Utilizing Large Language Models to Analyze PSR.exe Recorded Input for Computer Use

Authors: YUAN Tianyu, YUAN Tianyu

Date: 2025-03-21T00:00:00+00:00

Abstract

The rapid development of Large Language Models (LLMs) has opened new frontiers for automating complex workflows. This paper explores an innovative approach that leverages Large Language Models (LLMs) to parse and interpret data recorded by PSR.exe—a tool for capturing user mouse and keyboard operations—to simulate computer usage. We propose a methodology for extracting, analyzing, and replicating user interactions documented in MHT files. By decoding screenshots and extracting action sequences, we aim to develop an automated pipeline that enables applications to effectively simulate user operations. This workflow integrates BeautifulSoup (for XML parsing), base64 (for image decoding), and LLMs (for semantic analysis). The results demonstrate that our approach is lightweight, versatile, capable of ensuring precision and adaptability, while simultaneously reducing reliance on external tracking tools.

Full Text

Preamble

Utilizing Large Language Models to Analyze PSR.exe Recorded Input for Computer Use
YUAN Tianyu
JAN 13, 2025

Abstract

The rapid advancement of Large Language Models (LLMs) has opened new frontiers in automating complex workflows. This paper explores an innovative approach to computer use simulation by leveraging Large Language Models (LLMs) to parse and interpret data recorded by PSR.exe, a tool designed to capture user's mouse and keyboard operations. We propose a method to extract, analyze, and replicate user interactions recorded in MHT files. By decoding

screenshots and extracting action sequences, we aim to develop an automated process that enables applications to emulate user operations effectively. The workflow combines BeautifulSoup for XML parsing, base64 for image decoding, and LLMs for semantic analysis.

Results show that our method is lightweight, versatile, and capable of ensuring precision and adaptability while reducing dependency on external tracking tools.

1.1 Background and Motivation

The Problem Steps Recorder (PSR.exe) records user interactions with computer interfaces, generating a detailed step-by-step MHT file containing screenshots and action descriptions. However, these files are inherently static and lack the operational structure required for automated simulation. Large Language Models (LLMs) such as GPT-4 offer a powerful mechanism for understanding and transforming such static records into actionable workflows, paving the way for automation in computer use.

1.2 Research Objectives

This paper outlines an LLM-driven framework for parsing MHT files created by PSR.exe, with the following objectives:

1.3 Significance

Our approach offers a lightweight solution for automating computer operations without relying on resource-intensive tracking tools. By integrating LLMs with PSR.exe, we enable a streamlined and generalized process for analyzing and reproducing user actions across diverse applications.

2 Related Work

Interface automation has traditionally relied on screen-tracking tools or APIs such as Win32. While these solutions are effective, they impose significant performance overhead on host systems and often require deep integration with the operating system. These conventional approaches also struggle with portability and adaptability across different application environments, limiting their utility in heterogeneous computing ecosystems.

2.2 LLMs in Workflow Generation

LLMs have demonstrated remarkable proficiency in natural language understanding and code generation, making them particularly suitable for parsing semi-structured data and generating executable workflows. Notable applications include robotic process automation and user interface testing, where LLMs

can interpret ambiguous instructions and translate them into precise action sequences. This capability positions LLMs as ideal candidates for bridging the gap between static documentation and dynamic automation.

2.3 PSR.exe Utilization

PSR.exe has been widely used for troubleshooting and documentation purposes, providing a simple yet effective way to capture user interactions for diagnostic analysis. However, its utility in automation remains underexplored, particularly in conjunction with modern AI technologies. The static nature of its output has historically limited its application to manual review rather than automated processing.

3.1.1 Parsing MHT Files

The recorded MHT file is parsed using Python libraries such as BeautifulSoup. The file contains XML-like structures that encapsulate action descriptions, screenshots, and metadata. The parsing process involves two primary steps: extracting XML units from the HTML body and isolating action descriptions along with their associated screenshots. This structured extraction forms the foundation for subsequent semantic analysis and workflow generation.

3.1.2 Screenshot Decoding

Screenshots embedded in the MHT file are Base64-encoded JPEGs. Decoding is performed using Python's base64 and PIL libraries, which convert the encoded strings into usable image objects. These extracted images provide essential visual context for each action, enabling the LLM to understand the precise state of the user interface at each step.

