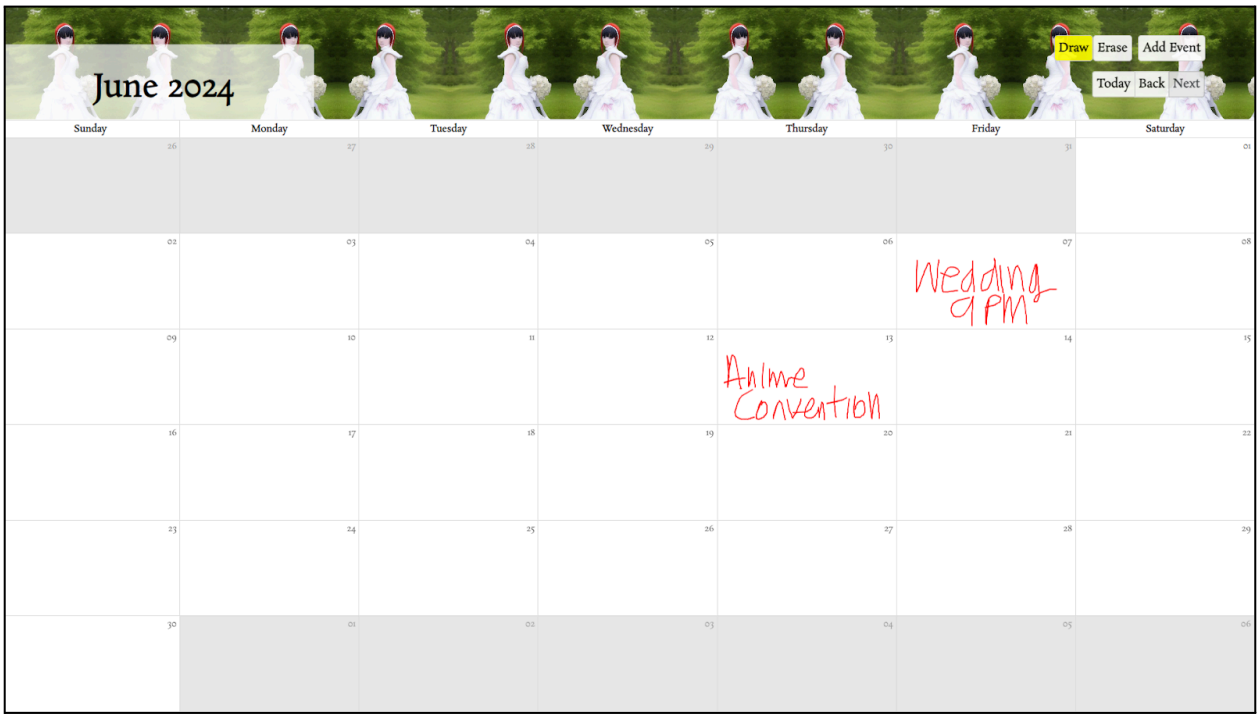


CalPal: A Multimodal Digital Wall Calendar

by

Devin Murphy and Jenny Moralejo

Abstract

Mobile calendaring platforms provide unparalleled convenience and accessibility, yet often lack the tactile engagement and personalized touch of physical mediums. This paper introduces CalPal, an innovative multimodal digital wall calendar designed to bridge this gap by blending digital convenience with the benefits of traditional calendars. CalPal integrates pen-based event tracking, seamless synchronization with Google Calendar, intuitive gesture-based navigation, and dynamic theming based on monthly events, distinguishing it from existing digital wall calendar solutions. Our system on average is able to detect the navigation gestures (flipping forward and flipping backward) with 98% and 87% accuracy respectively. Through a comprehensive user study, we also demonstrate user satisfaction with CalPal's intuitive interaction modalities and custom theming. Overall, these results indicate that CalPal represents a promising approach to harmonizing the strengths of digital and traditional calendaring practices, thereby providing users with a more holistic and insightful scheduling tool.

CalPal: A Multimodal Digital Wall Calendar

Devin Murphy
Massachusetts
Institute of Technology
devinmur@mit.edu

Jenny Moralejo
Massachusetts
Institute of Technology
moralejo@mit.edu

Abstract

Mobile calendaring platforms provide unparalleled convenience and accessibility, yet often lack the tactile engagement and personalized touch of physical mediums. This paper introduces CalPal, an innovative multimodal digital wall calendar designed to bridge this gap by blending digital convenience with the benefits of traditional calendars. CalPal integrates pen-based event tracking, seamless synchronization with Google Calendar, intuitive gesture-based navigation, and dynamic theming based on monthly events, distinguishing it from existing digital wall calendar solutions. Our system on average is able to detect the navigation gestures (flipping forward and flipping backward) with 98% and 87% accuracy respectively. Through a comprehensive user study, we also demonstrate user satisfaction with CalPal's intuitive interaction modalities and custom theming. Overall, these results indicate that CalPal represents a promising approach to harmonizing the strengths of digital and traditional calendaring practices, thereby providing users with a more holistic and insightful scheduling tool.

1. Introduction and Related Work

In recent years, traditional methods of organizing schedules, such as wall calendars and paper planners, have seemingly been overshadowed by the widespread adoption of mobile calendaring applications like Google Calendar and iCal. This shift can be attributed largely to the convenience afforded by digital platforms, allowing for instantaneous scheduling and effortless sharing across devices. However, despite the undeniable utility of digital calendars, there are inherent qualities of physical mediums that are absent in their digital counterparts. For instance, the tactile experience of writing on a paper calendar offers a sense of natural interaction and enables personalized annotations beyond the limitations of a standard keyboard. Moreover, research may indicate that users of paper calendars may experience a greater likelihood of plan execution, attributed to the medium's encouragement of a more holistic, big-picture approach to scheduling [4].

In light of these observations, we propose "CalPal," a multimodal digital wall calendar. CalPal combines the accessibility of digital calendars with the tangible benefits of traditional calendaring practices, offering features such as pen-based event tracking and display, synchronization with Google Calendar, intuitive gesture-based navigation, and generative theming based on the events in a month. In many commercially available digital wall calendars, such as the Mango Display [3] and DAKboard [2], we have found that interaction was mainly limited to display and touch, while CalPal also allows for interaction through the use of gesture. Additionally, these digital wall calendars do not allow for mobile calendar sync through recognition of the user's own handwriting. This is an important distinction, as research has shown handwriting results in better memory and therefore may be a more effective means of scheduling [7]. Finally, while the Mango Display and DAKboard have customizable themes, they do not automatically update to reflect the events of the current month. As noted by the author of [5], it feels like a missed opportunity that we rarely use our calendars to reflect on our past or present. Through CalPal's generative theming, we hope to provide a more insightful calendaring experience for users.

