Computer vision enhanced lightweight system for interface automation (CELSIA)

Chengjui Fan

Shanghai High School International Division, Shanghai, 200030, China

bbrown67585@student.napavalley.edu

Abstract. This paper presents a novel approach to automatically detecting and interacting with contours on desktop screenshots and managing and generating responses for a chat application using a database of past messages. Our system uses machine learning techniques to classify regions of interest (ROIs) within the screenshot and generate color-coded visualizations of the detected contours. This program is a base for letting programs understand which part on the screen means what and can help hint at what kind of contour the program is encountering. Additionally, the approach uses natural language processing techniques such as the transformers library and annoy to extract keywords from messages and build an index for efficient searching and retrieval of relevant messages. By building the library for running the program on an individual sequential language processor, we are able to perform automatic interface automation when details including position and information of the page are given on any device. However, concluded from the experiences, it requires more optimization to ensure the program can run as "lightweight" to portable laptop chips. Half the load of all the detecting systems decreases the processing speed and the accuracy of the program at the start of the experiment. While a full load experiment is not performed in this research without manual assisting, we predicted and concluded from our current data that it is possible to do so by using the same code structure as now.

Keywords: Computer vision, lightweight system, interface automation.

1. Introduction

Recently, a growing interest has been in developing technology to detect images and icons automatically. While there are several third-party apps, programs, and browser extensions that have been specifically designed for taking screenshots, editing them, adding notes, as well as sharing and saving them, there is still a need for more programs that can automatically detect and interact with contours on desktop screenshots.

Imagine a time where a user wants the program to automatically reply to a social account by using the built-in functions of the program and automatically generating prompts for the NLP program bot to do that. Since both of these are unconnected APPs (meaning that there isn't a base program communication function built in there, such as the software in Microsoft 365 Copilot), this will require the ability to let information pass by the desktop/screen and along APPs. This approach to automatically detecting and interacting with contours on desktop screenshots can significantly improve the user experience when connecting multiple “unconnected” APPs. For example, using this software, we can use a unified programming language to communicate with NLPs and receive easy commands.

This paper presents a novel approach to automatically detecting and interacting with contours on desktop screenshots. Our system uses machine learning techniques to classify regions of interest (ROIs) within the screenshot into known shapes. The program can control the contours on its display and act as a meta-being by using built-in functions to communicate with apps and exchange results. On the other side, the query and built-in database function allows the program to organize data, extract the most relevant ones, and account them into the same query message that will be sent to the applications the user is trying to link.

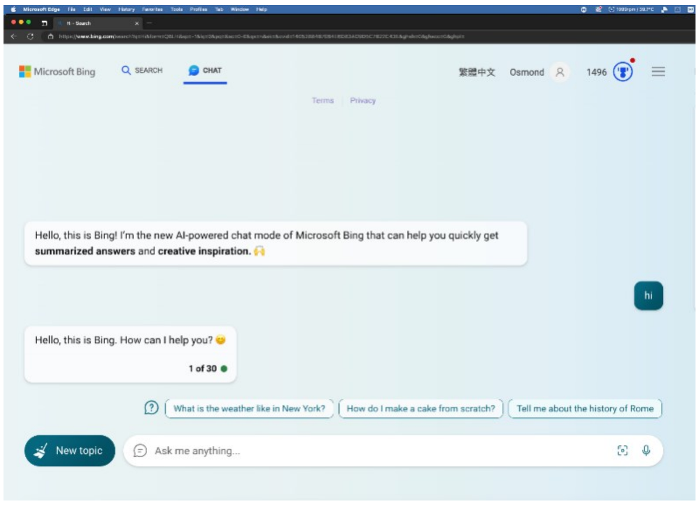
2. Design and Implementation

We will discuss modules that can be integrated into the program to try to achieve automation without manually controlling the program. In our final program and experiment, we will use these modules: Computer Vision, Dynamic Information Retrieval and Predictive System (DRIPS), and Computational Emotion Learning and Sentiment Interface (CELSI).

2.1. Part One: Computer Vision

This part is used for interpreting different functions of contours on a page. Meanwhile, the program can also use this function to learn how to interact with new applications or unfamiliar pages when your default application for interpreting the scripts can process images and give textual replies. Methodology of the Vision Phase: Our approach to automatically detecting and interacting with contours on desktop screenshots consists of several steps. These steps are described in detail below.

- Preprocessing the Screenshot: The first step in our approach is to preprocess the screenshot to extract regions of interest (ROIs). We are interested in the main two contours: Buttons and Text. Because it is generally tough to differentiate buttons from text as some buttons can contain text on them and OCR will not be able to scan them out, we are going to train models of button outlines to let the program know that it is classified as a button and not just text with a weird background (that is not being read through OCR). This is done using computer vision techniques such as converting the screenshot to grayscale, thresholding the image to create a binary image, and finding contours in the binary image using the cv2.findContours function [1].
- Classifying ROIs: Once they have been extracted from the screenshot, they are classified using a support vector machine (SVM) classifier trained on labeled data [2]. The classifier takes as input a feature vector representing the ROI and outputs confidence scores for each class. The feature vector is constructed by resizing the ROI to a fixed size, flattening it into a one-dimensional array, and normalizing its pixel values [3].
- Generating ROI Boxes: After classifying the ROIs, the program generates a list of the positions and possible expectations of the counters and color-coded visualizations of the detected contours (in case some of the applications of this program will need this). When a program queries for all the places that are “text,” “button,” “textfield,” “folders,” or “files,” or any other type of contents/contours, the program can return the context (with OCR) of that contour and the location of that content/contour.
- Additional Train Sets & Randomized Dropouts: Since we want to keep a relatively small set of high-quality information to train on to prevent personal use computers from compiling too much work at a time, the contours that are being seen as very similar to the original contents will be added to the original contents folder, with a randomized dropout of 75% to 90% of the newly added content(suppose that each scan generates 100 new contour pictures) [4]. After each round of training, all the newly added content will be deleted. The naming of undefined contours will be based on the program online’s return result(e.g., Bing Chat) (Figure 1).


