ISSN NO: 2249-3034

Artificial Intelligence Assisted Computer Troubleshooting

Dr.ECCLESTON
Computer Engineering
VIT Pune
Pune, India
amruta.bhawarthi@vit.edu

Dr.SAI
Computer Engineering
VIT Pune
Pune, India
rajesh.mahajan232@vit.edu

Dr.HENRY
Computer Science(Artificial Intelligence)
VIT Pune
Pune, India
dhaval.gujar23@vit.edu

Vedang Mahajan
Artificial Intelligence and Data Science
VIT Pune
Pune, India
vedang.mahajan23@vit.edu

Aarya Makanikar
Information Technology
VIT Pune
Pune, India
aarya.makanikar23@vit.edu

Vedant Mahale
Computer Engineering
VIT Pune
Pune, India
vedant.mahalle231@vit.edu

Piyush Male
Computer Engineering
VIT Pune
Pune, India
piyush.male232@vit.edu

Rushikesh Malegave
Computer Engineering
VIT Pune
Pune, India
rushikesh.malegave23@vit.edu

Abstract — Computers are fickle things. Even with the enormous amounts of fail saves implemented in today's computers, there is no knowing when an unexpected problem might occur on your device. Traditional troubleshooting often requires considerable manual efforts and extensive knowledge, thus leads to large resolution times and a hefty operational cost. This calls for a need of an easy-to-use, easy-to-understand method which can give solutions to everyday problems which arise when using a computer. We aim to provide a solution to this problem via an AI-assisted troubleshooting framework that uses a comprehensive knowledge base and real-time data analysis to provide accurate and efficient solutions in a user-friendly way. This study highlights the transformative potential of AI in computer support and makes way for future advancements in automated technical assistance.
Keywords — Artificial Intelligence, Large Language Model, Ollama, GPT, troubleshooting, laptop repairs, data extraction, service automation, machine learning

I. INTRODUCTION

Usage of AI has increased lately which has in turn increased the productivity of many people in STEM. This has helped people gain tons of knowledge and perform well. Artificial intelligence has become an integral part of every individual's life as it helps in giving best, proper solutions to every input problem.

Laptop has been one of the best modes to achieve so. If these laptops would face problems in working then definitely solving it should be our utmost priority. For the same our AI has been developed which gives proper solutions to any prompt related to laptops faults. This helps user get a solution in ease by simply inputting the problem. This AI system uses LLM (large language model) and prompt template as well. Whenever a user prompts a symptom then the same goes on to the prompt template which makes the inputting user prompt better thus increasing its effectiveness. It further goes on to LLM and creates an array of all possible problems that the input has. Once a problem is selected then the AI would search for the same in document search and would eventually go to the LLM and finally give a proper solution.

In this context, proper working of a laptop is crucial. Even if it faces any issue, addressing and executing the same is of equal importance. Our AI system helps in achieving it all with good performance, reduced errors, improved UI for a user-friendly approach and great ease while using the AI system.

Moreover, fixing a laptop is a time-consuming process and thus getting a solution at the very moment is of great ease and our AI helps in achieving so.

II. LITERATURE REVIEW

i. "An Empirical Study of the Naïve Bayes Classifier"

a. The study examines the naive Bayes classifier using Monte Carlo simulations. It finds that the classifier works best when features are independent or functionally dependent.
b. Despite its unrealistic assumptions, it remains effective in practice. Further research is needed to understand its performance with different data characteristics

ii. "An Expert System for Laptop Fault Diagnostic Assistance"

a. The paper introduces a laptop fault diagnosing system where users can generate reports by answering questions.
b. It aims to replace technicians to save costs and time, focusing on accessibility and expanding the knowledge base.
c. Based on user responses, the system generates a diagnostic report identifying potential root problems and recommending solutions.

iii. "The great transformer: Examining the role of large language models in the political economy of AI"

a. This paper delves into the impact of large language models, particularly the Transformer architecture, in AI.
b. It discusses their text generation capabilities and associated concerns like automated content creation and potential biases.
c. It suggests that more financial support should be directed towards independent AI research to promote the development of alternative techniques and systems.
d. The paper highlights the impact of the transformer architecture on the NLP landscape, emphasizing the importance of scale and computing power.

iv. "Research on online fault detection tool of substation equipment based on artificial intelligence"

a. This paper incorporates the framework based on artificial neural network methodology for fault identification and analysis.
b. Enhances the accuracy of fault identification and make the workstation fault free.

v. "Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications"

a. The article delves into the utility and risks of employing large language models (LLMs) in software engineering tasks, acknowledging concerns over semantic accuracy and environmental impact.
b. Emphasizing the importance of ethical considerations, it advocates for educating future software engineers on trust and improvement strategies for AI-driven assistants.

vi. "Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications"

a. This research paper focuses more on exploration and implementation of Turbo Prolog programming language along with a few exceedingly exceptional features of Object-Oriented based programming approach.
b. It also proposes a decent way of having a huge database management system. The rules proposed in this paper are in an IF-ELSE format.