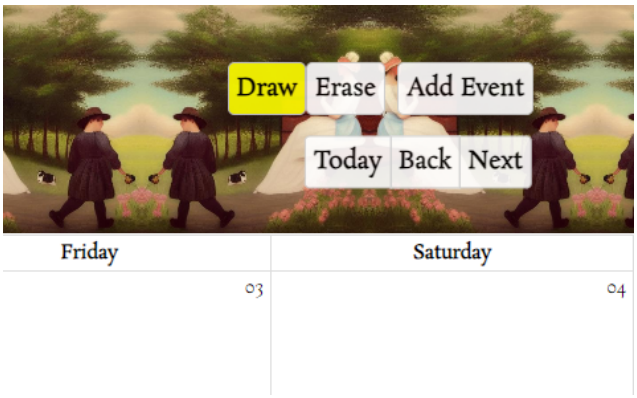
2. System Functionality

Our system is a digital wall calendar that supports touch based calendar and event manipulation, gesture based calendar flipping, event based generative calendar theming, online calendar syncing, and state maintenance.

2.1. Touch Based Functionality

Through touch, our system allows the user to interact with the calendar. The functionality currently supported includes: writing on the calendar, erasing writing, adding events, removing events, and flipping through calendar

Figure 1. Calendar Application Buttons



This figure shows the buttons of the calendar application interface. The "Draw" and "Erase" buttons allow the user to write and erase while the "Add Event" button adds the current writing as an event. "Today", "Back", and "Next" when toggled switch the month view accordingly.

months. This is interfaced through the button menu of the application as seen in Figure 1.

In order to write events on the calendar, the user can select the "Draw" button and begin drawing on the screen. Should they not like what they have written, they can select the "Erase" button which will automatically erase what they had last written.

In order to add events to the calendar the user must write in the event date box that the event should be scheduled for. Should the user be satisfied with the event they have written on the calendar, by tapping the "Add Event" button the event is added to the calendar. Should the event writing be unreadable, the user will be notified with an error asking them to try again. The system supports the addition of all-day events as well as timed events.

In order to remove events from the calendar after they have been added, the user can select "Erase", then tap anywhere on the text of the event they would like to remove, and the event will be removed. In "Erase" mode the user is unable to draw on the calendar.

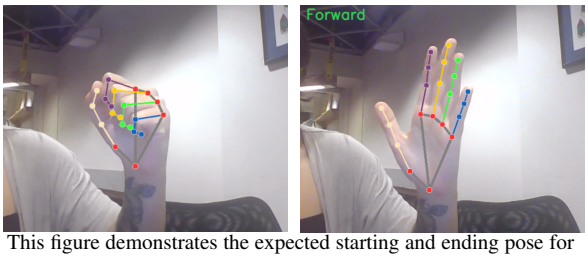
The user is additionally able to flip backwards and forwards through the calendar pages by tapping "Back" and "Next" respectively. Tapping on the "Today" button resets the calendar view back to the current month.

2.2. Gesture Based Functionality

In addition to using buttons to flip through calendar pages, we also support using gestures to flip through these pages. Currently the system supports a flip forward and flip backward gesture. The user is able to flip forward by first holding their hand upright in a closed fist and then opening their hand as shown in Figure 2. The user is able to flip backward

Figure 2. Forward Gesture Poses

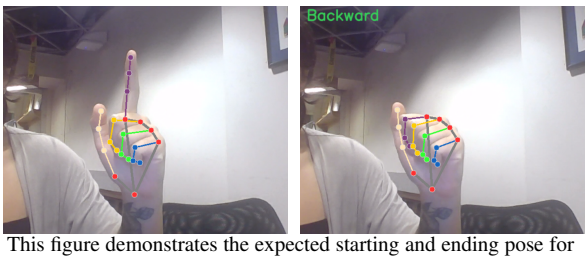
(a) Forward Gesture Start (b) Forward Gesture End



This figure demonstrates the expected starting and ending pose for forward gesture detection.

Figure 3. Backward Gesture Poses

(a) Backward Gesture Start (b) Backward Gesture End



This figure demonstrates the expected starting and ending pose for backward gesture detection.

by first holding their hand up right with only their pointer finger extended and then closing their fist as shown in Figure 3. When either the flip forward or flip backward gesture is completed as described above, the calendar page flips forward or backwards a page accordingly.

2.3. Generative Calendar Theming

The art shown in the calendar banner as seen at the top of Figure 4 is updated to reflect the events on the calendar. Every time an event is added or deleted, this art changes.

2.4. Online Calendar Syncing

Every time an event is added or deleted from the application, these changes are also reflected in the linked Google Calendar configured upon application start up.

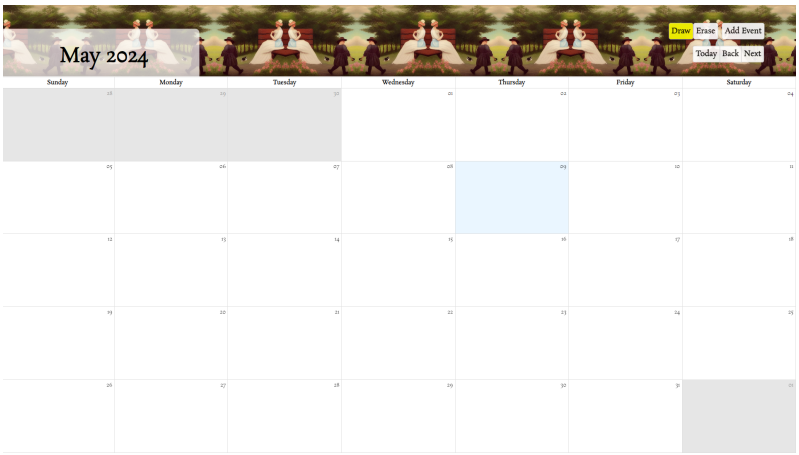
2.5. State Maintenance

Finally, the system is able to maintain 12 months worth of state, allowing for consistency upon system start up and shut down, as well as flipping between month to month.

2.6. Example Use Case

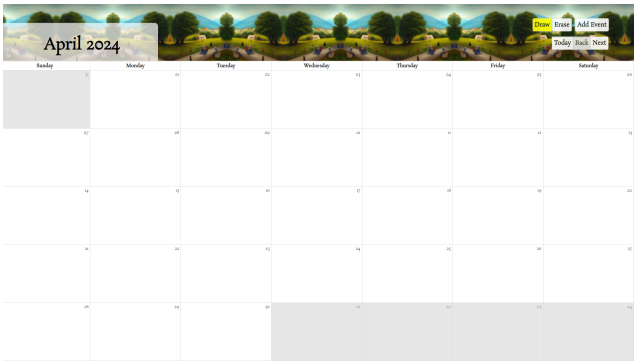
Upon loading up the calendar application, the user is greeted with the current month view of the application as seen in Figure 4.