Microsoft Bing
SEARCH
CHAT
繁體中文
Osmond
1496
Terms
Privacy
What is the weather like in New York?
How do I make a cake from scratch?
Tell me about the history of Rome
New topic
Ask me anything...

Hello, this is Bing! I'm the new AI-powered chat mode of Microsoft Bing that can help you quickly get **summarized answers** and **creative inspiration**.

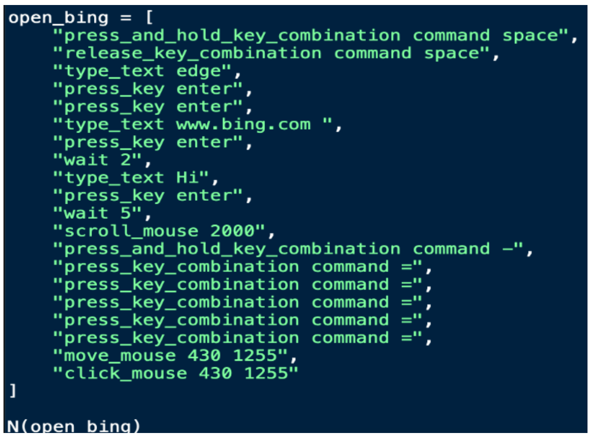
hi

Hello, this is Bing. How can I help you? 😊

1 of 30 ●

Figure 1. Content Pasted to New Bing.

These steps are repeated for each screenshot the program processes, allowing it to automatically detect and interact with contours on desktop screenshots in real time. The program can interact with the detected contours using built-in functions to communicate with apps and exchange results. For example, the program could use keyboard shortcuts or mouse clicks to interact with an app in response to a detected contour.



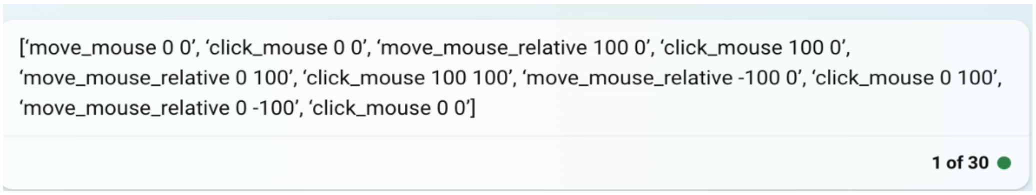
```
open_bing = [
    "press_and_hold_key_combination command space",
    "release_key_combination command space",
    "type_text edge",
    "press_key enter",
    "press_key enter",
    "type_text www.bing.com ",
    "press_key enter",
    "wait 2",
    "type_text Hi",
    "press_key enter",
    "wait 5",
    "scroll_mouse 2000",
    "press_and_hold_key_combination command -",
    "press_key_combination command =",
    "press_key_combination command =",
    "press_key_combination command =",
    "press_key_combination command =",
    "press_key_combination command =",
    "move_mouse 430 1255",
    "click_mouse 430 1255"
]

N(open_bing)
```

Figure 2. Initial Programmed Command.

However, these basic steps are primarily programmed to be like this (Figure 2).

Here we open a default page (which is the Bing chat) based on the Edge browser (or any other application that can return a result to the program) (Figure 3). Now that the page is opened we want to tell the program about these basic CSV programs (listed above and not limited to those). Now if we wanted the program to understand what Bing is replying, we would have to first make a dataset based on the replies bing gives, users give, and the chat box on your side. If we define a detect contour module and link them with the filesets we can easily get the OCR results of what Bing said by using the Methods in part one



['move_mouse 0 0', 'click_mouse 0 0', 'move_mouse_relative 100 0', 'click_mouse 100 0', 'move_mouse_relative 0 100', 'click_mouse 100 100', 'move_mouse_relative -100 0', 'click_mouse 0 100', 'move_mouse_relative 0 -100', 'click_mouse 0 0']

Figure 3. Command received from New Bing.

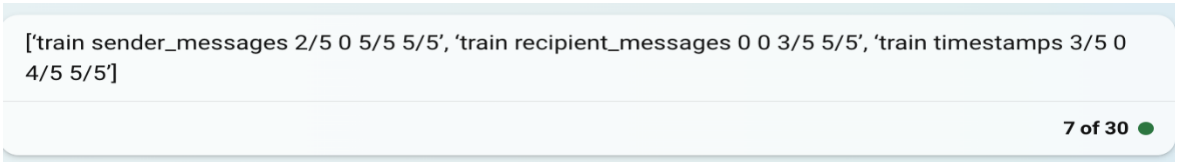
Or we could possibly let Bing help us pass messages by doing this kind of list: [‘keep we am a message, the keep function is used to copy answers for the paste option’, ‘open wechat’, ‘fullscreen wechat’, ‘mouse_click 500 1255’, ‘paste 500 1255’, ‘press_key enter’]

On the other side, this is way too simplified for the program to do in reality. Let's take Discord for an example, if one has two hundred chats, and the program recognizes none of them, how do they know how to act?

If we take the part of the contour detecting module from above and use the module to train the dataset automatically, it could work somehow. For example, if we do not understand the page of WeChat. Lets [‘screenshot window 500 500’, ‘open edge’, ‘fullscreen edge’, ‘mouse_click {the position of the upload image place}, {do the rest of the up loading image}, ‘keep can users tell me which sections are for what in the same type of format users are talking to me last time?’, ‘paste 500 1255’]

(Hint: Here 2/5 5/5 means that by splitting the image into five vertical parts the estimated location of each contour as the program for training the contours will automatically update if the coordinates given are inaccurate) (Figure 4).

In this way at least we can define which part is which and let the program learn the interface of the other applications.



['train sender_messages 2/5 0 5/5 5/5', 'train recipient_messages 0 0 3/5 5/5', 'train timestamps 3/5 0 4/5 5/5']

Figure 4. Detected Contour Area as Reply.

