SI568-Project 2 LangChain Generative AI Healthcare Solution

Jason Kemp, Rahat Naseem
Vaibhav Malla, Zhiyi Ji

04.19.2025

PROJECT OVERVIEW

- **Project Goal:** Enable healthcare professionals to quickly access section-specific answers from scientific research papers.
- **Core Functionality:**
  - Built a chatbot using LangChain and OpenAI GPT-4o
  - Supports real-time question answering over research PDFs
  - Runs on an interactive Streamlit web app

01

REASON 1

Scientific papers are dense and time-consuming to read.

02

REASON 2

Existing search tools lack section-level understanding.

03

REASON 3

There’s a need for intelligent, context-aware question answering.

MOTIVATION & PROBLEM

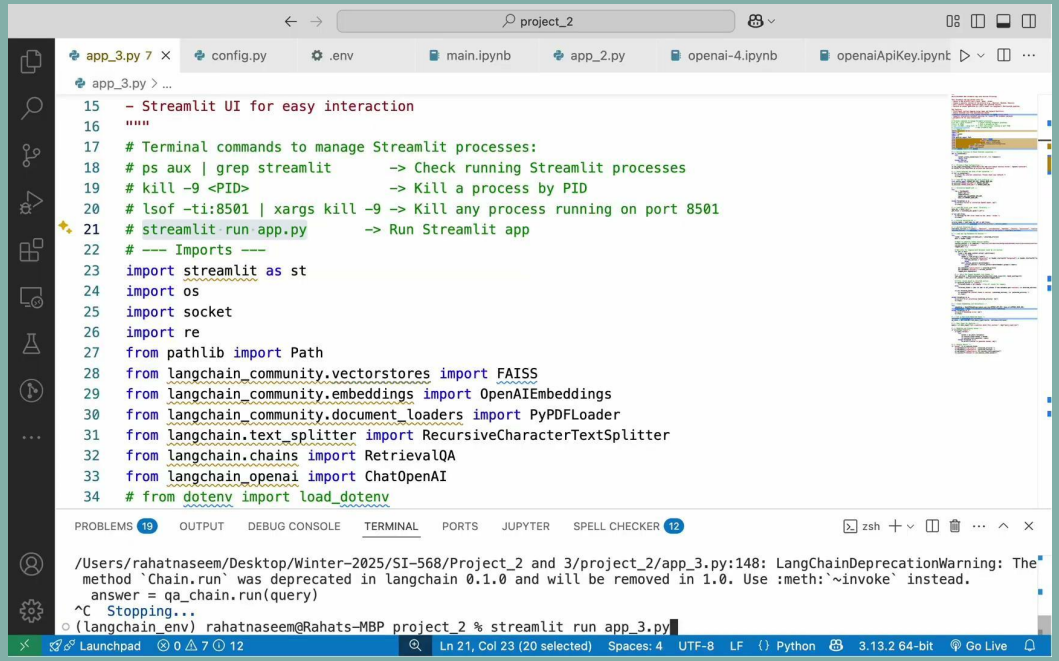
HOW IT WORKS - SETUP

- **Upload PDFs:** The system loads research papers from a folder.
- **Split into Chunks:** Each document is split into smaller text chunks for processing.
- **Generate Embeddings:** Chunks are converted into vector embeddings using **FAISS**.

HOW IT WORKS - GENERATION

- **User Interaction:** The user selects a document and a specific section
- **Retrieve Relevant Content: LangChain** searches for the most relevant chunks based on the question and selected section.
- **Answer Generation: GPT-4o** uses the retrieved content to generate an accurate, context-aware answer.

DEMO


project_2
app_3.py 7
config.py
.env
main.ipynb
app_2.py
openai-4.ipynb
openaiApiKey.ipynb
app_3.py > ...
15 - Streamlit UI for easy interaction
16 """
17 # Terminal commands to manage Streamlit processes:
18 # ps aux | grep streamlit -> Check running Streamlit processes
19 # kill -9 <PID> -> Kill a process by PID
20 # lsof -ti:8501 | xargs kill -9 -> Kill any process running on port 8501
21 # streamlit run app.py -> Run Streamlit app
22 # --- Imports ---
23 import streamlit as st
24 import os
25 import socket
26 import re
27 from pathlib import Path
28 from langchain_community.vectorstores import FAISS
29 from langchain_community.embeddings import OpenAIEmbeddings
30 from langchain_community.document_loaders import PyPDFLoader
31 from langchain.text_splitter import RecursiveCharacterTextSplitter
32 from langchain.chains import RetrievalQA
33 from langchain_openai import ChatOpenAI
34 # from dotenv import load_dotenv
PROBLEMS 19
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
JUPYTER
SPELL CHECKER 12
zsh
/Users/rahatnaseem/Desktop/Winter-2025/SI-568/Project_2 and 3/project_2/app_3.py:148: LangChainDeprecationWarning: The method `Chain.run` was deprecated in langchain 0.1.0 and will be removed in 1.0. Use :meth:`~invoke` instead.
answer = qa_chain.run(query)
^C Stopping...
(langchain_env) rahatnaseem@Rahats-MBP project_2 % streamlit run app_3.py
Launchpad
0 7 12
Ln 21, Col 23 (20 selected)
Spaces: 4
UTF-8
LF
Python
3.13.2 64-bit
Go Live

Evaluation

Select an article:

A Case-Control Clinical Trial on a Deep Learning-Based Classification ...

Select a section to ask about:

abstract

Ask a question about this section:

what information is provided in the abstract section

Submit

Article: `A Case-Control Clinical Trial on a Deep Learning-Based Classification System for Diagnosis of Amyloid-Positive Alzheimer's Disease.pdf`

Section: `abstract`

Question: what information is provided in the abstract section

Answer: I'm sorry, but the abstract section is not provided in the context you shared.

Ask a question about this section:

tell me what information is before the introduction section

Submit

Article: `A Case-Control Clinical Trial on a Deep Learning-Based Classification System for Diagnosis of Amyloid-Positive Alzheimer's Disease.pdf`

Section: `abstract`

Question: tell me what information is before the introduction section

Answer: The information before the introduction section includes details about specificity, positive predictive value (PPV), and negative predictive value (NPV), as well as the methods used for comparing demographics, such as independent samples t-test for continuous variables and chi-square test for categorical variables. Additionally, there is information about the assumed and targeted sensitivity/specificity under the null and alternative hypotheses, the sample size estimation, image acquisition parameters, study participants, and the network used for processing MRI data.

Deploy

what is the information provided before the introduction section

Submit

Article: `A Case-Control Clinical Trial on a Deep Learning-Based Classification System for Diagnosis of Amyloid-Positive Alzheimer's Disease.pdf`

Section: `abstract`

Question: what is the information provided before the introduction section

Answer: The information provided before the introduction section includes details about the methodology used in the study, such as the processing of MRI data, the construction of a feature vector that includes structural and textural information of the brain from the coronal slice, and the inclusion of the subject's age, sex, and slice location information. It also describes the use of a fully connected network to calculate the probability of a slice belonging to a patient with Alzheimer's Disease (AD) and how the probabilities are averaged to calculate a final score representing the subject's probability of having AD. Additionally, it mentions the anonymous processing of MRI data and the blinding of a researcher to the subjects' clinical diagnoses.

