International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Innovative Log Analysis with Advanced Language Models: Unearthing Unparalleled Insights

Nagavenkatabalaji Thota [1], Dr. Harikrishna Kamatham [2], Dr. Thayyaba Khatoon Mohammad [3]

[1]*Student, M.Tech in ML&DL - Department of AIML, nagavenkatabalajithota@gmail.com*
[2]*Associate Dean – School of Engineering, drhk@mallareddyuniversity.ac.in*
[3]*Head of Departmet – AIML, hodaiml@mallareddyuniversity.ac.in, thayyaba.khatoon16@gmail.com*
School of Engineering, Malla Reddy University, Maisammaguda, Dhulapally, Hyderabad, 500100, INDIA

A B S T R A C T

This project will help the User/Developer that will help the user fix errors and bugs. The project uses Question-and-Answer text-generation LLM embedded with the millions of logs that the enterprise system generates across the company.

Target users: Developers, QA and customer support teams.

The solution taps into the extensive logs, providing a unique edge in troubleshooting customer site issues. The users may face many errors and bugs coming from the systems. It can be a random error that broke the local Project development Environment or may be a customer bug issues. Hunting for bug issues online or reaching out for help consumes significant productive hours, impacting workflow and task completion.

A single-stop knowledge base for issues and log tickets saves approximately 30-35% of productive time. With the helper chat, we can ask a question regarding an error or issue you are facing. It will check and analyse millions of records from embedded log data and provide a response.

Hours of head scratching will be saved. That hours can be used in the productive way to develop the projects. Hence millions of dollars will be saved to the Enterprise.

LLM models: codegen-350M-mono if hardware resources are available, we can also use llama2 or falcon7b

Keywords: Log Analysis, Large Language Models (LLM), NLP

1. Introduction

In today's complex software landscape, developers, quality assurance (QA) teams, and customer support personnel encounter a multitude of errors and bugs. These issues can arise during local project development or manifest as customer-reported problems. The traditional approach to resolving these challenges involves manual searches, online forums, and reaching out for help, which consumes valuable productive hours.

To address this inefficiency, we propose a novel solution that leverages Question-and-Answer text-generation language models (LLMs) embedded with extensive logs generated by enterprise systems. Our system provides a single-stop knowledge base, enabling users to query specific errors or issues. The LLM processes these queries, analyzes millions of log records, and delivers targeted responses, significantly reducing troubleshooting time.

In this paper, we delve into the methodology, benefits, and LLM models recommended for implementing this solution. By enhancing troubleshooting efficiency, organizations can save time, improve workflow, and ultimately reduce costs.

1.1 Problem Statement

Software systems often encounter errors and bugs, whether they arise during local project development or manifest as customer-reported issues. The traditional process of hunting for solutions online or reaching out for help consumes valuable productive hours, impacting workflow efficiency. This paper will help the User/Developer that will help the user fix errors and bugs.

1.2 Proposed Project Research Plan

Our proposed solution uses question-and-answer text-generation which establishes a single-stop knowledge base that integrates issue tracking and log analysis. By querying the embedded log data using the LLM, users can ask questions related to specific errors or issues they face. The LLM processes millions of records and provides targeted responses, significantly reducing the time spent on troubleshooting.

1.3 How it Works:

1. Ask a question about an error or issue you are facing.
2. Project will then analyze millions of records from its embedded log data and provide you with a comprehensive response.

1.4 Current Scope:

Project is currently implemented on smaller LLM models, but we are exploring the use of larger models, such as Llama2 and Falcon7B, for improved accuracy.

1.5 Future Scope:

Additionally, Project can be integrated into existing platforms, such as Slack, or used as a standalone chat bot.

2. Literature Review

An overview of previous work on log analysis via NLP and LLM-based approaches.

