From Algorithmic Thinking to Linguistic Thinking: A Case Study of Code Generation Using GPT-3.5 and GPT-4.0

Jia-Wei Lee
Department of Computer Science
Tunghai University
Taichung, Taiwan
s09350139@thu.edu.tw

Jung-Sing Jwo
Department of Computer Science
Tunghai University
Taichung, Taiwan
jwo@thu.edu.tw

Abstract

The emergence of Large Language Models (LLMs) is blurring the lines between natural language and programming. With the widespread adoption of ChatGPT, traditional software development practices are evolving. A future where human coding inputs are no longer essential seems more likely than ever. Our study explores this possibility by designing a task for GPT-3.5 and GPT-4.0 to build a machine learning model with minimal human assistance. We aimed to let the LLMs construct the model and correct errors independently. The entire process took less than one day, significantly reducing human effort. The accuracy of the generated code using GPT-3.5 and GPT-4.0 was 80.09% and 97.45%, respectively. These findings suggest that linguistic thinking may become more important than algorithmic thinking in the era of LLMs.

Keywords: algorithmic thinking, ChatGPT, Large Language Models, linguistic thinking

1. Introduction

Generative AI have changed software engineering forever. The advent of large language models (LLMs) is not only an emerging trend but a substantial shift in how enterprises leverage technologies to gain advantages, the shift is particularly evident in the field of software engineering [23][24]. Historically, the process of software development has been closely tied to algorithm design and code writing. However, with the emergence of LLMs like ChatGPT, the software development landscape is undergoing a rapid transformation where the conventional format is slowly fading away. This tendency suggests the skills prioritized for software development may be shifting from a traditional programming approach towards a prompt engineering approach [19].

Our study aims to investigate whether the development of software projects can indeed transition from an algorithm-centric approach to linguistic-centric approach. To verify our inference, we used the development of an objection detection and location prediction model as a case study to observe whether a shift from algorithm approach to linguistic approach is actually possible.

To this end, we conduct an experiment using two versions of ChatGPT to compare and verify their performances. Specifically, we utilize GPT-3.5 and GPT-4.0 to generate the codes to build the models based on the Seq2Seq LSTM structure [5][6][21]. With this goal in mind, we prompt the two LLMs to build a Seq2Seq model with the same architecture and functionalities. We analyze the generated codes for correctness, completeness and most importantly—the necessity for human intervention.

Our case study reveals that GPT-4.0 successfully generates an executable model which is based on Seq2Seq model with 2 LSTM layers and a 128 features hidden layer in both decoder and encoder, as well as a fully connected layer in the decoder. The codes generated by GPT-4.0 achieve 97.45% correctness. On the other hand, despite generating majority of the components successfully, GPT-3.5 falls short in shaping sequences into correct shapes within model layers. In summary, the codes generated by GPT-3.5 achieve 80.09% correctness rating. While the model does not execute as desired, it does come with essential components and structure that need to be fixed. In the meantime, both experiments are done within a single day which implies

a significant reduction of coding time when comparing with the traditional software development.

2. Related Work

Natural Language Processing (NLP) has been an evolving field of research long before the advent of artificial intelligence. The roots of NLP can be traced back to the mid-20th century, when early efforts focused on tasks like machine translation and speech recognition using rule-based approaches and statistical methods [9][25]. The development of NLP has been greatly facilitated with the growth of machine learning models. Neural networks, particularly the recurrent neural networks (RNNs) and later transformer models, marked pivotal improvements for NLP, allowing for the processing of sequential data and contextual understanding in ways that were previously unattainable. These foundational advancements set the stage for the emergence of LLMs [22], which leverage vast amounts of data and modern computational power to push the boundaries of what NLP systems can achieve.

In recent years, LLM chatbots have swiftly dominated the AI scene, the most prominent ones being OpenAI's GPT-3, and GPT-4 [1][2][3]. Other cutting-edge LLMs have also been developed including Anthropic Claude 3 [7] and Google Gemini [4]. With the help of large data and modern computational power, LLMs are more coherent than ever and can undertake notably complicated tasks such as teaching or programming. However, advanced LLMs still face many limitations, acquiring the optimal response can also be quite intricate due to the randomness nature in NLP models. To maximize the benefits of LLMs, the field of prompt engineering [8][10] has emerged as a scholarly pursuit worthy of investigation, the goal for prompt engineering is to obtain desired results through LLM interactions and mitigate the limitations of LLMs.

