Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Towards LLMCI - Multimodal AI for LLM-Vision UI Operation

Husam Barham

hbarham@uop.edu.jo

University of Petra

Mohammed Fasha

University of Petra

Research Article

Keywords: AI, HCI, LLM, Computer Vision, Multi-Modal AI, User Interface, Human-Computer Interaction, LLMCI, Multi-Agents

Posted Date: July 22nd, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4653823/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Towards LLMCI - Multimodal AI for LLM-Vision UI Operation

Husam Barham[1*] and Mohammed Fasha[2*]

[1*]Department of e-Business, University of Petra, Amman, 10181, Jordan.
[2*]Department of Business Intelligence and Analytics, University of Petra, Amman, 10181, Jordan.

*Corresponding author(s). E-mail(s): hbarham@uop.edu.jo; mohammed.fasha@uop.edu.jo;

Abstract

Human-computer interaction (HCI) has evolved significantly, yet it still largely depends on visual communication through screens and manual input devices. While this paradigm is likely to remain dominant for the foreseeable future, this research suggests that existing user interfaces (UI) can also be leveraged by Large Language Models (LLMs) to interact with computers. By integrating vision models into a multimodal framework, LLMs can gain the ability to understand and operate UI elements, enabling them to retrieve information, run functions, and perform various tasks just like humans. The framework utilizes a vision model to communicate UI components and information to the LLM, which then leverages its language understanding capabilities to retrieve information, and operate keyboard and mouse inputs. This paper introduces a new element to Human-Computer Interaction (HCI), called LLM-Computer Interaction (LLMCI), which combines Large Language Models (LLMs) with computer vision via intelligent agents. These agents process user text commands and use visual perception to recognize visual and textual elements of computer interfaces. This allows the Multimodal AI to independently perform complex tasks and navigate applications in a way that resembles human behavior. We present a proof-of-concept framework that illustrates how the agent uses LLMs and computer vision to handle interface elements, complete tasks, and support users according to their instructions. This strategy closely imitates human interactions and suggests a path forward for enhancing HCI practices.

Keywords: AI, HCI, LLM, Computer Vision, Multi-Modal AI, User Interface, Human-Computer Interaction, LLMCI, Multi-Agents

1 Introduction

Human-computer interaction (HCI) has evolved significantly over the past few decades, transitioning from command-line interfaces to graphical user interfaces (GUIs) and now to more natural and intuitive interfaces such as voice and gesture recognition [1]. However, despite these advancements, HCI still largely depends on visual communication through screens and manual input devices [2].

While this paradigm is likely to remain dominant for the foreseeable future, recent advancements in artificial intelligence (AI), particularly in the field of large language models (LLMs), offer new possibilities for HCI. LLMs, such as GPT-3 [3] and LaMDA [4], have demonstrated remarkable capabilities in understanding and generating human-like text, raising the question of whether they can also be used to interact with computers through existing user interfaces (UI).

This research explores the potential of employing LLMs to understand and operate UI elements, paving the way for a new era of LLM-Computer Interaction (LLMCI). By integrating vision models into a multimodal framework, LLMs can gain the ability to "see" and interpret UI elements, enabling them to retrieve information, run functions, and perform various tasks just as humans do.

This paper proposes a novel multimodal framework for LLMCI and demonstrates its capabilities through an applied example. The framework leverages a vision model to identify and analyze UI elements [5], and an LLM to understand the context and the intent of the user's request. The LLM then uses this information to generate instructions for the vision model, which in turn interacts with the UI elements to complete the task. This research builds upon existing work in multimodal AI [6, 7]and LLM-based interaction [8, 9]. However, it differs from previous work by focusing specifically on the use of LLMs to understand and operate UI elements. The proposed framework employs a dedicated visual perception module to accurately locate and interpret UI components using only screenshots from the computer screen. This eliminates the dependency on underlying UI layout files and enables the LLM to operate effectively and leverage the vast amount of existing UI design functionalities, potentially enabling LLMs to interact with a wide range of applications and websites.

LLMCI builds upon the existing foundation of HCI by recognizing the importance of vision as the highest-bandwidth channel for information transfer to humans. UIs are specifically designed to convey information and facilitate control of computers through visual elements. LLMCI allows LLMs to tap into this existing infrastructure, enabling them to "see" and understand UIs just like humans do. This opens up new possibilities for human-computer collaboration, where LLMs can assist humans in a more integrated and seamless manner.

