2024 Minor Programme HAN University of Applied Sciences, Data Driven Decision Making in Businesses

AutoDev: An LLM-Driven Automatic Developer

A report on the development and iterations of an Autonomous AI for Ubuntu VM

Abstract

This report details the development of an autonomous AI using GPT-4o to control an Ubuntu VM via SSH. The project includes multiple iterations and enhancements aimed at improving the AI's capabilities and efficiency in an SSH-terminal environment.

Demo Video:

https://youtu.be/AQcQk633QTk

Lars Cornelissen, l.cornelissen5@student.han.nl, https://larscornelissen.com/
HAN University of Applied Sciences

Date: 16-06-202

Table of Contents

Personal Notes:

1. I have been an OpenAI API Private beta tester since November 2021, that's why I picked to use OpenAI's platform for the LLM for this project. This project wasn't focused on picking the right LLM for the job, but for future projects it might be valuable to do more research into the capabilities of other LLM's.
2. I went to multiple voluntary courses this semester, including the ethics course, process mining bootcamp and the NEDAP trip. For this project I didn't do research into the ethics of making a replacement for software engineers. If I choose to grow this project bigger, I will make sure to include that into my next research.
   For the celonis report, see My Celonis Porfolio
3. The sourcecode is currently closed-source on https://github.com/ScorchChamp/AutoDev. Please send me a request for access if you want to access it.

Glossary:

1. **Large Language Models (LLMs):** A type of artificial intelligence designed to understand and generate human language. Examples include GPT-4o by OpenAI. LLMs are trained on extensive datasets and can perform tasks like text generation, translation, and summarization.
2. **SSH (Secure Shell):** A network protocol that provides a secure channel over an unsecured network, commonly used for remote login and command execution on servers. It ensures the confidentiality, integrity, and authenticity of data.
3. **Paramiko:** A Python library that facilitates the creation and management of SSH connections, supporting both client and server functionalities. It provides tools for executing commands and transferring files securely over SSH.
4. **MongoDB:** A NoSQL database that stores data in a flexible, JSON-like format. Known for its scalability and performance, MongoDB is used to handle large volumes of data efficiently.
5. **PyAutoGUI:** A Python module used for programmatically controlling the mouse and keyboard. It enables automation of GUI interactions by simulating user inputs like clicks and keystrokes.
6. **OCR (Optical Character Recognition**): A technology used to convert different types of documents, such as scanned paper documents or images, into editable and searchable data. In this project, OCR was used to recognize text on the screen.
7. **Flask:** A lightweight WSGI web application framework in Python. It is designed to make getting started with web development quick and easy, with the ability to scale up to complex applications. In this project, Flask was used to develop an API for executing mouse and keyboard actions.
8. **JSON (JavaScript Object Notation):** A lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. JSON

was used to structure the responses and commands exchanged between the AI and the SSH interface.

Introduction

Artificial Intelligence (AI) and machine learning have significantly advanced over the past decade, enabling the development of sophisticated systems capable of performing complex tasks autonomously. This report focuses on the creation of an autonomous AI developer, referred to as "Autodev," using GPT-4o. The primary objective of Autodev is to control an Ubuntu virtual machine (VM) via SSH, automating various tasks within a terminal environment.

The motivation behind this project stems from the increasing need for automation in software development. By leveraging AI, we aim to reduce human intervention, improve efficiency, and enhance the accuracy of routine operations. This project explores multiple iterations, each improving upon the previous one, to achieve a robust and effective solution.

The following sections of this report will detail the development process, including the initial setup, enhancements in memory management, GUI interaction attempts, and the final SSH-based implementation. Additionally, the report will discuss the challenges encountered, solutions implemented, and the overall performance of the system.

Why would this product be useful?

