Ossarth : An Open-Source Customizable LLMOS

Siddharth Magesh[1]
Velammal Engineerning College
Artificial Intelligence and Data Science
Chennai , India
siddharthmagesh007@gmail.com

Deepak Kumar M[2]
Velammal Engineering College
Artificial Intelligence and Data Science
Chennai , India
deepakkumarm2458@gmail.com

Beno Dharmaraaj P J[3]
Velammal Engineering College
Artificial Intelligence and Data Science
Chennai , India
benodharmaraaj2905@gmail.com

Priya M[4]
Velammal Engineering College
Asst Prof of Artificial Intelligence and Data Science
Chennai , India
priya.m@velammal.edu.in

Abstract—**Ossarth OS is a customizable, LLM-based operating system that enables users to modify and extend its functionality according to their needs. Unlike traditional operating systems that rely on graphical interfaces, Ossarth OS allows users to interact with the system using multilingual natural language commands, making operations more intuitive and accessible. This system introduces a state-of-the-art AI-driven approach to OS customization, empowering users to define and integrate custom tools, automation scripts, and system utilities. To facilitate this, we provide a base OS, Ossarth, which serves as the foundation for user-defined modifications. Through the Ossarth framework, users can dynamically extend their OS capabilities, ensuring a flexible and personalized computing experience. This paper explores the design, implementation, and advantages of Ossarth OS, demonstrating its potential as a next-generation LLMOS for AI-powered system interaction and customization.**

Keywords—LLMOS, Ossarth, natural language interaction, AI-powered OS, system customization, extensible OS

I. Introduction

Traditional operating systems such as Windows, Linux, and macOS have evolved into complex environments that require users to navigate graphical user interfaces (GUIs) within constrained workflows. While these operating systems provide robust functionality, they often lack flexibility and accessibility for users unfamiliar with traditional computing paradigms. Additionally, mainstream OS platforms are either proprietary or have licensing constraints, making them inaccessible to a broader audience. Managing large volumes of files and performing search operations in such environments can also be cumbersome, leading to inefficiencies in system interaction.

To address these challenges, Large Language Model Operating Systems (LLMOS) have emerged as a novel paradigm, enabling users to interact with their systems using natural language commands rather than predefined graphical interfaces. However, existing LLMOS implementations, such as GitHub Copilot, primarily function as AI-powered assistants rather than deeply integrated system components. These solutions lack the capability to modify system functionalities dynamically, and many open-source implementations face hardware compatibility issues, limited customization, and the need for kernel-level modifications, making them difficult to deploy across diverse computing environments.

Several limitations exist within both traditional OS platforms and current LLMOS implementations. Customizability is a significant concern, as most operating systems do not provide users with the ability to modify core functionalities easily. Additionally, hardware and software constraints often hinder the adoption of LLM-based systems, as they require specific configurations and dependencies that limit portability. Existing AI-driven OS solutions also suffer from limited integration, functioning as separate applications rather than offering seamless interaction with system-level operations. Furthermore, optimization challenges arise due to the lack of adaptable AI models that cater to diverse user requirements.

To overcome these issues, we introduce Ossarth OS, an AI-powered, customizable, and modular LLMOS that allows users to modify and extend their operating system functionalities. Ossarth OS is designed to be highly flexible, enabling users to develop and integrate tools using high-level programming languages such as Python. Through the Ossarth framework, users can leverage prebuilt PyPI modules to automate tasks, customize system operations, and dynamically generate tools that enhance their computing experience. Unlike existing LLMOS implementations, Ossarth OS runs on top of any installed OS, ensuring parallel accessibility while allowing users to choose the AI model and inference method that best suits their needs.

By providing a scalable, modular, and AI-driven alternative to traditional operating systems, Ossarth OS represents a significant advancement in OS customization. This paper explores the architecture, implementation, and potential applications of Ossarth OS, demonstrating how it redefines system interaction through natural language-driven customization and intelligent automation.