Recent advancements in multimodal large language models (MLLMs) have shown remarkable progress in visual comprehension capabilities, making the realization of LLMCI increasingly feasible. However, as highlighted by Zheng et al. (2024), even state-of-the-art MLLMs like GPT-4V [10] though possess great ability to analyze User Interface, and appropriate what buttons needs to be clicked based on tasks, but still lack sufficient visual and spatial perception capabilities to decide the exact location for an element on user interface, and the capability to click on it, operate user interfaces effectively, particularly on mobile devices. Existing approaches attempt to address this limitation by leveraging user interface layout files (Yang et al., 2023d; Zheng et al., 2024). However, these methods are often hindered by the lack of access to such files in many real-world scenarios.

This paper presents a purely vision-based solution to this challenge: a novel LLMCI framework that enables LLMs to operate computer interfaces through visual understanding of UI elements. The framework utilizes a dedicated visual perception module to accurately locate and interpret UI components using only screenshots from the computer screen. This eliminates the dependency on underlying UI layout files and enables the LLM to operate effectively in a wider range of scenarios.

1.1 Related Work

Multimodal Machine Learning (MML) is a rapidly growing field that combines data from multiple modalities, such as text, images, audio, and sensor data, to solve complex problems. MML has been successfully applied to a wide range of tasks, including image captioning [11], speech recognition [12], and sentiment analysis [13]. One of the key challenges in MML is how to effectively represent and fuse data from different modalities. Several approaches have been proposed to address this challenge, including early fusion, late fusion, and hybrid fusion [14]. Early fusion combines data from different modalities at the feature level, while late fusion combines data at the decision level. Hybrid fusion combines both early and late fusion techniques. Another challenge in MML is the aligning of data from different modalities. This is particularly challenging when the data is unaligned, such as in the case of image captioning, where the images and captions are not paired. Several approaches have been proposed to address this challenge, including canonical correlation analysis (CCA) [15] and dynamic time warping (DTW) [16]. LLM-based interaction is a relatively new field that explores the use of LLMs to interact with computers. LLMs have been used to develop chatbots [17], generate code [18], and even control robots [19]. This research builds upon existing work in MML and LLM-based interaction by proposing a multimodal framework that enables LLMs to interact with computers through user interfaces. The framework leverages a vision model to identify and analyze UI elements, and an LLM to understand the context and intent of the user's request. The LLM then uses this information to generate instructions for the vision model, which in turn interacts with the UI elements to complete the task.

This paper is organized as follows: Section 2 describes the LLMCI framework and its key components, including the visual perception module, the self-planning capabilities, and the self-reflection methods. Section 3 presents several test use cases that were performed using the framework, showcasing its flexibility and potentials. Finally, Section 4 discusses the benefits and limitations of LLMCI, as well as future research directions.

2 Multimodal-Vision Framework for LLMCI

This section describes the proposed multi-modal framework for LLM-Computer Interaction (LLMCI). The framework consists of two main components: a vision model and a LLM.

2.1 Vision Models:

The vision model is responsible for identifying and analyzing UI elements on the screen. It uses a combination of object detection and optical character recognition (OCR) to extract information about the UI elements, such as their type, location, and text content. In the implemented framework, the vision model uses the YOLO object detection algorithm [20] to identify UI elements. In the context of LLMCI, YOLO is used to identify and locate various UI elements on the screen, such as buttons, text boxes, images, and other interactive components. Here's how YOLO contributes to the framework:

1. Real-time Detection: YOLO can process images in real-time, making it suitable for dynamic UI environments where elements might change or move. This allows the LLM to receive up-to-date information about the UI state.
2. Accurate Localization: YOLO provides bounding boxes for detected objects, which define the element's location and dimensions on the screen. These bounding boxes are crucial for subsequent OCR and interaction tasks.
3. Class Identification: YOLO can classify detected objects into predefined categories, such as "Button", "Text View", or "Image View". This information helps the LLM to understand the type of UI element and its potential functionality.

Once the UI elements have been identified, the framework utilizes EasyOCR model [21] to extract text content from within the bounding boxes. This text content provides additional context and meaning to the UI elements, allowing the LLM to better understand their purpose and function. Here's how OCR complements YOLO in the framework:

1. Text Extraction: EasyOCR is a capable OCR library that we use to recognize and extract text from the cropped regions of the screenshot defined by the bounding boxes provided by YOLO. This extracted text is then associated with the corresponding UI element.
2. Contextual Understanding: The extracted text, along with the element's type and location, are fed to the LLM. This combined information allows the LLM to infer the element's functionality and understand the overall context of the UI.
3. Actionable Instructions: Based on its understanding of the UI, the LLM can then generate specific instructions for interacting with the elements. For example, it might instruct the vision model to click on a button with a specific text label or enter text into a particular text box.