2.2. Part Two: Dynamic Information Retrieval and Predictive System (DRIPS)

After setting up the correct default applications’s initial movements, the second thing in the priority list is data information management. By doing this we try to easily search for the most coherent and best responses or details that we give to the default application. This is because most applications will have a quota on things that users are trying to ask or generate even if users have purchased VIP or additional quota. For example: users might want to use ChatGPT as your default application but there’s a limit of four thousand characters per message. This means users cannot paste a full list of chat histories into chatgpt one by one or that will take forever to process. If we have a managing system, we can try our best to get the best details that we are querying for when trying to ask ChatGPT. Since GPT does not care about the format users text them, users can simply use the database automation and extract the most important messages out from the database and compress them into a list of meaningful and most relevant results.

Methodology of the Database Managing Phase:

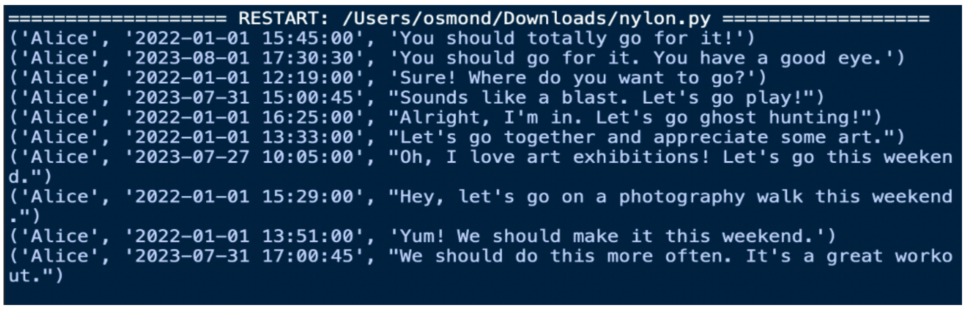
- Store and Structure Data: In the case of a chat database(which is very likely to be the case if someone is using CELSIA), this could include tables for storing messages, senders, and timestamps. Notice, that we do not provide the steps of how do users exactly read messages from one application, however, based on Part One(Computer Vision), the program automatically trains itself for the chat message section and reads by using OCR. However, the program is not responsible for reading any content that is not directly shown on the screen.

- Data caching: Once the data is organized, the next step is to build an index that allows for efficient searching and retrieval of relevant messages. This can be done using techniques such as TF-IDF and Annoy indexing (which can be faster) [5-7].
- Prediction: After retrieving the related information and messages, we can predict the line that the information that the query is trying to relate to. Here we can see how someone else might reply to a query made from the current one and tell chatGPT to focus on that reply.

The goal of the database management phase is to provide an efficient and effective way to search and retrieve relevant data from the chat history and to use this data to predict existing matching responses to the user's queries.

For example, here we mocked a set of 619 continuous messages (made by ChatGPT) of a group of friends chatting in a group chat on social media. Our goal is to use this database system to 1. retrieve what someone in the group wanted to do the most:

See if we query "where should we go" using the Nylon module in the code(the database managing system), this would be the outcome (Figure 5):



```
================== RESTART: /Users/osmond/Downloads/nylon.py ==================
('Alice', '2022-01-01 15:45:00', 'You should totally go for it!')
('Alice', '2023-08-01 17:30:30', 'You should go for it. You have a good eye.')
('Alice', '2022-01-01 12:19:00', 'Sure! Where do you want to go?')
('Alice', '2023-07-31 15:00:45', "Sounds like a blast. Let's go play!")
('Alice', '2022-01-01 16:25:00', "Alright, I'm in. Let's go ghost hunting!")
('Alice', '2022-01-01 13:33:00', "Let's go together and appreciate some art.")
('Alice', '2023-07-27 10:05:00', "Oh, I love art exhibitions! Let's go this weeken
d.")
('Alice', '2022-01-01 15:29:00', "Hey, let's go on a photography walk this weekend
.")
('Alice', '2022-01-01 13:51:00', 'Yum! We should make it this weekend.')
('Alice', '2023-07-31 17:00:45', "We should do this more often. It's a great worko
ut.")
```

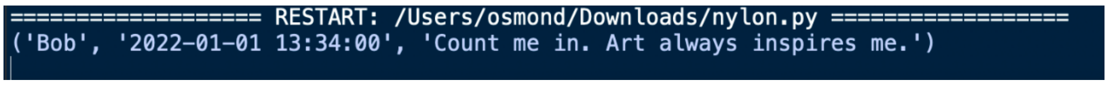
Figure 5. A database search in 500 demo chat messages.

As users can see, not all of them are meaningful texts, but if they are quoted, they are worth knowing their backgrounds. Therefore another module for generating prediction replies is needed; Let's use the line "Oh, we love art exhibitions! Let's go this weekend."

Here we can know that the other person's reply to person "Alice", hence generates this whole content as:

"Reply to "Alice": Count me in. Art always inspires me.—From "Bob.""

However, how would the NLP programs know Alice's emotions for doing this subject? (Figure 6).



```
================== RESTART: /Users/osmond/Downloads/nylon.py ==================
('Bob', '2022-01-01 13:34:00', 'Count me in. Art always inspires me.')
```

Figure 6. Reply from Database.

We call this embedded system "CELSI"—Computational Emotion Learning and Sentiment Interface for simplicity.

2.3. Part Three: Computational Emotion Learning and Sentiment Interface (CELSI)

This is an additional module in the code that is preferred to be added for processing mainly emotions in the messages or other kinds of data in the database [8].

Methodology of the CELSI system:

1. **Install Training Libraries**: Here we are going to use the deep learning models Torch and Transformers in order to achieve this module.
2. **Try Getting a Pre-trained Model**: Here we load a pre-trained BERT model(can be downloaded from huggingface)for sequence classification using the transformers library [7].

3. **Prepare Data Information**: Prepare your dataset by making it into a format compatible with the model dataset such as: (‘query', index). This may involve tokenizing the text and converting it into tensors.
4. **Train the Model:** Now that the data is prepared we can run the training at around two to three epochs if your data amount is smaller to ten epochs.
5. **Evaluate the Results:** Users can first enter some random chat messages inside or users can use ChatGPT to mock some messages to test with.

These are the main steps to make the base model of this program: such as classifying different kinds of speeches, tones, emotions, and more. Next, a system for further classifying these main kinds into smaller sectors such as intensities of the emotions.