2.1 Traditional NLP Approaches

Nguyen et al. [1] and colleagues present an innovative supervised ensemble learning approach that amalgamates results from different machine learning models to enhance the classification of abnormal security events. This method, tested on the ECML/PKDD 2007 and HTTP CSIC 2010 datasets, leverages selected features and syntactic properties of log lines. The authors highlight the necessity of evolving intrusion detection systems (IDSs) to operate effectively in diverse and hostile network environments. Their IDS shows a 10% improvement in accuracy over traditional supervised machine learning techniques, made possible by an online learning framework. The system utilizes various algorithms, such as decision stump, Naive Bayes, RBF Network, and Bayesian Network. A key innovation is the creation of a new pipeline that integrates IDS outputs to improve both accuracy and reliability. The IDS achieves accuracies of 0.9098 on the CSIC dataset and 0.9256 on the PKDD/ECML dataset, though this comes at the cost of increased computational requirements for model output integration.

Vartouni et al. [6] investigate innovative techniques for feature extraction in intrusion detection systems, employing deep belief networks and parallel feature fusion approaches. They implement synthetic feature extraction and fusion on bi-gram representations of log files, coupled with dimensionality reduction methods such as FICA, PCA, and KPCA. Their study evaluates the impact of dimensionality reduction, synthetic feature fusion, and baseline n-gram representations in training unsupervised anomaly detection algorithms like Isolation Forest, Elliptic Envelope, and one-class SVM. The effectiveness of these methods is tested using the ECML/PKDD 2007 and HTTP CSIC datasets. Their findings indicate that deep learning neural networks, particularly fusion models, significantly enhance feature extraction for anomaly detection. Specifically, the feature fusion approach achieves an accuracy of 0.8535 for Isolation Forest on the CSIC dataset and 0.8493 for Elliptic Envelope on the PKDD dataset, following extensive exploration and parameter optimization.

2.2 LLM-Based Approaches

In their recent study, Nam et al. [12] investigate the application of BERT for analyzing logs to predict failures in Virtual Machines (VMs). They propose a model where BERT generates sentence embeddings from each log entry, which are subsequently processed by a pre-trained convolutional neural network to forecast failures within a 2 to 30-minute timeframe. This model achieves a 0.74 accuracy rate on a proprietary OpenStack VM system log dataset used for testing and training. A key advantage of their BERT-CNN model is its ability to predict VM failures even for instances not encountered during the training phase.

Wang et al. [13] develop an intrusion detection system that uses word2vec and TF-IDF for feature extraction, combined with supervised learning methods for classification. This system is tested on supercomputer system log datasets, demonstrating that the combination of TF-IDF/word2vec with LSTM yields an accuracy of 0.99.

Qi et al. [14] introduce a deep learning-based anomaly detection system named Adanomaly, which detects anomalies by combining log key extraction using the Drain log parser ([1]) with an adversarial autoencoder for reconstructive error-based anomaly detection. Their model, evaluated on three different system log files, shows an accuracy range between 0.92 and 0.98. However, the reliance on the Drain log parser for log key representation may limit the generalizability of this approach.

2.3 Summary

The existing literature highlights several shortcomings in NLP-based log analysis:

- Most studies focus either on web application logs or system logs, yet real-world systems often operate under both conditions and should therefore be evaluated on both types of data.
- Research tends to utilize a narrow range of LLM architectures, predominantly BERT/RoBERTa models. However, state-of-the-art transformer language models offer various alternatives that possess distinct features or enhancements which have not yet been thoroughly examined in log analysis.
- Performance evaluations are typically limited to modified versions of LLMs, making it difficult to distinguish improvements due to optimizations from those inherent to the original LLMs.

This paper aims to investigate the adaptability of LLM-based models for log analysis by employing multiple language model architectures. We will assess both the original and fine-tuned versions of these models to understand the impacts and requirements of fine-tuning. Our study aims to establish a benchmark for LLM-based feature extraction in log analysis for intrusion detection.

We utilize CodeGen-Mono 350M, an autoregressive language model family designed for program synthesis, as described in "A Conversational Paradigm for Program Synthesis" by Erik Nijkamp, Bo Pang, and colleagues. The checkpoint in this repository, named CodeGen-Mono 350M, is initialized with CodeGen-Multi 350M and further pre-trained on a Python programming dataset. The "350M" indicates the number of trainable parameters.

