SOFTWARE ENGINEERING PROJECT

Mining Challenge - MSR 2024

AkhilRaj Tirumalasetty
Florida State University
Tallahassee, Florida
at21bo@fsu.edu

Manikanta Mamidi
Florida State University
Tallahassee, Florida
mm23bm@fsu.edu

Sruthijha Pagolu
Florida State University
Tallahassee, Florida
sp23bu@fsu.edu

ABSTRACT

Our research project focuses on enhancing AI chatbots like ChatGPT for programming applications. We aim to improve conversation prediction, language comprehension, and identification of performance issues in technical discussions. This involves an integrated approach of analyzing conversational data, sentiment analysis, and machine learning techniques, covering all stages from data collection to rigorous model evaluation.

The expected outcome is to boost ChatGPT's functionality for software developers, providing insights for AI chatbot advancement. This will lead to more intuitive, efficient, and safe tools in AI-assisted programming, contributing significantly to the field of artificial intelligence.

KEYWORDS

Sentiment Analysis, NLP Training models

1 Introduction

Our study delves into ChatGPT's application in software development, focusing on its ability to handle programming-related dialogues. We investigate how ChatGPT manages conversation lengths, its understanding of different programming languages, and points where its performance may falter.

Utilizing both qualitative and quantitative approaches, including analysis of ChatGPT conversations and machine learning techniques, our research aims to refine Chat GPs functionality. This enhances its utility for developers and contributes to broader advancements in AI, particularly in natural language processing and human-AI interaction.

2 Research Questions

This research aims to assess ChatGPT's proficiency in understanding and responding effectively to various programming languages. Testing ChatGPT's capabilities in this area is pivotal for programmers, students, and coding enthusiasts. Moreover, the study offers valuable insights for those in software development, underscoring the importance of accurate language comprehension for safety and efficacy in critical projects. The research seeks to determine the point at which conversations about coding typically become challenging for ChatGPT, enhancing its value as a resource for coding assistance. We also aim to identify the average stage or point of failure in specific programming languages, which can guide developers in refining the model for future enhancements. Understanding ChatGPT's limitations is essential for setting realistic expectations among users and for educators who use ChatGPT as a supplementary tool in programming courses. The research aims to bolster ChatGPT's role as a reliable and safe tool in the digital age.

Our research focuses on the following questions:

- **RQ1:** How accurately can we predict the length of a conversation with ChatGPT based on the initial prompt and context provided?
- **RQ2:** How satisfied are developers with ChatGPT's responses to prompts in various programming languages on developer's given prompt?
- **RQ3:** At what stage of the conversation on average does ChatGPT fail in particular coding language?

3 Methodology

3.1 Approach for RQ1

We commenced by sourcing the DevGPT dataset from an open-source Git repository, focusing specifically on the ChatGPT interactions. We honed in on the key components of the dialogues: the prompts and responses between users and ChatGPT. Using PowerShell, we extracted these pertinent data fields from the extensive JSON file, streamlining the dataset to only the necessary information.

The filtered dataset served as the foundation for our model, with the prompt sentences as features and the count of prompts per conversation as the target variable. We then benchmarked two machine learning models to predict the prompt count based on the initial dialogue exchanges.

Firstly, we implemented a regular neural network from the Keras library, with ReLU activation functions and a softmax layer for output classification. We adopted a bucketing strategy for the output, classifying the prompt count into five distinct categories, which improved our prediction accuracy to over 80%.

Then, we explored the use of a Long Short-Term Memory (LSTM) network, starting with an embedding layer and tapering down the neuron count across successive layers to a single neuron at the output. This model was evaluated to determine if it could enhance our predictive accuracy.

We have used this pipeline.