Additionally, we used a multi-modal captioning model IMP-v1-3b-2024 [22] image to text for describing the screen based on the LLM prompt input, and to provide a visual feedback loop to the LLM for the results of operations on screen. IMP-v1-3b is able to understand and generate natural language, and it can also reason about the relationships between text and images.

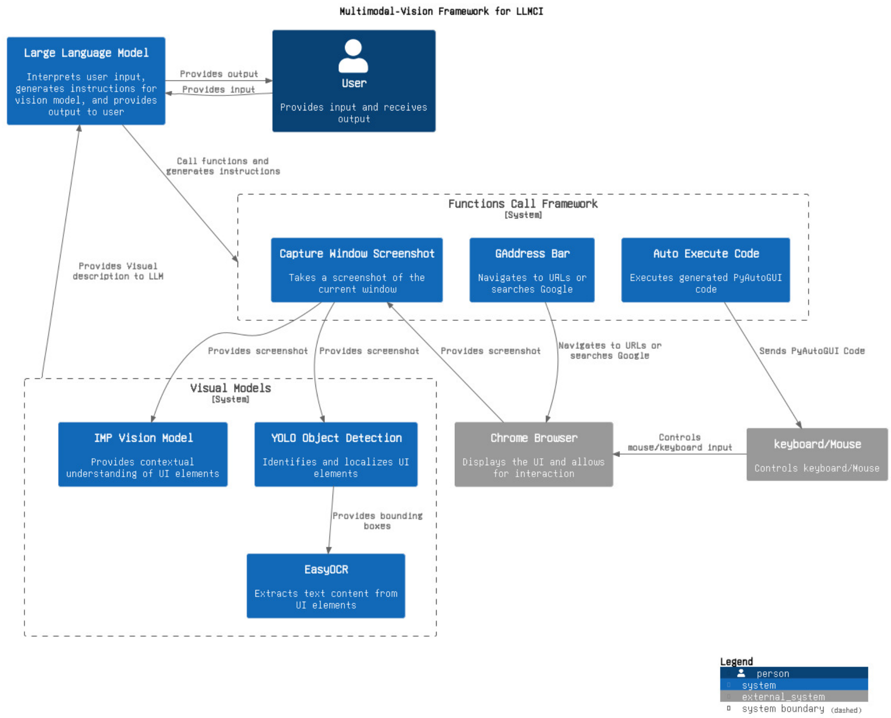

Fig. 1 LLMCI Visual-Interaction framework

2.2 Large Language Model:

In the implemented framework, the LLM used is GPT-3.5 [3], which is a large language model that has been trained on a massive dataset of text, additionally Autogen framework [23] has been used to design the interaction between the different elements. The LLM is responsible for understanding the context and intent of the user's request. It uses its knowledge of the world and its ability to understand natural language to interpret the user's request and generate responses that can be be divided into 2 main parts as shown in (refer Figure 1) which are instructions for desired system functions calling including function input arguments, and natural language responses to the user, as follow:

2.2.1 Function calling and instruction generation

LLMs have shown adequate ability in calling specific functions by generating JSON format responses [24] that include information for the requested function which programmatically facilitates running specific functions with the desired input arguments based on LLM interpretation of user requests. In the LLMCI framework, there are 4 main functions available for the LLM to choose and call from as required:

1. Navigate to URL or search Google: This function provides for LLM the ability to open Chrome window and navigate to a specific URL or to input

a search term in Google search engine. The JSON format that the LLM generates to run this function follows this LLM system message structure:

Listing 1 System Message to generate JSON format response thus navigate internet

```
{"name": "gaddressbar",
"description": "function␣to␣navigate␣URL,␣or␣search␣Google",
"parameters": {
"type": "object",
"properties": {
"url_or_search_term": {
"type": "string",
"description": "Write␣only␣the␣<URL>␣or␣the␣search␣term"}},
"required": ["url_or_search_term"]},}
```

2. Execute Code: This function auto-executes the code generated by LLM to control the mouse/keyboard. the JSON format that the LLM generates to run this function follows this LLM system message structure:

Listing 2 System Message to generate JSON response thus control mouse/keyboard

```
{"name": "auto_execute_code",
"description": "Write␣in␣python␣pyautogen␣code.",
"parameters": {"type": "object",
"properties": {"language": {
"type": "string",
"description": "language␣script␣type",},
"code": {"type": "string",
"description": Code to interact with screen elements",},},
"required":␣["language",␣"code"],},}
```