3.2.1 Structuring Steps into JSON

Each parsed step is formatted into JSON, containing the following fields:

```
{
  "step": 1,
  "description": "Open File Menu",
  "screenshot": "base64_{{{encoded}}_{{image}}}",
  "action": "click"
}
```

Incorporating LLMs, steps and screenshots are transmitted in batches to an LLM. The LLM processes the input, generating actionable descriptions in the format: "Application - Field - Action". Examples include "Notepad - File Menu - Click" and "Browser - URL Bar - Type". This standardized format ensures consistency across different applications and interaction types.

3.3.1 Eliminating Redundant Actions

Redundant steps such as unnecessary mouse scrolling are identified and removed through heuristic analysis. Additionally, actions involving dropdown menus are flagged for conditional checks, as these often require special handling to ensure proper state management during automation. This refinement process streamlines the workflow and eliminates potential sources of error during execution.

3.3.2 Workflow Summarization

Steps are grouped by application context to form cohesive workflows. This segmentation aids in the logical organization of tasks and enables parallel processing when appropriate. By clustering related actions, the system can generate more intuitive and maintainable automation scripts that reflect the user's original intent.

3.4 Visualization

The finalized workflow is converted into a Mermaid-compatible format and rendered as an HTML flowchart. The workflow depicts sequential and conditional branches, aiding in intuitive understanding and validation of the automation logic. This visual representation serves as both documentation and a verification tool for developers.

Implementation

4.1 Tools and Libraries

The implementation leverages several key technologies: - **Python**: For Base64 decoding and HTML parsing using BeautifulSoup - **Pillow**: For image processing and manipulation - **Mermaid.js**: For workflow visualization and flowchart generation - **LLM**: GPT-4 or equivalent models for step analysis and semantic interpretation

4.2.1 Parsing MHT Files

```
from bs4 import BeautifulSoup
import base64

# Parse MHT file
def parse_{mht}(filepath):
    with open(filepath, 'r', encoding='utf-8') as file:
        soup = BeautifulSoup(file, 'html.parser')
    return soup
```

4.2.2 Decoding Screenshots

```
from PIL import Image
from io import BytesIO

def decode_{screenshot}(base64_{data}):
    image_{data} = base64.b64decode(base64_{data})
    return Image.open(BytesIO(image_{data}))
```

4.2.3 Sending Data to LLM

```
{
  "description": "User action description",
  "screenshot": "path_{{{to}}_{{screenshot}}}"
}
```

5.1 Advantages

Our methodology offers several compelling benefits: - **Low System Overhead**: Operates without additional tracking tools, minimizing resource consumption - **High Accuracy**: Integrates screenshots for precise context, reducing ambiguity in action interpretation - **Automation-Friendly**: Outputs standardized action sequences and workflows that can be directly executed - **Actionable Sequences**: Extracted sequences are concise and structured, with enhanced clarity from visual context - **Workflow Visualization**: Generated flowcharts accurately represent user operations, facilitating automated replication and manual review - **Performance Metrics**: Our approach exhibits low resource overhead, making it suitable for real-time applications and deployment on resource-constrained systems

5.2 Limitations

Several constraints must be acknowledged: - **Dependence on Screenshot Quality**: Poor image quality may impact LLM interpretation and lead to incorrect action classification - **Model Limitations**: LLMs require fine-tuning for domain-specific tasks to achieve optimal performance and may struggle with highly specialized applications

5.3 Discussion

The proposed methodology successfully automates the parsing and interpretation of PSR records, demonstrating the feasibility of converting static documentation into dynamic workflows. However, challenges such as ambiguous descriptions or missing screenshots require further enhancements, such as integrating heuristic algorithms or advanced image recognition models. Additionally, the system's performance depends heavily on the LLM's ability to understand context, which may vary across different interface designs and application domains.

