PROJECT REPORT

FLUXOR – "SWIRL INTO A NEW ORDER"

A Project report submitted in partial fulfilment of the

requirements for the award of the degree of

Bachelor of Technology (Hons.)

in

Computer Science and Engineering

By

ARCHI AGRAWAL – 10

ARJUN SINGH RAJPUT – 11

JATIN KHETAN – 21

Under the Mentorship of

SHIVANSHU UPADHYAY SIR

Department of Computer Engineering & Applications

Institute of Engineering & Technology

GLA University

Mathura- 281406, INDIA

DECEMBER, 2024

DECLARATION

We hereby declare that the work presented in the B.Tech. (H) project titled **"FLUXOR"**, submitted to the **Department of Computer Engineering and Applications, GLA University, Mathura**, in partial fulfilment of the requirements for the award of the **Bachelor of Technology (Honors)** in **Computer Science and Engineering**, is an authentic record of our own work.

This project has been carried out under the guidance of **Mr. Shivanshu Upadhyay** (Technical Trainer). We affirm that this work is original and has not been submitted elsewhere for any other degree or diploma. All external contributions and references have been duly acknowledged.

SIGN:______________

NAME:

UNIVERSITY ROLL NO.:

SIGN:______________

NAME:

UNIVERSITY ROLL NO.:

SIGN:______________

NAME:

UNIVERSITY ROLL NO.:

CERTIFICATE

This is to certify that the project report titled **"FLUXOR"**, submitted by **TEAM FLUXOR** from the **CSE Department, GLA University**, is a record of the bonafide work carried out by the team during the academic year **2024-25**. The project was undertaken as part of the academic curriculum and demonstrates the team's deep commitment and dedication toward solving real-world problems in the domain of **AI-powered file management systems**.

The project **FLUXOR** aims to provide an intelligent, efficient, and user-friendly solution for managing and organizing digital files through advanced features such as **content-based classification**, **batch renaming**, **file grouping by extensions**, and an interactive **AI chatbot** for user assistance. The project work has been carried out under the guidance and supervision of **Mr. Shivanshu Upadhyay Sir** and has met all the objectives, specifications, and requirements as per the project evaluation criteria.

The team has shown exceptional technical expertise, creativity, and problem-solving abilities in designing and implementing **Fluxor**, demonstrating a thorough understanding of the core concepts and tools used in this project. The work completed is of high academic value and serves as an exemplary model of innovation and practical application in the field of **Computer Science and Engineering**.

We are pleased to acknowledge the successful completion of this project, which has been executed to our satisfaction and fulfils the academic requirements for the project evaluation.

Supervisor Signature: ________________

Name: MR. SHIVANSHU UPADHYAY SIR

Designation: TECHNICAL TRAINER

ACKNOWLEDGMENT

We would like to express our sincere gratitude to several individuals who have been instrumental in the successful completion of this project:

- First and foremost, We would like to thank my **project supervisor**, **Mr. Shivanshu Upadhyay Sir**, for his continuous guidance, unwavering support, and invaluable insights throughout the project. His expertise and encouragement were key to overcoming challenges and ensuring the project's success.
- We would also like to extend my heartfelt thanks to the **faculty members** of the **CSE Department**, whose technical support, advice, and feedback have contributed significantly to the overall development of this project.
- Special thanks to our **colleagues and classmates** for their constructive feedback, collaboration, and assistance throughout the project. Their suggestions and perspectives were invaluable.
- Lastly, We are deeply grateful to our **family and friends** for their constant encouragement, understanding, and emotional support, which kept me motivated and focused during the course of this project.

This project could not have been completed without the contributions and support of all these individuals.

TEAM FLUXOR

ABSTRACT

FLUXOR is an intelligent file management system that aims to transform how users organize and interact with digital files. Leveraging the power of artificial intelligence and machine learning, FLUXOR combines traditional file operations with advanced AI-driven features.

The project's key objectives are to develop a comprehensive file management application for the Windows operating system, implement intelligent algorithms for optimal file organization, and create a user-friendly adaptive interface. FLUXOR integrates cloud services, supports both online and offline modes, and implements secure work and personal modes to enhance productivity and privacy.