3. System Design

This paper explores the process of building an LLM (Language Model-based Learning) application using document loaders, embeddings, vector stores, and prompt templates. LLMs have become increasingly popular in natural language processing tasks due to their ability to generate coherent and contextually relevant text. We discusses the importance of LLMs, compares fine-tuning and context injection approaches, introduces LangChain, and provides a step-by-step process for building the LLM app.

Language is the primary medium through which humans communicate and express their thoughts and ideas. Understanding and processing human language has always been a fundamental challenge in the field of artificial intelligence. With the advancements in natural language processing (NLP), the development of sophisticated language models has paved the way for significant breakthroughs in various NLP tasks.

Language Model-based Learning (LLM) has emerged as a powerful approach to tackling these challenges. LLMs leverage deep learning techniques to model and understand the complex patterns and structures of human language. These models have demonstrated remarkable capabilities in generating coherent and contextually relevant text, enabling them to excel in tasks such as text generation, summarization, translation, and question-answering systems.

3.1 LLMs in Natural Language Processing Tasks

The integration of LLMs in natural language processing tasks has revolutionized the way we interact with text data. These models can learn from vast amounts of textual information and capture intricate relationships between words, phrases, and concepts. By leveraging this knowledge, LLMs can generate human-like text that aligns with the given context.

One of the key advantages of LLMs is their ability to generate coherent and contextually relevant text. Unlike traditional rule-based or statistical approaches, LLMs have the capability to generate language that follows grammatical rules, preserves context, and demonstrates a deep understanding of semantic relationships. This enables applications such as text summarization, where LLMs can generate concise and informative summaries by extracting key information from a given document.

Additionally, LLMs have been employed in machine translation systems, where they learn to map input text from one language to another, producing high-quality translations. These models have shown impressive performance, outperforming previous machine translation approaches and bridging the gap between languages.

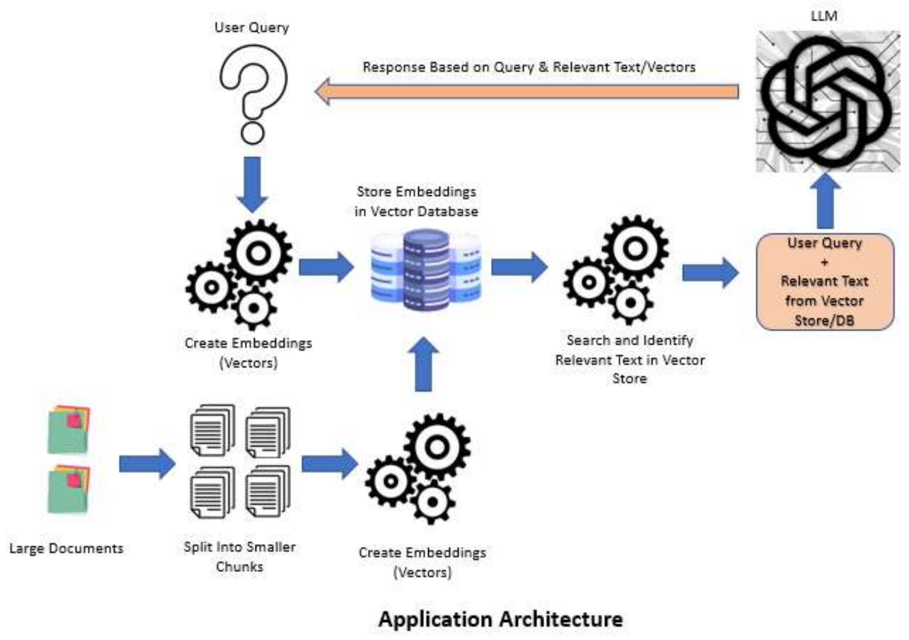
User Query
Response Based on Query & Relevant Text/Vectors
LLM
Store Embeddings in Vector Database
Create Embeddings (Vectors)
Search and Identify Relevant Text in Vector Store
User Query + Relevant Text from Vector Store/DB
Large Documents
Split Into Smaller Chunks
Create Embeddings (Vectors)
Application Architecture

Fig. 1 – Application Architecture

3.2 Coherent and Contextually Relevant Text Generation

The ability of LLMs to generate coherent and contextually relevant text is a result of their training on vast amounts of diverse textual data. These models capture patterns, dependencies, and contextual cues from this data, allowing them to generate text that aligns with the input context.