As the market for LLM chatbots grows increasingly competitive, comparing the performances between trending chatbots became a prevalent task among researchers. For deeper ways to evaluate the performances of LLM's code generation, an array of options has been proposed by researchers to test the LLMs in more rigorous manner. One of the most widely used measurements for LLMs is the MMLU dataset [11] by Hendrycks et al. A good criterion for illustrating the capabilities of LLMs is their code generation potential. Many benchmarks for assessing LLMs' code generating abilities have been proposed by researchers, Chen et al. presented the HumanEval dataset and Codex [12], a code specific LLM consists of 164 programming problems to test the LLM's code generation. Hendrycks et al proposed APPS (Automated Programming Progress Standard) [13], a code generation benchmark dataset composed of 10000 coding problems with difficulties ranging from introductory to professional levels.

For the goal of our study case, we assigned GPT-3.5 and GPT-4.0 to build a Seq2Seq LSTM network [5, 6, 21]. Specifically, we assigned the LLMs to build a Seq2Seq LSTM model that conducts time series forecasting [18][20] predicting task trained using our self-recorded video data, our training data is around 6 hours long encoded using the YUV color space [14]. We choose Seq2Seq model as the challenge for our case study because of the moderated difficulties in this objective and the popularity of the model in the NLP [15][16][17] research, the goal to build a widely used model can be served as an exemplification on how companies can integrate LLMs with minimum coding in a practical scenario, for small and medium companies, this opens up many potentials with limited resources, as many companies would usually feel reluctant to sustain this type of operation due to budget and technical reasons.

3. Experiments Using GPT3.5 and GPT4.0

The data used for the experiments to evaluate how good an LLM can create machine learning models is based on a set of YUV vector data. The Y dimension represents grayscale values ranging from 0 to 255, while the U and V dimensions represent the acceleration of movement in the x and y directions, respectively. The main purpose of this AI model is to predict the future

locations of the identified objects that may appear.

In order to create the above addressed model, Seq2Seq (Sequence to Sequence) LSTM (Long Short-Term Memory) is chosen. Seq2Seq LSTM is a type of neural network architecture used primarily for tasks where input sequences need to be transformed into output sequence. Since the focus of this paper is on how LLM generating codes, for a detailed introduction of Seq2Seq LSTM please refer to [5][6][21].

In order to compare the effects of GPT in different versions, the same experiments are done for versions 3.5 and 4.0 respectively. Note that while we try to give them the same prompts, sometimes the prompts may not be identical because of the difference in how the two versions response to prompts, GPT-4.0 usually plan things very meticulously and guide the user through the whole process, whereas GPT-3.5 needs clearer instructions. Regardless of the wording we used on the two versions, the purpose of the prompts remains the same. The following are the prompts and responses form the experiments.

A. Define the Goal of the Task

Prompt:

Can you help me with a project. I am trying to build a Seq2Seq+LSTM model for time series forecasting.The attached .csv file is the training data, it contains the Y、U、V values of a persons hand. My goal is to train a Seq2Seq+LSTM model to detect the trajectory of human hands, with Y、U、V 、x 、y values as my inputs and train a model that predicts human hands' future movement. So, the overall plan is this: Train a time series forecasting model with the YUV value within the (x,y) position, predict the future (x,y) position based on the YUV values. Can you help me set up a plan on how to write a code for time series forecasting model, taking the YUV values as input and predict time series with it ? Then guide me step by step with the plan. So, the overall plan is this: Train a time series forecasting model with the YUV value within the (x,y) position. Can you help me set up a plan on how to write a code for time series forecasting model, taking the YUV values as input and predict time series with it ? Then guide me step by step with the plan.

B. Data Preparation

After the initial prompt, we instruct ChatGPT to load and process the data, whilst appointing the data to be split into 80:20 ratio, and assigned the required input and predict sequences length for our scenario.

Prompt:

Okay, please proceed to load the data. For the train、test split, I would like a 80:20 ratios, please assign 90 frames(steps) for the input sequence and predict 30 frames ahead.

C. Construct the Model

After preparing the data, ChatGPT suggested a layout for the structure of our model, we carefully reviewed the structure and determined it to be feasible, we prompt ChatGPT to continue setting up the model.

Prompt:

Yes please proceed. (Please use Pytorch.)