3. Capture screen: Capture screen is a form of two functions, one is showscreen(), this function describes the screen to the LLM using imp multimodal vision model [22], and the other function is getelements() function which identify the elements in the screen, their classes and spatial locations along with text extracted associated to them, this function uses 2 models first YOLO for object Identification [20], then EasyOCR to extract text within each object [21]. The framework then returns extracted information as an organized list to the LLM, following this structure [**h, v, object class, text found in element based on OCR**], where **h** is the center horizontal constraint of the element, and **v** is the center vertical coordinate of the element, extracting the center coordinates of each element and feeding them to the LLM offers advantages in terms of data efficiency and certainty, and clicking location certainty instead of providing the LLM the complete coordinates of the bounding box for every object (refer Figure 2). Here's how OCR complements YOLO in the framework:

(a) Text Extraction: EasyOCR is used to recognize and extract text from the cropped regions of the screenshot defined by the bounding boxes provided by YOLO. This extracted text is then associated with the corresponding UI element.

(b) Contextual Understanding: The extracted text, along with the element's type and location, is fed to the LLM. This combined information allows the LLM to infer the element's functionality and understand the overall context of the UI.

(c) Actionable Instructions: Based on its understanding of the UI, the LLM can then generate specific instructions for interacting with the elements. For example, it might instruct the vision model to click on a button with a specific text label or enter text into a particular text box.

2.3 Interaction Loop:

The vision model and the LLM work together in an interaction loop to complete the user's request. The interaction loop begins with the user providing a request to the LLM. The LLM then interprets the user's request and generates instructions for the vision model. The vision model then executes the instructions and provides feedback to the LLM. The LLM then uses this feedback to refine its understanding of the user's request and generate new instructions for the vision model. This process continues until the user's request is completed.

1. LLM Code Generation:

   - Input: The LLM receives information about the UI elements, including their type, location (center coordinates), and text content, from the vision model. Additionally, the LLM has access to the user's request or desired action.
   - Code Generation Process: The LLM leverages its understanding of natural language and its knowledge of PyAutoGUI [25], a generic python library that allows control of mouse and keyboard based on a Python script, the LLM generates Python code that performs the desired action on the specified UI element in the screen such as clicking, typing, drag/drop, etc.

2. IPython for Code Execution:

   - Integration: The generated Python code is then passed to IPython for execution. IPython provides an interactive environment for executing Python code, allowing the LLM's instructions to be carried out in real-time.
   - Execution and Feedback: IPython executes the code, controlling the mouse and keyboard to interact with the UI elements as specified by the LLM. After execution, the vision model can capture another screenshot and analyze the updated UI state. This feedback is then provided to the LLM, allowing it to verify the success of the action and make any necessary adjustments for subsequent interactions.

This framework enables the LLM to dynamically interact with the UI based on the user's requests and the current state of the interface, The use of PyAutoGUI provides a wide range of functions for controlling the mouse and keyboard, allowing the LLM to perform various actions on the UI, while IPython facilitates real-time execution of the