III. Methodology/

Our Artificial Intelligence (AI) Assisted Computer Troubleshooting Tool leverages state-of-the-art AI technologies to provide an automated and efficient solution to diagnose and repair laptop issues. The tool's workflow combines a large language model (LLM), natural language processing (NLP) and data retrieval techniques to provide simple and user-friendly troubleshooting.
We first started out building a very user-friendly interface using Streamlit (it is an open-source Python framework for data scientists and AI/ML engineers to deliver interactive data apps – in only a few lines of code). Then the troubleshooting process began with taking in user input. The user is prompted to provide: the model of their laptop and a description of the problem symptoms. This information is crucial as it allows the system to provide the troubleshooting process for the specific hardware and software configurations of the laptop model in question.
For example, an Asus G15 2022 facing a dead screen may enter "screen not working" as a symptom.

Once the user input is collected, the data is formatted into a structure that can be efficiently processed by a large language model (LLM). To achieve this, we utilize the Langchain Framework of Python, which is designed for creating prompt templates that facilitate the interaction between structured data and language models. The Langchain prompt templates are used to convert the user's laptop model and problem symptoms into a query that can be understood and analyzed by the LLM.
The formatted query is then fed into a large language model, for that we tried using OpenAI's GPT-4 and Ollama. The model by OpenAI was faster than Ollama by a small margin. Still, we opted for using Ollama for our purpose as it was an open-source model and is updated almost every 4-5 days on the . The LLM analyzes the input to generate a list of potential problems based on the provided symptoms. The model's extensive training on diverse datasets enables it to infer possible issues that might not be immediately apparent through traditional troubleshooting methods.

The output from the LLM is an array of potential problems, each represented as a clickable button within the tool's user interface. The user is presented with these options and asked to select the problem they believe is most likely to be the cause of the issue. This interactive element ensures that the user remains engaged in the troubleshooting process and provides a level of control over the diagnosis. This also accounts in the fact that an average person may not be able to give a perfect description of the symptom and providing the user with potential problems may offer some insights to push the user in the right direction

After the user selects the most likely problem, the tool proceeds with another prompt template from the Langchain Framework which is employed to convert the selected problem into a prompt for llm to give us a step-by-step guide. The LLM processes this prompt to generate a detailed, easy-to-follow solution that guides the user through each step necessary to resolve the issue.

The final step involves presenting the step-by-step solution to the user through the tool's interface. The instructions are designed to be clear and comprehensive, ensuring that users of varying technical expertise can follow them effectively.
Our AI-assisted computer troubleshooting tool represents a significant advancement in automated technical support. By integrating user input, LLM analysis, and step-by-step solution generation, the tool offers a robust and user-friendly approach to laptop troubleshooting. This methodology not only reduces downtime and operational costs but also empowers users with the knowledge and tools needed to resolve technical issues efficiently.

APPROACH:
In the initial phase of development, we employed the Naive Bayes classifier, a straightforward probabilistic model grounded in Bayes' theorem. This model operates on the principle of conditional independence, assuming that the presence of a particular feature in a class is unrelated to the presence of any other feature. Despite its simplicity and computational efficiency, the Naive Bayes classifier yielded an accuracy of only 0.125. This subpar performance is attributed to the small size of our dataset, which undermined the model's ability to accurately capture the underlying patterns and dependencies in the data.
Recognizing the limitations of the Naive Bayes approach, we next explored Decision Trees and Random Forests. Decision Trees segment the dataset into smaller subsets based on the most significant predictor variables, creating a tree-like model of decisions. Random Forests, an ensemble learning method, combine multiple decision trees to enhance predictive accuracy and mitigate overfitting. Despite their robustness and interpretability, these models achieved an accuracy of only 0.5517241379310345. While an improvement over the Naive Bayes classifier, this accuracy level remained insufficient for practical application, primarily due to the limited dataset which restricted the depth and diversity of the decision trees.