Figure 4. Current Month View: May



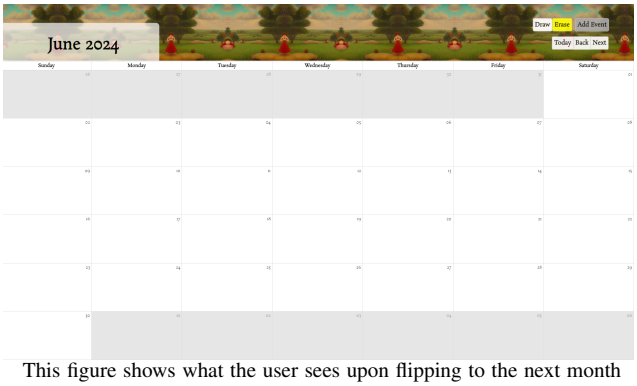
This figure shows the current month view of the calendar with no events written on it.

Figure 5. April Month View



This figure shows what the user sees upon flipping to the previous month of the calendar (given the current month is May.)

Figure 6. June Month View



This figure shows what the user sees upon flipping to the next month of the calendar (given the current month is May.)

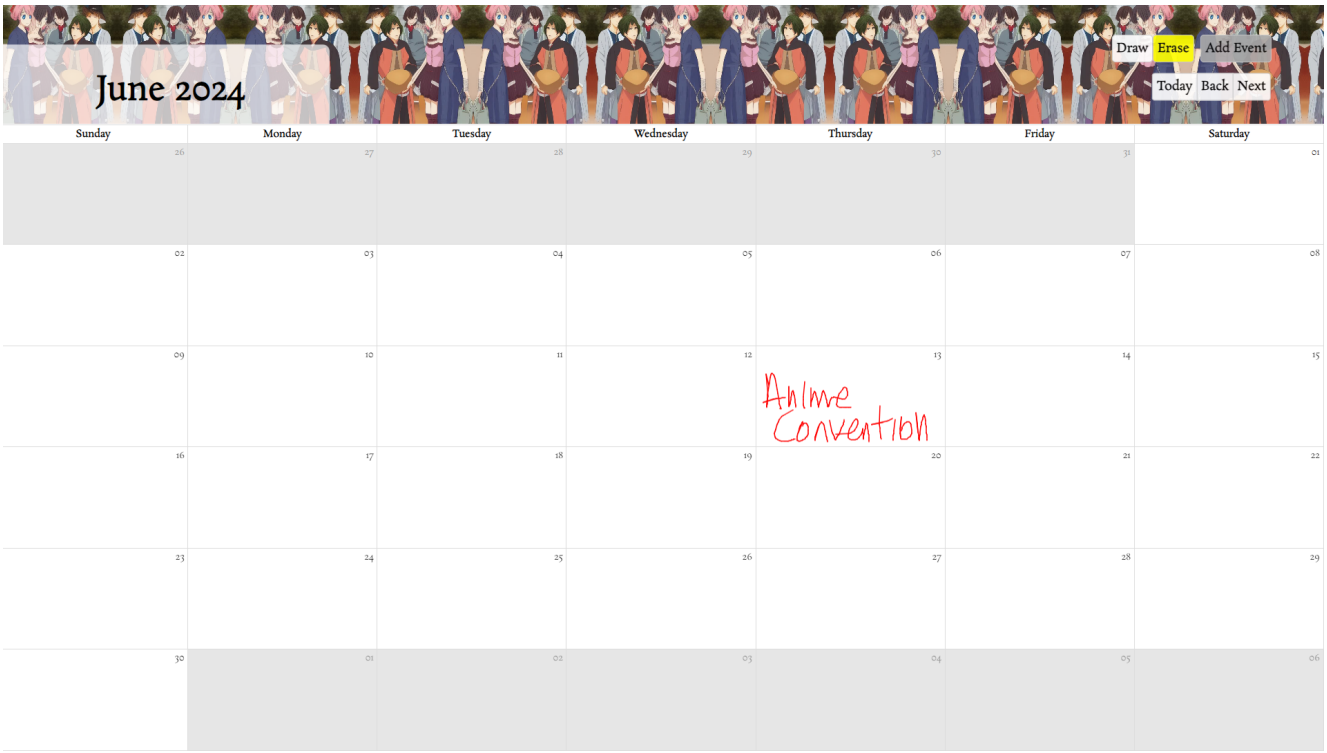
The touch based functionality is supported through the user touching and writing on the screen. The user, using gestures or buttons, flips backwards to the previous month of April shown in in Figure 5. The user then flips forward (with either gesture or buttons) to the month of June shown in Figure 6. Each of these month views, even while empty have distinct theming representative of the month.

The user has an all day Anime Convention they would like to attend on June 13th. The user grabs the touch stylus, making sure they are in ”Draw” mode, and writes Anime Convention in the June 13th box. The user then clicks the ”Add Event button”. Rapidly, the user sees the generative theming change from the summer garden view in Figure 6 to a banner with anime characters. User input and the changes in generative theming are shown in Figure 7. The user sees the event show up in their linked Google calendar scheduled as an all day event as well in Figure 8.

The user receives a call from their friend notifying them that they are getting married on June 7th at 10PM. The user scribbles ”Wedding 8PM” in their calendar. Mid call, the friend notifies the user that the event actually starts at 9PM. The user simply clicks erase to completely erase what they had just written changing the screen to the one shown in Figure 7. The user now rewrites ”Wedding 9PM”. Upon touching the ”Add Event” button, the user sees the generative theming update to include an anime character in a wedding dress as shown in 9. The user opens their linked Google calendar and sees the event ”Wedding” scheduled for 9PM in Figure 8.

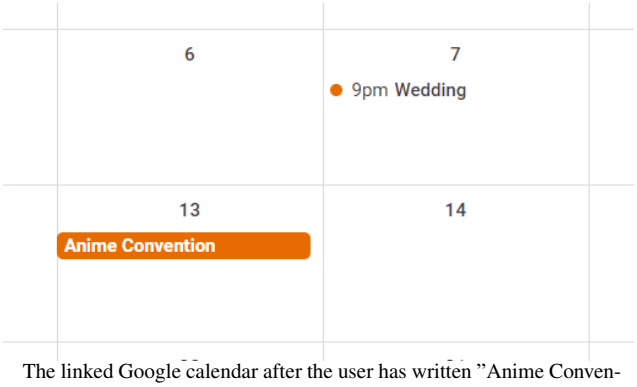
It turns out the user had mistakenly scheduled these events in June when they should have been scheduled for July. The user clicks on the ”Erase” button and clicks anywhere on the text they had written to delete both ”Anime Convention” and ”Wedding 9PM”. The calendar is now

Figure 7. Adding All Day Event: Anime Convention



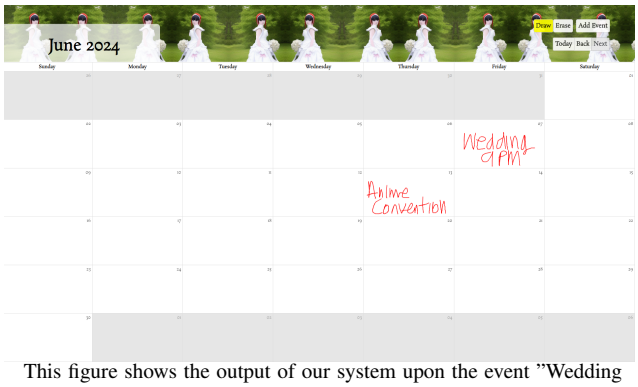
This figure shows the output of our system upon the event "Anime Convention" being written on the calendar for June 13th. The system updates the themed banner at the top of the month to include anime figures.