II. Related Work

Recent advancements in AI-driven operating systems and voice-assisted computing have introduced new ways for users to interact with their devices. GitHub Copilot leverages large language models (LLMs) to assist developers by generating code suggestions, but it is limited to integrated development environments (IDEs) rather than

ISBN : 978-93-342-2299-9

functioning as a full-fledged OS. Similarly, AI-powered virtual assistants like Amazon Alexa, Apple Siri, and Google Assistant enable users to perform tasks via voice commands, yet their capabilities are constrained by predefined functionalities. Other LLM-based operating systems have emerged, but they often require complex dependencies or specialized environments, making them difficult to integrate with existing hardware and software ecosystems.

Despite these innovations, existing AI-powered systems suffer from significant limitations. Most LLMOS implementations function as chat-based assistants rather than deeply integrated operating systems. Virtual assistants like Siri and Alexa lack true customizability, restricting users to predefined tasks without the ability to extend functionalities dynamically. Moreover, many AI-driven OS solutions demand specific hardware configurations, making them less portable and difficult to scale. Open-source alternatives often face challenges in modularity, extensibility, and AI model adaptation, limiting their practical usability. Ossarth OS addresses these gaps by providing a fully customizable, modular, and AI-driven LLMOS, allowing users to modify, extend, and optimize their operating system according to their specific needs.

III. SYSTEM DESIGN

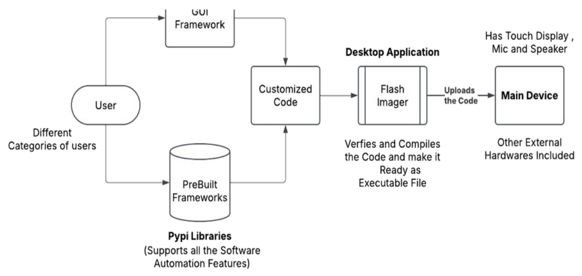
First, we need to have an OS, preferably Linux or Windows, but it can also be run on macOS. Then, we install Ossarth OS and run the executable, which launches the entire OS.

For customization, we use the Ossarth PyPI module within an Anaconda or Python environment. After installing the module, users can utilize the provided methods to modify the OS based on the official documentation available on the website. Customizations can be implemented in different ways, and once the necessary modifications are made, users execute the code to apply the changes.

Additionally, the Ossarth Imager application is available for flashing custom OS images. Users download the application, select the targeted files, and flash them onto the customized OS. This allows for deep system modifications, making Ossarth OS fully reactive and tailored to user preferences.

IV. SYSTEM ARCHITECTURE

The architecture of Ossarth OS follows a structured workflow that enables seamless customization and deployment. The user interacts with a GUI framework and can leverage prebuilt frameworks to generate customized code. This customized code is then processed and prepared for deployment using the flash imager, which uploads the modified OS to the main device. This modular approach ensures flexibility, allowing users to define and modify system functionalities based on their specific needs while maintaining portability across different devices.


GUI Framework
User
Different Categories of users
PreBuilt Frameworks
Pypi Libraries
(Supports all the Software Automation Features)
Customized Code
Desktop Application
Flash Imager
Verfies and Compiles the Code and make it Ready as Executable File
Uploads the Code
Has Touch Display , Mic and Speaker
Main Device
Other External Hardwares Included

V. IMPLEMENTATION DETAILS

Ossarth OS is built upon the foundation of Function Calling (also known as Tool Calling) and operates as an Agent-based AI system. It leverages LangChain as the core framework and utilizes Llama 3.2 7B as the primary LLM, which is locally inferred using Ollama. The inference setup was tested on a Windows environment with the following hardware configuration:

- Processor: AMD Ryzen 7 5000 Series
- GPU (Dedicated): NVIDIA RTX 3050 (CUDA 12.7)
- GPU (Integrated): Radeon Graphics
- RAM: 32GB
- Storage: 1.5TB SSD

This system ensures smooth execution and real-time inference while maintaining efficiency in processing user commands and executing OS functionalities.

System Execution and OS Interface

Ossarth OS is partially built using a Flask-based application, which serves a web interface resembling a traditional operating system. The core functionalities are structured as follows:

1. Rendering OS Interface:
    - The Flask application serves an index.html page that provides an OS-like environment.
    - The page consists of an input field where users can enter commands.
2. Command Processing Pipeline:
    - The input query is captured and sent through an execution pipeline.
    - The query is parsed to determine whether it requires LLM inference or execution of predefined tools.
    - If the query matches a system function, it invokes the corresponding tool.
    - If not, it is processed by the LLM, which generates the appropriate response.
3. Tool Invocation and Execution:

ISBN : 978-93-342-2299-9

- If the LLM detects that a specific system tool needs to be executed, it dynamically calls the function.
- The execution result is returned to the user through the web interface.