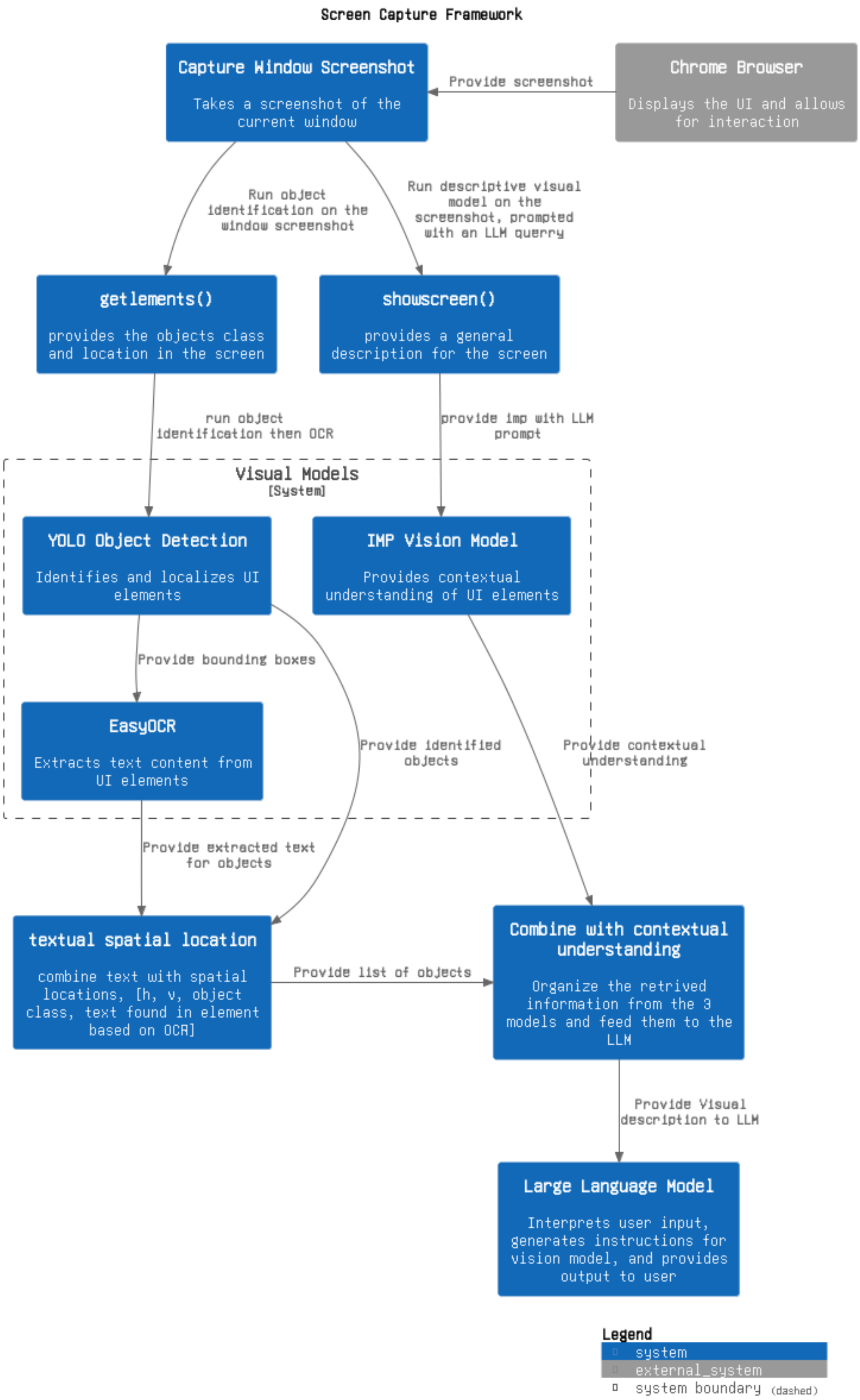

Fig. 2 Screen Capture Framework

generated code, making the interaction between the LLM and the UI more responsive and efficient.

Example 1. *Consider the following example: the user asks the planner to "draw a triangle at the center of the AutoDraw web app." The LLM interprets the user's request and generates the following instructions for the vision model:*

Open the AutoDraw web app.

Identify the drawing area.

Calculate the center coordinates of the drawing area.

Draw a triangle at the center coordinates. The vision model then executes these instructions and provides feedback to the LLM. The LLM then verifies that the triangle has been drawn correctly and provides feedback to the user, ***note:*** *video for this example is provided in the Annex.*

3 Implementation and Results

This section presents an applied example of the Multimodal-Vision Framework for LLMCI in action, specifically focusing on the LLM's ability to generate and execute PyAutoGUI code to interact with UI elements.

Scenario: The user provides the following instruction to the LLM:

"Go to this website https://getsitecontrol.com/p/0ewjmxr6 then type a message saying you are doing good then click on submit"

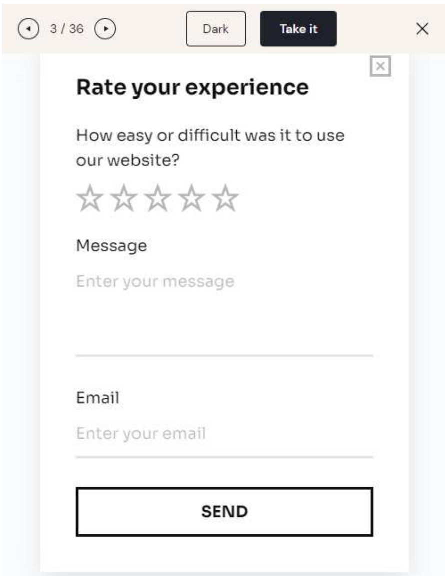

Fig. 3 Sample website form for testing LLMCI-Vision interaction, https://getsitecontrol.com/p/0ewjmxr6

Steps:

1. **Navigate to Website:** The LLM interprets the user’s request and identifies the need to navigate to the specified website. It calls the **gaddressbar(https://getsitecontrol.com/p/0ewjmxr6)** function with the URL as an argument.
2. **Analyze UI Elements:** The framework takes a screenshot of the webpage. The YOLO object detection algorithm analyzes the screenshot and identifies the UI elements, including their type, location (bounding boxes).
3. **Extract Text:** The bounding boxes are passed to the EasyOCR library, which extracts the text content from each element. extracted elements and texts for provided screenshot: **Mapped extracted text with class and coordinates:**