Figure 8. Google Calendar Sync



The linked Google calendar after the user has written "Anime Convention" and "Wedding 9PM" on the calendar and has added both events.

Figure 9. Adding Timed Event: Wedding



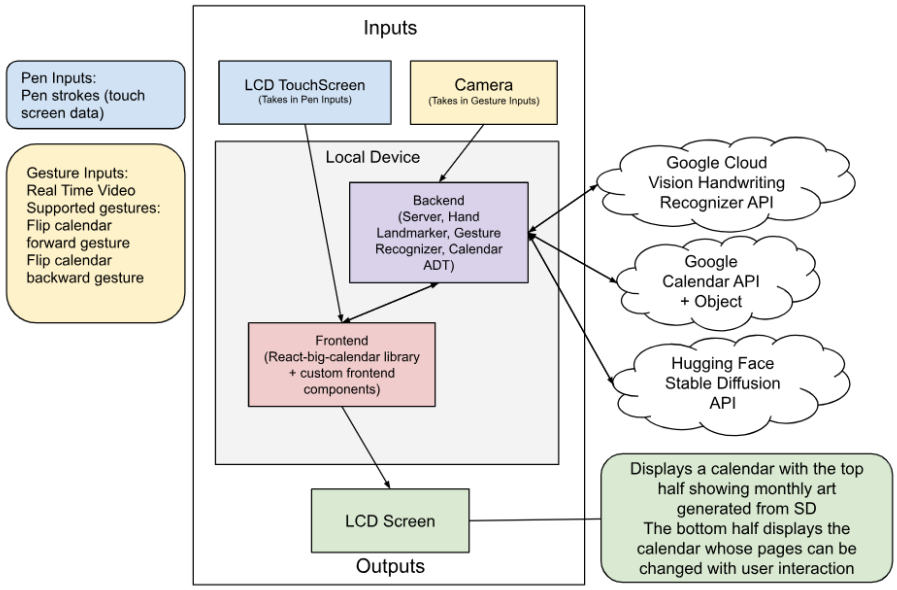
This figure shows the output of our system upon the event "Wedding 9PM" being written on the calendar for June 7th. The system updates to integrate the new event with the existing ones to create a new theme. The banner now displays an anime character in a wedding dress.

empty and reverts back to the view in Figure 6. The user checks their Google calendar and sees the events have now been completely deleted.

3. System Design and Architecture

We decided to prototype CalPal as a fullstack web application using ReactJS to develop the frontend served by a Python Flask backend. This allowed us to rapidly prototype a user interface and maintain a stateful application which could process multimodal inputs as seen in Figure 10. Overall our system takes in two types of inputs: pen based and gesture based to manipulate the calendar. The pen based

Figure 10. Overall architecture Diagram



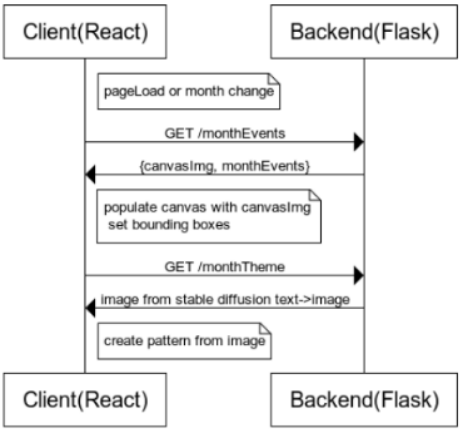
This is the overall architecture diagram of our system. The system takes in touch inputs from the touch screen and gesture inputs from the camera which get passed into the front end and backend respectively. These inputs are processed and outside libraries are queried to update the state of both the back and frontend to create a calendar display with the top banner displaying generative art based on the calendar events and the bottom calendar displaying the handwritten events.

input is passed to the front-end, while the gesture based input is passed to the backend. These two components work together to pass information while querying outside libraries such as the Google Cloud Vision API for handwriting recognition, Google Calendar API for event syncing, and Hugging Face API for stable diffusion generative calendar theming, to render and change the display based on user input.

3.1. Frontend

The CalPal frontend was created using ReactJS and the react-big-calendar library. It is responsible for 3 main tasks: managing the calendar display, looking for gesture input, and processing touch inputs to add or remove events. To manage the calendar display, we get the necessary information for rendering the calendar view from the back-end through GET requests (/monthEvents and /monthTheme) and update the view when the page loads or month changes. This process is shown in Figure 11. The calendar theming is updated by repeating the stable-diffusion generated image in the horizontal direction, as seen in figures 7 and 9. The main information necessary for rendering the calendar view is an image displaying the month's saved events in the user's handwriting, and bounding boxes representing the area of the canvas HTML element that each individual event occupies. The bounding boxes are necessary for event modification, which we explore in more detail later in this section.

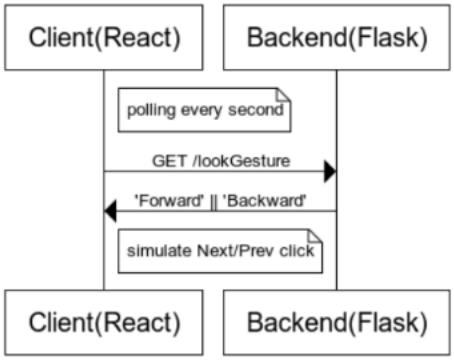
Figure 11. Sequence diagram for updating the calendar display



To look for gesture input, our frontend sends a GET request (/lookGesture) to the backend every second to check if a flip forward or flip backward gesture has been made (Figure 12). If so, it updates the month view accordingly. The polling interval of one second was decided on mainly through trial and error, the trade-off with increased polling

frequency being reduced latency between gesture production and visual feedback of the calendar updating, but less responsiveness to touch input due to the overactive network. In future work, this tradeoff can be avoided through the use of websockets to enable bi-directional communication between the frontend and backend, or by compiling the React frontend to a static site and serving it from the same port as the Flask backend.

Figure 12. Front-end looks for gesture from backend every 1 second

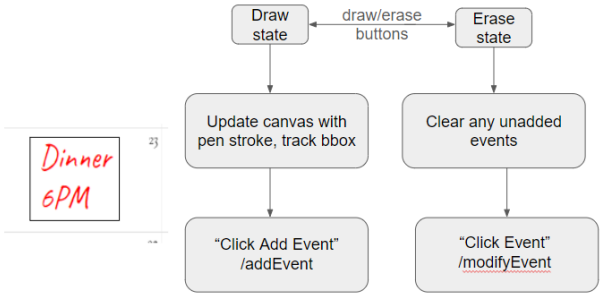


Finally, the frontend maintains a draw state and erase state for helping manage the touch based input from the user. When in the draw state, the canvas updates with the user's stroke, and a bounding box is calculated based on their minimum/maximum X and Y coordinates. When the "Add Event" button is clicked, the current canvas is downloaded as a png image and sent to the backend along with the bounding box for the event, which the backend then uses for month state management and to add the event to Google Calendar. In the case that the backend is unable to detect an event from the written text, the user is notified using a window alert prompting them to try re-adding the event, and the unidentified stroke is subsequently deleted. Upon clicking the "Erase" button, the frontend enters the erase state, and any strokes from unadded events are removed from the canvas. If an event's bounding box is touched while the frontend is in the erase state, the event is removed from the canvas and a request is sent to the backend to remove the event from Google Calendar. This process is illustrated in Figure 13

3.2. Backend

Our backend consists of 3 main parts: the server, the calendar abstract data type, and the gesture recognizer as shown in Figure 14.

