image. These bounding boxes represent areas where marks are expected to be found, such as voting selections or signatures. Within each bounding box, the algorithm checks for significant marks by assessing the darkness or density of black pixels. If the average darkness within a bounding box exceeds a certain threshold, the area is considered to contain a significant mark.

- **Visualization and Result Analysis**: The final step involves visualizing the processed image with bounding boxes drawn around detected marks. This visualization serves as a reference for manual verification and further analysis. It provides a clear indication of where the significant marks were detected on the ballot.

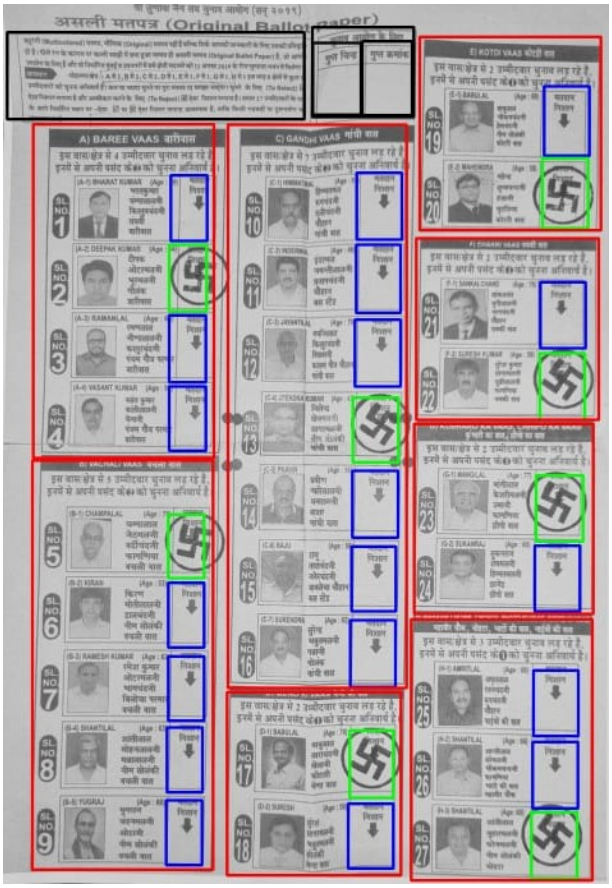

Fig2: Processed Scanned Ballot paper where the green bounding boxes are drawn around detected marks while the empty boxes are identified as blue boxes

This process of ballot image processing for mark detection allows for accurate identification of voting selections, even in the presence of scanner noise or other image distortions. By combining contour detection, perspective transformation, mark detection, and error handling, the system provides a robust approach to automated ballot

interpretation. The inclusion of SSIM-based checks and detailed color-coding for bounding boxes contributes to improved precision and visual clarity, enhancing the overall effectiveness of the system. The described methodology provides a robust approach to ballot analysis, enabling reliable and efficient interpretation of voting results.

III. HANDLING HTTP REQUESTS IN DJANGO

This section outlines the approach used in a Django-based web application to handle various HTTP requests for functionalities such as user authentication, image uploads, image retrieval, file deletion, and the integration of Streamlit for data visualization. The described methods utilize specific Django decorators for HTTP method restriction and CSRF (Cross-Site Request Forgery) exemption, along with proper exception handling and logging for error tracking and debugging.

- **HTTP Methods and CSRF Exemption**: The code snippet utilizes Django decorators to define HTTP methods allowed for each endpoint. The `@require_http_methods` decorator specifies whether a view accepts GET or POST requests, ensuring that only expected methods are processed. Additionally, the `@csrf_exempt` decorator allows the exemption of CSRF protection for specific endpoints, reducing constraints on incoming requests, which is helpful for APIs or non-browser clients.

- **User Authentication**: The 'auth' endpoint demonstrates a basic approach to user authentication using HTTP POST requests. The endpoint checks the received password against a predefined value (in this case, '0000'). If the password is correct, a success response is returned with a 200 status code. Otherwise, a failure response with a 401 status code is returned. The code employs exception handling to catch and log errors, ensuring robustness in the face of unexpected issues.

- **Image Uploads**: The upload endpoint handles the receipt of image files via POST requests. If an image file is present, it is stored in a specific directory (`settings.UPLOAD_ROOT`), and a corresponding database record is created using Django's ORM (Object-Relational Mapping). The endpoint returns a success response if the upload is successful or an appropriate failure response in case of errors, ensuring feedback to the client. Detailed logging is used to track each step of the upload process and record any exceptions.

