ENHANCING FILE ORGANIZATION IN THE MODERN COMPUTING ERA: AN AI-DRIVEN APPROACH WITH THE FILE SORTER APPLICATION

Chenyue Shao[1], Garret Washburn[2]

[1]Campion School, AgiouIoulianis street, Pallini, 15351, Athens, Greece
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

In the modern computing age, where it seems that almost everyone has a computer, the organization of individuals file structures has become increasingly more and more unorganized [1]. In an effort to solve this problem, the File Sorter application proposed in this paper enables a user to easily sort their files with the help of an AI model designed to sort files accurately and efficiently. The File Sorter application utilizes AI to predict where a file should go given a dataset [2]. To test this application, we employed experiments to design how time efficient the File Sorter was as well as how accurately it sorted files, in which both experiments displayed no inefficiency. The public should use the File Sorter application because it accurately and efficiently sorts files in varying manners, and it is good practice for an individual to maintain a healthy file structure because it enables a productive file structure workflow.

KEYWORDS

File Organization, AI Sorting, File Management, Productivity Tools

1. INTRODUCTION

Unorganized file structures have been a problem in the use of computers since their creation. Unlike other computer problems, the fault and natural existence of this problem is entirely placed onto the user, as the user is the mover and creator of files generally. Despite how the problem occurs, the existence of unorganized file structures has proven to dampen productivity and also can potentially lead to the loss of files in the overall structure. Research conducted on how users maintain their file structures demonstrate that the participants who didn't organize their file structures would search for a file 51% of the time, over five times more than the other participants who did organize their file structures [3]. Often, searching for a file will take over five minutes in Windows machines [4]. This time will be even longer if the user is searching in the C:/ drive, as a recent study has shown that when looking at the file search mechaning the "default Windows presentation is suboptimal - if changed, retrieval time could be reduced substantially". This problem, as Windows machines and even other machines all have long file search times, is one of importance because computers are used everywhere, even places of critical importance such as national security or critical infrastructure [5]. As this problem is undoubtedly one of self-doing, as a user must allow their files to become unorganized, the individuals that this problem affects could be anyone, but is inclined to be individuals who are generally unorganized. This inevitability of the file system becoming unorganized creates a necessity for a solution that makes file management easier and provides a base for organizing files.

“3 out of 20 sometimes used, 13 out of 20 rarely used 3 out of 20 every once in a while, and 1 out of 20 never used.”

“Through the course of use, document collections accumulate duplicate folders, folders that fail to reflect current job duties, files no longer relevant, and so on. All but two subjects said they would periodically go through their document collections to tidy up files and folders, deleting files no longer needed and adjusting the folder structure.”

Some other methodologies we used as a baseline for comparison of the efficacy of our application are the Easy File Organizer, habitual file sorting, and Online file organizers like OneDrive and Google Drive.

The Easy File Organizer is an application available on the Microsoft shop that enables the easy file sorting of a directory. Within the application, the way in which files are sorted is quite vague as well as, when sorted, are sometimes inaccurate in our testing. The File Sorter methodology proposed in this paper seeks to improve the file sorting capabilities by utilizing AI to accurately sort files [6].

Habitual file sorting is the habit one employs to maintain a healthy and organized structure. The major drawback of employing habitual file sorting is the requirement that one must consistently employ these habits to maintain the organized structure. Our intentions with the File Sorter are to negate this requirement and allow the user to organize their files in a few clicks.

Online file sorting services such as OneDrive and Google Drive are both solutions to maintaining file organization, as they allow the user to upload files and easily search for them [7]. However, the major drawback to this type of file sorting solution is that they require internet connection as well as the uploading of the files to be sorted. The File Sorter does not require internet connection and will sort files right in the file structure.