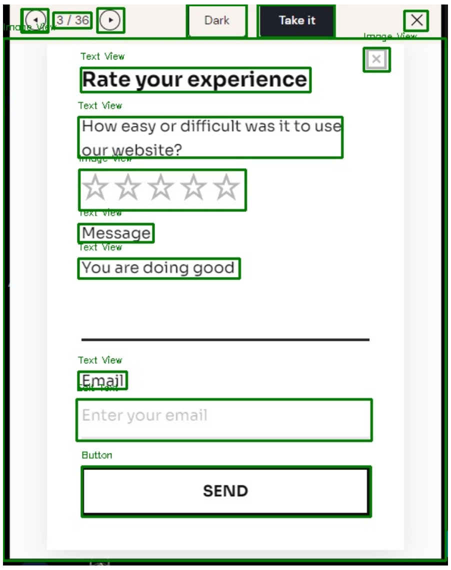

Fig. 4 Bounded boxes for detected elements and their coordinates

   [[258, 714, ’Button’, ’SEND’], [247, 169, ’Image View’, ’Dark’], [80, 168, ’Text View’, ’36’], [223, 237, ’Text View’, ’Rate your experience’], [478, 168, ’Image Button’, ”], [240, 304, ’Text View’, ’How easy or difficult was it to use our website?’], [185, 364, ’Image View’, ”], [115, 585, ’Text View’, ’Email’], [130, 414, ’Text View’, ’Message’], [124, 168, ’Image View’, ”], [259, 631, ’Edit Text’, ’Enter your email’], [36, 168, ’Image Button’, ”], [339, 169, ’Image View’, ’Take it’], [37, 168, ’Image View’, ”], [258, 491, ’Image View’, ’Rate your experience How easy or difficult was it to use our website? Message Enter your message Email Enter your email SEND’], [258, 520, ’Edit Text’, ”]]
4. **Image Description:** The image and extracted information are sent to the IMP vision model along the initial prompt for contextual understanding. The IMP vision model provides a summary of the relation of image to the user prompt, and potential functions, and then the combined information is sent to the LLM as follows:

[The image shows a screenshot of a website or app that asks the user to rate their experience. The user is required to enter their rating and provide a message explaining their feedback. The rating is displayed in the top left corner of the image, and the message box is located in the bottom right corner. The overall layout suggests that the website or app values user input and encourages them to share their thoughts on their experience.', "Here are the elements inside the webpage with x,y locations and object identification/OCR content of elements: ['Mouse Position: x=966, y=563', 'Window title is:Rate your experience', [80, 168, 'Text View', '36'], [247, 169, 'Image View', 'Dark'], [339, 169, 'Image View', 'Take it'], [223, 237, 'Text View', 'Rate your experience'], [240, 304, 'Text View', 'How easy or difficult was it to use our website?'], [130, 414, 'Text View', 'Message'], [258, 491, 'Image View', 'Rate your experience How easy or difficult was it to use our website? Message Enter your message Email Enter your email SEND'], [115, 585, 'Text View', 'Email'], [259, 631, 'Edit Text', 'Enter your email'], [258, 714, 'Button', 'SEND']]]

5. **Type Message:** The LLM analyzes the UI information and identifies the message box element based on its location and text content (e.g., "Enter message"). It generates PyAutoGUI code to click on the message box and type the message "You are doing good",

   Listing 3 Generated python script

```
{"language":"python",
"code":"import pyautogui\n
import time\n
# Click on the message box\n
pyautogui.click(130, 414)\n
# Type the message\n
pyautogui.write('You are doing good')"}
```

   The autoExecuteCode() function executes this code, and the message is typed into the message box.

6. **Click Submit:** The LLM identifies the submit button based on its location and text content (e.g., "SEND"). It generates PyAutoGUI code to click on the submit button:

   Listing 4 Generated python script

```
{"language":"python",
"code":"
# Click on the submit button
pyautogui.click(258, 714)"}
```

   The autoExecuteCode() function executes this code, and the submit button is clicked.