The proposed system includes an Intelligent File Organization Engine, AI-Driven Content Analysis, Adaptive User Interface, Secure Multi-Mode Operation, and a Smart Assistant with natural language query capabilities. Key features include intelligent file renaming, content-based sorting, advanced image classification, and a file recommendation system.

FLUXOR is being developed using Python, JavaScript, and frameworks like PySide/PyQt and TensorFlow/PyTorch. The project follows an Agile methodology with two-week sprints and comprehensive testing. The expected outcomes include a functional Windows application, implemented AI models, seamless online/offline integration, a user-friendly interface, and detailed documentation.

TABLE OF CONTENTS

1. Introduction

    1.1 Overview and Motivation

    1.2 Objectives

    1.3 Summary of Similar Applications

    1.4 Organization of the Project

2. Software Requirement Analysis

    2.1 Technical Feasibility

    2.2 Functional Requirements

    2.3 Non-Functional Requirements

3. Software Design

    3.1 System Architecture

    3.2 Design Methodology

    3.3 Detailed Design

4. Implementation and User Interface

    4.1 Development Environment

    4.2 Implementation Details

    4.3 User Interface Design

    4.4 Key Features

5. Software Testing

    5.1 Testing Methodology

    5.2 Test Cases

    5.3 Test Results

    5.4 Performance Analysis

6. Conclusion

    6.1 Project Outcomes

    6.2 Limitations

    6.3 Future Scope

INTRODUCTION

1.1 Overview and Motivation

Fluxor is an innovative Windows-based file management application designed to simplify and enhance the organization of digital files. Built using Python with the PySide/PyQt framework, Fluxor caters to the needs of users overwhelmed by the complexity and volume of their files. Its smart features allow users to classify, rename, and group files with ease, reducing manual effort and saving time.

The application's core is its ability to leverage user input and automated techniques for file management. Fluxor's user-centric design ensures adaptability for diverse use cases, from personal file organization to professional data management.

Motivation Behind Fluxor

The exponential growth of digital content has made file organization a critical challenge for users across industries. Whether it’s sorting photos, managing project files, or organizing downloads, users often face:

1. **Time Constraints:** Manually sorting and naming files is tedious and time-consuming.
2. **Errors in Manual Sorting:** File misplacements and inconsistent naming conventions can cause confusion.
3. **Lack of Automation:** Existing file management solutions are often limited in customization and require significant manual intervention.

Fluxor was born out of the need for a **smart, efficient, and user-friendly solution** that addresses these challenges. By combining automation, user interactivity, and AI-powered capabilities, Fluxor enables users to:

- Reduce clutter by categorizing files based on their content and extensions.
- Maintain a consistent naming system for better organization.
- Access files more efficiently, thanks to intelligent grouping and classification.
- Use a chatbot interface for enhanced usability and support.

Fluxor not only addresses the functional aspects of file management but also enhances the user experience, making it an indispensable tool in the digital era.

1.2 Objectives

The primary goal of Fluxor is to provide a seamless and intelligent file management experience that empowers users to organize their digital spaces effortlessly. The following objectives guide the development and functionality of the application:

1. Enhance File Organization Through Automation

- Automate the classification of files based on their content, metadata, and extensions.
- Provide dynamic categorization options that adapt to user-defined rules and preferences.

2. Simplify File Renaming Processes

- Enable batch renaming of directory files with intuitive rules and patterns.
- Ensure consistent and meaningful file naming conventions that improve accessibility.

3. Deliver a User-Friendly Chatbot Experience

- Offer an interactive chatbot feature to guide users through complex file management tasks.
- Address user queries, suggest file organization methods, and streamline operations through natural language commands.

4. Facilitate Grouping Based on File Extensions

- Provide tools to group files with similar extensions for quick sorting and processing.
- Support bulk operations on grouped files, such as moving, copying, or deleting.

5. Improve Productivity and Efficiency

- Minimize manual effort required for sorting and renaming files.
- Save time with robust automation and user-defined workflows.

6. Enhance Digital Workflows

- Integrate Fluxor into existing workflows for seamless file handling.
- Encourage better digital hygiene by reducing clutter and improving access to relevant files.

1.3 Summary of Similar Applications

Fluxor, as an AI-powered file manager, operates in a domain where several other applications have been developed to aid file organization and management. Here's a summary of similar applications currently available and how they compare with Fluxor:

1. FileBot

- **Overview:** FileBot is a popular tool primarily used for renaming and organizing media files like movies and TV shows. It retrieves metadata and renames files automatically.
- **Strengths:**
  - Excellent for media files with automated metadata fetching.
- **Limitations:**
  - Limited to media files and lacks a generalized file organization capability.

