CoCoT: Coding Companion Tutor

Malak Slim, Batool Ibrahim, Aya El-Saoudi, Hazem Khattar

***Abstract*— As coding has become one of the most important skills for competing in today's diverse workplaces and research fields, many students are required or motivated to learn it. However, challenges such as time constraints, financial limitations, and the inherent difficulty of programming can make it difficult to acquire these skills. To address these challenges, we developed CoCoT (Coding Companion Tutor), a chatbot designed to combine teaching qualities with programming expertise, specifically aimed at tutoring Python due to its simplicity for beginners. Following its development, we assessed its effectiveness in teaching Python programming through both internal evaluation and human-subject testing. Results showed CoCoT's strong ability in teaching Python, demonstrating significant improvement over the original base model, and high user satisfaction according to human-subject feedback. The full code used in this work can be found in our GitHub repository at MSBA-316-NLP-Project-CoCoT.**

I. Introduction

Nowadays, technology is deeply embedded in our daily routines, revolutionizing how we interact, learn, and conduct business. Accordingly, understanding these technologies and how to apply them is essential for the current generation to engage in future workplaces. A key skill in understanding and applying technology is coding. Therefore, coding has become a fundamental component across various educational levels, including high schools and higher education.

Despite its importance, coding presents challenges for many students because it requires conceptual understanding and procedural knowledge. Additionally, not all students have the motivation or time to enroll in coding workshops or training courses. To address these gaps, our project introduces "CoCoT" (Coding Companion Tutor), a new chatbot designed to assist programmers in learning and writing code within formal educational settings.

CoCoT is not just a generative language model; it operates as a teacher-based model specifically designed to act as a tutor, rather than functioning as a general chatbot for users. It can provide access to coding resources and instructions from anywhere and anytime that suits the students.

CoCoT is designed specifically to teach Python language. This choice is motivated by Python's reputation as an ideal language for beginners [1], making CoCoT suitable for users at various levels of programming knowledge, including those who are starting from scratch. Moreover, Python has become the most popular programming language today. A recent statistical study has shown that the number of Python job postings significantly exceeds the number of job postings for Java developers [2].

The remaining part of this report is structured as follows: Section II provides an overview of related works in the literature. Section III describes the data to build CoCoT and the preprocessing steps. Section IV outlines all the components involved in developing CoCoT. Section V presents a detailed analysis of the obtained results and the final evaluation. Section VI reveals the challenges and insights presented during our projects. Finally, Section VII offers conclusions and future directions.

II. Literature Review

The integration of large language models (LLMs) into educational technology has ushered in new opportunities for personalized and effective learning experiences, particularly in programming education. Educational chatbots, such as Python-Bot, illustrate the practical application of LLMs by offering assistance with basic Python syntax and semantics. Developed on the SnatchBot platform, Python-Bot utilizes Artificial Intelligence Markup Language (AIML) to match user queries with predefined responses, proving effective in improving comprehension and reducing dropout rates in programming courses [3]. Fine-tuning LLMs like LLaMA is crucial for adapting these models to specific educational tasks. The LLaMA-Adapter, introduced by Zhang et al. [4], employs a lightweight fine-tuning method that efficiently adapts the LLaMA model with minimal parameters and training time while preserving pre-trained knowledge. This method enhances instruction-following and multi-modal reasoning, demonstrating a superior balance between performance and training efficiency compared to other methods like Alpaca-LoRA.

Additionally, integrating diverse datasets is vital for developing robust educational tools. The "Me-LLaMA" project exemplifies this by creating a specialized medical LLM through extensive domain-specific data, achieving superior performance in medical tasks [5]. Furthermore, recent research by Denny et al. [6] highlights the desirable characteristics of AI teaching assistants in programming education. Their study reveals that students appreciate tools that provide instant support and scaffolding rather than direct solutions, emphasizing the potential of AI assistants to offer equitable, round-the-clock support and address the limitations of human teaching assistants. Overall, the convergence of educational chatbots and fine-tuned LLMs represents a significant advancement in educational technology, offering scalable and resource-efficient solutions that enhance learning outcomes and instructional effectiveness. Future research should continue exploring these technologies to maximize their educational impact.

III. Data Curation and Preprocessing

To fine-tune our model, we selected three different datasets, each contributing to one or more aspects that we wanted to incorporate into our chatbot. These datasets are text-based, featuring user prompts paired with corresponding responses. Additionally, the datasets are sourced from HuggingFace [7], ensuring high-quality and accessible data.