A Case-Control Clinical Trial on a Deep Learning-Based Classification System for Diagnosis of Amyloid-Positive Alzheimer's Disease

Jong Bin Bae[1,2*], Subin Lee[3*], Hyunwoo Oh[4], Jinkyeong Sung[5], Dongsoo Lee[4], Ji Won Han[1,2], Jun Sung Kim[1,5], Jae Hyoung Kim[6], Sang Eun Kim[7,8], and Ki Woong Kim[1,2,3]

[1]Department of Neuropsychiatry, Seoul National University Bundang Hospital, Seongnam, Republic of Korea
[2]Department of Psychiatry, Seoul National University, College of Medicine, Seoul, Republic of Korea
[3]Department of Brain and Cognitive Sciences, Seoul National University College of Natural Sciences, Seoul, Republic of Korea
[4]VUNO Inc., Seoul, Republic of Korea
[5]Institute of Human Behavioral Medicine, Seoul National University Medical Research Center, Seoul, Republic of Korea
[6]Department of Radiology, Seoul National University Bundang Hospital, Seongnam, Republic of Korea
[7]Department of Nuclear Medicine, Seoul National University Bundang Hospital, Seongnam, Republic of Korea
[8]Center for Nanomolecular Imaging and Innovative Drug Development, Advanced Institutes of Convergence Technology, Suwon, Republic of Korea

Objective A deep learning-based classification system (DLCS) which uses structural brain magnetic resonance imaging (MRI) to diagnose Alzheimer's disease (AD) was developed in a previous recent study. Here, we evaluate its performance by conducting a single-center, case-control clinical trial.

Methods We retrospectively collected T1-weighted brain MRI scans of subjects who had an accompanying measure of amyloid-beta (Aβ) positivity based on a 18F-florbetaben positron emission tomography scan. The dataset included 188 Aβ-positive patients with mild cognitive impairment or dementia due to AD, and 162 Aβ-negative controls with normal cognition. We calculated the sensitivity, specificity, positive predictive value, negative predictive value, and area under the receiver operating characteristic curve (AUC) of the DLCS in the classification of Aβ-positive AD patients from Aβ-negative controls.

Results The DLCS showed excellent performance, with sensitivity, specificity, positive predictive value, negative predictive value, and AUC of 85.6% (95% confidence interval [CI], 79.8–90.0), 90.1% (95% CI, 84.5–94.2), 91.0% (95% CI, 86.3–94.1), 84.4% (95% CI, 79.2–88.5), and 0.937 (95% CI, 0.911–0.963), respectively.

Conclusion The DLCS shows promise in clinical settings where it could be routinely applied to MRI scans regardless of original scan purpose to improve the early detection of AD.

Psychiatry Investig 2023;20(12):1195-1203

Keywords Alzheimer disease; Magnetic resonance imaging; Clinical trial; Deep learning.

INTRODUCTION

The number of individuals with dementia is increasing globally, with more than 130 million people expected to live with dementia in 2050.[1] Alzheimer's disease (AD) is the most prevalent type and cause of dementia,[2] with no cure currently available yet. Early detection of AD is crucial because it allows for advanced treatment planning and improves prognosis. However, due to the insidious nature of the disease, more than 60% of people living with dementia in the community go undetected.[3] In order to improve the accuracy and advance the timing of AD diagnosis, the National Institute on Aging-Alzheimer's Association proposed new diagnostic criteria for AD that incorporates neuroimaging biomarkers such as amyloid beta (Aβ) deposition and neuronal degeneration.[4,5] Assessment of Aβ deposition (performed using positron emission tromography [PET]) is an earlier and more specific biomarker of AD than assessments of neurodegeneration (performed using magnetic structural imaging [MRI]). However, PET has practical drawbacks in clinical practice because they involve radiation and are not available in all clinical settings.

Structural brain MRI is more widely available but less ex-

Received: February 16, 2023 **Revised:** August 13, 2023
Accepted: September 12, 2023

Correspondence: Ki Woong Kim, MD, PhD
Department of Neuropsychiatry, Seoul National University Bundang Hospital, 82 Gumi-ro 173beon-gil, Bundang-gu, Seongnam 13620, Republic of Korea
Tel: +82-31-787-7432, Fax: +82-31-787-4058, E-mail: kwkimmd@snu.ac.kr
*These authors contributed equally to this work.

© This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (https://creativecommons.org/licenses/by-nc/4.0) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Copyright © 2023 Korean Neuropsychiatric Association **1195**

KEY FEATURES

What makes it unique:

- Heuristic section detection
- Section-aware QA (Abstract, Methods, etc.)
- “Summary mode” = Full-document Q&A
- Real-time feedback if content is missing

- **Minimizes AI hallucinations:** By narrowing context to specific sections, the model provides more accurate, grounded answers.
- **Improves research efficiency:** Saves time for healthcare professionals by delivering quick, section-specific insights from scientific papers.