2. Duplicate Cleaner

- **Overview:** This application helps identify and remove duplicate files across directories, making it ideal for decluttering storage.
- **Strengths:**
  - Efficient at finding and managing duplicates.
- **Limitations:**
  - Focused only on duplicate detection; no support for classification or grouping.

SOFTWARE REQUIREMENT ANALYSIS

Fluxor is structured to provide modularity, scalability, and ease of development. The project is divided into distinct components to ensure seamless functionality and maintainability. Below is a detailed outline of the project's organization:

1. User Interface (UI)

The user interface is designed with PySide/PyQt, focusing on simplicity and interactivity.

- **Features:**
  - Intuitive layout with navigation panels for various features (Classification, Renaming, Chatbot, Grouping).
  - Drag-and-drop functionality for file input.
  - Interactive chatbot window for natural language queries.
- **Files:**
  - main_window.py: Controls the primary application window.
  - ui_design.ui: Contains the layout designed using PyQt Designer.

2. Core Functionalities

Fluxor's functionality revolves around four primary modules, each addressing a critical aspect of file management. These modules are implemented independently to ensure easy updates, seamless integration, and scalability. Each module is designed with a focus on automation, efficiency, and user interactivity.

a. Classification Module

Description:

The Classification Module leverages AI to analyze the content of files and categorize them accordingly. Unlike conventional methods that rely solely on file names or extensions, this module performs content-based analysis using **Natural Language Processing (NLP)** and **Machine Learning (ML)** algorithms.

This feature prompts users for feedback or additional inputs to refine its categorization rules, ensuring high accuracy and adaptability. The classification logic can be customized to meet user-specific needs, such as creating specialized categories or adjusting criteria based on file attributes.

Core Functionalities:

1. **Content Analysis:**
    - Extracts textual content from supported file formats (e.g., .txt, .docx, .pdf).
    - Analyzes keywords, phrases, and context to determine appropriate categories.
2. **User Prompts:**
    - Requests user input for ambiguous classifications.
    - Provides suggestions for categories based on past user preferences or AI predictions.
3. **Dynamic Rule Adaptation:**
    - Learns from user feedback and refines classification rules for improved accuracy over time.
    - Supports the creation of custom classification categories.

Files:

1. **classifier.py**
    - Implements the core logic for file content analysis.
    - Uses pre-trained NLP models for text extraction and classification.
2. **prompt_handler.py**
    - Handles user interactions during the classification process.
    - Integrates feedback to update classification rules dynamically.

Example Workflow:

1. A user uploads a folder containing various files (e.g., reports, invoices, contracts).
2. The module analyzes each file's content and assigns it to predefined categories (e.g., "Finance," "Legal," "Personal").
3. For uncertain classifications, it prompts the user to confirm or select a category.
4. The categorized files are stored in organized directories or virtual folders.

b. Renaming Module

Description:

The Renaming Module provides an efficient way to batch rename files within a directory. This is particularly useful for maintaining consistent naming conventions, improving file organization, and ensuring accessibility. The module supports user-defined rules for renaming, which can be based on metadata (e.g., date created, file type) or specific patterns.

Core Functionalities:

1. **Batch Renaming:**
    - Processes multiple files simultaneously, applying a single renaming pattern across all selected files.
2. **Pattern Recognition:**
    - Identifies existing patterns in file names and applies consistent modifications.
3. **Metadata Integration:**
    - Uses file metadata (e.g., timestamps, authors, extensions) to create meaningful file names.
4. **Error Handling:**
    - Validates new file names to avoid duplicates or conflicts in the directory.

Files:

1. **renamer.py**
    - Contains the core logic for file renaming.
    - Processes patterns and applies user-defined rules.

Example Workflow:

1. A user selects a folder with unorganized file names like "Document1.pdf," "Scan_2023.jpg."
2. The module prompts the user to define a renaming rule (e.g., "<Category><*Date*><Index>").
3. Files are renamed to follow the new pattern (e.g., "Invoice_2023-12-07_001.pdf").

c. Chatbot Module

Description:

The Chatbot Module introduces an interactive, AI-powered assistant that helps users navigate Fluxor's features. It simplifies complex operations by enabling users to issue natural language commands instead of manually configuring options. The chatbot acts as both a guide and a command executor, bridging the gap between users and Fluxor's functionalities.