A. Data Collection

The selected datasets are described as follows:

- *Dataset 1 - "codefuse-ai/CodeExercise-Python-27k" [8]:* This dataset contains 27,000 Python programming exercises presented in English, covering a wide range of topics from basic syntax and data structures to algorithm applications, database queries, and machine learning. This dataset was created with the assistance of a teacher model and Camel [9], though it has not undergone rigorous validation, so some questions or answers may contain errors or semantic duplicates. The creation process involved several steps: first, a seed set of Python knowledge points was curated. Each seed was then embedded into a fixed task template to produce a "Task Prompt," which was used by a teacher model to generate relevant exercise questions. Camel was employed to refine these prompts for greater accuracy and diversity. Subsequently, these refined prompts were input into the teacher model to generate the corresponding exercise questions and answers. Finally, each question was paired with its answer, and duplicates were removed. This dataset is a valuable resource for generating diverse Python exercises, though users should be aware of its potential imperfections.

- *Dataset 2 - "ibl-best-practices-instructor-dataset" [10]:* This dataset demonstrates a conversation between a student and a teacher. It consists of 380 samples, each including a student's question, a virtue expected from the tutor, and a response that answers the student's question. Such dataset reflects the features and qualities that should be presented in a good teacher without including any programming aspects or codes. We selected this dataset to ensure that our model is trained to exhibit these features.

- *Dataset 3 - "KrisPi/PythonTutor-Evol-1k-DPO-GPT4-vs-35" [11]:* This dataset supports the fine-tuning of models for Python code generation. It originates from the Evol-Instruct-Code-80k-v1 dataset [12], which is based on GPT-3.5 Turbo outputs. From this dataset, 1,000 samples containing Python code snippets were selected. GPT-4 was used to generate improved responses for these samples, aiming to enhance LIMA (Language Instruction for Model Alignment)-like "Python Tutor" Instruct fine-tuning, favoring GPT-4's responses over GPT-3.5 Turbo. The dataset was further refined by removing responses that included terms like"impossible" or"sorry", which were deemed refusals. The GPT-4 system prompt directed the model to produce Python code wrapped in triple backticks and comments, with each function including a detailed docstring.

B. Data Preprocessing

To prepare the selected datasets for the fine-tuning process, we followed a systematic approach to properly combine and preprocess them.

Dataset 1 was not directly accessible from HuggingFace. Therefore, we cloned its directory and extracted the necessary information by parsing the JSON Lines. The relevant prompt-response pairs were then saved as a DataFrame in a CSV file using 'Pandas' and subsequently converted into a HuggingFace Dataset object. Datasets 2 and 3 were directly loaded from HuggingFace.

Each dataset was then carefully preprocessed to ensure consistency in column names, aligning them with a unified format that includes a ***'prompt-response'*** pair for each dataset. Columns that are not needed for training - other than ***'prompt'*** and ***'response'*** - were removed from the corresponding datasets.

The prepared datasets were then combined into a single comprehensive dataset by concatenating them, resulting in a cohesive dataset with a total of 28,547 entries. This unified dataset encompasses both key aspects of the proposed chatbot: teacher traits and methods of educating, along with Python programming content.

Following that, we shuffled the combined dataset to ensure that the data was randomly distributed. This step helps in reducing any biases that may arise from the order of the entries, promoting a more robust and generalized model during the fine-tuning process. The shuffled dataset was then split into 80% for training and 20% for testing. The testing dataset was later used for evaluating the fine-tuned model.

Finally, we tokenized the dataset to convert the text into a format suitable for model input. This step involved transforming the prompt-response pairs into tokens that the model can process, ensuring compatibility with the model's input requirements and optimizing the efficiency of the fine-tuning process.

IV. Building the Chatbot

A. Model Selection

In selecting the ideal model for the CoCoT chatbot, we evaluated several candidates, which fall into two main categories:

1) Generalized multi-task language models: This category includes models such as GPT, LLaMA, and Mistral.
2) Specialized models: These models are fine-tuned specifically for programming or tutoring tasks.

While specialized models may provide superior performance in Python tutoring, we opted to choose to focus on generalized multi-task language models. This decision was made to ensure that our selected model can handle a broader range of prompts beyond Python and programming, thereby enhancing its overall capabilities. We selected LLaMA-3 8b

as our base model because it is open-source, which allows for structural modifications and adaptations to better meet our specific objectives.

LLaMA-3 stands out due to its advanced natural language processing capabilities and superior contextual understanding. This model excels in explaining complex programming concepts with clarity and precision, and its ability to generate contextually accurate code snippets makes it particularly effective for educational purposes. Although this model may demand more computational resources due to its large size, its advantages in delivering detailed, relevant responses, and open-source availability, outweigh this drawback, making it ideal for an interactive tutoring environment.