For example, in text completion tasks, LLMs can generate the most likely continuation of a given sentence, ensuring that the generated text is coherent and relevant to the preceding context. This has practical applications in auto-completion features, where LLMs can predict the next word or phrase as users type, providing real-time suggestions.

Moreover, LLMs have been employed in chatbot systems, enabling conversational agents to generate human-like responses. These models learn from dialogue datasets and generate contextually relevant responses, considering the conversational history to maintain coherence and relevance throughout the conversation.

LLMs have become invaluable in natural language processing tasks, offering the capability to generate coherent and contextually relevant text. The advancements in deep learning techniques, coupled with large-scale training data, have paved the way for LLMs to excel in tasks such as text generation, summarization, translation, and dialogue systems. Harnessing the power of LLMs opens up new possibilities for automating language-related tasks and creating more interactive and intelligent applications.

3.3 Fine Tuning

Fine-tuning is a popular approach in LLM development that involves adapting a pre-trained language model to perform specific tasks. Fine-tuning begins with leveraging a pre-trained LLM, which has been trained on a large corpus of general language data. The pre-training phase enables the model to learn rich linguistic representations and capture the statistical patterns of natural language.

To fine-tune an LLM for a specific task, we start with the pre-trained model and further train it on a task-specific dataset. This dataset consists of labeled examples that are relevant to the target task. During the fine-tuning process, the model's parameters are adjusted to optimize its performance on the specific task.

3.4 Context Injection in LLMs

Context injection, also known as prompt engineering, is an alternative approach to utilizing pre-trained LLMs without extensive fine-tuning. Instead of fine-tuning the entire model, context injection involves injecting specific context or prompts into the pre-trained LLM to guide its output generation for a specific task.

Prompt engineering offers flexibility and faster iteration cycles compared to fine-tuning. Developers can design prompts that incorporate desired input-output behavior and encode task-specific instructions. By carefully engineering prompts, it is possible to generate task-specific outputs from a pre-trained LLM without extensive retraining.

3.5 LangChain: Architecture and Components

LangChain is a powerful framework that provides a modular and efficient architecture for building LLM applications. It offers a streamlined workflow for document loading, text chunking, embedding generation, LLM selection, prompt template creation, and vector store creation. Let's explore the key components and their functionalities:

Document Loader:

The document loader component handles the loading of documents into the LangChain framework. It supports various document formats such as plain text, PDF, HTML, and more. The document loader ensures efficient and reliable ingestion of documents, enabling seamless integration with the rest of the pipeline.

Text Chunker:

The text chunker component splits the loaded documents into smaller text chunks. This step is particularly useful when dealing with large documents or when processing documents in a distributed manner. Text chunking enables parallel processing and improves the efficiency of subsequent steps, such as embedding generation and LLM inference.

Embedding Generator:

The embedding generator component takes the text chunks and generates embeddings for each chunk. Embeddings capture the semantic information of the text and represent it in a numerical vector form. LangChain leverages state-of-the-art language models and embedding techniques to generate high-quality embeddings that encode the contextual meaning of the text chunks.

LLM Selector:

The LLM selector component allows developers to choose the specific LLM model they want to use for their task. LangChain supports a wide range of pre-trained LLM models, such as GPT, BERT, and Transformer models. Developers can select the most suitable LLM based on their specific requirements, such as language generation, question-answering, or sentiment analysis.

Prompt Template Creator:

The prompt template creator component facilitates the creation of prompt templates for context injection. Prompt templates define the structure and instructions provided to the LLM for generating desired outputs. Developers can design templates that guide the LLM's behavior and tailor it to the task at hand. Prompt templates can include placeholders for dynamic inputs, allowing for flexible and customizable text generation.

Vector Store Builder:

The vector store builder component creates an efficient vector store for storing the generated embeddings. A vector store is a data structure that organizes and indexes the embeddings, allowing for fast and efficient retrieval. LangChain provides methods for building vector stores that enable efficient similarity searches, clustering, and other operations on the embeddings.