Core Functionalities:

1. **Natural Language Understanding (NLU):**
    - Interprets user commands such as "Classify all files in this folder" or "Rename files based on date."
2. **Interactive Assistance:**
    - Answers user queries about Fluxor's features and functionalities.
    - Provides real-time guidance during tasks (e.g., suggesting renaming patterns).
3. **Integration with Core Modules:**
    - Executes tasks by invoking the relevant modules (classification, renaming, grouping).
4. **Customizable Commands:**
    - Supports predefined commands as well as user-defined shortcuts for frequent operations.

Files:

1. **chatbot.py**
    - Implements the chatbot interface and logic.
    - Handles command parsing and intent recognition.
2. **response_generator.py**
    - Generates intelligent responses based on user queries.
    - Interfaces with other modules to perform requested actions.

Example Workflow:

1. A user types, "Group all files by type and rename them with today's date."
2. The chatbot interprets the request and executes the grouping and renaming tasks.

d. Grouping Module

Description:

The Grouping Module organizes files into logical groups based on extensions or other criteria. This feature is especially useful for users dealing with diverse file collections, as it helps reduce clutter and streamline access to related files.

Core Functionalities:

1. **Extension-Based Grouping:**
    - Automatically categorizes files by their extensions (e.g., .pdf, .docx, .jpg).
2. **Custom Criteria:**
    - Allows users to define grouping rules (e.g., file size, creation date).
3. **Bulk Actions:**
    - Supports batch operations on grouped files, such as moving them to folders or applying additional classification.

Files:

1. **grouping.py**
    - Contains the logic for grouping files based on extensions or custom criteria.
2. **file_operations.py**
    - Handles file movement, folder creation, and other operations required for grouping.

Example Workflow:

1. A user selects a folder with various file types.
2. The module groups files into subfolders like “Images,” “Documents,” “Videos.”
3. The grouped files are displayed in the UI, with options for further actions (e.g., renaming, classification).

3. Backend and AI Integration

- **Description:** The backend manages data processing and integrates AI models for intelligent file management.
- **Files:**
    - ai_models/: Directory containing pre-trained models for NLP and classification tasks.
    - data_handler.py: Handles data input/output and ensures compatibility with different file formats.
    - config.py: Central configuration file for managing parameters, file paths, and settings.

4. Database and Storage

- **Description:** Stores user preferences, classification logs, and chatbot interactions.
- **Files:**
    - database_handler.py: Manages SQLite or other lightweight databases for saving data.
    - logs/: Directory for storing log files for debugging and user feedback.

5. Utilities and Helper Functions

- **Description:** Provides reusable functions and tools for file handling and application support.
- **Files:**
    - utils.py: Contains utility functions for file validation, error handling, and miscellaneous tasks.
    - file_validator.py: Ensures files are compatible with Fluxor's features.

6. Documentation and Guides

- **Description:** Comprehensive documentation to assist developers and users.
- **Files:**
    - README.md: Overview of the project, installation instructions, and usage guidelines.
    - docs/: Directory for detailed documentation, FAQs, and troubleshooting guides.

7. Testing and Debugging

- **Description:** Ensures the robustness and reliability of the application.
- **Files:**
    - test_suite.py: Automated test cases for each module.
    - debug_tools.py: Tools for identifying and resolving issues.

8. Deployment and Packaging

- **Description:** Facilitates the distribution of Fluxor as a standalone Windows application.
- **Files:**
    - setup.py: Script for building and packaging the application.
    - requirements.txt: Lists Python dependencies.
    - build/: Directory for packaged application files.

Project Flow

1. **User Interaction:**
    - Users interact with the application through the UI or chatbot.
2. **Backend Processing:**
    - User commands trigger corresponding modules (classification, renaming, grouping).
3. **Output Delivery:**
    - Results (e.g., classified files, renamed directories) are displayed in the UI or exported to the file system.