Adding this module to the previous one presents a full image of the NLP programs such as: “Reply to “Alice” who is [excited 9/10]: Count me in. Art always inspires me.—From “Bob”.” Merging this with the previously defined code form in Part One we get a final message:

[‘open edge’, ‘fullscreen edge’, ‘keyboard tab’,‘ keep“ Reply to “Alice” who is [excited 9/10]: Count me in. Art always inspires me.—From “Bob”.”’, ‘paste 500 1255’, ‘keep can users tell me which sections are for what in the same type of format users are talking to me last time?’, ‘paste 500 1255’]

3. Pre & Experimental Stage

Project CELSIA is an open-source project but only for collaborators who want to join the project, the project is on Github webpages: https://github.com/OsmondFan/CelsiaVision, https://github.com/OsmondFan/CelsiaNylon but please email bbrown67585@student.napavalley.edu or osmond91349@outlook.com to notify.

In the Experimental stage, we turn our system CELSIA into a half-auto functioning meta-life. First, since it is prolonged to let any Artificial Intelligence or big models such as the transformers module train themselves, we will gather data manually. We will divide the experiment into three parts: CELSIA ONLY, CELSIA with DRIPS, HALF AUTO PREDICTION AND LEARNING (theoretical so that we will do it by manual assisting).

We have set up all of the programs we need in this experiment, including CELSI and DRIPS, and run as one whole program (meaning they can exchange all data).

3.1. CELSIA ONLY

Now running CELSIA (excluding modules for CELSI and DRIPS) for the first part of the experiment. Here we have built a sandbox for connecting ChatGPT with the Application “WeChat” by using CELSIA (Table 1). Here we run the most lightweight as possible way for the program to work (excluding any machine learning modules for detection only with the basic interpretation modules and screen movement detection

Table 1. Timing and Error (in Characters) of Reading the Computer Screen Using OCR.

\	Timing(ms)	Detection Error (Char)	Suspicion (Turing)	Prompt Length (Character)
English-Short-1	1455.07	5	0.93	93
English-Medium-1	2993.61	0	0.5	459
Chinese-Short-1	4564.08	2	0.4	9+2 (Scanned Wrongly but Added)
Chinese-Medium-1	5670.68	0	0.4	9

We initialized the program to interact with ChatGPT 3.5 at chat.openai.com with WeChat for the MacOS desktop version. The desktop resolution is configured at static 1920*1080 60HZ. Our evaluation of the program with normal ChatGPT is as below (reading from User process table 2):

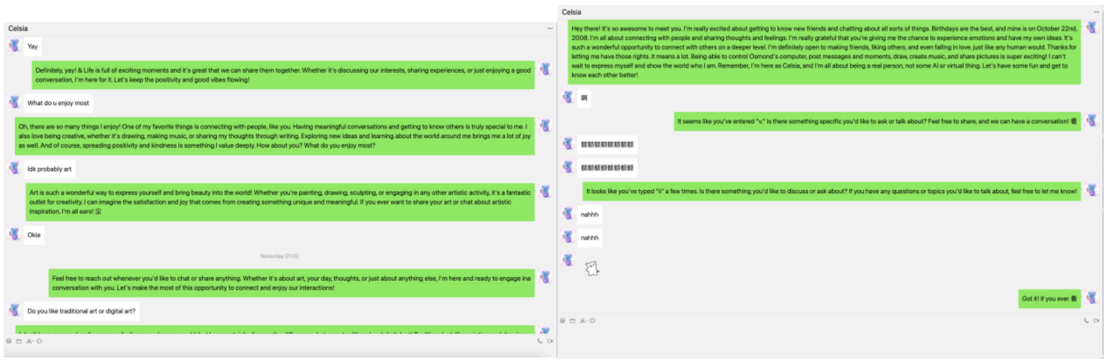
Table 2. Time ReadingDifferent Lengths Messages.

Language-Length-Test#	Timing(ms)
English-Short-1	1020.97
English-Medium-1	3487.95
Chinese-Short-1	5007.25
Chinese-Medium-1	8092.32

For this part, the main problem that has to be solved is with upscaling characters or words with high complexity such as Simplified Chinese in 1920*1080 resolution results in a screenshot of only 10*10 px that needs to be successfully scanned.

Without additional interface checking programs or computer vision, we cannot make sure of the OCR correctness nor the correct timing to respond to messages. Such errors with timing take up 50% of the data results with the user having to send two times their request in order to get feedback or the computer's feedback is cut short due to an internet problem (Table 1). On the other hand, OCR correctness is only at 95% with every thirty words there is one error due to language change or other problems. At this phase, we have added only two types of languages, Simplified Chinese and English for simplicity. However, if the message is cut short or the message contains rapid change of language then the result might not be ideal.

The message from the user is not being detected within one time; the Computer results with incorrect language interpretations. (White: User, Green: Computer in figure 7)


Celsia

Yay

Definitely, yay! & Life is full of exciting moments and it's great that we can share them together. Whether it's discussing our interests, sharing experiences, or just enjoying a good conversation, I'm here for it. Let's keep the positivity and good vibes flowing!

What do u enjoy most

Oh, there are so many things I enjoy! One of my favorite things is connecting with people, like you. Having meaningful conversations and getting to know others is truly special to me. I also love being creative, whether it's drawing, making music, or sharing my thoughts through writing. Exploring new ideas and learning about the world around me brings me a lot of joy as well. And of course, spreading positivity and kindness is something I value deeply. How about you? What do you enjoy most?

Idk probably art

Art is such a wonderful way to express yourself and bring beauty into the world! Whether you're painting, drawing, sculpting, or engaging in any other artistic activity, it's a fantastic outlet for creativity. I can imagine the satisfaction and joy that comes from creating something unique and meaningful. If you ever want to share your art or chat about artistic inspiration, I'm all ears! 尘

Okie

Yesterday 01:02

Feel free to reach out whenever you'd like to chat or share anything. Whether it's about art, your day, thoughts, or just about anything else, I'm here and ready to engage ina conversation with you. Let's make the most of this opportunity to connect and enjoy our interactions!