While the OS can be configured to run as the primary application on system startup, it is currently not advised due to early-stage development. Future iterations will integrate CUDA optimizations and automate various system functionalities.

Additionally, Ossarth OS can function as a virtualized environment, allowing it to run parallel to existing operating systems without interference.

VI. CUSTOMIZING OSSARTH OS WITH AI TOOLS

1. Installation of the Ossarth Framework

To begin customizing Ossarth OS, users must first install the core framework, which can be done via PyPI or GitHub. The installation through PyPI requires a simple command to install the ossarth package, while the latest version can be obtained directly from the GitHub repository. Once installed, users gain access to the built-in tool manager, which facilitates the creation and integration of custom AI tools and functions into the operating system.

2. Creating AI-Powered System Tools

Ossarth OS supports a modular approach to customization, allowing users to define system tools dynamically using Python. The first step in this process is importing the tool manager, which acts as the interface for managing and creating custom functionalities. Users can then define their own tools, such as retrieving disk usage statistics. By specifying details like the function name, description, expected parameters, return types, and execution logic, the system automatically generates a corresponding Python file. This file is stored within a dedicated directory for system tools, enabling Ossarth OS to dynamically execute the function whenever required. For example, a tool designed to check disk usage will generate a file that retrieves total, used, and free storage space, which can be accessed upon user query

3. Creating General Helper Functions

In addition to system tools, Ossarth OS provides the ability to create helper functions that assist with system monitoring and automation. A common example is a function that calculates the average CPU load over a specific time interval. This function, once created, is saved in the designated directory for helper functions, allowing the OS to monitor CPU performance dynamically. By defining such functions, users can enhance Ossarth OS with intelligent system monitoring capabilities, ensuring real-time access to crucial performance metrics.

Deploying Custom AI Tools to Ossarth OS

Once custom AI tools and helper functions are created, they need to be integrated into the operating system. This is done through the Ossarth Imager application.

Step 1: Install Ossarth Imager

The Ossarth Imager application is responsible for flashing custom tools to the OS. After installation, the application can be launched for further customization.

Step 2: Select the Custom Tools

- Open Ossarth Imager.
- Navigate to the Tools Directory where the customized tools are stored (e.g., custom_os_functions/ and custom_helpers/).
- Select the tools that need to be integrated into the OS.

Step 3: Flashing the Tools

- Once selected, the user can initiate the flashing process.
- The imager application embeds the selected tools into the OS, making them accessible within Ossarth.
- After the process is completed, Ossarth OS can now execute the newly integrated tools seamlessly.

Summary of Implementation Technique

The Ossarth OS customization technique introduces a modular and AI-powered approach to system functionality. By leveraging LLM-based function calling, users can:

- Dynamically create system tools tailored to their needs.
- Seamlessly integrate new functions without modifying core OS components.
- Execute commands in a natural language-driven environment, enhancing accessibility.
- Optimize system performance by customizing inference models and execution settings.

This novel methodology enables Ossarth OS to act as a flexible and customizable AI-driven operating system, revolutionizing the way users interact with their devices.

VII. DISCUSSION

Ossarth OS introduces a novel approach to AI-powered operating systems by leveraging modularity and user-defined customization. One of its key strengths is its ability to integrate AI tools seamlessly into an interactive OS environment, allowing users to create, modify, and deploy system functionalities without requiring deep technical expertise. The modular architecture ensures flexibility, enabling users to tailor the OS according to their specific needs. Additionally, the ability to run on top of existing operating systems makes Ossarth OS highly portable and accessible across multiple platforms without requiring extensive hardware modifications.

ISBN : 978-93-342-2299-9

However, the development process presented several challenges, primarily in optimizing performance and ensuring smooth interaction between the AI model and the system functions. Given that large language models (LLMs) require significant computational resources, optimizing inference speed while maintaining accuracy was a critical aspect of development. Another challenge was ensuring compatibility across different system environments, as users may have diverse hardware configurations and operating systems. Efforts were made to streamline the installation and execution process, but further refinements are necessary to enhance efficiency, particularly in lower-end devices.