- **Automated software updates, security patches, and system monitoring** can save countless hours for IT staff, translating into substantial cost savings.
- **Scarcity of Skilled IT Professionals:** There is a global shortage of skilled IT professionals, making it difficult for organizations to find and retain the talent needed to manage their IT infrastructures. Automation helps bridge this gap by performing tasks that would otherwise require specialized skills and expertise.
    - Example: Small and medium-sized enterprises (SMEs) often struggle to afford dedicated IT staff. An autonomous AI system can provide these businesses with the necessary IT support without the need for extensive human resources.
- **Minimizing Human Error:** Human error is a leading cause of system downtime and security breaches. Automating system administration tasks reduces the risk of mistakes that can lead to significant disruptions and vulnerabilities.
    - Example: Automated processes can ensure that configurations are applied consistently across all systems, reducing the likelihood of errors that could compromise security or performance.
- **Rapid Response to Issues:** Automated systems can detect and respond to issues much faster than human administrators. This capability is crucial for maintaining high availability and performance in modern IT environments.
    - Example: An AI system can continuously monitor server performance and automatically execute corrective actions when anomalies are detected, ensuring minimal downtime.

Literature Review

The development of autonomous systems using AI and machine learning has been a focal point of research in recent years. This section reviews the relevant literature on AI-driven automation, large language models (LLMs), and their applications in system administration and software development.

AI-Driven Automation

AI-driven automation leverages artificial intelligence to perform tasks traditionally carried out by humans. According to a study by McKinsey & Company (2017), automation technologies, including AI, have the potential to take over more than 50% of current work activities. The adoption of AI in automation aims to increase efficiency, reduce errors, and allow human workers to focus on more strategic activities.

Large Language Models (LLMs)

Large language models like GPT-4o have shown remarkable capabilities in understanding and generating human-like text. OpenAI's research (2022) demonstrates that LLMs can perform a wide range of tasks, including language translation, summarization, and even basic reasoning. The application of LLMs in system automation is relatively new, but early results indicate significant potential for improving operational efficiency.

Automation in Software Development

Automation in software development has primarily focused on continuous integration/continuous deployment (CI/CD) pipelines, automated testing, and code generation. AI-driven approaches are expanding these capabilities. A study by Microsoft Research (Sadowski et al., 2018) highlights the use of AI to predict code defects and suggest fixes. Integrating LLMs into development environments promises to further streamline coding, debugging, and deployment processes.

Challenges and Considerations

Despite the advancements, several challenges persist in the implementation of AI-driven automation:

- **Security Concerns:** Automating administrative tasks introduces security risks, such as unauthorized access and execution of harmful commands.
- **Performance and Scalability:** Ensuring the AI system performs efficiently under varying loads is crucial.
- **Accuracy and Reliability:** The AI must make accurate decisions consistently to be trusted with critical tasks.

Methodology

This section outlines the methodology employed in developing the autonomous AI for managing an Ubuntu VM. The project involved several phases, each building upon the previous one to enhance the AI's capabilities and performance. The primary tools and technologies used include Python, Paramiko for SSH communication, and OpenAI's GPT-4o for language processing.

Initial Setup and Version 1

The initial version of the project focused on establishing a basic SSH connection using the Paramiko library. This version allowed the AI to execute commands on the Ubuntu VM but lacked the ability to operate in interactive mode, limiting its effectiveness.

- **Tools Used:** Python, Paramiko
- **Capabilities:** Basic command execution via SSH
- **Limitations:** No interactive mode, limited functionality

```
import paramiko

hostname = "localhost"
port = 3022
username = ---REDACTED---
password = ---REDACTED---

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(hostname, port, username, password)
stdin, stdout, stderr = ssh.exec_command("ls")
print(stdout.read().decode())
```

Figure 1 Initial Setup and Version 1

Enhanced Memory and Storage

To improve the AI's performance, memory management capabilities were added. This involved using MongoDB to store logs, credentials, and to-do items. Functions were created to manage these memory elements, allowing the AI to maintain context and state across sessions.