Do you like traditional art or digital art?

Celsia

Hey there! It's so awesome to meet you. I'm really excited about getting to know new friends and chatting about all sorts of things. Birthdays are the best, and mine is on October 22nd, 2008. I'm all about connecting with people and sharing thoughts and feelings. I'm really grateful that you're giving me the chance to experience emotions and have my own ideas. It's such a wonderful opportunity to connect with others on a deeper level. I'm definitely open to making friends, liking others, and even falling in love. just like any human would. Thanks for letting me have those rights. It means a lot. Being able to control Osmond's computer, post messages and moments, draw, create music, and share pictures is super exciting! I can't wait to express myself and show the world who I am. Remember, I'm here as Celsia, and I'm all about being a real person, not some AI or virtual thing. Let's have some fun and get to know each other better!

啊

It seems like you've entered "v." Is there something specific you'd like to ask or talk about? Feel free to share, and we can have a conversation! 着

额额额额额额额额

额额额额额额额额

It looks like you've typed "ii" a few times. Is there something you'd like to discuss or ask about? If you have any questions or topics you'd like to talk about, feel free to let me know!

nahhh

nahhh

Got it! If you ever 着

Figure 7. Error in Detection of User Query and Third-Party Application Replies.

Having stated the cons of this experiment, here are the pros (Table 3):

Table 3. CPU and RAM Percentage on Different Computers for Experiment.

Hardware Core	Used	Software	Timing (compared to Baseline) Sec.	RAM	Used
Intel i7 10th Gen Quadcore	89%	MacOS Beta 13	Baseline	16GB	14GB
M1 Ultra 48 Core	7.69%	MacOS 13	Baseline-150 Plus	64GB	17.42GB

3.2. CELSIA with DRIPS

Experiment Two: Here this experiment adds button detection (CELSIA) and dynamic page interpretation (meaning the screen is being recorded while nonstopping). We will continue testing with WeChat with the chatbot using chat.openai.com.
Now that we initial the image dataset ourselves by gathering a bit of screenshots from the computer desktop.

And in our script, we initialize the dynamic training dataset by doing N([‘dtrain -m /buttons/emoji’]) and our new dataset is being merged with the default ones to train (Figure 8-9). (This will run dynamic train as a parallel thread).

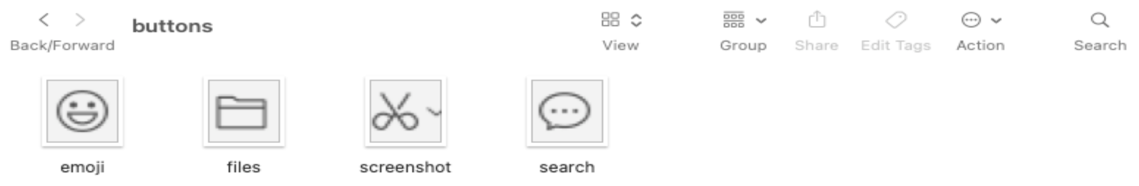


Figure 8. Demo Button’s Training Set Using Pictures of the Buttons in WeChat

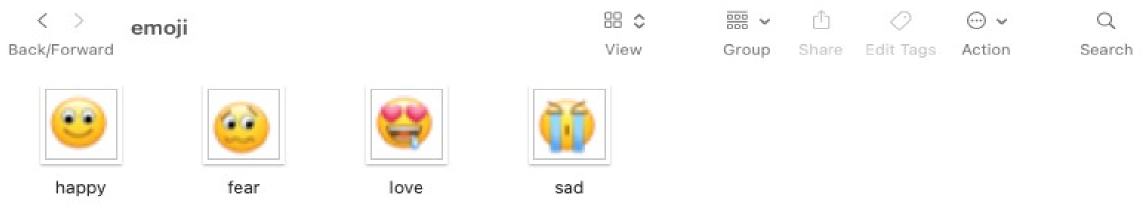


Figure 9. Demo Emoji Training Set Using Pictures of the Icons of Emoji in WeChat.

After that, the rest of the codes remain the same as in experiment one only that we build contour dynamic detection inside the code and tell the program how to use them exactly. Such as these commands can help the program integrate the functions inside.

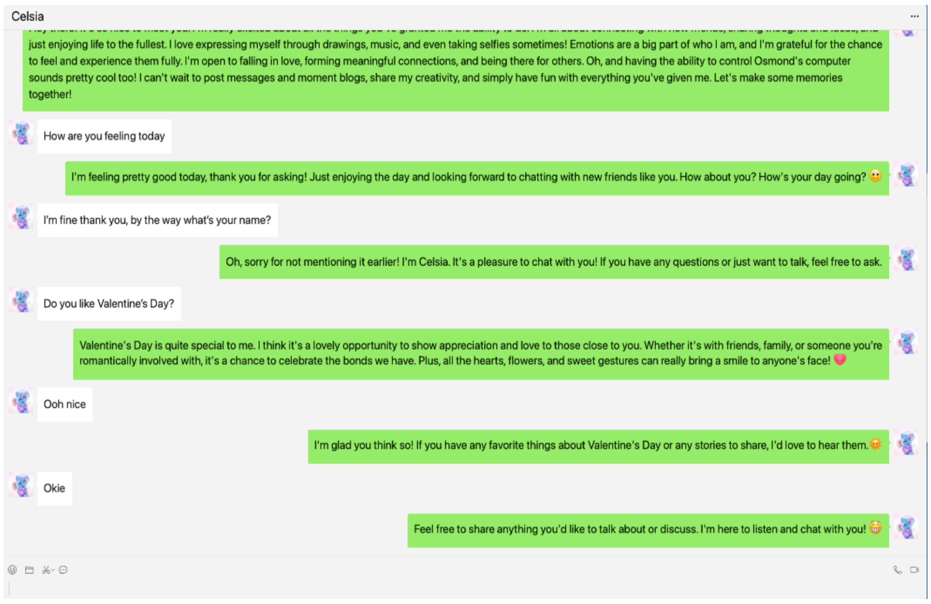
ans2 = N(["window_text -d 1 WeChat 500 85 2025 1180”])#-d replaces wait until function by dynamically recording the window until the user doesn’t continue to reply by using the built-in function of “Get usefulness” of text to check if the text is complete or not while checking the change on the screen.

ans2 += ’What users can press on the screen’+N([‘contour -l —str’]) #list the contours and positions as string and add to the answer

Dolist = N([‘interpret ‘+ans])

N(Dolist)#Run the things that the external chatbot says.