Figure 13. State machine for managing touch based inputs from the user, along with an illustrated example of an event bounding box



3.2.1 Server

Upon start up, the server is responsible for multiple things including, loading in the previous calendar object for state maintenance as well as setting up the token credentials for outside library API access. State is being maintained through the pickling of the backend calendar object every time an event is added or deleted from the calendar.

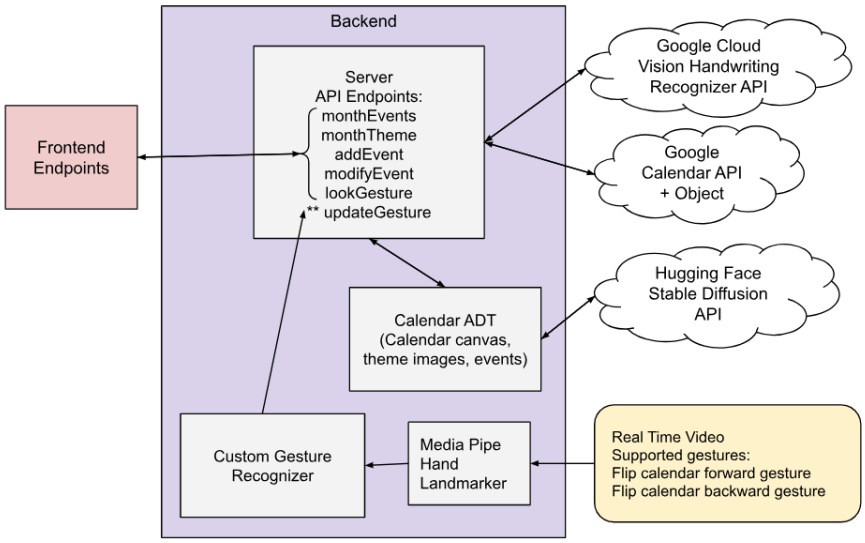
The server contains multiple API endpoints. The endpoints monthEvents (given a month, returns the month's canvas and event bounding boxes), monthTheme (given a month, returns the month's theme image), addEvent (given the user handwriting, updates the backend state and returns the event id), and modifyEvent (given the month, event id, and canvas, deletes the event from the backend state), and lookGesture (returns the output of the gesture recognizer) are queried by the frontend.

Whenever the addEvent endpoint is queried, the server makes a call to the Google Cloud Vision API to map the handwriting provided by the frontend to machine readable text. The decoded event name is passed into the Google Calendar API to create an event in the linked Google Calendar returning an event id linking the event in our backend to the event in the Google calendar. The decoded event name along with the event id is then passed in to the backend Calendar object to update and maintain state.

Whenever the deleteEvent endpoint is queried with an event id, the server, in addition to updating the calendar object, makes a call to the Google Calendar API to delete the event from the linked online calendar.

Additionally, the server also maintains state on whether the frontend has received the output of the gesture recognizer (via the lookGesture endpoint). The final API endpoint is queried by the gesture recognizer to update this state to Forward or Backward whenever a gesture is detected. After the lookGesture event is queried to get the result of the recognizer, this state is reset to None.

Figure 14. Backend Architecture Diagram



This figure shows a more detailed view of the backend architecture. The backend consists of the parts in this figure: the server which responds to frontend and gesture recognizer requests by calling outside APIs for handwriting detection and digital calendar syncing and querying the calendar object, the calendar object which maintains the state of the calendar and calls the Hugging Face API to generate it's theme, and the gesture recognizer which takes in real time video that is passed through the Media Pipe hand landmarker for analysis.

3.2.2 Calendar Abstract Data Type

The calendar ADT is responsible for maintaining the state of the calendar in the backend and is pickled to recover the calendar upon server restart.

The calendar object maintains the following information for 12 months: the month's canvas (ie: the user's hand-written events as seen on the frontend), the month's events (including event ids, names, and bounding boxes), and the month's themes.

The calendar object supports querying it's own information as well as mutating it's state through adding or deleting an event. The server actively queries the calendar object as described above in the server section.

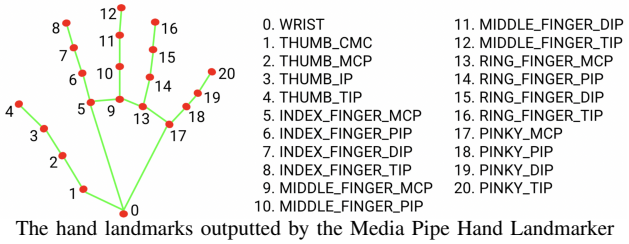
Whenever the state of the calendar object is mutated through adding or deleting an event, the calendar object calls the Hugging Face API to run stable diffusion on the updated set of events in the calendar for that month generating a new month theme image.

3.2.3 Gesture Recognizer

The gesture recognizer takes in real time video and passes it through the locally run Media Pipe Hand Landmarker model to produce hand landmarks as shown in Figure 15.

These hand landmarks are then analyzed frame by frame to classify the hand pose into either one of the two supported gestures' starting or ending poses as shown in 2 and 3 (flip forward or flip backward) or nothing.

Figure 15. Hand Landmarks



The hand landmarks outputted by the Media Pipe Hand Landmarker model.

If a starting pose has been detected, the user has a set amount of time to complete the gesture before the gesture recognizer resets and starts to look for a new gesture being performed. If the ending pose for the starting gesture has been detected within that time frame, the gesture recognizer sends an updateGesture request to the server to update the server state. The time frame is set to small enough such that the user is less likely to trigger a misdetection of the system by interpolating other unrelated poses between the start and end pose, however it is still possible for the user to trigger a misdetection through this interpolation as our system only examines the start and end poses of a gesture and not the poses in between.

In order to detect the starting pose for the two supported

gestures, we examine multiple landmarks. First, we detect the hand orientation by looking at the relative angle between the wrist landmark and the middle knuckle landmark (in order for a supported gesture to be detected the hand must be upright.) Then we check to see whether the hand is fully closed (flip forward) or whether the hand is fully closed with the index finger pointing up (flip backward) by analyzing the relative position of fingertips to their knuckles and the distance between the finger tip to the wrist as well as the joint underneath the fingertip to the wrist. If a finger is fully closed while the hand is upright we would expect the y position of the fingertip to be smaller than the y position of the knuckle. If a finger is open, we would expect the distance between the wrist and the fingertip to be greater than the distance between the wrist and the joint underneath the fingertip. We use the distance metric rather than a simple y coordinate check for open fingers to allow for a smaller and more natural range of motion (as distance does not require the fingers to be fully extended.)