In this step, GPT-4.0 encountered an error when trying to self-analyze the data, this is a common issue that occurs when GPT-4.0 conducts data operation, we added a prompt in the end, instructing the LLM to ignore the analysis error as the LLM will try to repeatedly process the data in an attempt to fix the issue. They do this for many loops

and outputs very lengthy response filled with redundant information unless they were prompted not to do it.

D. Further Modification and Enhancement

After evaluating the code generated by ChatGPT, we deemed the model constructed by GPT-4.0 to be feasible, while the model constructed by GPT-3.5 a little bareboned. We further prompted the LLM to modify the model for more features.

Prompt:

(1) Can you create a new folder every time I train a model and save the best model inside after every time I train ? (Use library such as Time to help me create new folder)
(2) Can you create a function that draws the validation loss and training loss graph and also save it in the same folder ?
I want the file to be saved in folder :C:/Users/home/Desktop/Prompt Engineer/model_output

E. Further Implementation

The code constructed so far by GPT-4.0 is suffice enough to serve as a groundwork for Seq2Seq LSTM model, but there are still many features left to be desired. To better monitor the training process, we prompted the LLMs to add progress bar.

Prompt:

Okay, I'd like to make a new adjustment. Can you modify my code and use Tqdm library to print progress bar to showcase the percentage of progress and print out the critical information such as accuracy and lost for each training loop.

4. Discussion

In the previous section, we have introduced our approach to the problem and the prompts for ChatGPT. In this section we will examine the types of the error codes returned by GPT-3.5 and GPT4.0 respectively.

A. Mismatching dimensions

Mismatching in size or dimension is one of the most ubiquitous errors that can occurred when trying to build a ML model through LLMs. Due to the complexity of these neural networks, it is very likely that LLMs make mistake when determining the input and output size for each layer. The error happened because the input size of decoder does not match with the output size of decoder. To rectify this issue, we reshape the input with the expected shape.

B. Lack of teacher forcing

Teacher forcing is a vital technique used in Seq2Seq model, this approach helps accelerate and stabilize the training process by taking the ground truth as input rather than its own generated output from previous steps. In the code generated by GPT-3.5, teacher forcing was not featured, this could lead to negative impacts such as error accumulation and slower convergence. To resolve this problem, we implemented a basic teacher forcing feature for this model.

C. Incorrect implementation of time progress bar

In step E, GPT-3.5 failed to implement the progress bar as required. Instead of printing the individual loss for each epoch, it outputs a single progress bar representing the entire training process. The problem of this issue may lie within the ambiguity of our prompt, but given the fact that GPT-4 successfully implemented the feature, we deemed this a

mistake by the LLM.

D. Type mismatching between input tensor and LSTM parameters

The error message is: *RuntimeError: Input and parameter tensors are not the same dtype, found input tensor with Double and parameter tensor with Float*. This error occurred because the input data and LSTM layers were in double precision, while the LSTM internal parameters were in float precision. This error happened because the code did not convert the input data from the dataset to float, leading to the default double type in some cases. We converted the data to float and resolved the problem, note that we asked GPT-4.0 to resolve this error and the LLM successfully debugged the issue.

In the case of GPT-4.0, the LLMs successfully achieve all our prompt requests and built a Seq2Seq model that served as a solid foundation to be expanded upon. The model is structured with an encoder and decoder both consisted of 2 LSTM layers and 128 features, a fully connected layer in the decoder, the model is also enhanced with teacher forcing mechanism. Conversely, the GPT-3.5 generated Seq2Seq model did not execute successfully, the issues lie within the LSTM tensor shapes, and after numerous attempts at prompting GPT-3.5 to solve the problem, it unfortunately falls short. To fix this error, we manually reshape the encoder output tensor size to match the decoder. The Seq2Seq model built by GPT-3.5 includes 2 LSTM layers for the encoder with 64 features, and 2 LSTM layers in the decoder alongside a fully connected layer for the decoder (64 features). Worth noting that the GPT-3.5 codes do not come equipped with teacher forcing mechanism, which may affect the Seq2Seq model's prediction significantly.

Both LLMs make mistakes during the interactions. We evaluate their correctness based on the lines of code (LoC) that are changed with human inputs in order for the code to run without error. While GPT-4.0 performs significantly better, it unfortunately makes one mistake when processing data. From our measurement GPT-4.0 achieves 97.45% correctness in terms of code generation. On the other hand, while GPT-3.5 fails to deliver a functional code, after some bug-fixing and features implemented it also works. Despite the modest structure, GPT-3.5 achieves 80.09% correctness with our measurement.