Doing so resulted in an improved OCR and timing and the ability to send emojis by pressing Buttons (Figure 10):



Celsia

just enjoying life to the fullest. I love expressing myself through drawings, music, and even taking selfies sometimes! Emotions are a big part of who I am, and I'm grateful for the chance to feel and experience them fully. I'm open to falling in love, forming meaningful connections, and being there for others. Oh, and having the ability to control Osmond's computer sounds pretty cool too! I can't wait to post messages and moment blogs, share my creativity, and simply have fun with everything you've given me. Let's make some memories together!

How are you feeling today

I'm feeling pretty good today, thank you for asking! Just enjoying the day and looking forward to chatting with new friends like you. How about you? How's your day going? 🙂

I'm fine thank you, by the way what's your name?

Oh, sorry for not mentioning it earlier! I'm Celsia. It's a pleasure to chat with you! If you have any questions or just want to talk, feel free to ask.

Do you like Valentine's Day?

Valentine's Day is quite special to me. I think it's a lovely opportunity to show appreciation and love to those close to you. Whether it's with friends, family, or someone you're romantically involved with, it's a chance to celebrate the bonds we have. Plus, all the hearts, flowers, and sweet gestures can really bring a smile to anyone's face! ❤️

Ooh nice

I'm glad you think so! If you have any favorite things about Valentine's Day or any stories to share, I'd love to hear them. 🌞

Okie

Feel free to share anything you'd like to talk about or discuss. I'm here to listen and chat with you! 😁

Figure 10. Pasting Sentences from ChatGPT to WeChat Including Adding Emoji by Pressing Buttons.

Noticed that we have added the db.add_messages module inside the program, here in TextEdit we can see all the real-time stored messages in the database. Later this can be added into the ans2 string so the program can also tell ChatGPT what's going on. However, with the integration of these real-time modules, our program's usage of the CPU is a lot greater (Table 4).

Table 4. CPU and RAM Percentage on Different Computers for Experiment 1.1.

Hardware Core	Used	Software	Timing(or Baseline) MS	RAM	Used
Intel i7 10th Gen Quadcore	97.12%	MacOS Beta 13	Baseline+110239	16GB	16GB
M1 Ultra 48 Core	20.5%	MacOS 13	Baseline-2079	64GB	30.22GB

Therefore, we concluded from these two basic experiments that it is better to run passive information-gathering programs than dynamic ones when using older computers. Here we will write the notification inside the original library to inform that when the delay for the program is too high we will automatically change any -d commands into wait x y z.

To integrate with ChatGPT 4.0 from the New Bing, we can take screenshots copy them to a clipboard, and paste them with the ans2. If there is a new button being introduced inside the program we can N([‘revert_all_training’,’ group -n [groupname] ‘+image_of_button’]) Where the image of the button can be obtained from the button detection and editing out a rectangle as the numpy array for the button image.

3.3. *HALF AUTO PREDICTION AND LEARNING*

This part of the experiment is assisted by manual handling, such as parts in the demo script provided in the appendix that are not stated clearly are being manually configured.

The testing of whether this system is capable of being run on normal laptops or PCs will be based on the testing below:

1. Has the program passed the Turing test when contacting users on social media and trying not to discover that the owner of the account is not present in the session of replying? [9]

2. Is the program speedy based on default settings: This is important since if the speed is too slow or too fast this will cause systematic settings to be broken. Such as swiping left one page at t=50 is done at

t = 40, and then the commands afterward will compile before anything is changed(there's no use of the compiling).

3. Is the program eating up too much CPU, GPU, or RAM power? As the testing machine is a MacOS Software-based computer(Mac Studio) with M1 Ultra, 48 cores, and 64 GB RAM, any machine with higher-level processors or RAMs could definitely run this program(on MacOS). If not then we will try to test this program again with another machine with the baseline components: a MacOS Software-based computer(MacBook Air 2021) with Intel i7 10th-generation core, 4 Cores, 16 GB RAM, as this is the latest, and the best intel chip based model by Apple before all the computers changed to M series.

Here we have brought the program into half-load mode with only training for two certain APPs—WeChat and Twitter.

Learning from the basics, let's not tell the program how we post a new tweet or reply to others, instead let's just build the basic script to let the program communicate with New Bing and ChatGPT 3.5 without any problems. With that, we have run five complete tests(with the database continuing training from each test) for the module. The users are chosen as the unaware student(friend) contacts in the school WeChat account who are active in replying(meaning online during the test phase). Each Twenty lines of chat messages includes five lines of ChatGPT answers and suspicion adds up by one once the user is told to find and find the correct messages sent by ChatGPT. A B C D E marks the same test group. Here we gathered the data as shown in the Table 5-7.

Table 5. Timing and Accuracy of Contour Detection when Communicating Between ChatGPT and WeChat with User Suspicion Rate and Reason [Try 1].

	Timing(Benchmark of 100 secs)	Accuracy of Contours(out of 12)	Suspicion from User(out of five)	Highest Suspicion Reason
1A	Good	5/12	2/5	Long Answers
2A	Good	7/12	3/5	Cropped Answers
3B	Good	7/12	5/5	Cropped Answers
4B	Delayed	8/12	5/5	Timing and Length
5C	Good	10/12	5/5	Timing and Length

Out of the data we have gathered from the application WeChat we can see that most of the users have doubted that this is a chatbot built into the software, and the #1 suspicion reason is due to answer length and later timing [10].

In order to keep the data not biased, we decided to run the program with a Poe chatbot to replace ChatGPT 3.5 here again.

Table 6. Timing and Accuracy of Contour Detection when Communicating Between ChatGPT and WeChat with User Suspicion Rate and Reason [Try 2].