The method of solving this problem that is provided in this paper is a program called The File Organizer. The File Organizer is a desktop application that, when pointed at a directory in the user’s operating system, will organize all files either by extension or in alphabetical order, depending on the option the user chooses. Upon choosing a folder directory, the File Organizer will organize them based on the function the user selects, being organized by file type or alphabetically, and place them in subfolders title either by the extension or by the letter of the alphabet it starts with. In the case of the File Organizer sorting by extension, The File Organizer application utilizes machine learning to organize files in the directory to place them into subfolders such as ‘documents’ or ‘images.’ In the case of sorting by alphabetical order, the program reads the first letter of each file and places them into subfolders according to this first letter. The File Organizer is a productive solution to this problem, as it not only makes finding specific files much easier, as it rearranges files into places that are easier to find, but also does so with all files, making all of them easier to find. Another method of solving this problem, which is a lot more common in Windows machines, are background services that maintain and organize the file structure of the OS for the user all by themselves. The method that this paper introduces differs from the ‘file organizer service method’ as The File Organizer requires the user to organize their files themselves with the tool, giving a user a better idea of where their files are at.

The experiments carried out that are displayed in this paper have to do with testing to see how much time File Sorter application takes to sort folders with a varying number of files and how accurately it sorts those folders. To set up the File Sorter sorting time experiment, we took folders of varying amounts of files and measured the time it took for the File Sorter to complete sorting. From this experiment, we discovered that the time to file amount slope was consistently linear, implying that the number of files the File Sorter receives is not a threat to the stability of the application. To test the accuracy of the File Sorter AI model, we created some test folders that we used the File Sorter to sort and then checked and recorded how many files were not accurately sorted. Our findings were that the File Sorter application was consistently accurate when sorting these folders.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Organize Files

A major component of The File Organizer application is the machine learning model that categorizes where a file should be organized based on the extension. Throughout development, we noticed that the AI model tended to inaccurately organize files, which we eventually discovered to be a case of not enough train data [8]. There were a few solutions we could have implemented to sort out this problem, however, we decided to add more data for the AI model to use, as this was the simplest solution. Another minor problem encountered during development, was how to properly fit the data to the model as well as utilize the classifier for a classification matrix, which we solved by creating a vectorizer object for predictions and a classifier object for the matrix.

2.2. Sorting Algorithm

Another major component of the File Organizer Application is the function to organize files alphabetically. A major problem we identified after implementing this function into the application was that the sorting algorithm doesn’t take into account the capitalization of the first letter, enabling files that start with both ‘s’ and ‘S’ for example to be sorted into the same folder. To implement this change and take this into account in the algorithm, we could add an if statement check that checks if a letter is capitalized or not, and then if it is capitalized, we could sort it accordingly and create a new folder if necessary.

2.3. The Structure and Format

The last major component included in the File Organizer application is the graphical user interface that utilizes the Tkinter library [9]. The most prominent challenge that we encountered during development of the GUI was how to implement the structure of the GUI as well as formatting all items on the GUI. When developing the structure of the GUI, it was important to keep in mind all the components that had to be fit onto the GUI, as the size available on the GUI very much limited what we could put on it. Additionally, we also wanted to make sure the application couldn’t be expanded and had to do some research to figure out how to lock the window size. When putting the items onto the GUI, we were constantly going back and forth between the window size parameter and where to put the item, as we wanted to keep the GUI as space efficient as possible.

3. SOLUTION

Upon starting the File Organizer application, the user is greeted by the graphical user interface. This graphical user interface contains the logo of the application, and two buttons labeled 'Sort by Extension' and 'Sort by Alphabetical.' Upon clicking either of these buttons, a new folder selection page opens, in which the user can select a folder that they would like to sort either by extension or alphabetically. If the user selects the 'Sort by Extension' button, the sort_by_extension function executes and sorts the files with an AI model, which is trained on a set of data containing file types and where they should be sorted, into separate categorial folders. Conversely, if the user selects the 'Sort by Alphabetical' button, all of the files inside of the folder will be sorted into folders of the letter in which the file starts with. To create and manage the AI model that we used to sort files into categorial folders by extension we used the sklearn python library, and specifically utilized the MultinomialNB classifier base as the model we trained with our data. As opposed to the extension sorting function, the function used to sort file alphabetically does not use any AI, and really only parses filenames as strings. For the graphical user interface that acts as the structure for the entire application, the python library tkinter was utilized to create a simple and easy to use GUI [10]. To make the graphical user interface more official and user friendly, the use of a custom designed logo was employed.