- **Image Retrieval**: The `get_image` endpoint allows retrieval of images by name via HTTP GET requests. If the specified image exists on the server, it is returned with an appropriate content type (image/jpeg). If the image is missing or still being processed, relevant responses are provided, including a "Try again" message for processing images. This endpoint also implements error handling to manage unexpected scenarios and record errors in the log.

- **File Deletion**: The delete endpoint is designed to remove records and files associated with a specific Android device. It uses background threading to handle the deletion of physical files, preventing blocking on the main thread. If successful, a 200 status code is returned, while failure scenarios are appropriately logged and managed with a 401 status code response.

- **Streamlit Integration**: The streamlit endpoint is responsible for starting a Streamlit server for visualization. It initiates a background thread to run the Streamlit command, allowing asynchronous server startup. The endpoint constructs the Streamlit server's URL based on request metadata and provides it to the client through a rendered HTML page.

Fig3: Live Voting Dashboard which automatically updates every 30 seconds

IV. ANDROID APPLICATION ARCHITECTURE

The Lunawa Elections Project's frontend is an Android application designed to capture images of paper-based ballots and communicate with the backend server for processing. The app is built with a focus on ease of use, seamless integration with backend services, and robust user interaction. Below is an overview of the structure and functionalities of key Java classes in the Android application:

- `MainActivity.java`
  Purpose: MainActivity serves as the entry point for the Android application, managing the initial setup, user authentication, and navigation to other parts of the app.
  Features:

  - User Authentication: Handles user login processes, ensuring secure access to the application. The login process involves entering a password, typically "0000", to authenticate users.
  - Server URL Configuration: Allows the user to set or reset the server URL, enabling the app to point to different backend endpoints. This flexibility is crucial for deployment and testing.
  - Navigation: After a successful login, MainActivity redirects users to the CameraActivity, where they can capture images of ballots.

- `CameraActivity.java`
  Purpose: CameraActivity manages the camera functionality within the app, allowing users to take pictures and view the status of image processing.
  Features:
  - Camera Access and Permissions: Handles camera permissions and manages the capture of images upon user request. It ensures users have granted the necessary permissions to use the camera.
  - Image Upload: After capturing an image, CameraActivity sends it to the backend server for processing. The app uses Retrofit for HTTP communication, providing a smooth upload experience.
  - Session Management: Includes a logout feature that returns the user to MainActivity, promoting secure session handling.
  - Status Display: Shows the count of successfully processed images, providing feedback to the user.

- `RetrofitClient.java`
  Purpose: RetrofitClient provides a singleton instance of Retrofit for making HTTP requests to the backend server.
  Features:
  - HTTP Client Configuration: Configures and maintains a Retrofit client with logging capabilities, enabling efficient communication with backend services.
  - Dynamic Server URL: Supports updating the base URL at runtime, allowing the app to connect to different server instances. This flexibility is useful for development, testing, and production.

Additional Files: `activity_main.xml`, `activity_camera.xml`, `fragment_url.xml`
Purpose: These XML files define the layouts for MainActivity, CameraActivity, and a fragment for server URL configuration, respectively.
Usage: The XML files control the user interface, dictating the layout of elements such as buttons, text fields, and camera views. This structure ensures a user-friendly interface and intuitive interaction.