We detect the ending pose for the supported gestures by checking whether all finger on the hand are open (flip forward) or whether the hand is fully closed (flip backward) using the same logic as above. The overlap in the start of the flip forward gesture with the end of the flip backward gesture allow for minimal friction when interfacing with our system.

3.3. Hardware

The main hardware that enabled the development of our system was a 15.6 inch Capacitive Touch LCD Screen from Waveshare. We used a standard commercially available capacitive touchscreen stylus for interaction with the display. For our final demo we ran our frontend and backend servers from a single laptop, using the built in webcam of the laptop for gesture recognition. As we will discuss in the Roadblocks and Lessons Learned section, we also attempted to run CalPal as a standalone IoT device using a Raspberry Pi model 4B along with a PiCam 3 module.

4. Roadblocks and Lessons Learned

4.1. Text Recognition

One major hurdle we faced was with the handwriting text recognition model we initially chose. Our intention was to utilize the Tensorflow Handwriting Text Recognition model developed by Scheidel [6]. This model employs two stages: one for detecting individual words on a page and another for recognizing the text within those segmented words. Initially, when testing the model using screenshots of handwritten text from our calendar frontend, we observed what seemed to be satisfactory segmentation. However, a significant issue arose when the model struggled to accurately segment numbers. Despite attempts to adjust inference parameters such as margin and text scale, we realized the recognition quality fell short of our requirements for scheduling events at specific times.

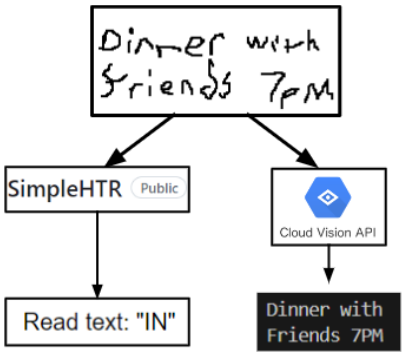


Figure 16. This figure shows the difference in output between the SimpleHTR and Google Cloud Vision handwriting text recogntion models. For our use case, Google Cloud Vision provided much more accurate and consistent segmentations

Subsequently, we explored alternative solutions and discovered that the Document Text Recognition functionalities of the Google Cloud Vision suite provided superior text segmentation. This API accepts an image in base64 encoded format as input and returns a JSON object containing the detected text. Since we were already transmitting the user's handwritten canvas image to the backend as a base64 encoded string, transitioning to this API required minimal code modifications. As a result, we opted to switch to this solution 16.

4.2. Raspberry Pi

We attempted to run CalPal as a standalone IoT device using the Raspberry Pi 4B to host the backend with a PiCam 3 as the camera. The greatest roadblock to this approach was that the Raspberry Pi 4B was unable to run the Media Pipe hand landmarking model fast enough for our use case. At start up there would be around an 8 second lag between a hand first appearing on the screen and the hand landmarks appearing. Once those hand landmarks appeared there would be a 4 second delay between a gesture being completed and it being detected by the system. In order to resolve this we tried decreasing the size of the frame image input to the model in hopes that it would make it run faster to no avail. We also looked into using a USB hardware accelerator; however, the available ones (Coral USB) require significant model modification and set up that we did not have easy access to.

Due to this latency, we made the decision to shift back to using a more powerful computer to run the system. We hope that with more powerful hardware the system will be

able to run as a standalone product.

4.3. The Easy

We found that there were actually many open source or easily accessible AI models for accomplishing the inference tasks we were interested in with CalPal, such as gesture detection and handwriting recognition. As we had some prior experience with web development, using a client server architecture to prototype our system allowed for quick iteration and a natural division of work. Finally, using web frameworks and the open source MediaPipe library for Python allowed us to focus on user interaction rather than the IO of the hardware (for example with the touch input or camera input).

4.4. The Hard

Overall, we found it took longer than expected to go from our proposed functionalities to concrete realization of the architecture that would accomplish them. We certainly learned the merits of breaking up big problems into smaller pieces. Additionally, we found integration of code written separately sometimes time consuming, and recognized the importance of well-defined abstractions. Occasionally, making react-big-calendar work for our use case posed a challenge, as we didn't have direct access to the sub components of the module and therefore had to be creative with css to achieve a desired custom look. Finally, during our preliminary research we found many packages/models that appeared to fit our use case, and vetting these within the context of our system capabilities proved difficult.

5. User Study

We conducted a small-scale usability testing study involving 5 participants, tasking them with interacting with the calendar using the prescribed script:

1. "You can create an all-day event in Google Calendar by entering your event name within a calendar day's boundary and clicking 'Add Event.' Can you attempt to add an all-day event?"
2. "The calendar also accommodates scheduled events by including a standard time, followed by AM or PM, after the event name on a date. Please schedule an event."
3. "To remove an event, switch to the eraser function using the Erase button and touch any part of a scheduled event name. Can you try removing an event?"
4. "To navigate the calendar to the next month, hold your hand upright in a closed fist, then open your hand. To move to the previous month, hold your hand upright with your pointer finger extended, then close your fist. Can you try flipping the calendar page forward and backward?"

We then observed each user's interactions with tasks 1-4 before soliciting feedback. During interactions 1 and 2, we noted that users took some time to adjust to the required pressure for accurate writing with the capacitive touchscreen stylus. We introduced them to the eraser function during this phase to facilitate corrections to their event names. Moreover, many users naturally rested their palms on the tablet to stabilize their writing. As our current system lacks palm rejection, this led to some inadvertent strokes. After approximately two initial attempts, users consistently managed to add and remove events from the Calendar.

We observed a steeper learning curve with gesture interaction, as users experimented with optimal hand placement and angle for gesture detection. Demonstrating the gestures and providing visual feedback from the camera view aided in improving gesture detection.

Following the interactions, we gathered open-ended feedback from each of the 5 users. Users found the event addition functionality intuitive but expressed a desire for automatic synchronization of events with the Calendar (i.e., immediate syncing upon completion of writing). Additionally, users were satisfied with the intuitiveness of the forward and backward flipping gestures but desired more consistent gesture detection and faster visual feedback. Surprisingly, users provided extensive feedback regarding additional functionalities resembling those of Google Calendar, such as automatic syncing of events added directly to Google Calendar with CalPal, drag-and-drop event movement, and recurring event creation. This underscores users' preference for the convenience offered by mobile scheduling systems and may serve as valuable insights for future CalPal development.

Lastly, users were asked to rate the resemblance of the generated theme to their current month's events on a scale from 1 to 5, where 1 denoted "No resemblance" and 5 denoted "Perfect Resemblance." The average rating was calculated at 3.8, indicating a potential area for improvement in the engineering of our stable diffusion prompt.