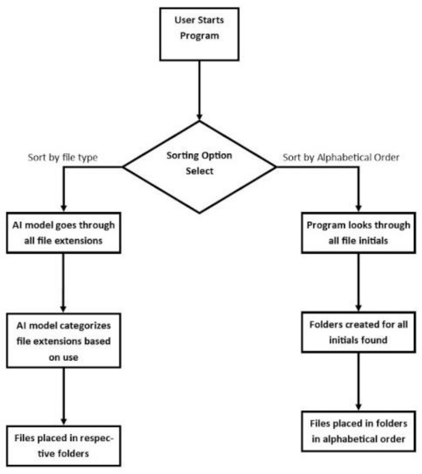

User Starts Program
Sorting Option Select
Sort by file type
Sort by Alphabetical Order
AI model goes through all file extensions
AI model categorizes file extensions based on use
Files placed in respective folders
Program looks through all file initials
Folders created for all initials found
Files placed in folders in alphabetical order

Figure 1. Overview of the solution

The AI model used to sort files into categorial folders based on extension employs the use of the sklearn python library and uses the MultinomialNB classifier model to sort the files. After the model makes a prediction of where the file should be sorted, the organize file's function moves the file into that folder.

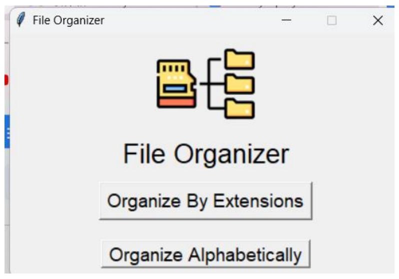
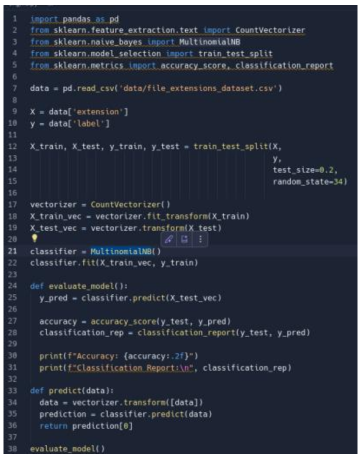
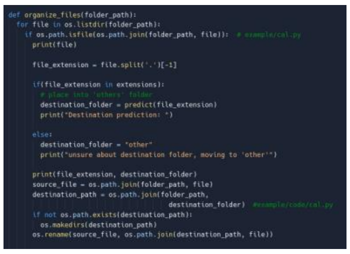


Figure 2. Screenshot of the file organizer



```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

data = pd.read_csv('data/file_extensions_dataset.csv')

X = data['extension']
y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=34)

vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

classifier = MultinomialNB()
classifier.fit(X_train_vec, y_train)

def evaluate_model():
  y_pred = classifier.predict(X_test_vec)

  accuracy = accuracy_score(y_test, y_pred)
  classification_rep = classification_report(y_test, y_pred)

  print(f"Accuracy: {accuracy:.2f}")
  print(f"Classification Report:\n", classification_rep)

def predict(data):
  data = vectorizer.transform([data])
  prediction = classifier.predict(data)
  return prediction[0]

evaluate_model()
```



```python
def organize_files(folder_path):
  for file in os.listdir(folder_path):
    if os.path.isfile(os.path.join(folder_path, file)):  # example/cal.py
      print(file)

      file_extension = file.split('.')[-1]

      if(file_extension in extensions):
        # place into 'others' folder
        destination_folder = predict(file_extension)
        print("Destination prediction: ")

      else:
        destination_folder = "other"
        print("unsure about destination folder, moving to 'other'")

      print(file_extension, destination_folder)
      source_file = os.path.join(folder_path, file)
      destination_path = os.path.join(folder_path,
                                      destination_folder)  #example/code/cal.py
      if not os.path.exists(destination_path):
        os.makedirs(destination_path)
      os.rename(source_file, os.path.join(destination_path, file))
```