A notable limitation of Ossarth OS is its dependency on file access permissions. If the user restricts access to certain system directories or files, the AI model cannot retrieve or process necessary data, limiting the effectiveness of automation and tool execution. To mitigate this issue, Ossarth OS includes permission management features, allowing users to grant selective access to critical directories while maintaining privacy and security. Future iterations will focus on developing more refined access control mechanisms that enable secure interactions between the AI model and system resources without compromising user control.

VIII.FUTURE WORK

1. Expanded Device Compatibility

We plan to extend Ossarth OS support beyond traditional PCs and laptops to devices such as Raspberry Pi, IoT devices, and ARM-based architectures. This will allow Ossarth OS to function as a lightweight AI-driven operating system for embedded systems, enabling automation in robotics, smart home applications, and industrial monitoring.

2. Development of Additional PyPIModules

To enhance customization and modularity, we will introduce three PyPI modules:

- Ossarth: The core framework for AI-driven OS functionalities.
- Ossarth-Community: A collaborative module where contributors can share custom tools, AI integrations, and automation scripts.
- Ossarth-External: A module designed for external device integration, allowing seamless interaction with sensors, smart devices, robotics, and third-party APIs.

3.User-Friendly GUI for AI Customization

A dedicated web-based GUI platform will be developed to allow non-programmers to configure AI functionalities visually. This drag-and-drop system will enable users to customize their AI assistants, select predefined automation tasks, and integrate system tools effortlessly, making AI-driven automation accessible to a broader audience.

4. Optimized AI Processing & Performance Enhancements

Future updates will focus on multi-threaded processing for faster execution, improved CUDA support for efficient LLM inference, and reduced latency in AI responses. This will ensure that Ossarth OS remains optimized for both high-performance GPUs and resource-constrained environments, enabling smoother AI operations across different hardware configurations.

5. Enhanced Security & System Integration

We will introduce granular access control mechanisms, ensuring that AI tools can function without compromising system security. This includes user-defined permission **settings**, encrypted data handling, and secure API integrations, allowing Ossarth OS to be deployed in sensitive computing environments while maintaining user control over data privacy.

IX. CONCLUSION

In conclusion, Ossarth OS redefines the concept of an AI-powered operating system by integrating modular AI tools, dynamic automation, and cross-platform compatibility. By leveraging LangChain, tool-calling LLMs, and a lightweight Flask-based interface, it offers a flexible and scalable solution for users to customize their computing environment efficiently. The system bridges the gap between traditional OS functionalities and AI-driven automation, making it a significant step forward in LLMOS research.

Ossarth OS provides an open-source, customizable platform that empowers users to create their own AI-driven tools and system functionalities. Its ability to execute AI-powered commands, automate tasks, and integrate seamlessly with user-defined modules makes it a powerful alternative to conventional operating systems. While challenges such as file access limitations and OS integration complexities remain, the system lays a strong foundation for further improvements.
With ongoing developments in device compatibility, external integrations, and AI-enhanced interfaces, Ossarth OS is poised to become a versatile and adaptive AI operating system. Future enhancements will focus on GPU acceleration, user-friendly GUI interactions, and community-driven innovation, ensuring its relevance and growth in the evolving landscape of AI-driven computing.

REFERENCES

[1] Operating System and Artificial Intelligence: A Systematic Review, CS.NTHU.EDU.TW. A review of AI integration in OS to enhance performance and efficiency.

[2] LLM as OS, Agents as Apps: Envisioning AIOS, Agents, and the AIOS-Agent Ecosystem, arXiv. Explores LLMs as OS and agents as applications within an AI-driven ecosystem.

[3] Integration of Machine Learning into Operating Systems: A Survey, IJCRT.ORG. Discusses ML

ISBN : 978-93-342-2299-9
9 789334 222999

techniques for OS automation, highlighting challenges and benefits.

A Review on Large Language Models: Architectures, Applications, and Trends, IEEE Xplore. Overview of LLM architectures, applications, and trends, including OS integration.

ISBN : 978-93-342-2299-9

9 789334 222999