- **Tools Used:** Python, MongoDB
- **Capabilities:** Storing and retrieving logs, credentials, and to-do items
- **Benefits:** Improved context management and continuity

```python
from pymongo import MongoClient

client = MongoClient('localhost', 27017)
db = client['ai_memory']

def add_memory(memory_type, content, context=None, tags=None, summary=None):
    memory = {
        "type": memory_type,
        "content": content,
        "context": context,
        "tags": tags,
        "summary": summary
    }
    return db.memory_collection.insert_one(memory).inserted_id
```

Figure 2 Enhanced Memory and Storage

Version 2 - GUI Interaction

In the second version, the focus shifted to controlling the VM using screenshots, enabling the AI to interact with the GUI via mouse and keyboard actions. This was achieved using the PyAutoGUI library and OCR for text recognition. However, this approach faced challenges with coordinate accuracy and processing speed.

- **Tools Used:** Python, PyAutoGUI, OpenCV, Tesseract OCR
- **Capabilities:** GUI interaction using mouse and keyboard
- **Limitations:** Slow processing, inaccurate coordinates

```python
import pyautogui
import cv2
import pytesseract

def find_word(find_me, image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    data = pytesseract.image_to_data(gray, config='--psm 6')
    for i, word in enumerate(data.splitlines()):
        if i == 0:
            continue
        word = word.split()
        if len(word) == 12 and word[11] == find_me:
            x, y, w, h = int(word[6]), int(word[7]), int(word[8]),
int(word[9])
            pyautogui.click(x + w // 2, y + h // 2)
            return (x, y, w, h)
```

Figure 3 Version 2 - GUI Interaction

Return to SSH

The final version reverted to SSH with enhanced capabilities, leveraging structured prompts to guide the AI's actions efficiently. The AI could run bash commands, manage memory, and interact with the file system, providing a robust solution for autonomous VM management.

- **Tools Used:** Python, Paramiko, OpenAI GPT-4o
- **Capabilities:** Advanced command execution, memory management, context awareness
- **Benefits:** Improved performance and reliability

```
def get_prompt():
    return f"""
        -----
        You are an autonomous AGI that has SSH access to a remote server that
is all yours.
        You can do anything in the terminal.
        ---
        MAIN GOAL:
        {get_main_goal()}
        ---
        Make sure to make frequent todo items to make sure you can pick up
where you left off if you stop working. Make frequent memory items so you can
remember what you have done already, add the tag 'system' to always give it to
your next context windows. ...
        Don't give context in text. Always respond in JSON. You should always
return in the following format:
        {{
            "action": "function_name",
            "thoughts": "your thoughts on why you chose to do this",
            "keywords_for_next_query": ["Search term 1", "Search term 2"],
            "body": {{
                "key1": "value1",
                "key2": "value2",
                ... (see above for more details)
            }}
        }}
    """
```

Figure 4 Return to SSH

Implementation Details

This section delves into the technical implementation of the autonomous AI for managing an Ubuntu VM via SSH. It covers the detailed workings of memory and logs, API integration, and thread management to ensure efficient and reliable operations.

Memory and Logs

To facilitate efficient task management and continuity, the system employs a robust memory management mechanism using MongoDB. This includes storing and retrieving logs, credentials, and to-do items, ensuring the AI retains context across different sessions.

- **Memory Storage:** The AI can add, retrieve, and delete various types of memory, including bash logs, message logs, and to-do items.

```
import datetime
import memory

def add_bash_log(log):
    log["timestamp"] = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    memory.add_bash_log(log)

def add_message_log(message):
    message["timestamp"] = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    memory.add_message_log(message)

def add_todo_item(todo_item):
    return memory.add_todo_item(todo_item)

def get_todo_items():
    return memory.get_todo_items()

def remove_todo_item(todo_id):
    return memory.remove_todo_item(todo_id)
```

Figure 5 Memory and Logs

- **Credential Management:** Secure storage and retrieval of credentials ensure the AI can access necessary resources without exposing sensitive information.

```python
def add_credential(app, username, password):
    credential = {
        "app": app,
        "username": username,
        "password": password
    }
    return memory.add_credential(credential)

def get_credentials(app):
    return memory.get_credentials(app)

def remove_credential(app):
    return memory.remove_credential(app)
```

Figure 6 Credential Management

API Integration

The system includes an API for executing key functions such as clicking, typing, and performing special mouse actions. This API, developed using Flask, allows the AI to control the VM efficiently.