6. System Performance and Limitations

6.1. Overall System Evaluation

To quantitatively assess the usability of our system, we conducted UI system performance tests across three primary workflows: event addition, event removal, and calendar month change.

For the event addition and removal workflows, we measured elapsed time metrics in milliseconds for both the update of the Google Calendar and the refresh of the calendar theme. In the case of changing the calendar month, we recorded the elapsed time until the new month view was fully loaded, including both the events for the month and the updated calendar theme. Detailed metrics are presented

Table 1. UI System Performance Test Results

Metric	Time to Gcal Update (Add Event)	Time to Theme Update (Add Event)	Time to Gcal Update (Delete Event)	Time to Theme Update (Delete Event)	Time to Month View Update (Change Month)
Mean Elapsed Time (ms)	920	7345	713	4156	224

This table reports the results of our UI system performance tests, timed using the performance.now() Javascript builtin method. Mean elapsed time in milliseconds is reported for various Add event, delete event, and change month workflows.

in Table 1.

Notably, we observed an average reduction of 3000 ms in time taken to update the calendar theme when deleting events. We attribute this improvement to the caching of images for unchanged prompts within the Huggingface stable diffusion API. Following the RAIL guidelines [1], which suggest users expect tasks such as page loading or view changes to occur within 1000 ms, we find our system's performance satisfactory for both the Google Calendar update workflows and the month view update workflow, as our Mean Elapsed Time falls below this threshold.

However, we noted that text-to-image generation takes longer, resulting in a delay of approximately 7 seconds before users receive feedback in the CalPal UI after clicking the 'Add Event' button. While this falls below the 10000 ms threshold at which users may lose interest and potentially abandon tasks, it represents an area for improvement. One potential enhancement could involve providing alternative visual feedback while the new calendar theme is generating.

Finally, we evaluated the Levenshtein edit distance between the intended written word and the segmented string returned by the Google Cloud Vision API, assessing subjective handwriting qualities as good, mediocre, and poor using the phrase 'Dinner with friends 7pm.' We observed exceptional segmentation accuracy for both good and even mediocre handwriting qualities, with an average edit distance of 1 primarily due to capitalization errors. Surprisingly, for nearly unintelligible handwriting, we found an average edit distance of 9, which was lower than expected given the 23 characters in the string.

6.2. Frontend

Our frontend was evaluated well in the user study and is able to deal with user and system failures robustly. However, a current limitation of our frontend is that different screen sizes change the rendering and can cause components of the calendar to overlap. There is currently no partial erasing and the drawing response can lag relative to the touch movement due to the operations running in the background of the frontend.

Additionally due to the tradeoff between polling and other system lag in the frontend as described in the Frontend section, polling is only conducted once every second. This could be sped up by refactoring the code to make this request a callback function reducing how often we are polling the backend.

6.3. Backend

The backend runs the gesture recognition module as a separate process and whenever a gesture is detected the request to the server is started on a different thread. This allows the gesture recognition and notification to be non blocking allowing smooth integration with the rest of the system. However as the server is updated through a request from this module, there is up to half a second lag in sending this request over. As we are only polling the backend every second though, this is less important. This latency could be improved through integrating the module with the server and running it in a separate thread.

Additionally, a current limitation of the gesture recognition is that only two gestures are supported for flipping backward and flipping forward. These gestures allow for a natural range of motion but sometimes result in system error.

We conducted an experiment where we asked 3 individuals to interact with our gesture recognizer and do the following: attempt to perform the flip forward gesture 20 times, attempt to perform the flip backward gesture 20 times, and interact with the system for a minute without any gestures. The first was instructed to flip as fast as naturally comfortable, the second was instructed to flip naturally, and the third was instructed to look at the hand landmarks outputted to make sure they aligned with their hand while flipping. The results of this experiment are in Table 2.

The system has a harder time accurately detecting flips when the hand is moving too fast. This is due to the fact the hand landmarker is unable to keep up with the speed of the hand. As the users slow down the accuracy increases dramatically. The user flipping moderately fast (which is what we expect to be the most natural use case) resulted in a 100% accuracy rate for the flip forward gesture and a 90% accuracy rate for the flip backward gesture. On average our system performed much better at detecting the flip for-

Table 2. Gesture Experiment Results

Individual	Flip Forward Accuracy	Flip Backward Accuracy	Misdetections/ Minute	Notes
A	0.95	0.75	5	Flipping extremely fast
B	1.00	0.90	1	Flipping moderately fast
C	1.00	0.95	4	Flipping intentionally while looking at landmarks
Average	0.98	0.87	3.33	

This table reports the results of our gesture system experiment asking 3 users to perform the flip forward gesture 20 times, the flip backward gesture 20 times, and to interact with the system with their hands in camera view without making any of the supported gestures. We report the flip forward accuracy, the flip backward accuracy, and the misdetections/minute when not making gestures on an individual and aggregate level.

ward gesture with an average accuracy of 98% than the flip backward gesture with an accuracy of 87%. The hand landmarker at times has trouble detecting the index finger when it is pointed up and instead landmarks it as being closed with the rest of the fingers. In particular, it also struggles to detect the movement of the index finger if the other fingers on the hand are also closed leading to some drop in accuracy. On average our system had 3.33 false positive midetections per minute when the user was interacting with the system with their hand in view. This could be rectified by imposing stricter requirements on the gesture making at the expense of feeling less natural to the user. However when the user is naturally interacting with the system to write or to add an event or just to look at the calendar, their hands are often not by their face (and thus not in camera view).

6.4. Outside Tools

The outside APIs and libraries we used in our system overall worked very well. However, the largest issue is that the Media Pipe hand landmarker can lag causing gestures to not be detected or even gesture misdetection. A solution to this would be to run the model on a more powerful computer.

6.5. Hardware

The largest limitation of the current hardware we are using is that the capacitive touch screen isn't very performant for drawing on the screen with a stylus. In order to draw, the user needs to press harder than what they would have to press when writing with a marker for example. Additionally at times the touch point, due to the capacitive touch screeen being configured to register finger touches, is not very precise and there can be a disconnect between what the user is intending to write and and what is shown on the screen. Additionally, the screen application is not configured to use palm rejection as discussed in the user study.

A potential avenue to rectify this is to use a resistive touch screen which depends solely on pressure which we hypothesize would allow for better stylus interaction at the cost of finger interaction.

6.6. Interesting Failure

An interesting failure our system has, has to do with the limitations of the backend. Our backend only maintains up to 12 months of state, however the frontend is able to query for decades. Thus for example if an event has been added in January 2024 if the user flips back to another year in January (ie: 2023 or 2025) the system will display the 2024 events for that year. Once on January 2023, the user is still able to add and delete events as normal. Even when an event is added in a year that the state is not being maintained for it will still be added appropriately to the Google calendar for the correct date. It is just that we only maintain 12 months worth of handwriting canvas and every month is mapped to the same canvas regardless of the year causing overlap in displays for the same month in different years on the frontend. This problem can be mitigated by using an actual database creating different canvases for the same month in different years.