Fig4: Login screen of the Android App

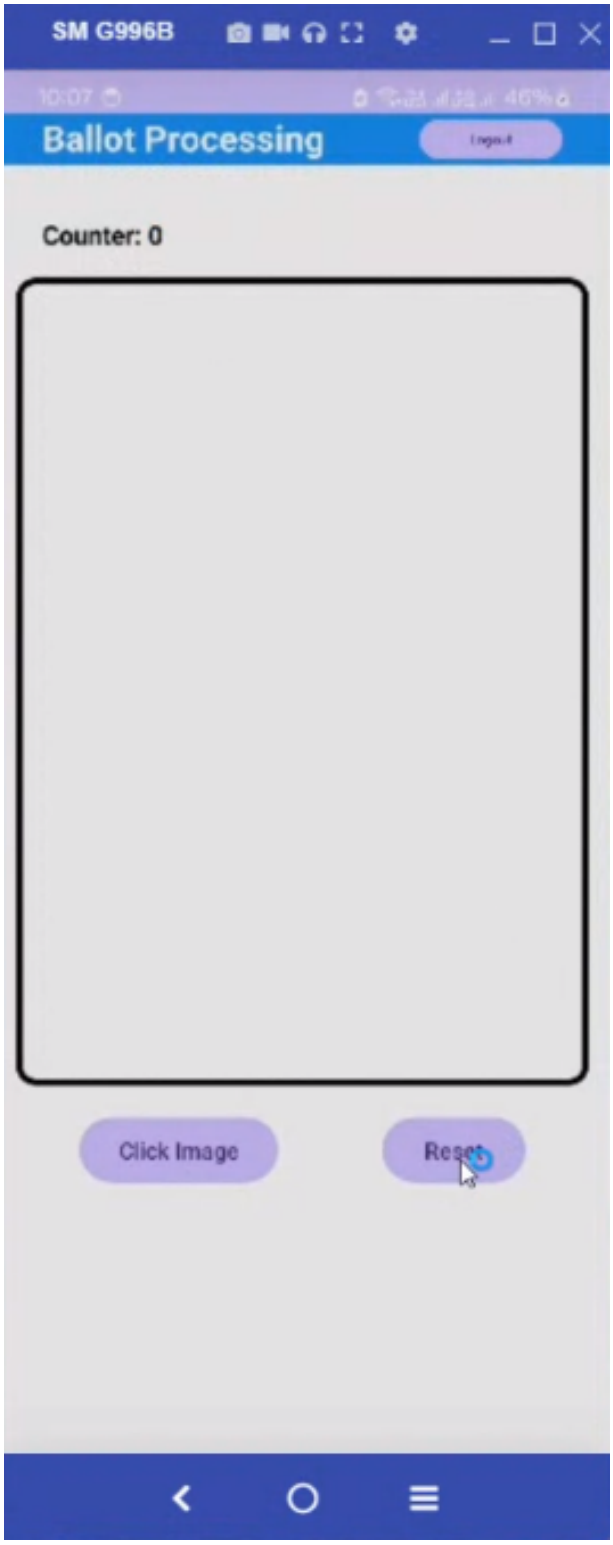

Fig4: Android App - Ballot Paper Capture Screen

V. FEATURES

The Lunawa Elections Project is designed with several features to streamline paper-based offline elections and provide real-time updates on vote counts. Below is an overview of the key features:

- **Real-time Election Vote Updates**: The system provides a live dashboard that displays the count of votes in real-time. This feature allows election organizers and observers to monitor the progress of the election process without delays, fostering transparency and reducing the likelihood of human errors in vote counting. The dashboard is built using Streamlit, enabling dynamic updates as votes are processed.
- **Automatic Ballot Processing**: To reduce manual work and expedite the vote counting process, the project includes an automated system that processes images of ballots to determine votes. The backend, built with Django, receives images from the Android application and uses predefined logic to identify and tally votes. This automation improves efficiency and accuracy in the election process.
- **Android Application**: An Android application is a core component of the system, allowing users to capture and submit images of ballots. The app is developed in Java with XML for user interface design. After capturing an image, the app automatically uploads it to the Django backend, where it's processed to determine the vote count. This approach replaces manual data entry with automated image-based processing.
- **Comprehensive Logging and Error Handling**: To ensure robustness, the project includes comprehensive logging and error handling mechanisms. Errors encountered during image processing, data storage, or other operations are logged for troubleshooting and debugging. This feature is crucial for maintaining system reliability and aiding in future enhancements or issue resolution.
- **Secure Authentication and Access Control**: To maintain security, the project implements basic authentication mechanisms for accessing the Android application and backend resources. Users need to enter a password to access the app and submit images, preventing unauthorized access to sensitive data. This feature ensures that only authorized personnel can interact with the system.
- **Easy Setup and Installation**: The project provides straightforward setup and installation procedures for both the backend and the Android application. Detailed instructions are included to help users quickly get the system up and running, whether for development, testing, or deployment purposes. This feature minimizes the barrier to entry for those implementing the system.
- **Flexible Deployment Options**: The system is designed to be flexible in terms of deployment. The

backend can be run on a local server for development or hosted on a cloud platform for broader access. The Android application can be installed on any compatible device, providing versatility for fieldwork or testing.

These features combine to create a robust and efficient system for conducting paper-based offline elections, with real-time updates and automated processing playing key roles in achieving the project's goals.