4. Implementation

4.1 Design Analysis

Any Enterprise application will have enormous logs that tells about the system state, errors etc. The solution taps into the extensive logs, providing a unique edge in troubleshooting customer site issues. The users may face many errors and bugs coming from the systems. It can be a random error that broke the local Project development Environment or may be a customer bug issues. Hunting for bug issues online or reaching out for help consumes significant productive hours, impacting workflow and task completion.

A single-stop knowledge base for issues and log tickets saves approximately 30-35% of productive time. With the helper chat, we can ask a question regarding an error or issue you are facing. It will check and analyse millions of records from embedded log data and provide a response.

Hours of head scratching will be saved. That hours can be used in the productive way to develop the projects. Hence millions of dollars will be saved to the Enterprise.

4.2 Log Files

There will be serval types of logfiles associated to any enterprise software. Below are some of the Logfiles.

1. Database Logs
2. Application logs
3. Event Logs

4. Cassendra Logs

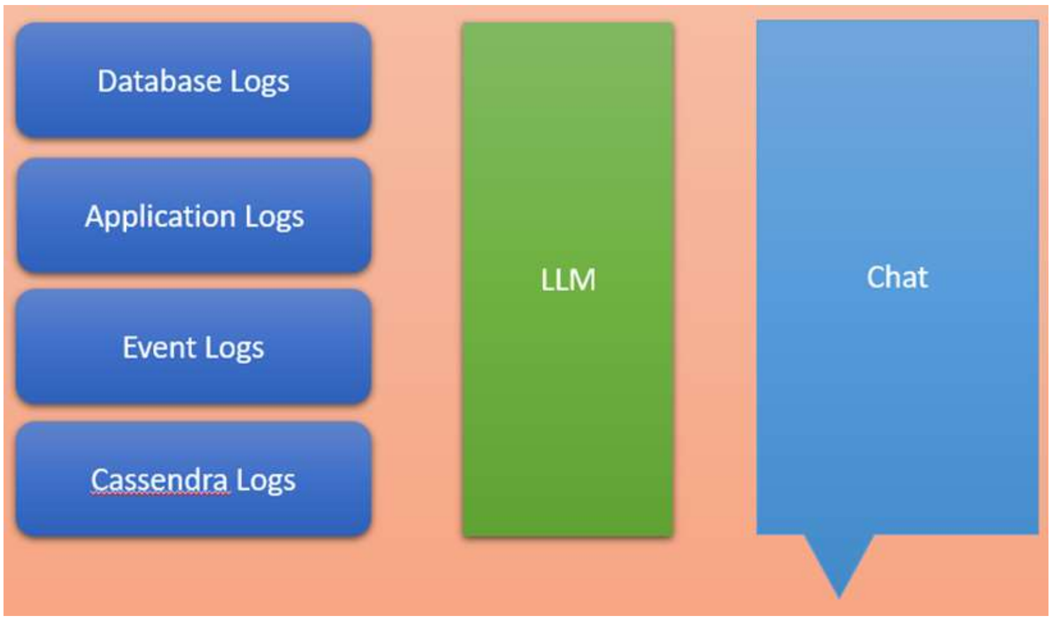

Fig. 2 – Project Components

Fig. 2 shows the various components involved in this project.

4.3 Benefits of this solution

The benefits of this project are multifaceted, encompassing advancements in technology, methodology, and practical application refer Fig. 3:

Reduces L1 Support Workload:

- By automating the analysis of logs using advanced language models (LLM), the project can significantly reduce the workload on Level 1 (L1) support teams.
- LLM can efficiently process and interpret large volumes of log data, enabling quicker identification and resolution of common issues without human intervention.
- This reduction in L1 support workload allows support staff to focus on more complex and high-priority tasks, ultimately improving overall efficiency and customer satisfaction.

Identifies Root Causes:

- Leveraging LLM for log analysis enhances the capability to accurately pinpoint the root causes of system errors and failures.
- Advanced language models can recognize patterns, anomalies, and correlations within log data that might go unnoticed by manual inspection.
- By identifying root causes with greater precision, the project enables proactive problem-solving, preventing recurring issues and minimizing downtime.