7. **Feedback and Confirmation:** After each interaction step (typing and clicking), the vision model re-analyzes the UI to capture any changes. The LLM receives

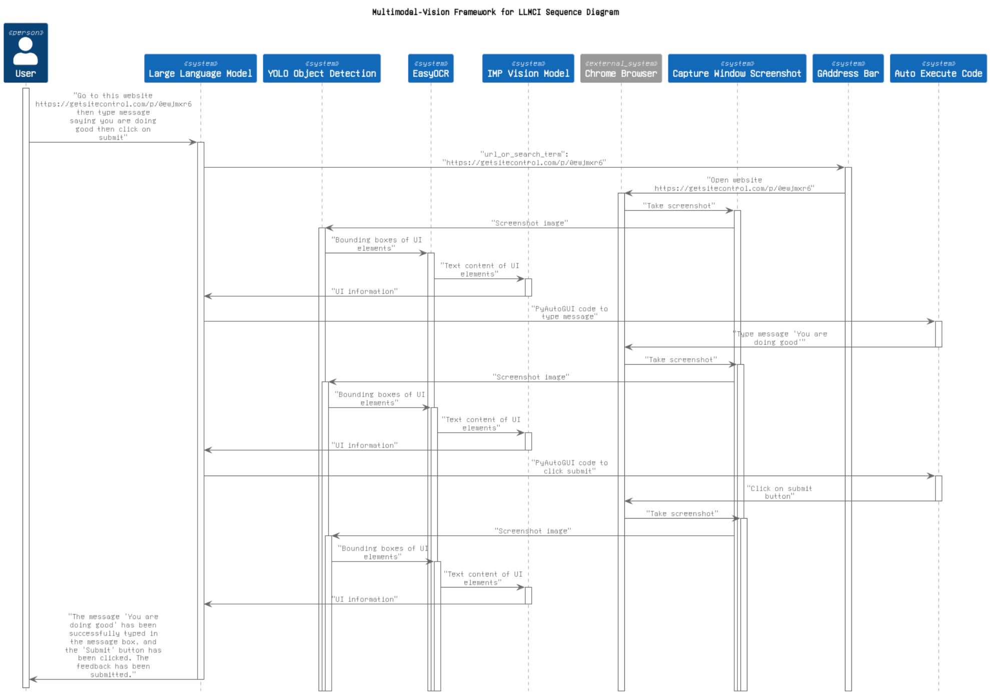

Fig. 5 LLMCI-Vision scenario sequence diagram

feedback from the vision model and confirms that the actions were performed successfully. The LLM provides output to the user, confirming that the message was typed and the feedback was submitted [”The message ”You are doing good” has been successfully typed in the message box, and the ”Submit” button has been clicked. The feedback has been submitted. Is there anything else you would like to do on this website?”]. The complete sequence diagram is presented (refer Figure 5), and a video record of the scenario is in the Annex.

This example demonstrates the LLM’s ability to understand the user’s request, analyze the UI elements, generate appropriate PyAutoGUI code, and execute the code to interact with the website. This showcases the potential of the Multimodal-Vision Framework for LLMCI to enable LLMs to perform complex tasks that involve interacting with computer interfaces.

4 Conclusion and Future Work

In conclusion, this paper introduces a groundbreaking framework for Large Language Model-Computer Interaction (LLMCI), which integrates computer vision and language understanding to enable LLMs to interact with user interfaces (UIs) in a manner resembling human behavior. By combining vision models with LLMs, the proposed framework facilitates the understanding and operation of UI elements, empowering

LLMs to retrieve information and execute tasks across a wide range of applications and websites.

Our multimodal framework leverages object detection, optical character recognition (OCR), and natural language understanding to enable LLMs to interpret user requests, analyze UI elements, generate PyAutoGUI code, and execute interactions with the UI. This approach eliminates the dependency on underlying UI layout files and enables dynamic interaction based solely on screenshots, thereby enhancing the versatility and adaptability of LLMCI. And allowing LLM to interact with the UI through simulated mouse and keyboard clicks.

The implemented framework utilizes YOLO for object detection, EasyOCR for text extraction, and IMP-v1-3b-2024 for image-to-text description which work together. Furthermore, the LLM can generate Python code using PyAutoGUI to interact with the UI elements based on the user's request.

The applied examples demonstrated the efficacy of the framework in performing complex tasks such as filling out forms on websites, showcasing its potential to revolutionize human-computer interaction. However, there are still areas for improvement and future research directions. Future work could focus on enhancing the accuracy and efficiency of UI element detection and classification, optimizing the vision-captioning model to provide more informative feedback to the LLM, and further refining the interaction loop between the LLM and the UI. Additionally, noise reduction techniques for captured image information and reinforcement learning for PyAutoGUI outputs could be explored to improve the robustness and adaptability of the framework across diverse UI environments.