IMPACT

- Support for multi-section queries
- Direct PDF uploads
- Chat history saving

FUTURE WORK

```python
"""
Multi-Document Q&A Streamlit App with Section Filtering

This Streamlit web app allows users to:
- Select a PDF article from a local `data/` folder
- Choose a specific section of the article (e.g., Abstract, Methods, Results)
- Ask a natural language question about the selected section
- Receive an answer generated by a GPT-4 model via LangChain's RetrievalQA pipeline

Key Features:
- Intelligent section tagging using regex and keyword heuristics
- Robust internet and file availability checks
- Embedding generation using OpenAI Embeddings and FAISS vector store
- Supports interactive document querying for research and academic purposes
- Streamlit UI for easy interaction
"""
# Terminal commands to manage Streamlit processes:
# ps aux | grep streamlit       -> Check running Streamlit processes
# kill -9 <PID>                 -> Kill a process by PID
# lsof -ti:8501 | xargs kill -9 -> Kill any process running on port 8501
# streamlit run app.py       -> Run Streamlit app
# --- Imports ---
import streamlit as st
import os
import socket
import re
from pathlib import Path
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains import RetrievalQA
from langchain_openai import ChatOpenAI
# from dotenv import load_dotenv; we dont use this anymore

# --- Utility Function to Check Internet Connection ---
def is_connected():
    try:
        socket.create_connection(("8.8.8.8", 53), timeout=3)
        return True
    except OSError:
        return False

# --- Streamlit Page Configuration ---
st.set_page_config(page_title="Multi-Doc Q&A with Robust Section Filter", layout="centered")
st.title("📄 Ask Questions by Article and Sections")

# --- Check Internet and Stop if Not Connected ---
if not is_connected():
    st.error("No internet connection. Please check your network.")
    st.stop()

# --- Load API Key and Base URL from config.py ---
from config import OPENAI_API_KEY, OPENAI_BASE_URL
os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY
os.environ["OPENAI_BASE_URL"] = OPENAI_BASE_URL

# --- Initialize OpenAI LLM ---
try:
    llm = ChatOpenAI(
        model="gpt-4o",
        temperature=0,
        openai_api_key=OPENAI_API_KEY,
        base_url=OPENAI_BASE_URL
    )
except Exception as e:
    st.error(f"Failed to initialize OpenAI model: {e}")
    st.stop()

# --- Load PDF Files from `data/` Directory ---
data_dir = Path("data")
pdf_files = list(data_dir.glob("*.pdf"))

if not pdf_files:
    st.warning("No PDF files found in the `data/` folder.")
    st.stop()

# --- Article Selection UI ---
article_names = [pdf.name for pdf in pdf_files]
selected_article = st.selectbox("Select an article:", article_names)

# --- Section Selection UI ---
available_sections = ["summary", "abstract", "introduction", "methods", "results", "discussion",
"conclusion", "unknown"]
selected_section = st.selectbox("Select a section to ask about:", available_sections)

# --- Load and Tag Documents by Section ---
try:
    loader = PyPDFLoader(str(data_dir / selected_article))
    docs = loader.load()

    # Regex to identify common section headers
    section_pattern =
re.compile(r"^(abstract|introduction|background|methods|results|discussion|conclusion)$", re.IGNORECASE)

    current_section = "unknown"
    tagged_docs = []

    # Heuristic for tagging each document chunk by its section
    for doc in docs:
        lines = doc.page_content.strip().splitlines()
        for line in lines:
            header = line.strip().lower()
            if header.startswith("objective") or header.startswith("background") or
header.startswith("methods"):
                current_section = "abstract"
                break
            elif section_pattern.match(header):
                current_section = section_pattern.match(header).group(1).lower()
                break
        doc.metadata["source_file"] = selected_article
        doc.metadata["section"] = current_section
        tagged_docs.append(doc)

    # --- Split the Tagged Document into Chunks ---
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
    all_chunks = text_splitter.split_documents(tagged_docs)

    # Filter chunks based on selected section
    if selected_section == "summary":
        filtered_chunks = all_chunks  # Use all chunks for summary
    else:
        filtered_chunks = [doc for doc in all_chunks if doc.metadata.get("section") == selected_section]


    if not filtered_chunks:
        st.warning(f"No content found in section '{selected_section}' for {selected_article}.")
        st.stop()

except Exception as e:
    st.error(f"Error processing {selected_article}: {e}")
    st.stop()

# --- Create Embeddings and VectorStore ---
try:
    embedding = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY, base_url=OPENAI_BASE_URL)
    vectorstore = FAISS.from_documents(filtered_chunks, embedding)
except Exception as e:
    st.error(f"Embedding error: {e}")
    st.stop()

# --- Set up Retrieval-Based QA Chain ---
retriever = vectorstore.as_retriever()
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)

# --- User Input for Question ---
query = st.text_input("Ask a question about this section:", key="query_input_box")

# --- Generate and Display Answer ---
if st.button("Submit"):
    if query.strip():
        try:
            answer = qa_chain.run(query)
            st.session_state.answer = answer
            st.session_state.question = query
        except Exception as e:
            st.error(f"Failed to generate answer: {e}")

# --- Display Result ---
if "answer" in st.session_state:
    st.markdown(f"**Article:** `{selected_article}`")
    st.markdown(f"**Section:** `{selected_section}`")
    st.markdown(f"**Question:** {st.session_state.question}")
    st.success(f"**Answer:** {st.session_state.answer}")
```

Project 2: LangChain Generative AI Healthcare Solution

Team Members: Jason Kemp, Rahat Naseem, Vaibhav Malla, Zhiyi Ji

📄 Multi-Document Q&A Streamlit App with Section-Aware Retrieval

This interactive Streamlit app allows users to ask questions about research articles (PDFs) by selecting both the **document** and a specific **section** such as *Abstract*, *Methods*, *Results*, etc. The system uses OpenAI's GPT models via LangChain to provide answers based on semantically retrieved content chunks enriched with metadata.

Features

Multi-Document Support

- Automatically loads all PDF files from the `/data/` folder.
- Users can select the target document via a dropdown menu.

Section-Aware Contextual QA

- Detects and tags sections like `abstract`, `introduction`, `methods`, `results`, `discussion`, and `conclusion`.
- Supports a **"summary"** mode that uses the full document for broader Q&A.

Heuristic Section Detection

- Identifies section headers even when not explicitly labeled.
- Recognizes abstract-like content from keywords like *Objective*, *Methods*, *Results*.

Filtered Retrieval

- Limits answer generation to only the content from the selected section.
- Reduces hallucination by narrowing context.

Enhanced User Experience

- Real-time internet connection check before launching the app.
- Helpful feedback when content is missing for a section.
- Clean and intuitive interface with dropdowns for articles and sections.

Notes

All answers are generated by filtering and embedding only the selected section for improved relevance. If a section is not found, the app provides a clear warning. The "summary" mode enables global reasoning over the full document.

Future Enhancements

Allow multi-section selection Display chunk previews before Q&A Save chat history for each document Add PDF upload capability directly through the app

Tech Stack

- **Python 3.10+**
- **Streamlit**
- **LangChain**
- **FAISS** (Vector store)
- **OpenAI GPT-4o** via custom base URL
- **PyPDFLoader** for document ingestion

Directory Structure

project/ data/ Your article PDFs go here

Usage

1. **Install dependencies**:

   pip install -r requirements.txt 2 **Run the App**: streamlit run app_with_robust_section_detection.py

LED PROJECT

Team #8

Ali Al-Ogaili | Rahat Naseem | Ruiyang Wu | Ruizhe Wang

Data Source and Parsing Background

Data source: https://api.open.fec.gov/developers/

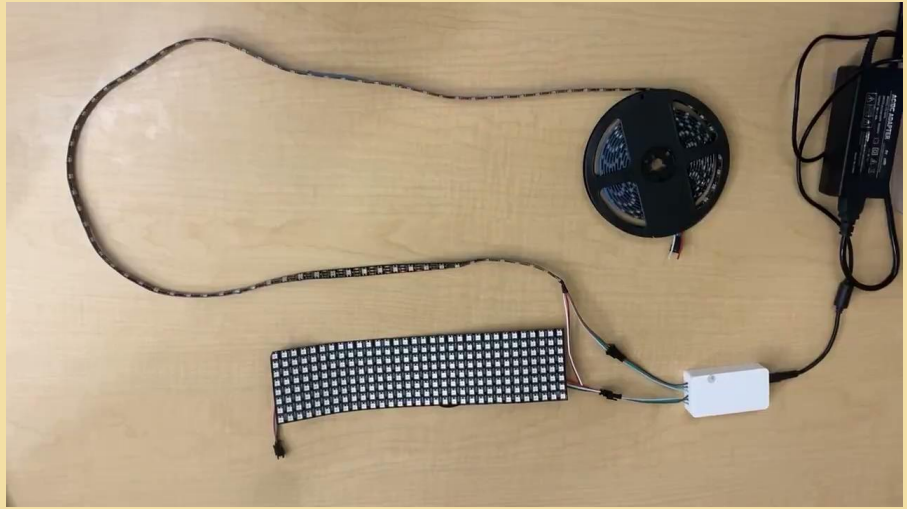
Our project leverages real-time campaign finance data from the Federal Election Commission (FEC) API as our primary data source. This API provides detailed financial records of political candidates, including total receipts, disbursements, and net differences in funding. To extract and utilize this data, we implemented the get_federal_contributions() function, which queries the API, retrieves relevant financial metrics, and determines the candidate's party affiliation using a separate API call. The parsed data is then mapped to our LED matrix display, where colors dynamically change based on political party and spending trends. Deep blue represents Democratic candidates, red represents Republican candidates, and purple is used for others. Additionally, the top and bottom rows of the matrix indicate financial standing, with green for surplus and orange for deficits. By integrating efficient data parsing and real-time updates, our system ensures an accurate and visually intuitive representation of campaign funding trends.