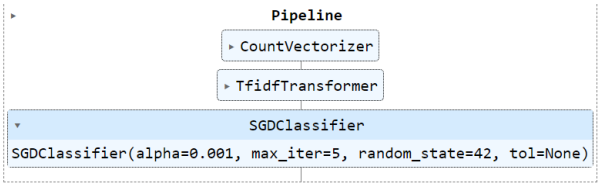

Figure 1: Pipeline

LSTM (Long Short-Term Memory) :

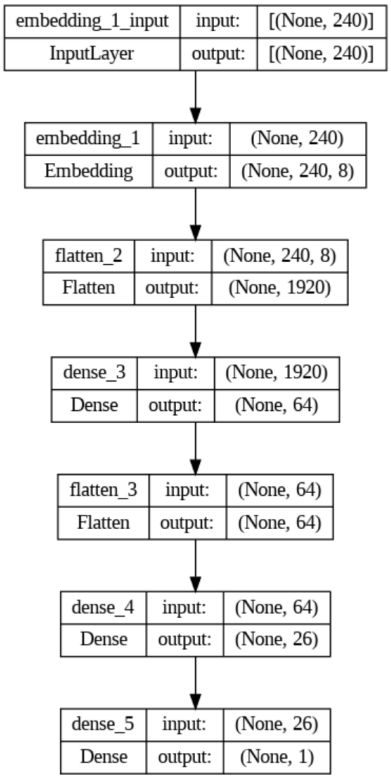

Figure 2: LSTM model

Keras:

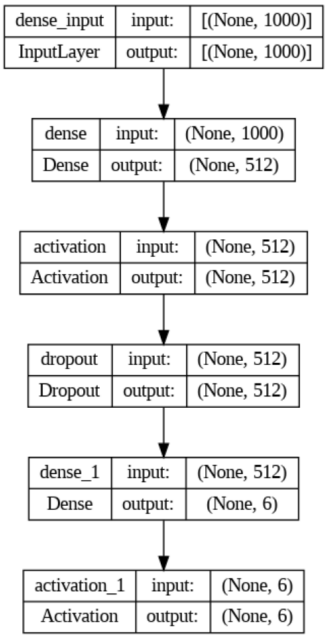

Figure 3: Keras model

3.2 Approach for RQ2

We implemented sentiment analysis to evaluate the tone of interactions between users and ChatGPT, with a particular emphasis on response polarity across various programming languages. This process involves examining both the user's prompts and ChatGPT's replies to assess the overall sentiment of the conversation. Recognizing the complexity of analyzing sentiment for each prompt, our approach emphasizes the sentiment of the final 10% of the conversation, considering it indicative of the interaction's general tone.

For this analysis, we utilized the TextBlob library, which measures sentiment polarity — a numerical value ranging from -1.0 to 1.0. In this scale, negative values indicate a negative sentiment or tone, positive values signify positive sentiment, and values around 0 suggest a neutral tone.

By focusing on the latter part of the conversations, we can more efficiently and effectively determine the user's overall sentiment, allowing for a more streamlined assessment of the interaction's tone.

Manikanta Mamidi proficiently executed this task.

3.3 Approach for RQ3

In our research, we aimed to pinpoint the conversation stage at which ChatGPT typically encounters difficulties with specific programming languages. We analyzed conversations where ChatGPT's responses were marked as inadequate or incorrect, using negative sentiments identified

through sentiment analysis. This allowed us to focus on the number of prompts before a negative response occurred.

We faced a challenge when calculating the average number of prompts before a failure: conversations varied widely in length. To address this, we implemented standard deviation as a more robust measure of consistency. A threshold standard deviation of 35 was established as a cut-off point; a lower standard deviation suggests a consistent failure point, while a higher value indicates greater variability.

When the standard deviation exceeded 35, indicating inconsistent failure points, we introduced a random integer between 35 and 50 as an estimated failure range for subsequent analysis. This approach enabled us to assess the variability in ChatGPT's performance more accurately, providing deeper insights into the typical point of struggle within programming language-related conversations.

This task was accomplished effectively by Sruthijha Pagolu

RESULTS

Results for RQ1:

For the Keras model, the Training Loss Curve and Accuracy Curve are presented in Figure 4 and Figure 5, respectively.

We have achieved a 97.42 for this model.