5.4 Future Work

Several avenues for improvement are planned: - **Enhanced Image Analysis**: Integrate specialized vision models for deeper analysis of screenshots and improved element detection - **Adaptive Learning**: Train LLMs with domain-specific data to improve accuracy for particular application categories - **User Interaction**: Develop interactive interfaces for user feedback during workflow generation, enabling human-in-the-loop refinement

6 Conclusion

This paper presents a novel framework integrating PSR.exe and LLMs for computer use simulation. By parsing MHT files, refining steps, and visualizing workflows, our methodology enables efficient automation without additional tracking tools. Future work will explore adaptive improvements, including real-time feedback loops and expanded application domains, ultimately aiming to create a robust, general-purpose automation platform.

7 References

1. https://github.com/u3588064/WorkflowLearner
2. Vaswani, A. et al. “Attention Is All You Need.” NeurIPS, 2017.

```
import base64
import json
import email
from email import policy
from email.parser import BytesParser
from bs4 import BeautifulSoup
from PIL import Image
from io import BytesIO

# Parse MHT file for images
def parse_{{{mht}}_{{for}}}_{images}(file_{path}):
    with open(file_{path}, 'rb') as file:
        msg = BytesParser(policy=policy.default).parse(file)

    images = {}
    for part in msg.iter_{parts}():
        if part.get_{{{content}}_{{type}}}() == 'image/jpeg':
            content_{location} = part.get('Content-Location')
            base64_{data} = part.get_{payload}(decode=False)
            images[content_{location}] = base64_{data}
    return images

# Parse MHT file for steps
def parse_{{{mht}}_{{for}}}_{steps}(file_{path}):
```

```
    with open(file_{path}, 'r', encoding='utf-8') as file:
        content = file.read()

    soup = BeautifulSoup(content, 'html.parser')
    xml_{data} = soup.find('script', {'id': 'myXML'}).string
    xml_{soup} = BeautifulSoup(xml_{data}, 'xml')

    steps = []
    for action in xml_{soup}.find_{all}('EachAction'):
        step_{data} = {
            'step': action.get('ActionNumber'),
            'description': action.find('Description').text,
            'screenshot': action.find('ScreenshotFileName').text,
            'action': action.find('Action').text
        }
        steps.append(step_{data})
    return steps

# Decode Base64 image
def decode_{{{base64}}_{{image}}}(base64_{data}):
    image_{data} = base64.b64decode(base64_{data})
    image = Image.open(BytesIO(image_{data}))
    return image

# Process steps with LLM
def process_{{{steps}}_{{with}}}_{llm}(steps, images):
    results = []
    for step in steps:
        if step['screenshot'] in images:
            base64_{data} = images[step['screenshot']]
            screenshot = decode_{{{base64}}_{{image}}}(base64_{data})
            screenshot.save('/work/' + step['screenshot'], format='JPEG')

            # Mock LLM processing function
            llm_{output} = process_{{{with}}_{{llm}}}(step, screenshot='/work/' + step['scre
            step['llm_{output}'] = llm_{output}
        results.append(step)
    return results

# Organize steps
def organize_{steps}(results):
    organized_{steps} = []
    for result in results:
        if result not in organized_{steps}:
            organized_{steps}.append(result)
    return organized_{steps}
```

```
# Summarize workflow
def summarize_{workflow}(organized_{steps}):
    workflow = {}
    for step in organized_{steps}:
        app = step['description'].split('-')[0]
        if app not in workflow:
            workflow[app] = []
        workflow[app].append(step)
    return workflow

# Main function
def main(mht_{{{file}}_{{path}}}):
    steps = parse_{{{mht}}_{{for}}}_{steps}(mht_{{{file}}_{{path}}})
    images = parse_{{{mht}}_{{for}}}_{images}(mht_{{{file}}_{{path}}})
    results = process_{{{steps}}_{{with}}}_{llm}(steps, images)
    organized_{steps} = organize_{steps}(results)
    workflow = summarize_{workflow}(organized_{steps})

    with open('results.json', 'w', encoding='utf-8') as file:
        json.dump(results, file, ensure_{ascii}=False, indent=4)

    print("Final Action Sequence and Workflow:")
    print(json.dumps(workflow, ensure_{ascii}=False, indent=4))

if __{name}_ == "__{{{main}}_{":}}
    mht_{{{file}}_{{path}}} = 'Recording_{{{20241120}}_{{2136}}}.mht'
    main(mht_{{{file}}_{{path}}})
```

Note: Figure translations are in progress. See original paper for figures.