#1

Requirement #1

Make all the light strip lights blue/ then red/ then green.

- cycle_strip_colors() iterates through the colors and updates all 300 LEDs simultaneously.
- Used get_color_value() to retrieve RGBW values and format the command.
- Applied time.sleep(1) to hold each color for one second before switching.
- Ended the sequence with turn_off_led() to power down the display.

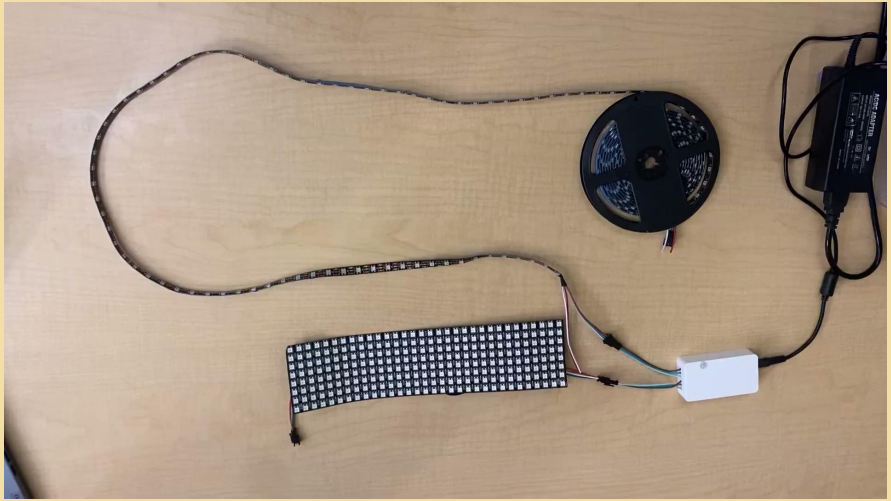


#2

Requirement #2

Make all the lights on the matrix blue/ then red then/ green.

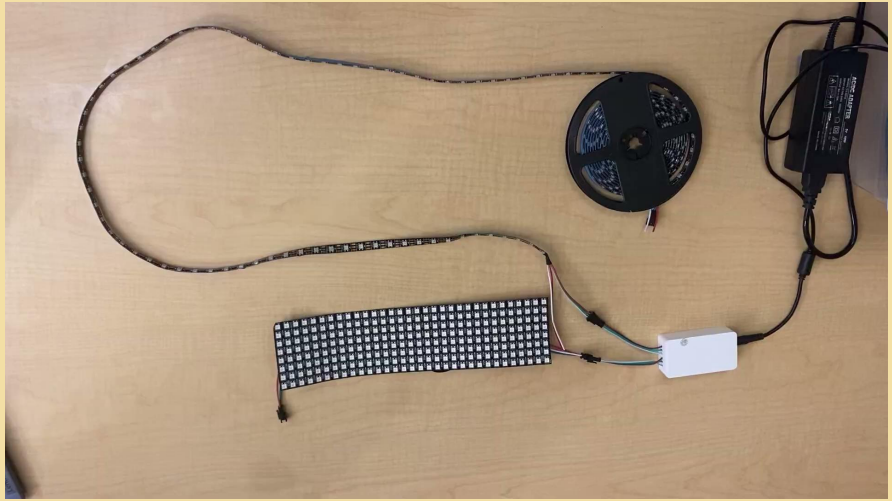
- cycle_matrix_colors() updates all 256 LEDs on the matrix simultaneously.
- Used get_color_value() to retrieve RGB values and format the command.
- Set the LED strip to black (0,0,0,0) to keep only the matrix active.
- Applied time.sleep(1) to hold each color for one second before switching.
- Ended the sequence with turn_off_led() to power down the display.



Requirement #3

Make every other light maize (yellow) and blue. (E.g., the first light is maize, second is blue, third is maize, and so on.)

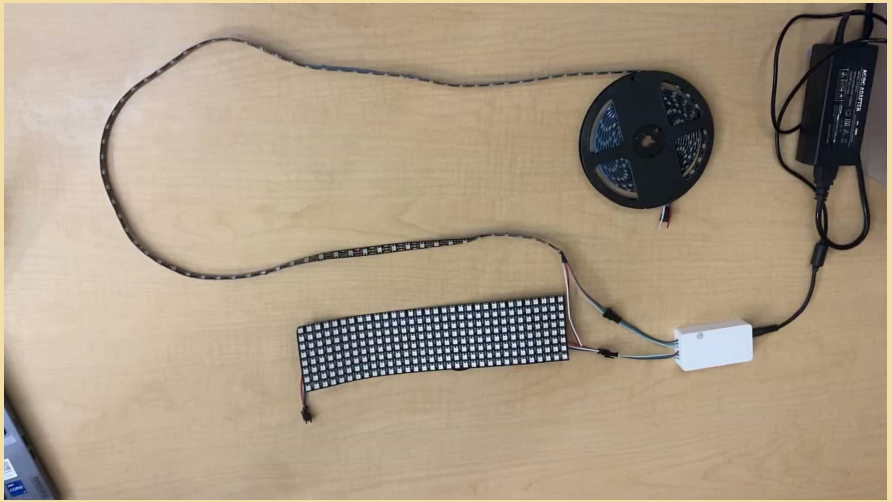
- alternating_maize_blue() sets even-indexed LEDs to maize and odd-indexed LEDs to blue.
- Used get_color_value() to retrieve RGBW values for the strip and RGB values for the matrix.
- Constructed two comma-separated command strings to apply the pattern.
- Sent the pattern using send_led_command() and held it for five seconds.
- Ended the sequence with turn_off_led() to power down the display.



Requirement #4

Make the lights alternate back and forth from maize to blue in two different ways. All lights change together. (E.g., all lights maize, then all lights blue, maize, and so on.)

- toggle_maize_blue() switches all LEDs to maize, then blue, repeating for five cycles.
- Used get_color_value() to retrieve RGBW values for the strip and RGB values for the matrix.
- Applied time.sleep(delay) to hold each transition for one second.
- Ended the sequence with turn_off_led() to power down the display.