Saves Productivity Time:

- The project saves significant productivity time by automating the process of log analysis, which otherwise would require manual effort.
- With LLM, the time spent by IT personnel on manually reviewing and troubleshooting logs is drastically reduced.
- This saved time can be reallocated to more strategic tasks such as system optimization, innovation, and enhancing overall IT infrastructure.

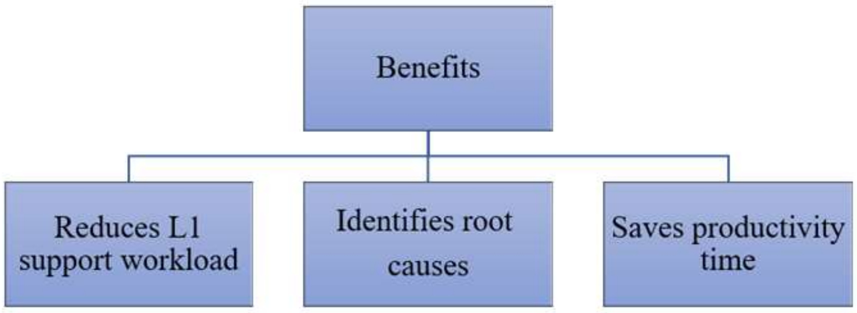

Fig. 3 – Benefits

5. Implementation

The process of implementing this project involves several key steps, each critical to achieving the desired outcomes. These steps are outlined below:

1. Collect all the log files from different sources.
2. Load multiple and process documents.
3. Build Embeddings
4. Create Db
5. Make a retriever.
6. Model preparation
7. Chat

5.1 Collect all the log files from different sources

To begin the implementation, the first step involves gathering log files from various sources such as servers, applications, databases, and network devices. These log files contain valuable information about system events, errors, and user activities. Depending on the environment, this process may involve configuring log collection agents, setting up centralized logging systems, or manually retrieving log files from different locations. Ensuring comprehensive coverage and proper organization of log files is crucial for accurate analysis and effective troubleshooting.

5.2 Load multiple and process documents

Once the log files are collected, the next step is to load them into the system for processing. This involves parsing and extracting relevant information from each log file, such as timestamps, log levels, error messages, and associated metadata. Additionally, preprocessing techniques may be applied to clean and normalize the log data, removing noise and irrelevant information. Loading and processing multiple documents efficiently is essential for handling large volumes of log data and ensuring scalability and performance.

5.3 Build Embeddings

After preprocessing the log data, the next step is to build embeddings or numerical representations of the log messages. Embeddings capture the semantic meaning and context of log messages, enabling advanced analysis techniques such as similarity matching and clustering. This step typically involves training word embeddings using techniques like Word2Vec, GloVe, or BERT on the log data corpus. By generating embeddings, the system can understand the relationships between different log messages and identify common patterns and anomalies.

5.4 Create Database

With the processed log data and embeddings ready, the next step is to create a database for storing and querying the log information efficiently. Depending on the requirements and scale of the project, this database may be relational (e.g., MySQL, PostgreSQL) or NoSQL (e.g., MongoDB, Elasticsearch). The database schema should be designed to accommodate the structured and unstructured nature of log data, allowing for fast retrieval and analysis. Indexing strategies and optimization techniques may be employed to improve query performance and reduce latency in accessing log information.

5.5 Make a Retriever

Following the database creation, a retriever component is developed to enable efficient retrieval of log data based on user queries or search criteria. This retriever utilizes indexing and search algorithms to quickly locate relevant log messages from the database, taking into account factors such as time range, severity level, keywords, and contextual information. The retriever component plays a crucial role in enabling real-time log analysis, troubleshooting, and incident response by providing fast access to historical log data.