SOFTWARE DESIGN

The software design of **Fluxor** is centered on modularity, scalability, and user-centric functionality. It follows a layered architecture to separate the core logic, user interface, and AI-driven capabilities for easier development and maintenance. Below is a detailed breakdown of the software design:

1. Architectural Overview

Fluxor adopts a **three-layered architecture**, as shown below:

a. Presentation Layer (Frontend)

- **Purpose:** Provides the user interface (UI) for interacting with the application.
- **Components:**
- PySide/PyQt-based GUI for file input, output, and interactive features.
- A chatbot window for natural language interaction.
- **Key Modules:**
- main_window.py
- ui_design.ui

b. Business Logic Layer (Core Functionality)

- **Purpose:** Processes user requests and implements core features (classification, renaming, grouping, chatbot).
- **Components:**
- Classification logic for content-based file organization.
- File renaming and grouping based on extensions.
- AI-driven chatbot for user assistance.
- **Key Modules:**
- classifier.py, renamer.py, grouping.py, chatbot.py.

c. Data Layer (Backend)

- **Purpose:** Manages data storage, retrieval, and AI model integration.
- **Components:**
- Database for logs, preferences, and interactions.
- **Key Modules:**
- data_handler.py, database_handler.py.

2. Module Design

a. Classification Module

- **Design:**
- Incorporates user feedback to refine classification accuracy.
- **Inputs:** File content and user prompts.
- **Outputs:** Categorized files into virtual or physical folders.

b. Renaming Module

- **Design:**
- Implements batch renaming based on user-defined rules (e.g., date, metadata, patterns).
- Validates file names to avoid duplication or conflicts.
- **Inputs:** Directory path and renaming rules.
- **Outputs:** Renamed files with consistent conventions.

c. Grouping Module

- **Design:**
- Groups files by extensions or custom user-defined criteria.
- Supports bulk actions like moving or copying grouped files.
- **Inputs:** List of files and grouping preferences.
- **Outputs:** Grouped folders or categories.

d. Chatbot Module

- **Design:**
- Acts as a bridge between users and the core modules.
- **Inputs:** User commands or queries in text.
- **Outputs:** Executed actions or responses to user queries.

4. Key Algorithms

a. Classification Algorithm

- Combines content-based analysis with user-defined rules.

- Steps:

1. Extract file content using parsers (e.g., text, PDF).
2. Apply NLP models to classify based on predefined categories.
3. Prompt user for feedback to refine the classification.

b. Renaming Algorithm

- Uses pattern matching and metadata extraction for consistent naming.
- Steps:

1. Parse existing file names and extract metadata.
2. Apply user-defined rules or templates.
3. Validate and rename files in batches.

c. Grouping Algorithm

- Groups files by analyzing extensions or custom tags.
- Steps:

1. Read file extensions or other attributes.
2. Sort files into categories.
3. Perform grouping actions (e.g., moving files into folders).

5. Data Flow

a. User Interaction:

1. User interacts with the UI or chatbot.
2. Inputs are processed by the controller layer.

b. Core Processing:

1. Input is forwarded to the relevant module (classification, renaming, grouping).
2. Modules interact with AI models or backend storage as needed.

c. Output Delivery:

1. Processed results are sent back to the UI.
2. Results are displayed or executed (e.g., renamed files, grouped folders).

6. Security and Error Handling

a. Security Measures:

- Prevent unauthorized access to sensitive files through user permissions.
- Validate user inputs to avoid harmful commands.

b. Error Handling:

- Provide clear error messages for invalid operations.
- Log errors for debugging purposes.

7. Future Scalability

- **Support for More File Types:** Extend classification and grouping for media, archives, and custom file formats.
- **Cloud Integration:** Add support for cloud storage platforms like Google Drive or OneDrive.
- **Advanced Chatbot Capabilities:** Improve chatbot interactions using GPT-based models for more conversational responses.

IMPLEMENTATION AND USER INTERFACE

The implementation of Fluxor focuses on integrating artificial intelligence, machine learning, and efficient file management features into an intuitive and user-friendly Graphical User Interface (GUI). The entire system is built using Python, with the help of libraries like PySide2/PyQt5 for the UI, scikit-learn and spaCy for AI tasks, and os and shutil for file operations.

This section provides an in-depth view of both the implementation process and the user interface design of Fluxor.