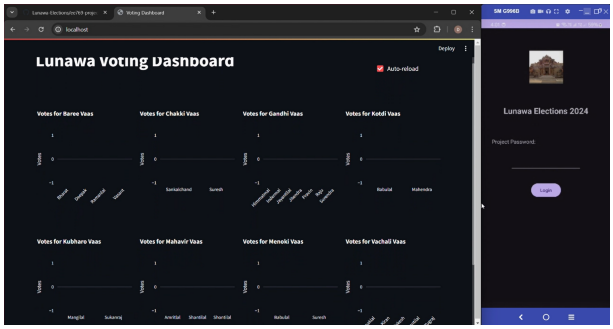

Fig5(a): Live Voting Dashboard

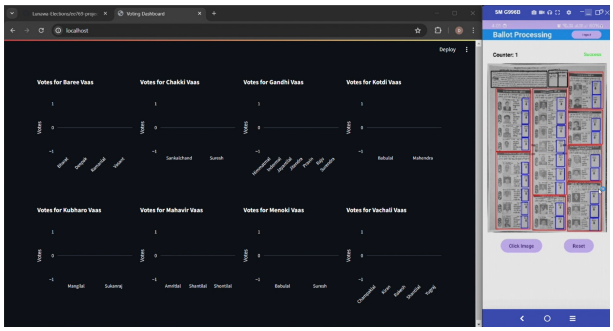

Fig5(b): Voting Dashboard - Initial Stage: beginning of the voting process, with scanned ballot paper with no votes.

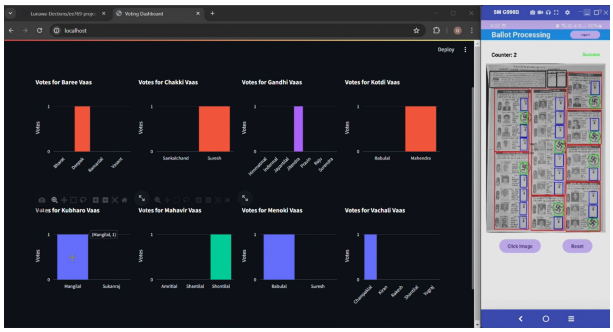

Fig5(c): Updated Voting Dashboard: point in the process where a single ballot paper with votes have been scanned, suggesting a progress update

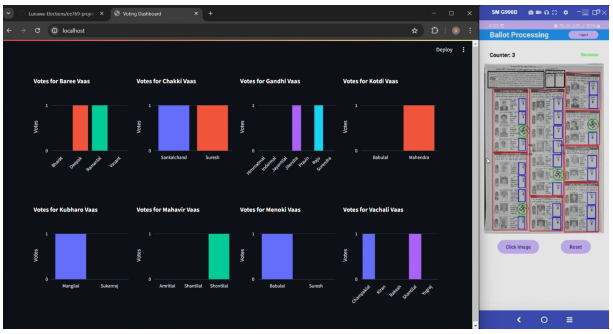

Fig5(d): Updated Voting Dashboard on further scanning the ballot papers

VI. CONCLUSION

The Lunawa Elections Project, designed for the NGO Lunawa Helping Hand in Mumbai, represents a significant step toward digitizing traditional paper-based offline elections. By implementing a streamlined process for counting votes and providing real-time updates through a dashboard, this project has not only increased efficiency but also reduced human error, ensuring a more transparent and reliable voting process.

The comprehensive system incorporates a robust backend built with Django, allowing secure and efficient image processing, while the frontend, created with Streamlit, offers an intuitive platform for visualizing voting results. The Android application, developed with Java, facilitates the seamless capture and upload of ballot images, enhancing the project's user-friendliness and versatility.

Throughout the implementation, key features such as real-time election vote updates, automatic ballot processing, and a straightforward setup and installation process have been achieved. The incorporation of various functionalities and endpoints ensures smooth interaction between the backend and frontend, allowing for scalable and maintainable operations.

Moreover, the detailed design of Django models, API endpoints, and views reinforces the system's reliability and scalability, enabling future enhancements and adaptations as needed. This thoughtful approach to development fosters an environment conducive to growth and innovation, positioning the Lunawa Elections Project as a template for other organizations seeking to digitize their election processes.

In summary, the Lunawa Elections Project exemplifies the successful integration of technology in facilitating offline elections. By combining the power of Django, Streamlit, and an Android application, the project has established a foundation for improved election processes. This successful implementation not only supports the NGO's mission but also serves as a blueprint for future projects in similar domains.

VII. CODE REPOSITORY

The codes developed for the project can be found here.

VIII. VIDEO DEMO

For a comprehensive demonstration of the Lunawa Elections project, watch the video demo via the following link: Video Demo.

IX. ACKNOWLEDGEMENT

We extend our heartfelt appreciation to our course instructor, Prof. Amit Sethi, for providing us with the opportunity to work on this project and guiding us throughout the process. We would also like to thank the teaching assistants of the course for their assistance in various aspects of the project. Their prompt response and willingness to help made the project progress smoother.