Given the inadequacies of traditional machine learning models, we turned to Large Language Models (LLMs) which use transformers for their advanced capabilities in handling complex, context-rich data.
LLMs, such as OpenAI's GPT-3, are pre-trained on extensive datasets and employ sophisticated architectures like transformers. These models excel in understanding and generating human-like text, making them well-suited for troubleshooting tasks that require nuanced comprehension and contextual analysis. Unlike traditional models, LLMs can leverage transfer learning, where the pre-trained model is fine-tuned with a smaller dataset, thereby mitigating the limitations imposed by data scarcity.

We initially integrated OpenAI's LLM into our system, recognizing its superior performance and reliability. The model demonstrated a significant improvement in diagnostic accuracy and response quality. However, the requirement for constant internet connectivity to access the model, coupled with the high operational costs, posed significant challenges. These constraints limited its practicality for widespread deployment, especially in environments with restricted internet access or budgetary limitations.

To address the limitations of O
Ollama's LLM, which operates offline. This offline capability was a critical factor in the decision of choice, as it ensured uninterrupted functionality regardless of internet availability and reduced ongoing operational costs.
The initial implementation of Ollama's LLM presented performance issues, notably slower response times, which could detract from the user experience. To mitigate this, we undertook extensive optimization efforts. These included:

- Model Pruning: Reducing the size of the model by removing less critical parameters and layers, thereby improving computational efficiency without significantly sacrificing accuracy.
- Quantization: Converting the model weights to lower precision formats, which decreased the memory footprint and accelerated inference times.
- Parallel Processing: Using multi-threading and parallel processing techniques to increase the model's response speed.

Through these optimization strategies, we successfully reduced the latency and improved the runtime efficiency of Ollama's LLM, making it a viable solution for our AI-assisted computer troubleshooting program.