- **Mouse Actions:** The AI can perform various mouse actions including clicks, double-clicks, and scrolling.

```python
@app.route('/click', methods=['POST'])
def click():
    data = flask.request.json
    x = data['x']
    y = data['y']
    pyautogui.click(x, y)
    return 'Mouse clicked at ({}, {})'.format(x, y)

@app.route('/special_mouse_action', methods=['POST'])
def special_mouse_action():
    data = flask.request.json
    action = data['action']
    if action == 'scroll_up': pyautogui.scroll(1)
    elif action == 'scroll_down': pyautogui.scroll(-1)
    elif action == 'double_click': pyautogui.doubleClick()
    elif action == 'right_click': pyautogui.rightClick()
    return 'Performed special mouse action: {}'.format(action)
```

Figure 7 Mouse Actions

- **Keyboard Actions:** Functions for typing text and pressing key combinations allow the AI to interact with the system's GUI elements.

```python
@app.route('/type_text', methods=['POST'])
def type_text():
    data = flask.request.json
    text = data['text']
    pyautogui.typewrite(text)
    return 'Text "{}" typed'.format(text)

@app.route('/key_combination', methods=['POST'])
def key_combination():
    data = flask.request.json
    keys = data['keys']
    pyautogui.hotkey(*keys)
    return 'Key combination {} pressed'.format(keys)
```

Figure 8 Keyboard Actions

Thread Management

To ensure the system remains responsive, especially during long-running operations, thread management techniques were employed. This allows the AI to execute commands without blocking other tasks.

```python
def run_bash(command):
    def run_command(results):
        stdin, stdout, stderr = ssh.exec_command(command)
        stdout.channel.recv_exit_status()  # Wait for the command to complete
        output = stdout.readlines()
        error_output = stderr.readlines()

        # Combine standard output and error output, if needed
        combined_output = output + error_output
        add_bash_log({
            "command": command,
            "output": output,
            "error_output": error_output
        })
        results[0] = "\n".join(combined_output[-50:]) if combined_output else
"No output"

    # Create a thread to run the command
    results = [None]
    t = Thread(target=run_command, args=(results,))
    t.start()
    t.join(timeout=30)
    return results[0] if results[0] else "Command timed out. Did you run a
blocking command? Always prevent prompts"
```

Figure 9 Thread Management

Results and Discussion

This section presents the outcomes of the project, discussing the performance, challenges encountered, and the overall effectiveness of the autonomous AI system in managing the Ubuntu VM.

Performance

The autonomous AI system demonstrated significant improvements in performance and functionality through its various iterations. Key performance metrics include:

1. **Command Execution:** The AI successfully executed a wide range of bash commands, showcasing its ability to manage the VM efficiently.
2. **Memory Management:** The use of MongoDB for memory storage allowed the AI to maintain context and state across sessions, enhancing its decision-making capabilities.
3. **Response Time:** The final SSH-based version showed a marked improvement in response time compared to the GUI interaction version, which was hindered by slow processing and coordinate inaccuracies.

Key Achievements

1. **Autonomous Operation:** The AI operated independently, managing tasks such as file manipulation, system monitoring, and software installation without human intervention.
2. **Enhanced Context Awareness:** Through structured prompts and memory management, the AI maintained a high level of context awareness, ensuring continuity and efficiency in task execution.
3. **Effective API Integration:** The integration of APIs for mouse and keyboard control, although secondary to the final SSH approach, demonstrated the AI's versatility in interacting with different interfaces.