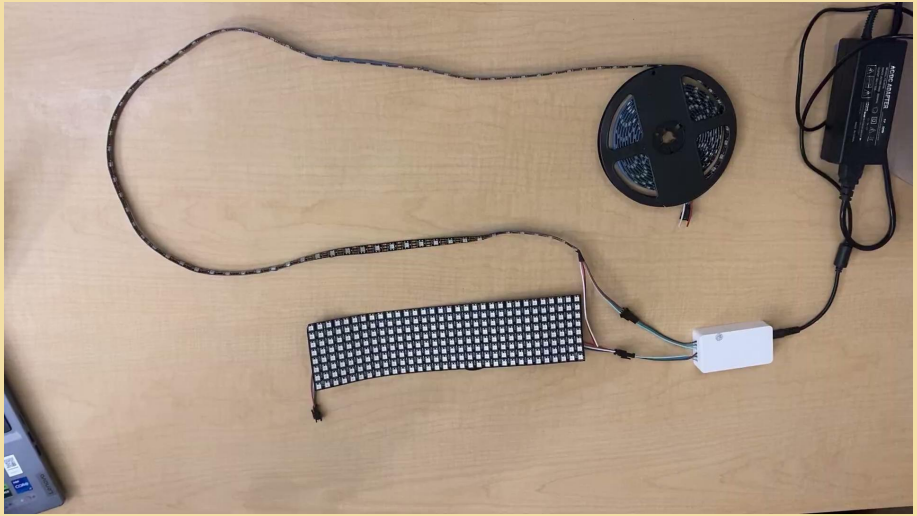


Requirement #5

Maize-Green Fade Effect

We created a smooth fading effect that transitions between maize and green colors across both the LED strip and matrix. This creates a visually appealing dynamic light sequence.

- Implemented toggle_maize_green_fade(), which gradually shifts between maize and green using fade_transition().
- Used incremental RGB blending to create a smooth color transition.



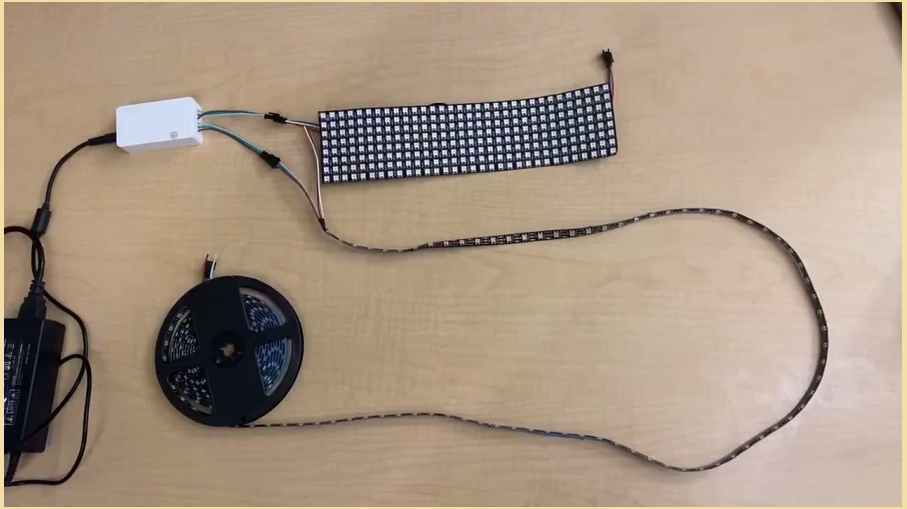
Bonus

Requirement #6

Displaying 'UMSI' on LED Matrix

We wrote "UMSI" on LED matrix using a maize-on-blue color scheme.

- Used Python Image Library to create an image with a blue background and maize letters.
- Designed a 5x5 pixel pattern for each letter and mapped them onto the LED matrix.
- Converted the image into a zigzag column-row format to match the LED hardware layout.



Requirement #7

API Integration:

- Fetched real-time campaign data (party, receipts, disbursements) using the FEC API.

Dynamic Color Assignment:

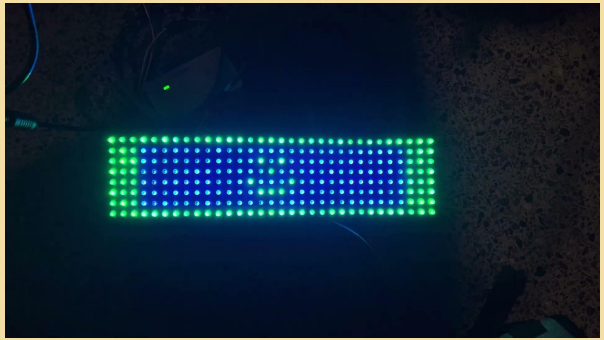
- **Party colors:** Blue (Dem), Red (Rep), Purple (Other).
- **Spending difference:** Green (Happy), Orange (Sad), Pale Green (Neutral).

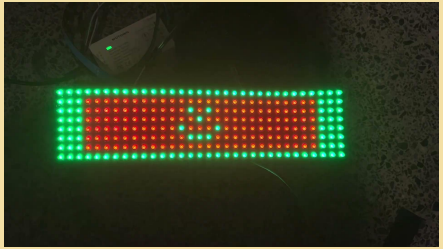
Matrix Visualization:

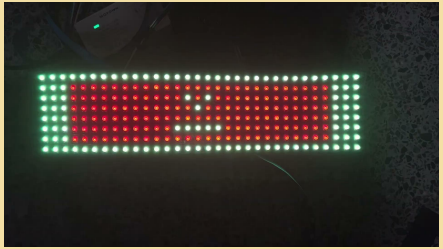
- Top & bottom rows + side columns reflect spending difference.
- Middle rows represent party color.
- Centered smiley (happy/sad/neutral) shows financial performance.

Technical Details:

- Converted data to column-major format for accurate LED mapping.
- Handled API errors with default values and friendly messages.
- Added delay to control display timing and avoid flickering.







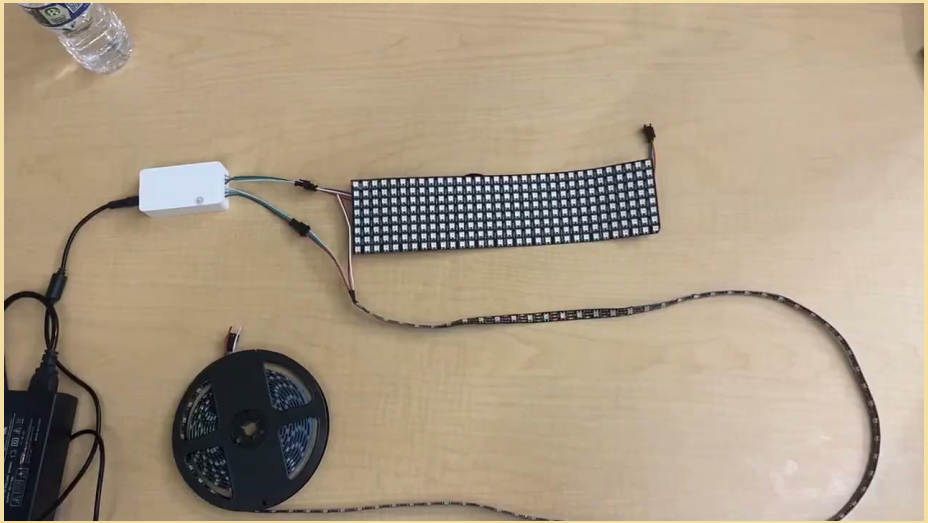
Bonus

Requirement #8

Scrolling Time & 'UMSI' Display

We designed a live scrolling display that alternates between the current time and "UMSI" on the LED matrix. This continuously updating effect makes it a useful and visually appealing installation for the UMSI student lounge.