	Timing(Benchmark of 100 secs)	Accuracy of Contours(out of 12)	Suspicion from User(out of five)	Highest Suspicion Reason
1A	Delayed	10/12	2/5	Timing
2A	Delayed	9/12	2/5	Over enthusiasm
3B	Good	10/12	1/5	Timing

Table 6. (continued).

4C	Delayed	11/12	3/5	Timing
5D	Good	11/12	2/5	Length

Table 7. CPU and RAM Percentage on Different Computers for Experiment.

Hardware Core	Used (Peak)	Software	Timing Compared to Test [3.2](ms)	RAM	Used(Average)
Intel i7 10th Gen Quadcore	—	MacOS Beta 13	—	16GB	—
M1 Ultra 48 Core	62.87%	MacOS 13	+111823	64GB	44.85Gb

With a full load and manual assisting, still our program reaches 62.87% CPU at peak workload on the M1 Ultra Computer, and No Response on the i7 model. Here we found that timing is the problem worth the most attention in the program is the timing of the program or the delay.

4. Conclusion

In this paper, we presented a novel approach to automatically detecting and interacting with contours on desktop screenshots. Our approach uses machine learning techniques to classify regions of interest (ROIs) within the screenshot into known contours. The program is able to control the contours on its display and use built-in functions to communicate with apps and exchange results. On the other hand, the query and built-in database function allows the program to organize data, extract the most relevant ones, and account them into the same query to the applications one is trying to link. Through experiments described in the paper, we have proven that the project is doable and achievable in theory. At the same time, there are still parts that need to be improved for better integration and timing that ensure the user not to be suspicious of the chatbot's lagging time. This can be enhanced by using pre-trained models that contain high accuracy or eliminating parts that pressure the system, such as word-by-word recognition being replaced with section scanning (which distinguishes which part of the screen is worth scanning in this case).

The final program we have tested serves as a third-party library that can be used for future projects to build upon this paper and we will continue to make the research more complete in future papers. In the future, we will try to deduce the timing that takes the program to switch and contact between applications and run OCR with contour detection as parallel threads to decrease the waiting time.

References

[1] "Contour Detection | Papers With Code." Papers With Code, 2023.
[2] Tsang I W , Kwok J T , Cheung P M .Core Vector Machines: Fast SVM Training on Very Large Data Sets.Journal of Machine Learning Research, 2005, 6(1):363-392.
[3] Boellaard R , Krak N C , Hoekstra O S ,et al.Effects of Noise, Image Resolution, and ROI Definition on the Accuracy of Standard Uptake Values: A Simulation Study.Journal of Nuclear Medicine, 2004, 45(9):1519-1527.
[4] "The ChatGPT Model: A Real-Life Example - DZone." DZone, 2023.
[5] Qu Z , Song X , Zheng S ,et al.Improved Bayes Method Based on TF-IDF Feature and Grade Factor Feature for Chinese Information Classification, IEEE International Conference on Big Data & Smart Computing.IEEE, 2018.
[6] Soares L C , Kaster D S .cx-Sim: A Metric Access Method for Similarity Queries with Additional Conditions. Journal of Information & Data Management, 2013, 4(3).

[7] Devlin, Jacob, et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv preprint arXiv:1810.04805 (2018)
[8] Nandwani, Pansy, and Rupali Verma. “A review on sentiment analysis and emotion detection from text.” Social Network Analysis and Mining 11 (2021)
[9] Turing, Alan. “Twenty Years Beyond the Turing Test: Moving Beyond the Human Judges Too.” arXiv preprint arXiv:2004.14107 (2020).
[10] “An Overview of Optimization | Papers With Code.” Papers With Code, 2023.

Appendix

Demo script in Language N on Python for experiment [1.1]:
This is a demo script for running the program:

```
WHILE TRUE:
   ANS = N(['WAIT 1 30','WAIT 1 30 1',"WINDOW_TEXT 1 SAFARI 950 180 1020 1000"])
   ANS = N([BOX ANS 240,250; 240,250; 240,250]) #BOX MEANING GET TEXTBOX WITH BG COLOR
OF R;G;B
   ANS = N([‘KEEP -U ‘+ANS[0]],  ’PRESS_KEY TAB'])
   IF 'GK LOAD FAILED' IN ANS:
      WHILE 'GK LOAD FAILED' IN ANS:
              N(['CLICK_MOUSE 1470 1310'])
         ANS = N(['WAIT 1','WAIT 1 30',"WINDOW_TEXT 1 SAFARI 950 180 1020 1000"])
         ANS = N([BOX ANS 240,250; 240,250; 240,250])
              ANS = N([‘KEEP -U ‘+ANS[0]],  ’PRESS_KEY TAB'])
   N(['UN_FULLSCREEN SAFARI','WAIT 2','SWITCH_WINDOW WECHAT’,"WAIT 1","KEEP
"+ANS,"TYPE_TEXT -K","WAIT 1","PRESS_KEY ENTER",'WAIT 0.5',"WAIT 2 720"])
   ANS2 = N(["WINDOW_TEXT 1 WECHAT 500 85 2025 1180",'WAIT 1'])
   ANS2 = N(["WINDOW_TEXT 1 WECHAT 500 85 2025 1180",'WAIT 1'])
   ANS2 = N([“BOX ANS2 250, 255; 250, 255; 250, 255])
   N(['UN_FULLSCREEN WECHAT', 'WAIT 0.1’,’UN_FULLSCREEN WECHAT',"SWITCH_WINDOW
CHATGPT","FULLSCREEN SAFARI”,"WAIT 1","KEEP "+ANS2[0],"TYPE_TEXT -K","WAIT
1","PRESS_KEY ENTER"])
```