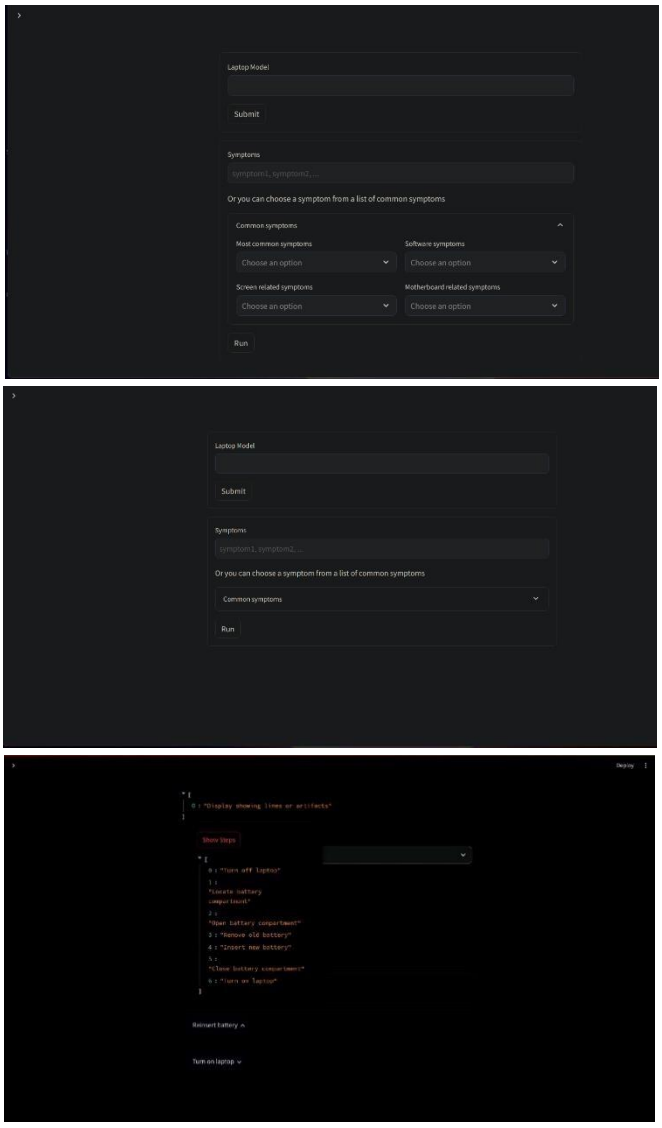

Fig. Front end of the program

ISSN NO: 2249-3034

CORE CODE:

```
import streamlit as st
from Chain import prompt_template_for_diagnosis,
prompt_template_for_solution,
prompt_template_for_solution2
import asyncio
from langchain_community.llms import Ollama
import re
from langchain.memory import
ConversationSummaryBufferMemory

async def chain(symptom, prompt):
    llm = Ollama(model="llama3")
    diagnosis_chain = prompt | llm
    response1 = diagnosis_chain.invoke(symptom)
    pattern = r'\[([^]]+)\]'
    extracted_list = re.findall(pattern, response1)
    extracted_list = [item.strip('\"\' ') for item in
extracted_list[0].split(',')]
    return extracted_list

async def main():
    if 'steps' not in st.session_state:
        st.session_state.steps = []
    if 'selected_problem' not in st.session_state:
        st.session_state.selected_problem = None
    st.write(st.session_state.symptom_list)

    symptom_query_dict = {
        "laptop_model": st.session_state.laptop_model,
        "symptom": st.session_state.symptom_list
    }
    with st.form(key="problem_selection_form"):
        st.session_state.selected_problem = st.selectbox(
            "Please select a problem you wish to see the solution
to",
            await chain(symptom_query_dict,
prompt_template_for_diagnosis.messages[0].prompt),
            placeholder="Select problem...",
            index=None
        )
        if st.form_submit_button("Run"):
            query_dict = {
                "laptop_model": st.session_state.laptop_model,
                "problem": st.session_state.selected_problem
            }
            st.session_state.steps = await
chain(symptom=query_dict,
prompt=prompt_template_for_solution.messages[0].prompt)
    for step in st.session_state.steps:
        with st.form(key=step):
            with st.popover(step):
                if st.form_submit_button("Show Steps"):
                    query_dict2 = {
                        "laptop_model":
st.session_state.laptop_model,
                        "instruction": step,
                        "problem": st.session_state.selected_problem
                    }
                    st.write(await chain(symptom=query_dict2,
prompt=prompt_template_for_solution2.messages[0].promp
t))

if __name__== '__main__':
    asyncio.run(main())
```

IV. Results and

The objective to develop an AI-assisted computer troubleshooting program is a massive one but today we are capable of diagnosing and solving technical issues using this approach. The research and development phase highlighted the limitations of traditional machine learning models when faced with small datasets and the superior capabilities of LLMs in such contexts.

The transition from OpenAI's LLM to Ollama's offline model, followed by targeted optimization efforts, enabled us to develop a robust, efficient AI-assisted troubleshooting tool. This phase underscored the importance of model selection and optimization in the development of practical AI applications.The end product also demonstrated just how much of a help and time save it can be while also highlighting the problems which may arising when using Artificial Intelligence such a system.

As LLMs continue to evolve, they will become even more adept at understanding nuanced technical issues, providing more accurate and contextually relevant solutions. Future advancements could see the integration of multimodal capabilities, allowing the AI to interpret and diagnose issues based on a combination of text, images, and perhaps even video inputs. This would enable users to simply upload screenshots or short videos of the problem, with the AI providing detailed, step-by-step resolutions.

Moreover, the system could be improved with real-time learning that will be able to adapt to new software updates and emerging technologies ensuring that the troubleshooting remains current and up-to-date. Additionally, the natural language processing (NLP) could be enhanced for more intuitive and human-like interactions, making the troubleshooting process seamless and less frustrating for the user. By employing advanced data analytics, the program could identify recurring issues and trends, providing valuable insights for better product development, ultimately reducing downtime and pitfalls which could occur during development.

In enterprise settings, integration with IT service management (ITSM) tools could streamline support workflows, automatically creating and updating tickets based on the AI's analysis and recommendations. Lastly, the program could be expanded to support multiple languages, making it accessible to a global audience, hence removing a large barrier for universal tech education.

V. Acknowledgment

We, the authors, would like to thank Vishwakarma Institute of Technology (VIT), Pune for the gracious opportunity and our project mentor, Mrs. Amruta A. Bhawarthi for her useful insights and guidance throughout our project.

vii. References

[1] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[2] Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

[3] Vaswani, A., et al. (2017). "Attention is All You Need". Advances in Neural Information Processing Systems, 30.

[4] Brown, T., et al. (2020). "Language Models are Few-Shot Learners". arXiv preprint arXiv:2005.14165