1. Implementation of Core Functionalities

a. Classification Module

- Content Extraction & NLP Models:
  The module leverages NLP models from spaCy and scikit-learn to process file contents. Pre-trained models help with text classification, where custom rules or categories are applied based on user prompts.
- Implementation Flow:

1. Content Extraction: Files (e.g., .txt, .pdf) are parsed using appropriate parsers.
2. Preprocessing: Text data is cleaned (e.g., removing stop words, stemming).
3. Model Prediction: The cleaned text is classified using a machine learning model trained on labeled data (e.g., reports, invoices).
4. User Prompt: If the classification is uncertain, the user is prompted for input to improve accuracy.
5. Key Files:

- classifier.py: Implements classification logic using AI models.
- prompt_handler.py: Manages user interaction for classification adjustments.

b. Renaming Module

- Pattern-Based Renaming:
  The renaming module allows batch renaming by applying rules based on patterns, metadata, or user preferences. It uses Python's regular expressions (regex) to find and replace specific parts of the file names.
- Implementation Flow:

1. User-Defined Rule Input: Users specify renaming patterns or choose from pre-configured templates.
2. Renaming Logic: The system applies the specified pattern, validates against naming conflicts, and renames the files.
3. Error Handling: The module handles errors like conflicting names or invalid characters in file names.
4. Key Files:

- renamer.py: Core renaming logic using regex and metadata.
- rule_parser.py: Interprets and parses the user-defined renaming rules.

c. Chatbot Module

- Natural Language Understanding:
  The chatbot module is implemented using a combination of NLU (Natural Language Understanding) and predefined command mappings. The system recognizes commands such as "Rename files by date" or "Group all images," processes them, and interacts with the backend.
- Implementation Flow:

1. User Command Input: The user types a command or query in natural language.
2. Intent Recognition: The chatbot parses the input using a pre-trained NLU model to understand the intent.
3. Action Execution: Based on the recognized intent, the chatbot triggers the relevant module (classification, renaming, etc.).
4. Key Files:

- chatbot.py: Implements chatbot logic and interfaces with backend functionalities.
- response_generator.py: Handles response generation and context management for user interactions.

d. Grouping Module

- Extension-Based Grouping:
  The grouping module organizes files by their extensions (e.g., .pdf, .jpg, .mp3). It enables users to manage their files better by categorizing them into meaningful groups.

- Implementation Flow:

1. File Analysis: The system scans files in the selected folder and categorizes them based on extensions.
2. Grouping Logic: Files are grouped into virtual or physical folders (e.g., all PDFs are moved to a "Documents" folder).
3. User Customization: Users can adjust grouping criteria if desired (e.g., group by size or creation date).
4. Key Files:

- grouping.py: Contains logic for sorting files by type.
- file_operations.py: Handles file movements and folder creation.

2. User Interface (UI) Design

The Fluxor GUI is designed to be clean, intuitive, and user-friendly, with a focus on ensuring that users can access key functionalities with minimal clicks.

a. UI Framework:

- The PySide2/PyQt5 library is used to create a native, cross-platform application that ensures a responsive and visually appealing interface.
- The GUI includes the following elements:
    - Main Window: Displays the file management interface, where users can drag-and-drop files, select directories, and view results of operations.
    - Sidebar: Offers quick access to core functionalities like classification, renaming, and grouping.
    - Tabs: Allow the user to easily switch between different modules (classification, renaming, chatbot, etc.).
    - Progress Indicators: Show real-time progress for tasks like classification, renaming, and grouping.

b. Components of the GUI:

1. Main Dashboard:
    - File Input Area: A section where users can select files or folders to apply the functionalities to.
    - Operation Panels: For each core feature, a corresponding panel allows users to configure parameters (e.g., classification categories, renaming patterns).

- File Output Area: Displays the results of operations (e.g., classified files, renamed files).

2. Tabs for Core Functionalities:

   - Classification Tab: Displays a list of files with suggested categories. Users can refine classifications and confirm or modify suggestions.
   - Renaming Tab: Provides an interface to input renaming rules, preview renamed files, and execute bulk renaming.
   - Grouping Tab: Displays files organized by extensions or user-defined criteria. Allows for additional actions like moving or copying files to grouped folders.
   - Chatbot Tab: A chat window where users can enter natural language commands and receive real-time assistance.