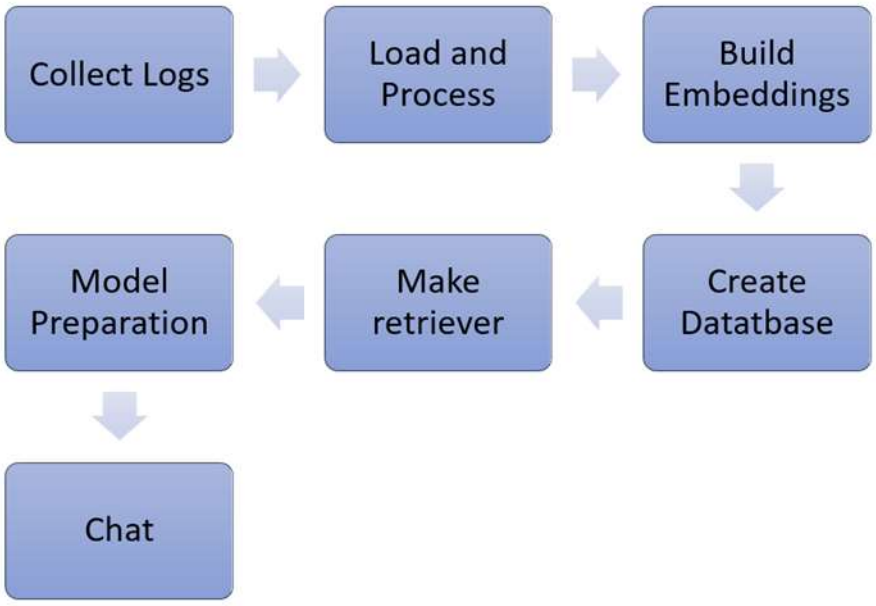

Fig. 3 – Implementation Steps

5.6 Model Preparation

Before performing advanced analysis tasks such as anomaly detection or predictive modeling, the log analysis system requires model preparation. This involves selecting appropriate machine learning or statistical models, training them on historical log data, and fine-tuning model parameters for optimal performance. Additionally, feature engineering techniques may be applied to extract relevant features from log messages and enhance the predictive capabilities of the models. Model preparation ensures that the log analysis system is equipped with accurate and effective predictive models for detecting and responding to anomalies and issues.

5.7 Chat

Finally, the log analysis system may incorporate a chat interface or conversational agent to provide a user-friendly interface for interacting with the system. This chat interface allows users to query log data, perform analysis tasks, and receive insights and recommendations in a conversational manner. Natural language processing (NLP) techniques are employed to understand user queries, extract relevant information, and generate meaningful responses. The chat interface enhances the usability and accessibility of the log analysis system, enabling users to interact with log data in a more intuitive and efficient manner.

6. Conclusion

This paper presents a valuable tool for users, particularly developers, QA teams, and customer support staff, to effectively address and resolve errors and bugs. By leveraging a Question-and-Answer text-generation LLM embedded with the extensive logs generated across an enterprise system, this project provides a unique advantage in troubleshooting and resolving issues that arise in customer sites and local development environments.

The solution targets the common challenge of time-consuming bug tracking and error resolution, which often disrupts workflow and delays task completion. By integrating a comprehensive knowledge base of issues and log tickets, the system significantly reduces the time spent on troubleshooting, saving approximately 30-35% of productive hours. This efficiency gain translates to substantial cost savings for enterprises, as the time saved can be redirected towards more productive development activities.

With the help of an interactive chat feature, users can query the system about specific errors or issues they encounter. The system then analyzes millions of records from embedded log data to provide accurate and timely responses, effectively eliminating hours of unproductive effort spent on manual troubleshooting.

By employing advanced LLM models such as codegen-350M-mono, and potentially llama2 or falcon7b if hardware resources permit, the solution ensures robust performance and scalability. Overall, this project stands to enhance productivity, streamline error resolution processes, and deliver significant financial savings for enterprises.

References

[1] Egil Karlsen, Xiao Luo, Nur Zincir-Heywood, Malcolm Heywood.: "Benchmarking Large Language Models for Log Analysis, Security, and Interpretation" arXiv, 2023 https://doi.org/10.48550/arXiv.2311.14519

[2] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, Caiming Xiong.: "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis" *ICLR*, 2022 https://doi.org/10.48550/arXiv.2203.13474

[3] Boffa, M., Milan, G., Vassio, L., Drago, I., Mellia, M., Ben Houidi, Z.: Towards nlp-based processing of honeypot logs, 314–321 (2022) https://doi.org/10.1109/EuroSPW55150.2022.00038