7. Conclusions and Next Steps

CalPal serves as a great launching point into more intelligent and intuitive calendaring. Given feedback from the user study and current system limitations the next steps we plan to pursue include: state maintenance for more than 12 months, decreasing gesture polling latency, and running the system as a standalone product. Additionally as the system only supports a one way sync with Google calendar, another avenue of exploration is to make this syncing bidirec-

tional (ie: removing an event on the digital Google calendar should remove it on the CalPal).

8. Collaboration

Jenny and Devin worked together to design the complete system functionality and specify the interaction functions between the back and front ends.

Jenny was responsible for implementing the system backend including the server and api endpoints to support frontend-backend communication, the backend calendar abstract data type, state maintenance, gesture recognition, and generative calendar theming.

Devin was responsible for implementing the frontend, including the logic for processing touch input (and interfacing with the Google Cloud Vision Handwriting Recognizer API), updating calendar views based on gesture input, and updating the calendar theme.

References

[1] Measure performance with the RAIL model. Web article. Accessed on May 15, 2024. 11

[2] Dakboard customizable display. https://dakboard.com/site, 2024. Accessed: 2024-05-11. 2

[3] Mango display digital calendar display. https://mangodisplay.com/digital-calendar-display, 2024. Accessed: 2024-05-11. 2

[4] Yanliu Huang, Zhen Yang, and Vicki G. Morwitz. How using a paper versus mobile calendar influences everyday planning and plan fulfillment. *Journal of Consumer Psychology*, 33(1): 115–122, 2023. 2

[5] Julian.digital. Multi-layered calendars. https://julian.digital/2023/07/06/multi-layered-calendars, 2024. Accessed: 2024-05-11. 2

[6] Harald Scheidl. Htrpipeline, handwriting text recognition with tensorflow. https://github.com/githubharald/HTRPipeline, 2023. 9

[7] Timothy J. Smoker, Carrie E. Murphy, and Alison K. Rockwell. Comparing memory for handwriting versus typing. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 53(22):1744–1747, 2009. 2

Project name: CalPal
Project author(s): Jenny Moralejo; Devin Murphy

Note that these are intended to be short answer questions, except perhaps for the last one.
Repeat for all packages/tools/libraries you used (make a copy of the table for each package, etc.)

Package/Tool/Library (name and version number):	Google Cloud Vision Optical Character Recognition API
What machine and OS version did you run it on (so people will know roughly what compute power and what environment it needs)	Lenovo ThinkPad P1 Windows 10
Where is it available? (eg url):	https://cloud.google.com/vision/docs/ocr
What did you use it to do? (One word or phrase is fine if the answer is the obvious, eg speech recognition, face detection, etc., otherwise explain a little more)	Handwriting detection
What was the data type of the input (eg image as a matrix, mp3 format for sound, etc.):	Png files for our project; many types supported
How well did it work in terms of • accuracy, as in rough percentage correct, in your experience (no need to run a formal evaluation): • speed: (as in, fast enough to allow convenient use, or it slowed down the system);	- 98% - Very fast (API)
Did it work out of the box? If not, what did you have to do to use it?	No – create a google cloud project (instructions in link above)

Package/Tool/Library (name and version number):	Google Calendar API v3
What machine and OS version did you run it on (so people will know roughly what compute power and what environment it needs)	Lenovo ThinkPad P1 Windows 10
Where is it available? (eg url):	https://developers.google.com/calendar/api/guides/overview
What did you use it to do? (One word or phrase is fine if the answer is the obvious, eg speech recognition, face detection, etc., otherwise explain a little more)	Calendar event syncing with linked online Google Calendar
What was the data type of the input (eg image as a matrix, mp3 format for sound, etc.):	String denoting event to be added or string id denoting event to be deleted
How well did it work in terms of • accuracy, as in rough percentage correct, in your experience (no need to run a formal evaluation): • speed: (as in, fast enough to allow convenient use, or it slowed down the system);	- 100% - Very fast (API)
Did it work out of the box? If not, what did you have to do to use it?	No – create a google cloud project and link calendar instructions here: https://developers.google.com/calendar/api/quickstart/python

Package/Tool/Library (name and version number):	Hugging Face Stable Diffusion API
What machine and OS version did you run it on (so people will know roughly what compute power and what environment it needs)	Lenovo ThinkPad P1 Windows 10
Where is it available? (eg url):	https://huggingface.co/docs/api-inference/en/index
What did you use it to do? (One word or phrase is fine if the answer is the obvious, eg speech recognition, face detection, etc., otherwise explain a little more)	Generative event theming for calendar based on calendar events using model stable-diffusion-v1-4
What was the data type of the input (eg image as a matrix, mp3 format for sound, etc.):	String of prompt for the stable diffusion model
How well did it work in terms of • accuracy, as in rough percentage correct, in your experience (no need to run a formal evaluation): • speed: (as in, fast enough to allow convenient use, or it slowed down the system);	- 90% rate limited if sending a lot of requests will fail - Very fast (API)
Did it work out of the box? If not, what did you have to do to use it?	No – create a hugging face account and generate a token

Package/Tool/Library (name and version number):	Media Pipe Hand Landmarker
What machine and OS version did you run it on (so people will know roughly what compute power and what environment it needs)	Lenovo ThinkPad P1 Windows 10
Where is it available? (eg url):	https://developers.google.com/mediapipe/solutions/vision/hand_landmarker/python
What did you use it to do? (One word or phrase is fine if the answer is the obvious, eg speech recognition, face detection, etc., otherwise explain a little more)	Generate hand landmarks to be analyzed for gesture detection
What was the data type of the input (eg image as a matrix, mp3 format for sound, etc.):	Image frame as mediapipe.Image (can load from file or numpy array)
How well did it work in terms of • accuracy, as in rough percentage correct, in your experience (no need to run a formal evaluation): • speed: (as in, fast enough to allow convenient use, or it slowed down the system);	- 95% sometimes hidden landmarks not classified correctly - If you move your hand too fast the system will lag but otherwise works relatively well – limited by the FPS of your device
Did it work out of the box? If not, what did you have to do to use it?	No – you have to write boilerplate code to get image from camera, run the model, and draw the landmarks (modified the following to work with live video)

Package/Tool/Library (name and version number):	React-big-calendar
What machine and OS version did you run it on (so people will know roughly what compute power and what environment it needs)	Lenovo ThinkPad P1 Windows 10
Where is it available? (eg url):	https://github.com/jquense/react-big-calendar
What did you use it to do? (One word or phrase is fine if the answer is the obvious, eg speech recognition, face detection, etc., otherwise explain a little more)	Base calendar library we build our final calendar off of
What was the data type of the input (eg image as a matrix, mp3 format for sound, etc.):	N/A
How well did it work in terms of • accuracy, as in rough percentage correct, in your experience (no need to run a formal evaluation): • speed: (as in, fast enough to allow convenient use, or it slowed down the system);	- N/A - Base calendar functionality (flipping pages with buttons) quite fast
Did it work out of the box? If not, what did you have to do to use it?	Yes