In summary, the proposed LLMCI framework represents a significant advancement in the field of LLM-computer interaction, offering a promising path toward seamless collaboration between humans and AI agents in interacting with computer interfaces. Further research and development in this area hold the potential to transform the way we interact with technology in the future. And enabling LLMs to perform more complex tasks on our behalf.

5 Declarations

The dataset used and GitHub code repository would be provided by the corresponding author upon a reasonable request.

5.1 Funding and/or Conflicts of interests

The authors have no relevant financial or non-financial interests to disclose.

References

[1] Rogers, Y.: HCI Theory: Classical, Modern, and Contemporary. Springer

[2] Dix, A.: Human–computer interaction: A stable discipline, a nascent science, and the growth of the long tail. Interacting with computers **22**(1), 13–27 (2010)

[3] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)

[4] Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al.: Lamda: Language models for dialog applications. arXiv preprint arXiv:2201.08239 (2022)

[5] Chen, J., Xie, M., Xing, Z., Chen, C., Xu, X., Zhu, L., Li, G.: Object detection for graphical user interface: Old fashioned or deep learning or a combination? In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1202–1214 (2020)

[6] Baltrušaitis, T., Ahuja, C., Morency, L.-P.: Multimodal machine learning: A survey and taxonomy. IEEE transactions on pattern analysis and machine intelligence **41**(2), 423–443 (2018)

[7] Atrey, P.K., Hossain, M.A., El Saddik, A., Kankanhalli, M.S.: Multimodal fusion for multimedia analysis: a survey. Multimedia systems **16**, 345–379 (2010)

[8] Shuster, K., Xu, J., Komeili, M., Ju, D., Smith, E.M., Roller, S., Ung, M., Chen, M., Arora, K., Lane, J., et al.: Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. arXiv preprint arXiv:2208.03188 (2022)

[9] Roller, S., Dinan, E., Goyal, N., Ju, D., Williamson, M., Liu, Y., Xu, J., Ott, M., Shuster, K., Smith, E.M., et al.: Recipes for building an open-domain chatbot. arXiv preprint arXiv:2004.13637 (2020)

[10] Yang, Z., Li, L., Lin, K., Wang, J., Lin, C.-C., Liu, Z., Wang, L.: The dawn of lmms: Preliminary explorations with gpt-4v (ision). arXiv preprint arXiv:2309.17421 **9**(1), 1 (2023)

[11] Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3156–3164 (2015)

[12] Palaskar, S., Sanabria, R., Metze, F.: End-to-end multimodal speech recognition. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5774–5778 (2018). IEEE

[13] Soleymani, M., Garcia, D., Jou, B., Schuller, B., Chang, S.-F., Pantic, M.: A survey of multimodal sentiment analysis. Image and Vision Computing **65**, 3–14 (2017)

[14] Lahat, D., Adali, T., Jutten, C.: Multimodal data fusion: An overview of methods,

challenges, and prospects. Proceedings of the IEEE **103**(9), 1449–1477 (2015)

[15] Hotelling, H.: Relations between two sets of variates. Biometrika **28**(3/4), 321–377 (1936)

[16] Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing **26**(1), 43–49 (1978)

[17] Shuster, K., Chang, M.W., Lapedriza, A.G.: BlenderBot 3: a Deployed Conversational Agent that Continually Learns to Responsibly Engage (2022)

[18] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto: Evaluating Large Language Models Trained on Code (2021)

[19] Ahn, S., Shridhar, M., Hajishirzi, H.: Do as I Can, Not as I Say: Grounding Language in Robotic Affordances (2022)

[20] Jocher, G., Chaurasia, A., Qiu, J.: Ultralytics YOLOv8. https://github.com/ultralytics/ultralytics

[21] Vedhaviyassh, D.R., Sudhan, R., Saranya, G., Safa, M., Arun, D.: Comparative analysis of easyocr and tesseractocr for automatic license plate recognition using deep learning algorithm. In: 2022 6th International Conference on Electronics, Communication and Aerospace Technology, pp. 966–971 (2022). https://doi.org/10.1109/ICECA55336.2022.10009215

[22] Shao, Z., Ouyang, X., Yu, Z., Yu, J.: Imp: An Emprical Study of Multimodal Small Language Models (2024). https://huggingface.co/MILVLG/imp-v1-3b

[23] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., Wang, C.: Autogen: Enabling next-gen llm applications via multi-agent conversation framework. (2023)

[24] Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., Ji, H.: Executable code actions elicit better llm agents. arXiv preprint arXiv:2402.01030 (2024)

[25] Sweigart, A.: Pyautogui documentation. Docs **25** (2020)

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Video1.mp4
- Video2.mp4
- Video3.mp4