[4] Karlsen, E., Copstein, R., Luo, X., Schwartzentruber, J., Niblett, B., Johnston, A., Heywood, M.I., Zincir-Heywood, N.: Exploring semantic vs. syntactic features for unsupervised learning on application log files. In: 2023 7th *Cyber Security in Networking Conference (CSNet*), pp. 1–7 (2023). https://doi.org/(nPress)

[5] Nguyen, H.T., Franke, K.: Adaptive intrusion detection system via online machine learning. In: 2012 12th *International Conference on Hybrid Intelligent Systems (HIS)*, pp. 271–277 (2012). https://doi.org/10.1109/HIS.2012.6421346

[6] Moradi Vartouni, A., Teshnehlab, M., Sedighian Kashi, S.: Leveraging deep neural networks for anomaly-based web application firewall. *IET Information Security* 13(4), 352–361 (2019)

[7] Bhatnagar, M., Rozinaj, G., Yadav, P.K.: Web intrusion classification system using machine learning approaches. In: 2022 *International Symposium ELMAR*, pp. 57–60 (2022). https://doi.org/10.1109/ELMAR55880.2022.9899790

[8] Farzad, A., Gulliver, T.A.: Log Message Anomaly Detection and Classification Using Auto-B/LSTM and Auto-GRU (2021)

[9] Tmer Sivri, T., Pervan Akman, N., Berkol, A., Peker, C.: Web intrusion detection using character level machine learning approaches with upsampled data, pp. 269–274 (2022). https://doi.org/10.15439/2022F147

[10] Adhikari, A., Bal, B.K.: Machine learning technique for intrusion detection in the field of the intrusion detection system. (2023)

[11] Copstein, R., Karlsen, E., Schwartzentruber, J., Zincir-Heywood, N., Heywood, M.: Exploring syntactical features for anomaly detection in application logs. it - Information Technology 64(1-2), 15 – 27 (2022) https://doi.org/10.1515/itit-2021-0064

[12] Nam, S., Yoo, J.-H., Hong, J.W.-K.: Vm failure prediction with log analysis using bert-cnn model, 331 – 337 (2022) https://doi.org/10.23919/CNSM55787. 2022.9965187

[13] Wang, M., Xu, L., Guo, L.: Anomaly detection of system logs based on natural language processing and deep learning. In: 2018 4th *International Conference on Frontiers of Signal Processing (ICFSP)*, pp. 140 – 144 (2018). https://doi.org/10.1109/ICFSP.2018.8552075

[14] Qi, J., Luan, Z., Huang, S., Wang, Y., Fung, C., Yang, H., Qian, D.: Adanomaly:Adaptive anomaly detection for system logs with adversarial learning. In: NOMS2022-2022 *IEEE/IFIP Network Operations and Management Symposium*, pp. 1 – 5. IEEE Press, ??? (2022). https://doi.org/10.1109/NOMS54207.2022.9789917

[15] Seyyar, Y.E., Yavuz, A.G., nver, H.M.: Detection of web attacks using the bert model. In: 2022 30th *Signal Processing and Communications Applications Conference (SIU)*, pp. 1 – 4 (2022). https://doi.org/10.1109/SIU55565.2022.9864721

[16] Guo, H., Yuan, S., Wu, X.: Logbert: Log anomaly detection via bert, 1 – 8 (2021).IEEE

[17] Shao, Y., Zhang, W., Liu, P., Huyue, R., Tang, R., Yin, Q., Li, Q.: Log anomaly detection method based on bert model optimization. In: 2022 7th *International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, pp. 161 – 166 (2022). https://doi.org/10.1109/ICCCBDA55098.2022.9778900

[18] Guo, H., Lin, X., Yang, J., Zhuang, Y., Bai, J., Zheng, T., Zhang, B., Li, Z.: TransLog: A Unified Transformer-based Framework for Log Anomaly Detection (2022)

[19] Le, V.-H., Zhang, H.: Log-based Anomaly Detection Without Log Parsing (2021)

[20] CodeGen (CodeGen-Mono 350M) [Salesforce/codegen-350M-mono · Hugging Face]

[21] Create a Chatbot with Gradio [www.gradio.app]