Figure 3. Screenshot of code 1

After importing all of the necessary libraries and modules, the engine.py program loads the csv file containing the data and sorts it by the column's 'extension' and 'label,' which are necessary to train the AI model. Upon creating the train_test_split necessary for training the AI model, the program then creates both a vectorizer and a classifier, with the CountVectorizer() and MultinomialNB() models separately, and fits the data to each model so they may be used for predictions [15]. Below the initialization of the vectorizer and classifier is the evaluate_model function, which serves as a way for the user to see how accurate both the classifier and vectorizer are. Below this function lays the predict function, which will decide on which category the files should be in. This predict function is then called from the organize_files function on every file, and depending on the prediction the predict function returns, the organize_files function, if necessary, will create the folder and put the file into that folder.

The organize_files_alphabet function serves as the backend function to sort all of the files within a directory into subfolders of a certain letter. In each of these folders, each file that starts with the corresponding letter of the subfolder is sorted into that subfolder by this function.

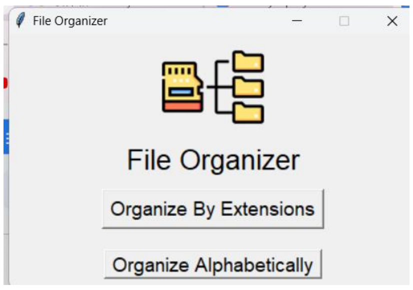
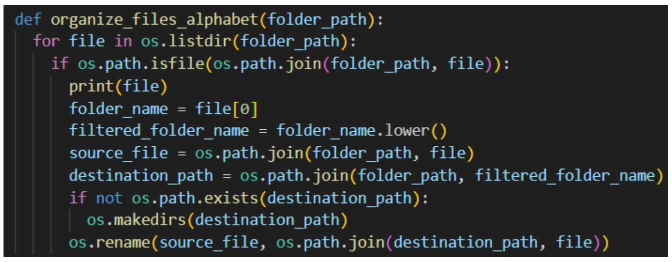


Figure 4. Screenshot of the file organizer

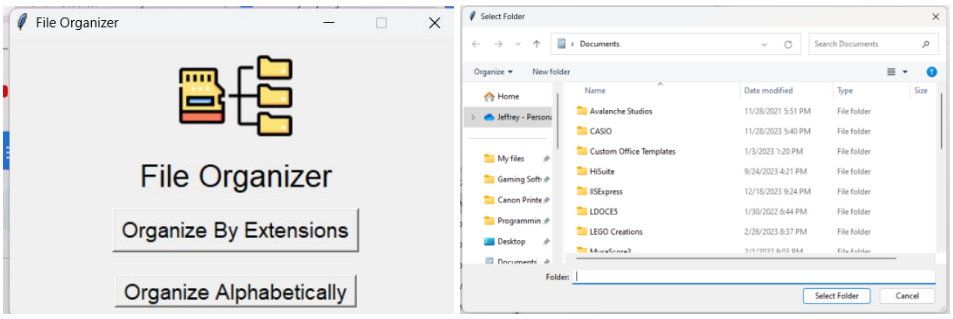


```python
def organize_files_alphabet(folder_path):
  for file in os.listdir(folder_path):
    if os.path.isfile(os.path.join(folder_path, file)):
      print(file)
      folder_name = file[0]
      filtered_folder_name = folder_name.lower()
      source_file = os.path.join(folder_path, file)
      destination_path = os.path.join(folder_path, filtered_folder_name)
      if not os.path.exists(destination_path):
        os.makedirs(destination_path)
      os.rename(source_file, os.path.join(destination_path, file))
```