- Built on Requirement 6, adding real-time numbers alongside "UMSI."
- Applied a scrolling effect, shifting the display pixel-by-pixel for smooth movement.
- Updated the time dynamically in real-time using datetime and ZoneInfo.



```
import requests
import datetime
import time
from PIL import Image, ImageDraw
from PIL import ImageFont

class LEDController:
    def __init__(self, device_name=None, api_key=None, strip_id=None <= 300, matrix_id=None <= 1,
matrix_width=None <= 11):
        self.device_name = device_name
        self.api_key = api_key
        self.base_url = f'https://developer.nanoleaf.com/api/v1/{api_key}/device/{device_name}'
        self.ctrl_url = f'https://ws3dd.smart.nanoleaf.com/v1/{api_key}/device/{device_name}'

        self.strip_id = strip_id
        self.matrix_id = matrix_id
        self.matrix_width = matrix_width
        self.matrix_size = matrix_id * matrix_width

        # Encode color maps are current
        self.color_map = {
            'red': [255, 0, 0, 0],
            'red': [255, 0, 0, 0],
            'green': [0, 255, 0, 0],
            'blue': [0, 0, 255, 0],
            'off': [0, 0, 0, 0]
        }

        self.matrix_colors = {
            'blue': [0, 0, 255],
            'red': [255, 0, 0],
            'green': [0, 255, 0],
            'white': [255, 255, 255],
            'deep blue': [0, 0, 139],
            'orange': [255, 165, 0],
            'purple': [128, 0, 128],
            'pink': [255, 192, 203],
            'off': [0, 0, 0]
        }

    def send_ind_command(self, strip_values=None, matrix_values=None):
        '''Send command to update LED strip and matrix.'''
        if strip_values and matrix_values:
            raise ValueError('Only one LED command id')
        try:
            response = requests.put(self.base_url, json=payload, timeout=(1,
            response.raise_for_status()
        except requests.exceptions.RequestException as e:
            print(f'Failed to send LED command: {e}')

    def turn_off_ledpanel():
        '''Turn off all LEDs.'''
        try:
            response = requests.put(self.ctrl_url, timeout=(1)
            response.raise_for_status()
        except requests.exceptions.RequestException as e:
            print(f'Failed to turn off LEDs: {e}')

    def get_color_value(self, color_name=None):
        '''Retrieve RGB(w) values for specified color.'''
        color_name = color_name.lower()
        if hasattr(self, 'color_map':
            return getattr(self.color_map, color_name, self.color_map['off'])
        elif hasattr(self, 'matrix'):
            return getattr(self.matrix_colors, color_name, self.matrix_colors['off'])
        else:
            raise ValueError('Invalid LED type. Use strip' or 'matrix.')


class LEDRequirements:
    def __init__(self, controller=LEDController):
        self.controller = controller

    def cycle_strip_colors(self, delay: int = 1) -> None:
        '''Cycle through strip colors (blue, red, green) while turning off the matrix.'''
        for color_name in ['blue', 'red', 'green']:
            print(f'Setting strip color to {color_name}')
            self.controller.strip_led
                matrix_value = '_join([*0, 0, 0*] * self.controller.matrix_size)
                self.controller.send_ind_command(strip_values, matrix_values)
                print(f'Error sending command: {e}')
                time.sleep(delay)

    def cycle_matrix_colors(self, delay: int = 1) -> None:
        '''Cycle through matrix colors (blue, red, green) while turning off the strip.'''
        for color_name in ['blue', 'red', 'green']:
            matrix_value = '_join([*0, 0, 0*] * self.controller.matrix_size)
self.controller.send_ind_command(strip_values, matrix_values)
            print(f'Matrix color set to {color_name}.')
            time.sleep(delay)

    def alternating_wave_blue(self, delay: int = 1) -> None:
        '''Alternate between blue and red strip colors in a wave pattern.'''
        wave_strip = self.controller.strip_led * self.controller.strip_leds
        wave_matrix = self.controller.get_color_value('blue', 'matrix')
        wave_matrix = self.controller.get_color_value('blue', 'matrix')
        wave_strip = self.controller.get_color_value('red', 'strip')
        wave_matrix = self.controller.get_color_value('red', 'matrix')

        for i in range(strip_leds):
            strip_values = '_'.joint([wave_strip if i % 2 == 0 else wave_strip for i in
range(self.controller.strip_leds)])
            matrix_pattern = '_'.joint([wave_matrix if i % 2 == 0 else blue_matrix for i in
range(self.controller.matrix_size)])

        self.controller.send_ind_command(strip_pattern, matrix_pattern)
        #print(f'Alternating waive and blue pattern displayed.')
        time.sleep(delay)

    def toggle_wave_blue(self, cycles: int = 1, delay float = 1.0) -> None:
        '''Toggle between waive and blue colors for a specified number of cycles.'''
        wave_strip = self.controller.get_color_value('wave', 'strip')
        wave_matrix = self.controller.get_color_value('wave', 'matrix')
        blue_strip = self.controller.get_color_value('blue', 'strip')
        blue_matrix = self.controller.get_color_value('blue', 'matrix')

        for i in range(cycles):
            strip_values = '_'.joint([wave_strip if i % 2 == 0 else blue_strip for i in range(
'_'.joint([blue_strip * self.controller.strip_leds
            matrix_values = '_'.joint([wave_matrix * self.controller.matrix_size] if i % 2 == 0 else
'_'.joint([blue_matrix * self.controller.matrix_size]
            self.controller.send_ind_command(strip_values, matrix_values)
            #print(f'Cycle {i + 1}/{cycles} completed.')
            time.sleep(delay)

    def fade_transition(self, start_color_str, end_color_str str, steps: int) -> list:
        '''Generate a fade transition between two colors.'''
        start_rgbw = [int(x.strip()) for x in start_color_str.split(',')]
        end_rgbw = [int(x.strip()) for x in end_color_str.split(',')]

        return [
            '_'.joint([int(start + (end - start) * i / steps) for start, end in zip(start_rgbw,
end_rgbw)]
            for i in range(steps + 1)
        ]

    def toggle_blue_green_fade(self, cycles: int = 1, fade_steps: int = 75, total_fade_duration: float
= 1.0) -> None:
        '''Fade between blue and green colors for the specified number of cycles.'''
        blue_strip = self.controller.get_color_value('blue', 'strip')
        blue_matrix = self.controller.get_color_value('blue', 'matrix')
        green_strip = self.controller.get_color_value('green', 'strip')
        green_matrix = self.controller.get_color_value('green', 'matrix')

        delay_per_step = total_fade_duration / fade_steps

        for cycle in range(cycles):
            #print(f'Making cycle {cycle + 1}/{cycles}')

            # Blue to green fade
            fade_strip_values = self.fade_transition(green_strip, blue_strip, fade_steps)
            fade_matrix_values = self.fade_transition(green_matrix, blue_matrix, fade_steps)

            for step_index, (strip_value, matrix_value) in zip(fade_strip_values, fade_matrix_values):
                strip_values = '_'.joint([strip_value * self.controller.strip_leds]
                matrix_values = '_'.joint([matrix_value * self.controller.matrix_size]
                self.controller.send_ind_command(strip_values, matrix_values)
                time.sleep(delay_per_step)

            # Green to blue fade
            fade_strip_values = self.fade_transition(green_strip, blue_strip, fade_steps)