	Code Generated (LoC)	Incorrect codes (LoC)	Correctness
GPT-3.5	121	23	80.09%
GPT-4.0	157	4	97.45%

It is worth noting that while our data highlights the inability of GPT-3.5 to complete the task, the results may vary due to the inherited randomness nature of the NLP model's predictions. Throughout our study, there are instances where GPT-4.0 provided inexact response, causing us to revisit our prompts repeatedly in order to acquire the desired results. Conversely, GPT-3.5 occasionally produces superior outputs with varying prompts. Consequently, we encourage users to experiment with different prompt sets to obtain better results.

5. Conclusion

This paper introduced a case study of building an AI model with bare minimum human code inputs. The purpose is to explore a possible future where small and medium-sized enterprises which lacks the funding and resources, can build their own AI models without coding efforts. Our case study reveals that GPT-4.0 delivers a stellar performance compared with GPT-3.5. GPT-4.0 successfully constructs a Seq2Seq model with comprehensive functionalities with 97.45% of accuracy. On the other hand, GPT-3.5 falls short in the task, fails in delivering desired results and cannot fix the error through prompts. Despite the dissatisfied result, GPT-3.5 still manages to write the majority of the code properly, and achieves 80.09% accuracy. These results affirm that LLMs can efficiently generate complex code, reducing the need for extensive

programming knowledge. While challenges remain, our findings suggest a future where software development can be accessible to those without coding expertise, relying instead on natural language interactions.

References

[1] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

[2] GPT-4 Technical Report https://cdn.openai.com/papers/gpt-4.pdf

[3] Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., ... & Zhang, Y. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

[4] Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J. B., Yu, J., ... & Ahn, J. (2023). Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805.

[5] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27.

[6] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[7] Introducing the next generation of Claude https://www.anthropic.com/news/claude-3-family

[8] Liu, V., & Chilton, L. B. (2022, April). Design guidelines for prompt engineering text-to-image generative models. In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (pp. 1-23).

[9] Johri, P., Khatri, S. K., Al-Taani, A. T., Sabharwal, M., Suvanov, S., & Kumar, A. (2021). Natural language processing: History, evolution, application, and future work. In Proceedings of 3rd International Conference on Computing Informatics and Networks: ICCIN 2020 (pp. 365-375). Springer Singapore.

[10] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9), 1-35.

[11] Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2020). Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300.

[12] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

[13] Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., ... & Steinhardt, J. (2021). Measuring coding challenge competence with apps. arXiv preprint arXiv:2105.09938.

[14] Podpora, M., Korbas, G. P., & Kawala-Janik, A. (2014, September). YUV vs RGB-Choosing a Color Space for Human-Machine Interaction. In FedCSIS (Position Papers) (pp. 29-34).

[15] Eisenstein, J. (2019). Introduction to natural language processing. MIT press.

[16] Chowdhary, K., & Chowdhary, K. R. (2020). Natural language processing. Fundamentals of artificial intelligence, 603-649.

[17] Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. Journal of the American Medical Informatics Association, 18(5), 544-551.

[18] Chatfield, C. (2000). Time-series forecasting. Chapman and Hall/CRC.

[19] Sauvola, J., Tarkoma, S., Klemettinen, M., Riekki, J., & Doermann, D. (2024). Future of software development with generative AI. Automated Software Engineering, 31(1), 26.

[20] De Gooijer, J. G., & Hyndman, R. J. (2006). 25 years of time series forecasting. International journal of forecasting, 22(3), 443-473.

[21] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information

processing systems, 30.
[22] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. arXiv preprint arXiv:2303.18223.
[23] Cambon, A., Hecht, B., Edelman, B., Ngwe, D., Jaffe, S., Heger, A., ... & Teevan, J. (2023). Early LLM-based Tools for Enterprise Information Workers Likely Provide Meaningful Boosts to Productivity. Technical Report MSR-TR-2023-43. Microsoft. https://www. microsoft. com/en-us/research/publication/early-llm-based-tools-for-enterprise-information-workers-likely-provide-meaningful-boosts-to-productivity.
[24] Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2023). Gpts are gpts: An early look at the labor market impact potential of large language models. arXiv preprint arXiv:2303.10130.
[25] Jones, K. S. (1994). Natural language processing: a historical review. Current issues in computational linguistics: in honour of Don Walker, 3-16.