Figure 5. Screenshot of code 2

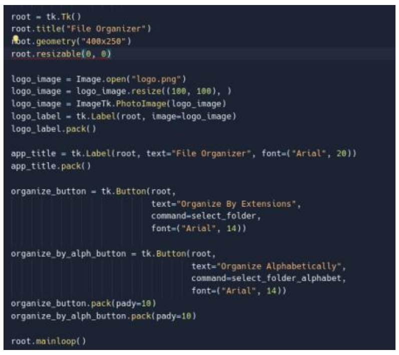
When the organize_folder_alphabet function is called, the folder_path parameters are passed for the function to use. When the function starts, it creates a for loop that will iterate through every file within the folder_path and will check if the file is a valid file. After checking, the function then grabs the ‘folder_name’ that should be sorted into, which is just the first letter of the file, and converts it to lowercase. Upon creating these, the function then creates the ‘source_file’ based on the original files location and the new ‘destination_path’ location based on where the folder should be moved too. The destination_path is then checked to see if it already exists, and if it doesn’t, it will be created so that files may be sorted into it. Then the function os.rename is used to take the source file and move it into the new destination_path. Really only the os python module was employed within this function, as there was no need for anything else.

The Graphical User Interface within the File Organizer application enables the user to interact with and use the backend functions [14]. The python module tkinter was used to create the GUI, as well as manage the main loop of the interface. Additionally, tkinter provides the application a screen for them to select a folder in which they would like to sort.



Name	Date modified	Type	Size
Avalanche Studios	11/28/2021 5:51 PM	File folder	
CASIO	11/28/2023 5:40 PM	File folder	
Custom Office Templates	1/3/2023 1:20 PM	File folder	
HiSuite	9/24/2023 4:21 PM	File folder	
IISExpress	12/18/2023 9:24 PM	File folder	
LDOCE5	1/30/2022 6:44 PM	File folder	
LEGO Creations	2/26/2023 8:37 PM	File folder	

Figure 6. Screenshot of the documents



```python
root = tk.Tk()
root.title("File Organizer")
root.geometry("400x250")
root.resizable(0, 0)

logo_image = Image.open("logo.png")
logo_image = logo_image.resize((100, 100), )
logo_image = ImageTk.PhotoImage(logo_image)
logo_label = tk.Label(root, image=logo_image)
logo_label.pack()

app_title = tk.Label(root, text="File Organizer", font=("Arial", 20))
app_title.pack()

organize_button = tk.Button(root,
                            text="Organize By Extensions",
                            command=select_folder,
                            font=("Arial", 14))

organize_by_alph_button = tk.Button(root,
                                    text="Organize Alphabetically",
                                    command=select_folder_alphabet,
                                    font=("Arial", 14))
organize_button.pack(pady=10)
organize_by_alph_button.pack(pady=10)

root.mainloop()
```

Figure 7. Screenshot of code 3

As seen in the image above, the very first step of creating the tkinter graphical user interface is to initialize the ‘root’ window. Upon initializing the window, we define the title of the root window as well as the geometry and its resizable capacity, which we purposefully set to (0,0) because we do not want it to be resizeable. After setting up the window structure, we go ahead and open up the logo image and set it up in the root window so it may be displayed. The next thing the code does is set the ‘app_title’ to be “File Organizer" and display it in the window. After setting up the display, the code then defines the two buttons for each major function in this program. The organize_button has the parameter command=select_folder, which will then go ahead and trigger a chain from select_folder, which will prompt the user to select a directory, to the organize_filesfunction, which will actually organize files with AI. The organize_by_alph_button has the same kind of flow, but has the unique functions select_folder_alphabet and organize_files_alphabet.

4. Experiment

4.1. Experiment 1

One potential weak spot identified in the application is how the number of files in a folder can impact the applications performance and overall time to sort. It is important that the program sorts of files in a manner that is time efficient because that is its primary function.

For this experiment, the plan is to sort folders of varying amounts of files and see how each increasing size impacts the performance of the application as well as the operating system. The first step of this experiment will be to create different unsorted folders, starting at the size of 50 files and increasing by 50 files until we reach at least 400 files. Depending on how the increased size impacts the performance, we may go ahead and increase the size to maybe 500, however, we will see how the performance is impacted before going there. This will provide us a good understanding of how the number of files will impact the application and operating system.