3. Interactive Elements:

   - Buttons: For triggering specific actions (e.g., "Classify Now," "Rename Files," "Start Grouping").
   - Dropdown Menus: For selecting options such as file types, categories, or renaming patterns.
   - Text Fields: For inputting file paths, renaming patterns, or custom grouping criteria.
   - Progress Bars: Indicate ongoing operations for file classification, renaming, and grouping.

4. Notification System:

   - Status Messages: Inform users about the progress or completion of operations.
   - Error Alerts: Display user-friendly messages when an error occurs (e.g., invalid file name during renaming).

c. User Flow Example:

1. Start Fluxor: The user launches Fluxor and is presented with the main window.
2. Select Files: The user drags and drops files into the file input area.
3. Choose Operation: The user selects one of the tabs (e.g., Classification, Renaming, Grouping).
4. Interact with the Module:

   - In the Classification Tab, Fluxor automatically analyzes files and assigns categories.
   - In the Renaming Tab, the user selects a renaming pattern, and Fluxor previews the changes.

3. Integration and Testing

a. Integration:

Fluxor's modules are designed to function both independently and together. The system is integrated to allow seamless interaction between modules. For example, after classifying files, the user can easily switch to the Grouping Tab to organize classified files.

b. Testing:

- Unit Testing: Individual modules (e.g., classifier, renamer) are tested to ensure proper functionality.
- Integration Testing: Ensures the modules work together smoothly, allowing users to switch between classification, renaming, and grouping tasks without issues.
- UI Testing: Verifies that the GUI is intuitive and that all user interactions trigger the expected results.

SOFTWARE TESTING

Software testing is a critical step in ensuring that Fluxor functions as intended and delivers a reliable, bug-free user experience. Fluxor's core functionalities, which include classification, renaming, grouping, and chatbot interactions, require rigorous testing to verify their correctness, performance, and integration. This section outlines the types of testing conducted, testing methodologies, and specific test cases for Fluxor.

1. Types of Software Testing for Fluxor

a. Unit Testing

Unit testing focuses on testing individual components of Fluxor to verify that each part of the application works as expected in isolation. Each module in Fluxor (e.g., classification, renaming, grouping, and chatbot) is tested independently to ensure that it performs its assigned function correctly.

- Key Tools:
    - pytest: A popular testing framework for Python that helps in writing simple and scalable tests.
    - unittest: Python’s built-in unit testing framework, providing tools to create and run test cases.
- Modules to be Tested:

1. Classification Module:

- Test case to check whether files are classified correctly based on content.
- Test case to simulate user inputs for refining classification.

2. Renaming Module:

- Test case to verify that files are renamed according to specified rules.
- Test case for error handling when file name conflicts occur.

3. Grouping Module:

- Test case to ensure files are grouped accurately based on extensions or user-defined criteria.

4. Chatbot Module:

- Test case to check chatbot’s ability to recognize and act on commands (e.g., "Group all PDFs," "Rename files by date").

b. Integration Testing

Integration testing focuses on ensuring that different modules within Fluxor work together seamlessly. Given that Fluxor includes multiple interconnected modules, it is crucial to verify that data and operations flow correctly between them.

- Key Areas to Test:

1. Classification and Grouping: After classifying files, test whether the files are automatically grouped into the correct folder or category.
2. Renaming and Grouping: After renaming files, test if the grouped folders reflect the renamed files correctly.
3. Chatbot Integration: Test the chatbot's ability to trigger the correct backend operations when users provide commands (e.g., initiating classification or renaming tasks).

c. Functional Testing

Functional testing ensures that Fluxor's core features (classification, renaming, grouping, and chatbot) perform as expected when interacting with the user. Each feature is tested in a real-world environment with the goal of identifying defects in functionality or design.

- Test Scenarios:
    - Classification Test: Upload a variety of files (e.g., reports, invoices, images) and verify that the classification is accurate.
    - Renaming Test: Choose a set of files with non-standard names and verify that renaming rules are applied correctly (e.g., renaming files based on creation date).
    - Grouping Test: Test if files of various types (e.g., .pdf, .jpg, .mp3) are grouped correctly into folders.
    - Chatbot Test: Interact with the chatbot using natural language queries and verify if it responds appropriately.

d. Performance Testing

Performance testing ensures that Fluxor can handle a large number of files and complex tasks without performance degradation. This type of testing is particularly important when Fluxor is used with large file collections.