            fade_matrix_values = self.fade_transition(green_matrix, blue_matrix, fade_steps)

            for step_index, (strip_value, matrix_value) in zip(fade_strip_values, fade_matrix_values):
                strip_values = '_'.joint([strip_value * self.controller.strip_leds]
                matrix_values = '_'.joint([matrix_value * self.controller.matrix_size]
                self.controller.send_ind_command(strip_values, matrix_values)
                time.sleep(delay_per_step)

    def draw_name_with_pixels(self, name str <= 3) -> None:
        '''Draw (MAX) 3 on the LED matrix with zigzag column-row mapping.'''
        matrix_size = self.matrix_size
        letter_index = {'A': 0, 'B': 1, 'C': 2}

        big_rgb = topleftmap(left, self.controller.get_color_value(background_color, 'matrix').split(',', ))

        letter_rgb = topleftmap(left, self.controller.get_color_value(letter_color, 'matrix').split(',', ))

        img = Image.new('RGB', (self.controller.matrix_width, self.controller.matrix_leu), color=big_rgb)

        draw = ImageDraw.Drawing(

        # Letter positions
        letter_positions = {
            'A': [(1,0),(0,1),(2,1),(0,2),(1,2),(2,2),(0,3),(2,3),(0,4),(2,4)],
            'B': [(0,0),(1,0),(0,1),(2,1),(0,2),(1,2),(0,3),(2,3),(0,4),(1,4)],
            'C': [(1,0),(2,0),(0,1),(0,2),(0,3),(1,4),(2,4)],
        }

        letter_positions = {'0': [], '1': [], '2': [], '3': []}

        for letter, y_offset in letter_positions.items():
            for row, pattern in enumerate(letter_patterns[letter]):
                for col, pixel in enumerate(pattern):
                    if pixel == '1':
                        # Calculate position with zigzag mapping
                        img.putpixel((x_offset + col, row + vertical_offset), letter_rgb)

        # Convert image to zigzag LED value
        matrix_values = []
        for y in range(5):
            for x in range(self.controller.matrix_width):
                rgb = img.getpixel((x, y))
                matrix_values.append('_'.joint(map(str, rgb)))

        matrix_value_str = '_'.joint(matrix_values)
        strip_value = '_'.joint([*0, 0, 0*] * self.controller.strip_leu)

        self.controller.send_ind_command(strip_value, matrix_value_str)
        #print(f'Displayed {name} on matrix.')
        time.sleep(delay)

    def __list_ind(self, controller=LEDController):
        self.controller = controller

    def draw_scrolling_time_ascii(self, delay float = 0.1, duration int = 30) -> None:
        '''
        Scrolls the current time and '(MAX)' text across the LED matrix for a specified duration.

        # Colors
        background_color = 'blue'
        text_color = 'white'

        # Get RGB values using the controller's method
        bg_rgb = topleftmap(left, self.controller.get_color_value(background_color, 'matrix').split(',', ))

        text_rgb = topleftmap(left, self.controller.get_color_value(text_color, 'matrix').split(',', ))

        # Set time zone (adjust as needed)
        time_zone = ZoneInfo('America/New_York')

        # Character patterns (5x5 grid)
        char_patterns = {
            '0': [(1,0),(2,0),(0,1),(3,1),(0,2),(3,2),(0,3),(3,3),(1,4),(2,4)],
            '1': [(1,0),(0,1),(1,1),(1,2),(1,3),(0,4),(1,4),(2,4)],
            '2': [(1,0),(2,0),(0,1),(3,1),(3,2),(2,3),(1,4),(0,4),(3,4)],
            '3': [(1,0),(2,0),(0,1),(3,1),(2,2),(3,3),(0,4),(1,4),(2,4)],
            '4': [(2,0),(1,1),(3,1),(0,2),(2,2),(3,2),(2,3),(2,4)],
            '5': [(0,0),(1,0),(2,0),(3,0),(0,1),(0,2),(1,2),(2,2),(3,3),(0,4),(1,4),(2,4)],
            '6': [(1,0),(2,0),(0,1),(0,2),(1,2),(2,2),(0,3),(3,3),(1,4),(2,4)],
            '7': [(0,0),(1,0),(2,0),(3,0),(3,1),(2,2),(1,3),(0,4)],
            '8': [(1,0),(2,0),(0,1),(3,1),(1,2),(2,2),(0,3),(3,3),(1,4),(2,4)],
            '9': [(1,0),(2,0),(0,1),(3,1),(3,2),(1,2),(2,2),(3,3),(1,4),(2,4)],
            '(': [(2,0),(1,1),(0,2),(0,3),(1,4),(2,4)],
            'M': [(0,0),(4,0),(0,1),(1,1),(3,1),(4,1),(0,2),(2,2),(4,2),(0,3),(4,3),(0,4),(4,4)],
            'A': [(1,0),(2,0),(0,1),(3,1),(0,2),(3,2),(0,3),(3,3),(0,4),(3,4)],
            'X': [(0,0),(4,0),(1,1),(3,1),(2,2),(1,3),(3,3),(0,4),(4,4)],
            ')': [(2,0),(3,1),(4,2),(4,3),(3,4),(2,4)],
            ':': [(2,1),(2,3)],
            ' ': [],
            '-': [(1,2),(2,2),(3,2)]
        }

        end_time = time.time() + duration
        while time.time() < end_time:
            # Get the current time in ss.ss.'12:45'
            current_time = datetime.datetime.now(time_zone).strftime('%S%M')

            # Message to scroll
            full_message = f'{current_time}  (MAX)  '

            # Create scrollable image (width depends on characters)
            text_width = len(full_message) * 6  # 6 each character (5 + 1 pixels) + 1 pixel space
            img = Image.new('RGB', (text_width, 5), color=bg_rgb)
            draw = ImageDraw.Draw(img)

            # Draw characters on the image
            x_offset = 0
            for char in full_message:
                if char in char_patterns:
                    pattern = char_patterns[char]
                    for row in range(5):
                        for col in range(5):
                            if (col, row) in pattern:
                                img.putpixel((x_offset + col, row), text_rgb)  # Pixel set to

                x_offset += 6  # Move to the next character

            # Scroll the image left pixel-by-pixel
            for shift in range(text_width - self.controller.matrix_width + 1):
                frame = img.crop((shift, 0, shift + self.controller.matrix_width, 5))

                # Convert to RGB values for LED mapping
                matrix_values = []
                matrix_value = []
                for y in range(self.controller.matrix_width):
                    for x in range(self.controller.matrix_width):
                        rgb = frame.getpixel((x, y))
                        matrix_value.append('_'.joint(map(str, rgb)))
                        if len(matrix_value) == 5:
                            matrix_values.extend(matrix_value)
                            matrix_value = []

                # Send to LED
                matrix_value = '_'.joint(matrix_values)
                strip_value = '_'.joint([*0, 0, 0*] * self.controller.strip_leu)
                self.controller.send_ind_command(strip_value, matrix_value_str)
                time.sleep(delay)

            # Pause before next scroll cycle
            time.sleep(1.0)

        #print('Completed scrolling display.')

class LEDCampaign:
    def __init__(self, controller=LEDController, campaign_api_key=str):
        self.controller = controller
        self.campaign_api_key = campaign_api_key # Campaign data API key
        self.campaign_base_url = f'https://app.nano.fun/api/{ campaign_api_key}' # Campaign data API base URL
def campaign_spending_matrix_update(self, delay int = 1) -> None:

    '''
    Change matrix colors based on campaign spending and daily utilization.
    - Top 4 bottom rows: row color chosen on spending difference
    - Middle rows: row color based on utilization
    - Middle rows: row color chosen on spending
    - Centered smiley face (happy/sad/neutral) with row_color

    '''
    spending, campaign, dishonoured, difference = self.get_data()[ control=False]
    #print(f'{*data}: {spent}/{dishonoured}: {difference}, {f}')dishonoured:
{difference}, {f}')difference: {difference}, {f}')
    # this part is to test all the functionality of the code.
```