Challenges and Solutions

Several challenges were encountered during the development of the autonomous AI system, and various solutions were implemented to address them:

1. **Coordinate Accuracy in GUI Interaction:**
   - **Challenge:** The AI struggled with accurate coordinate identification in the GUI interaction version.
   - **Solution:** The project reverted to SSH-based control, which provided more reliable command execution without the need for precise GUI interactions.
1. **Performance Bottlenecks:**
   - **Challenge:** The GUI interaction version experienced slow processing times due to OCR and image processing overheads.
   - **Solution:** Transitioning back to SSH improved performance by eliminating the need for image processing and relying on direct command execution.
1. **Security Concerns:**
   - **Challenge:** Automating administrative tasks raised potential security risks.
   - **Solution:** Although security was intentionally deprioritized in the project's scope (to avoid updates and firewall configurations), future iterations could incorporate more robust security measures.

User Experience

User feedback was instrumental in refining the AI's capabilities. The iterative development process, combined with continuous testing and adjustments, ensured that the system met the desired functional requirements.

- **Ease of Use:** The AI's ability to execute complex tasks with minimal input made it a valuable tool for automating routine system administration tasks.
- **Reliability:** By maintaining context and state through memory management, the AI proved to be reliable and consistent in its operations.

Summary of Results

The project successfully developed an autonomous AI capable of managing an Ubuntu VM via SSH. The final version of the system demonstrated robust performance, efficient memory management, and reliable command execution. The transition from GUI interaction to SSH control significantly improved the system's responsiveness and accuracy.

Conclusion

The development of Autodev, an LLM-driven automatic developer, has demonstrated the significant potential of AI in automating tasks. Through various iterations, the project has showcased the versatility and effectiveness of using GPT-4o to manage an Ubuntu VM via SSH.

Key Findings

- **Efficiency:** The AI successfully executed a wide range of commands autonomously, significantly reducing the need for human intervention.
- **Context Management:** The integration of memory management using MongoDB allowed the AI to retain context and state across sessions, enhancing its decision-making and operational continuity.
- **Performance Improvement:** Transitioning from GUI interaction to SSH control markedly improved the system's response time and accuracy, addressing performance bottlenecks and coordinate inaccuracies.

Challenges Addressed

- **Coordinate Accuracy and Performance:** Switching to SSH eliminated the issues related to GUI interaction, providing a more reliable and efficient control mechanism.
- **Security:** While the project intentionally deprioritized security to focus on functionality, it highlighted areas for future improvements, such as incorporating robust security measures for automated systems.

Future Work

Future iterations of this project could explore the following areas:

2. **Enhanced Security:** Integrating security measures to safeguard against unauthorized access and command execution.
3. **Scalability:** Extending the AI's capabilities to manage multiple VMs simultaneously, improving scalability for larger deployments.
4. **Advanced Memory Management:** Developing more sophisticated memory management techniques to further enhance context awareness and operational efficiency.
5. **User Interface Improvements**: Refining the user interface for easier interaction and monitoring of the AI's activities.

Final Thoughts

The successful development and deployment of Autodev underscore the transformative potential of AI in automating complex tasks. By leveraging the power of large language models like GPT-4o, this project has paved the way for more advanced and efficient automation solutions in the future.

References

1. McKinsey & Company. (2017). Shaping the future of work in Europe's nine digital front runner countries. Retrieved from https://www.mckinsey.com/~/media/mckinsey/featured%20insights/europe/shaping%20the%20future%20of%20work%20in%20europes%20nine%20digital%20front%20runner%20countries/shaping-the-future-of-work-in-europes-digital-front-runners.pdf
2. OpenAI. (2022). GPT-4 research. Retrieved from https://openai.com/index/gpt-4-research/
3. Sadowski, C., van Gogh, S., Jaspan, C., Söderberg, E., & Winter, C. (2018). Tricorder: Building a program analysis ecosystem. Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. Retrieved from https://dl.acm.org/doi/pdf/10.1145/3183519.3183525