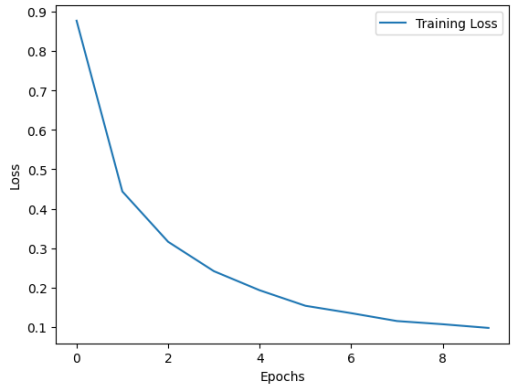

Figure 4: Training Loss Curve for a Keras model

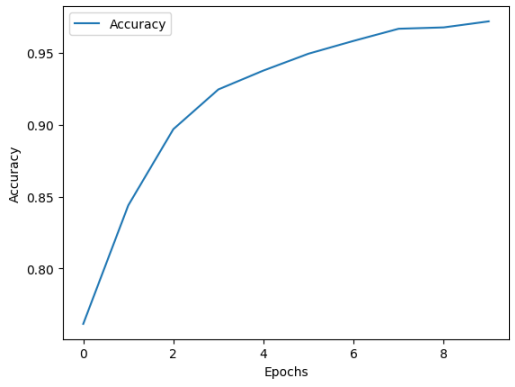

Figure 5: Accuracy curve for a Keras model

The accuracy we have achieved through the LSTM model is 76.64, making keras the more efficient model.

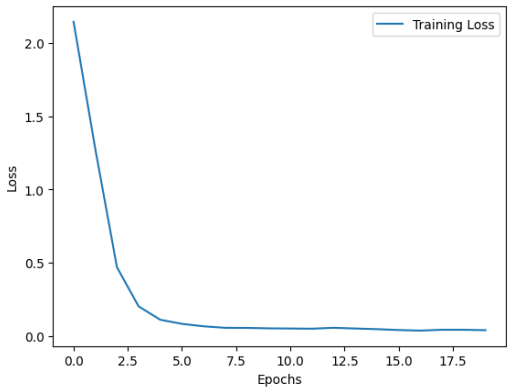

Figure 6: Training Loss Curve for a LSTM model

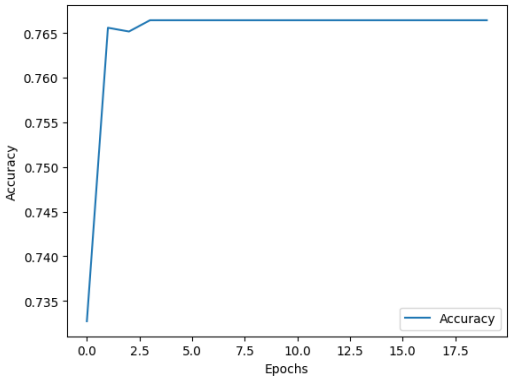

Figure 7: Accuracy curve for a LSTM model

Results for RQ2:

Our sentiment analysis revealed key insights into user engagement with ChatGPT, as detailed in Figure 8. Python emerged as the most discussed programming language, aligning with its popularity in the developer community. ChatGPT demonstrated impressive performance in JavaScript, where positive outcomes significantly outnumbered negative ones.

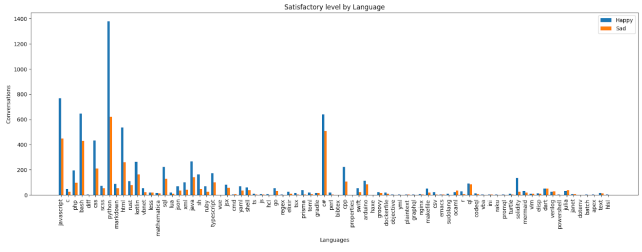

Figure 8: Programming Language Satisfaction Ratings

Conversely, languages like OCaml and Julia showed higher counts of unsatisfactory interactions. This may reflect intrinsic challenges these languages pose to ChatGPT, or perhaps higher user expectations that are not being met. Such data is invaluable for guiding enhancements in ChatGPT, indicating specific areas for improvement.

Interestingly, Figure 9 indicates a predominant neutrality in responses regarding C# solutions, suggesting that while users are not overly dissatisfied, there is room to elevate the satisfaction level with ChatGPT's responses in this language.