light-of-information-2025-team_8

This project controls two sets of LEDs—an LED strip and an LED matrix—using Python and a RESTful API. It demonstrates various lighting patterns, animations, and text displays. Additionally, it fetches campaign finance data from the OpenFEC API to dynamically set the LED matrix colors based on spending differences and party affiliation.

Key Features

- **Synchronous/Asynchronous LED Control**
  Control color changes on the LED strip and matrix in tandem or independently.
- **Text and Animations on the Matrix**
  Display static or scrolling text (e.g., "UMSI" or the current time), using the Pillow library (`PIL`).
- **Color Patterns**
  - **Alternating Maize and Blue**
    Sets every other LED to maize or blue.
  - **Toggle Maize and Blue**
    Switches the entire strip and matrix between maize and blue for a specified number of cycles.
  - **Fading Transitions**
    Smoothly fades between colors over a given number of steps.
- **Campaign Data Integration**
  Uses the OpenFEC API to:
  1. Retrieve federal campaign contributions, disbursements, and party affiliation.
  2. Dynamically change matrix colors (and display a smiley face) based on the spending difference.

Dependencies

Make sure you have the following libraries installed:

- **requests**: For making HTTP POST/GET requests to the LED API and the FEC API.
- **datetime** / **time** / **zoneinfo**: For time handling and scheduling delays.
- **Pillow (PIL)**: For drawing text and images (used for generating the "UMSI" letters and scrolling text).
- **Python 3.9+** (or any version that supports `zoneinfo`).

You can install these using:

```
pip install requests Pillow
```

(Ensure you're running a Python version that includes `zoneinfo`, typically 3.9+.)

How It Works

LEDController Class

```
class LEDController:
    def __init__(self, device_name: str, api_key: str, strip_len: int = 300, matrix_len: int = 8, matrix_width: int = 32):
        ...
```

- **Purpose**: Handles low-level operations to send commands to the LED strip and matrix via a RESTful API.
- **Key Methods**:
  - `send_led_command(strip_values: str, matrix_values: str)`: Sends color data to the device.
  - `turn_off_led()`: Turns off all LEDs.
  - `get_color_value(color_name: str, led_type: str) -> str`: Retrieves the appropriate RGB or RGBA values for a given color.

LEDRequirements Class

```
class LEDRequirements:
    def __init__(self, controller: LEDController):
        ...
```

- **Purpose**: Implements various lighting patterns and animations.
- **Key Methods**:
  1. `cycle_strip_colors(delay: int = 1)`: Cycles the strip through a set of colors while the matrix is off.
  2. `cycle_matrix_colors(delay: int = 1)`: Cycles the matrix through a set of colors while the strip is off.
  3. `alternating_maize_blue(delay: int = 5)`: Sets alternating maize and blue LEDs on both the strip and matrix.
  4. `toggle_maize_blue(cycles: int = 5, delay: float = 1.0)`: Toggles the entire strip/matrix between maize and blue.

5. `fade_transition(start_color_str: str, end_color_str: str, steps: int) -> list`: Helper method that returns a list of colors for smooth fading transitions.
6. `toggle_maize_green_fade(cycles: int = 5, fade_steps: int = 20, total_fade_duration: float = 1.0)`: Fades between maize and green multiple times.
7. `draw_umsi_with_pil(delay: int = 5)`: Uses `PIL` to draw the letters "UMSI" onto an 8x32 matrix image, then sends it to the LEDs.
8. `draw_scrolling_time_umsi(delay: float = 0.2, duration: int = 30)`: Continuously scrolls the current time and the text "UMSI" across the matrix.

LEDRCompaign Class

```
class LEDRCompaign:
    def __init__(self, controller: LEDController, campaign_api_key: str):
        ...
```

- **Purpose**: Integrates with the [OpenFEC API]() to retrieve real-time campaign finance data and display a corresponding color pattern on the LED matrix.
- **Key Methods**:
  1. `campaign_spending_matrix_colors(delay: int = 1) -> None`:
     - Fetches candidate party affiliation, receipts, disbursements, and calculates the difference.
     - Sets matrix colors based on the party and spending difference.
     - Draws a smiley face (happy, sad, or neutral) in the center.
  2. `get_federal_contributions(...)`: Retrieves top candidates based on receipts, disbursements, etc.
  3. `get_candidate_party(candidate_id: str)`: Fetches party affiliation for a given candidate.

Usage

1. **Clone or Download** the repository (or copy the code into a `.py` file).
2. **Install Dependencies**:

3. `pip install requests Pillow`
4. **Set API Keys**:
   - Replace `"YOUR_LED_API_KEY"` with the actual API key for your LED device.
   - Replace `"YOUR_CAMPAIGN_API_KEY"` with your [OpenFEC API]() key.
5. **Run the Code**:

6. `python main.py`
7. **Observe** the LED strip and matrix:

    - They will cycle through various patterns and eventually display the campaign spending results.

Configuration

- **LED Strip Length**: Set `strip_len` in `LEDController` if you have a strip of a different length.
- **Matrix Dimensions**: Adjust `matrix_len` (rows) and `matrix_width` (columns) to match your LED matrix specifications.
- **Colors**: The dictionary definitions in `LEDController` can be customized to your preferred colors or to match your LED capabilities.
- **FEC API Parameters**: In `LEDRCompaign`, you can customize the `office`, `election_year`, and other parameters to fetch data for different elections or offices.