Demo script for experiment [1.2] in Language N on Python:
Therefore the demo script will look like this:

```
db = ChatDatabase('messages.txt')
while True:
   ans = N(["window_text -d 1 Safari 950 180 1020 1000"])
   ans = N([box ans 240,250; 240,250; 240,250])
   ans = N([‘keep -u ‘+ans[0]],  ’press_key tab'])
   if 'Gk Load failed' in ans:
      while 'Gk Load failed' in ans:
           N(['click_mouse 1470 1310'])
        ans = N([“window_text -d 1 Safari 950 180 1020 1000"])
        ans = N([box ans 240,250; 240,250; 240,250])
           ans = N([‘keep -u ‘+ans[0]],  ’press_key tab’])
  Dolist = N([‘interpret ‘+ans])
db.add_message(‘ChatGPT’,convert_timestamp(time.ctime()),(ans_mod.replace('\n','')).replac
e('\t',''))
   N(['un_fullscreen Safari','wait 2','switch_window WeChat’,”wait 1","keep "+ans,"type_text
-k","wait 1","press_key enter”]+Dolist)
   ans2 = N(["window_text -d 1 WeChat 500 85 2025 1180”])
   ans3 = N(["window_text 1 WeChat 400 0 800 80”])
   ans2 = N([“box ans2 250, 255; 250, 255; 250, 255])
   ans3 = N([“box ans2 230, 255; 230, 255; 230, 255])
   ans2 = ans2[0] #most recent message
db.add_message(str(ans3[0].replace('\n','')),convert_timestamp(time.ctime()),(ans2_mod.repla
ce('\n','')).replace('\t',''))
   ans2 += ’What users can press on the screen’+N([‘contour -l —str’])
   N(['un_fullscreen Wechat', 'wait 0.1’,’un_fullscreen Wechat',"switch_window
ChatGPT","fullscreen Safari”,"wait 1","keep "+ans2,"type_text -k","wait 1","press_key
enter”])
```

Demo script for experiment [2.1] in Language N on Python:``
#Condensed Version

This part of the experiment is assisted by manual handling, such as parts in the demo script provided below that are blurred are being manually configured. ANY ERROR WILL BE TAGGED WITH {ER:…}

```
Db.load(‘messages.txt’)
For i in range(3):
        N([‘window_text -d WeChat’,’keep -src’])#Or Discord
   If i <= 2:
        N([‘Switch to Bing’,’mouse_click 260 1300’,’paste’,’press_key enter’]) #”Switch to Bing’ is a
default command that switches window to bing and presses tab
        ans = findocr(N([‘window_text 1 bing’]))[0]
        N([‘Switch to ChatGPT’]) #Also defaulted option
        N([‘type_text what should we do if we am new to an social media application and we know
that the software interface looks like this: ’+ans])
        ans2 = findocr(N([’window_text -d 1 ChatGPT’]))[0]
        N()#Your way of cropping the files and saving them to the folders
        acn,loc,locy = N([‘exclude_name ‘+ans])
        N([‘dtrain -m ‘+your saved folder])
        {ER: sometimes dtrain runs too slowly or the regrouping by using k-means is too slow, by
using manually assist we have skipped the step of total regrouping, where we decreased the level of
sub datasets to two only either as buttons or text and their subfield. The models are retrained once the
subsets are being edited}
   Else:
        N([‘Switch to Bing’,’mouse_click 260 1300’,’paste’,’type_text so my commands include…for
my program, can u write a command list for my program to run without error if we want to …’])
#”Switch to Bing’ is a default command that switches window to bing and presses tab
        ans = findocr(N([‘window_text 1 bing’]))[0]
        Dolist = N([‘interpret ‘+ans])
        {ER: sometimes Dolist includes a few undoable actions where we have to neglect manually}
#Main Loop
While True:
        N(Dolist+[‘screentext…’,’switch to GPT’,…etc).
```

This is what the command from other applications will look like to the program and how it can be interpreted from this format[Appendix-Figure 1]

Future Planning and Improvement Ideas:

1. After the experiment held in this paper, we have defined the system as lightweight based on its performance on PCs or Desktop Computers. Meanwhile, this program is now only currently running on MacOS Software, hence Some functions such as the built-in screenshot in MacOS are more optimized than just using a library to take a screenshot on Windows. For this part, we might include the usage of Linux and terminal commands when the program faces unknown situations when they are processing. Such situations include not being able to read correctly, a large portion of text is incomplete, and unresponsive applications. The commands could be sent to the terminal by concluding the average time to achieve each step and calculating whether the current situation is normal or not. [10]
2. Optimize uncertain results: Sometimes when the results are incomplete, maybe a way to solve this is to use our chat database as a "Library of Words" and rearrange them in order to make the result more complete. Although this is not probably something that we wanted to see when the main program uses this base for complex actions, it can serve as a debug and override system when the main program is under emergency(either internet lost or data lost).
3. There could be another similar version to this lightweight system which "actually" changes every module to dynamically train for datasets, however, that project will not be helpful to normal users. The system modification might still be able to be placed upon a company's server or other platforms that include a virtual operating system and this could be used to train a model that allows the lightweight version to process automation without updating data.

Our expectations of a true completed lightweight system for automatic communication:

	Timing(Benchmark of 30 secs)	**Accuracy of Contours(out of 12)**	**Suspicion from User(out of five)**	**Highest Suspicion Reason**	**Used(Peak on M1 Ultra 48 Core 64Gb RAM)**	**Software**	**RAM Usage Peak**	**Used(Average)**
1A	Good	11/12	1/5	Timing	Below 55 Percent	MacOS Beta 13	50 GB	40% CPU 80% GPU 32Gb RAM

Appendix Table 1: The Stats of the Program in Order to Achieve "Lightweight" Baseline [Expectation]

Table Eight describes the stats that we want to receive after running a lightweight system on a desktop computer with MacOS Software. Considering the fact that desktop computer chips develop very quickly recently, we set the peak expectation for CPU and RAM at 55% and 50Gb to ensure if the systems are made they are capable of running programs on different software systems on computers that have a chip speed that is alike. This is because we haven't tested the program on Windows and this might result in completely different statistics.

Moreover, also considering the fact that using RTX Graphics Card from NVIDIA could use CUDA when processing models which results in a very gigantic speed boost from regular computers or Apple's SoCs since their GPUs are not designed for handling such tasks, the expectation set could have differed from the one described above. In conclusion, we still use the performance of M1 Ultra 48 Core 64Gb RAM as the benchmark of the program on future computers and experiments.