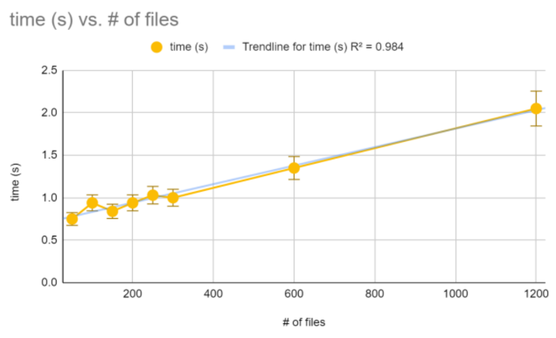


Figure 8. Figure of experiment 1

After collecting our data, nothing really stood out or was surprising about how the number of items affects how long it takes to sort. As seen in the graph, the trendline is considerably linear, implying that there is a constant rate at which the number of items in the folder being sorted will affect the time it takes to sort it. This is actually good news for us and the application, because it implies that, while the application may reach a point in which it overloads with too many items, that the only limiting factor of the applications ability to sort is the computer hardware and storage capacity. We suspect the reason for this linearly shaped trendline is due to the fact that as the operating system encounters more files to sort into directories with labels, it therefore takes the same amount of time to sort each file individually regardless of size.

4.2. Experiment 2

Another potential weak spot in the application is how accurately the file sorting AI model predicts where a file extension should go. It is important that this AI model works proficiently, as it is the main component for deciding where files should go to make it easier for the user to find their files.

For this experiment, we intend to create multiple folders that contain file items to be sorted of various file types. The objective is to use the file sorter AI model to sort the file items and see how accurately it is able to sort them. We will compare the outputted sorted directory directly to the csv file data that we trained the AI model on to see if all of the extensions of all of the file items are put into their correct sub-directories. We will then create a table in excel, and input, for each one of these folders, the number of files sorted correctly as well as the number of files sorted into incorrect directories.

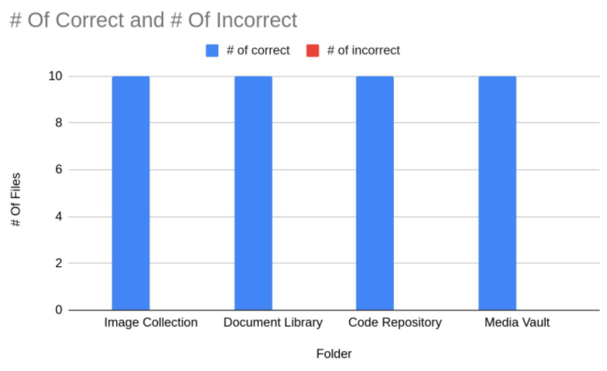


Figure 9. Figure of experiment 2

In the experiment, we encountered very assured results that the file sorting AI model created for this application is accurate, as all of the files were sorted into the correct sub-folders. One of the surprising aspects we discovered throughout the experiment was that one of the HTML files was placed in the code folder rather than in the web folder, which we initially assumed was incorrect. However, aftering inspecting the data source, we discovered that .html files are given both types of webs and code in the dataset multiple times. Although this can be considered an error, it doesn't show that the AI model is anyway wrong, rather that the AI model was given conflicting data, and sorted the HTML file into one of the folders it was given in the dataset, which is still correct. Moving forward, the conflicting data within the dataset is something that should be fixed.

5. Related Work

The Easy File Sorter, an application on the Microsoft shop, is an application that sorts files in a very similar fashion compared to the File Sorter method displayed in this paper [11]. However, in our testing we have found that it is considerably vague in where it sorts its files and doesn't have as many categories and levels of detail as our File Sorter application does. After trying out the app, we noticed that the Easy File Sorter had a tendency to misplace some files into different folders as well. Compared to applications proposed in this paper, the File Sorter AI model is trained to sort files by extension more accurately and the application is able to run on any operating system, not solely windows.

Another method for organizing files is to maintain a habit to name and accurately organize the files within the file structure accordingly [12]. One drawback, however, to maintaining the file system organization by relying on habit, is that it takes consistent effort to employ this file maintenance strategy. However, when compared to our File Sorter application, our method only takes a few clicks, and the files in whatever folder you would like will be sorted accurately. The method proposed in this paper, while it may enable the development of bad file sorting habits, solves the problem of needing consistent effort to maintain an organized file system structure.