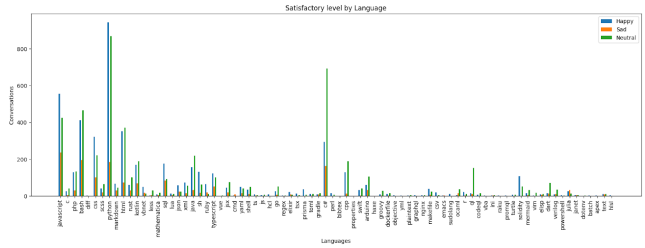

Figure 9: ChatGPT Language Proficiency Chart

Figure 10 presents a comparison between the number of satisfied and unsatisfied interactions with ChatGPT. The data clearly shows a predominance of satisfied (happy) occurrences over unsatisfied (sad) ones, indicating that a majority of users were pleased with ChatGPT's responses.

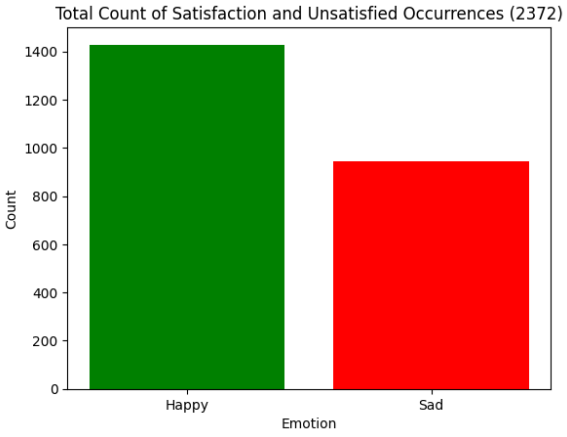

Figure 10: Overall satisfaction of users

Results for RQ3:

Figure 11's bar chart illustrates the average number of prompts after which ChatGPT incorrectly responds, broken down by programming language. The variation across languages suggests that ChatGPT's proficiency varies, with stronger performance in some languages than others.

The chart highlights significant differences in the number of prompts before failure for each language. This could be attributed to factors like language complexity, question specificity, or the volume of available training data. Languages such as JavaScript, TypeScript, and Python, which are extensively used, show higher bars, implying that ChatGPT can sustain accurate responses over more prompts in these languages, likely due to more comprehensive training data.

Conversely, languages represented by shorter bars, such as R, TSX, and Diff, indicate quicker failure rates, pinpointing areas for potential improvement in ChatGPT's training. Generally, more popular languages exhibit higher bars, suggesting a correlation between the abundance of training data and ChatGPT's ability to maintain accuracy over extended interactions.

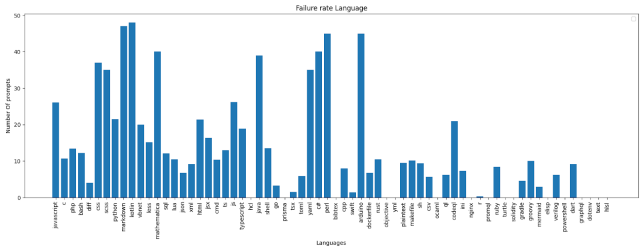

Figure 11: ChatGPT Language Proficiency Chart

CONCLUSION

In conclusion, our study provides a comprehensive understanding of ChatGPT's interaction with programming languages, enriched by our analysis using Keras and LSTM models. These models demonstrated significant variations in training loss and accuracy, underscoring the nuances in ChatGPT's ability to handle different programming languages. The study highlights ChatGPT's adeptness with popular languages like Python and JavaScript, while also pointing out areas for improvement in less common languages. Our approach, combining sentiment analysis with machine learning models, not only sheds light on ChatGPT's current strengths and weaknesses but also provides a roadmap for enhancing its performance in software development contexts. The findings from this study contribute to the broader understanding of AI capabilities in natural language processing and offer a foundation for future advancements in AI-driven chatbot technologies.

REFERENCES

"Language Models are Few-Shot Learners"
Link: https://arxiv.org/abs/2005.14165

“Nuances are the Key: Unlocking ChatGPT to Find Failure-Inducing Tests with Differential Prompting”
Link: https://ieeexplore.ieee.org/document/10298538