- Key Aspects to Test:

1. Load Testing: Simulate the uploading and processing of thousands of files to measure how Fluxor performs under heavy load.

2. Response Time Testing: Measure the response times for operations such as file classification, renaming, and grouping to ensure tasks are completed within acceptable time limits.
3. Memory Usage: Ensure that Fluxor does not consume excessive memory during long-running tasks.

e. User Interface (UI) Testing

UI testing focuses on verifying that Fluxor's graphical interface is intuitive, responsive, and free of visual bugs. It ensures that all UI elements, such as buttons, sliders, text fields, and progress bars, work as expected and provide appropriate feedback to the user.

- Key Areas to Test:

1. Functionality of Buttons: Test whether the buttons (e.g., "Classify Now," "Rename Files," "Start Grouping") trigger the expected actions.
2. Correct Display of Files: Ensure files are correctly displayed in the output area after classification, renaming, or grouping.
3. Responsive Design: Test whether the UI adapts correctly to different screen sizes and resolutions.

- Test Scenarios:

1. Ease of Navigation: Test whether users can quickly navigate between different tabs (classification, renaming, grouping) and complete tasks efficiently.
2. User Feedback Mechanisms: Evaluate whether the system provides clear and helpful feedback during each task (e.g., progress bars, error messages).
3. Chatbot Effectiveness: Assess whether the chatbot's responses are helpful and whether it provides the necessary guidance to users.

2. Specific Test Cases for Fluxor

Below are some specific test cases for Fluxor's core modules:

a. Test Case 1: File Classification

Objective: Verify that Fluxor classifies a file based on its content.

- Steps:

1. Upload a .pdf report file containing financial data.
2. Wait for Fluxor to classify the file based on its content.

3. Confirm the file is placed in the "Finance" folder.
4. Check that a user prompt is displayed for ambiguous content.

- Expected Outcome:
  The file is correctly classified into the “Finance” category, and the system prompts the user if the classification is uncertain.

b. Test Case 2: Batch Renaming

Objective: Ensure that Fluxor can batch rename files based on user-defined patterns.

- Steps:

1. Select multiple files with inconsistent names (e.g., "Doc1.pdf," "Scan_2023.jpg").
2. Specify the renaming pattern as "Category_<Date>_<Index>."
3. Confirm the renaming operation.

- Expected Outcome:
  All files are renamed according to the specified pattern, such as “Invoice_2023-12-07_001.pdf.”

c. Test Case 3: File Grouping by Extension

Objective: Verify that Fluxor groups files by their extensions.

- Steps:

1. Upload a folder with mixed file types (e.g., .pdf, .jpg, .mp3).
2. Choose the option to group files by extension.
3. Check that the files are placed into respective folders: “Documents,” “Images,” and “Audio.”

- Expected Outcome:
  The files are correctly grouped based on their extensions, with no errors.

d. Test Case 4: Chatbot Command Execution

Objective: Test if the chatbot can recognize and execute a user command.

- Steps:

1. Open the chatbot interface.
2. Type the command "Group all PDFs."
3. Wait for the system to group all .pdf files.

- Expected Outcome:
  The chatbot recognizes the command and executes the file grouping function correctly.

  e. Test Case 5: UI Responsiveness

  Objective: Ensure that Fluxor's UI is responsive on different screen sizes.
- Steps:
  - Open Fluxor on a desktop with a large screen.
  - Resize the window to a smaller size.
  - Check that all UI elements adjust correctly and remain accessible
  - .Expected Outcome:
    The UI adjusts to different screen sizes without visual glitches, and all controls remain functional.

CONCLUSION

6.1 Project Outcomes

Fluxor is an innovative, AI-powered file management system that simplifies tasks like file classification, renaming, grouping, and providing user assistance via a chatbot. Designed with efficiency and user experience in mind, it leverages advanced algorithms for content-based classification and customizable file operations. With robust testing ensuring its functionality, performance, and scalability, Fluxor offers a reliable and intuitive solution for users to better manage their digital files. Its modular design also allows for future updates and enhancements, making it a powerful tool for both individual and organizational use.

6.2 Limitations

The current system is limited to Windows OS and relies on internet connectivity for cloud-based features.

6.3 Future Scope

Future plans include:

- Cross-platform support.
- Collaborative tools for organizational use.