Services like Onedrive and Google Drive offer file sorting services that make the ability to search through files much easier and can even organize files in a quick and easy manner [13]. However, both of these services require the internet and that the files to be sorted be uploaded to their respective services. Additionally, the way in which Google Drive organizes files is through the use of their search function, which must be repeated everytime a user is looking for a specific file. Compared to the method proposed in this paper, the File Sorter application does not require any internet connection and will consistently organize files within the actual file structure.

6. CONCLUSIONS

Given more time to develop this project, after noticing some of the other methodologies implementing this strategy, the ability to create custom sorting rules is something we would like to implement into the File Sorter application. This would likely look like a table in which the user can create the folder name and input the file extension types they would like to be sorted into this folder. This would enable the user to sort, not only accurately, but in a manner that is more to how they would like their own file structure to be sorted. Additionally, the AI model that is used for predicting where a file should be sorted could be a bit more lightweight, as we believe we can make it lighter. We would also like to add more extension types to the dataset, so that the AI model will be even more accurate and have a wider range of extension types it can accurately sort.

This research paper has provided me with many opportunities for learning and given me many insights into how the public tends to sort their files as well as how to implement ideas to create solutions to problems like this. Thank you very much for reading about my project, it was a lot of fun!

REFERENCES

[1] Gill, Sukhpal Singh, et al. "Modern computing: Vision and challenges." Telematics and Informatics Reports (2024): 100116.
[2] Li, Xiaoxiu, et al. "Design and Implementation of Survey and Design Enterprise File Sorting System." Proceedings of the 6th International Conference on Computer Science and Application Engineering. 2022.
[3] Cusack, Thomas R., and Lutz Engelhardt. "The PGL file collection: File structures and procedures." Wissenschaftszentrum Berlin für Sozialforschung (2002).
[4] Bird, Christian, et al. "Extrinsic influence factors in software reliability: A study of 200,000 windows machines." Companion Proceedings of the 36th International Conference on Software Engineering. 2014.
[5] Yu, Yang, et al. "A feather-weight virtual machine for windows applications." Proceedings of the 2nd international conference on Virtual execution environments. 2006.
[6] ZareHarofte, Samaneh, et al. "Recent advances of utilizing artificial intelligence in lab on a chip for diagnosis and treatment." Small 18.42 (2022): 2203169.
[7] Quick, Darren, and Kim-Kwang Raymond Choo. "Google Drive: Forensic analysis of data remnants." Journal of Network and Computer Applications 40 (2014): 179-193.
[8] McLaren, Bruce M. "Extensionally defining principles and cases in ethics: An AI model." Artificial Intelligence 150.1-2 (2003): 145-181.

[9] Moore, Alan D. Python GUI Programming with Tkinter: Design and build functional and user-friendly GUI applications. Packt Publishing Ltd, 2021.

[10] Banerjee, Ishan, et al. "Graphical user interface (GUI) testing: Systematic mapping and repository." Information and Software Technology 55.10 (2013): 1679-1694.

[11] Ribas, Lluis, David Castells, and Jordi Carrabina. "A linear sorter core based on a programmable register file." XIX Conference on Design of Circuits and Integrated Systems, DCIS. 2004.

[12] Hicks, Ben J., et al. "Organizing and managing personal electronic files: A mechanical engineer's perspective." ACM Transactions on Information Systems (TOIS) 26.4 (2008): 1-40.

[13] Daryabar, Farid, et al. "Forensic investigation of OneDrive, Box, GoogleDrive and Dropbox applications on Android and iOS devices." Australian Journal of Forensic Sciences 48.6 (2016): 615-642.

[14] Bode, Brett M., and Mark S. Gordon. "MacMolPlt: a graphical user interface for GAMESS." Journal of Molecular Graphics and Modelling 16.3 (1998): 133-138.

[15] Fiok, Krzysztof, et al. "Explainable artificial intelligence for education and training." The Journal of Defense Modeling and Simulation 19.2 (2022): 133-144.