B. Training and Evaluation

For the training process, we loaded the selected base model "LLaMA-3 8b" from HuggingFace along with its tokenizer, and employed the 'QLoRA' framework (Quantized Low-Rank Adaptation) and 'transformers' library with a series of carefully selected hyperparameters. Using 'QLoRA' offers the advantage of combining quantization with low-rank adaptation techniques to fine-tune pre-trained language models more efficiently while reducing memory usage.

The prepared training dataset, as described in section III-B, was used with a batch size of 1, using gradient accumulation steps of 8 to effectively simulate a larger batch size. The 'AdamW optimizer' with 8-bit precision was utilized, balancing performance with memory efficiency. The learning rate was set to $2e-4$, and gradient clipping was applied with a 'max_grad_norm' of 0.3 to manage exploding gradients. Training was configured for a maximum of 30 steps, with a 'warmup_ratio' of 0.03 and a linear learning rate scheduler. Mixed precision training was used, and weight decay was set to 0.01 to regularize the model and avoid overfitting. The model's performance was monitored by logging the training loss, which demonstrated a steady improvement: the loss decreased from 0.59 at step 10 to 0.51 at step 20, reaching a minimum of 0.47 by step 30, as shown in Figure 1, illustrating the model's learning progress. Note that we implemented various iterations to fine-tune the model hyperparameters by monitoring the training loss. We eventually converged to the parameters' values discussed earlier. Although our original plan was to use the validation dataset loss as a metric, we used only the training loss due to challenges discussed in Section VI.

The fine-tuned model was then saved along with its tokenizer for later evaluation and implementation purposes.

C. Implementation

In this section, we detail the implementation of our chatbot system "CoCoT". We used 'LangChain' to handle conversational AI capabilities, ensuring a robust and responsive interaction. Additionally, we develop a user interface using 'Gradio' to facilitate seamless user interactions.

1) Implementing the Chatbot using LangChain: To implement the chatbot, we began by loading the saved fine-tuned model and tokenizer, initializing a text generation pipeline using the transformers library. We then created a LangChain pipeline with the text generation pipeline and set up an LLMChain using this pipeline and a predefined prompt template. The LLMChain is utilized to prompt the model and generate responses, leveraging the knowledge acquired during the fine-tuning process.

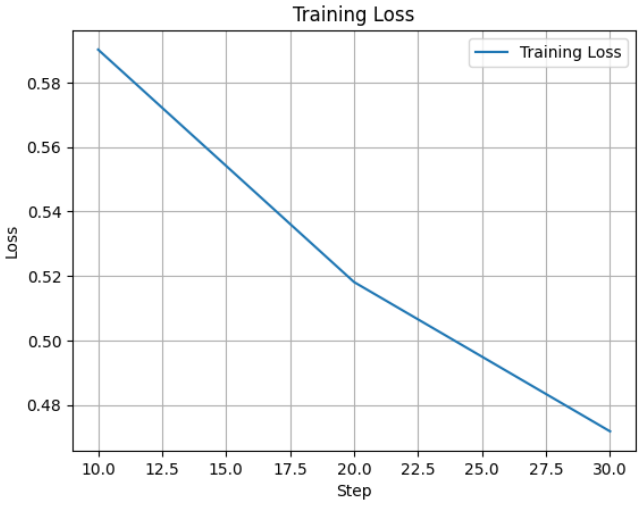


Fig. 1: Progression of Training Loss Over Training Steps

2) User Interface (UI): To facilitate user interactions with the chatbot, we created a user interface using Gradio. This involved integrating the created LLMChain into the interface, allowing users to interact with the chatbot seamlessly and receive responses generated by the model.

The interface features a chat-like page with a text box for users to input their queries and view the entire chat history. Additionally, it includes a clear button to reset the chat and remove all messages. Below the chatbot's responses, there are *"Like"* and *"Dislike"* buttons allowing users to provide feedback on the responses. Currently, this feedback mechanism is in place for user interaction, but future plans include integrating this feedback as a feature to enhance the chatbot's learning and performance. The intuitive design including the selected theme for the UI ensures a user-friendly experience. Figure 2 shows a screenshot of the created UI.

D. Logical Inference

While the primary objective of CoCoT is to teach coding skills, it is also important to incorporate additional tools that allow CoCoT to leverage advanced reasoning and problem-solving capabilities. This is achieved by equipping CoCoT with inference tools that allow it to derive new information or conclusions based on existing facts and rules, rather than relying solely on the dataset it was trained on to generate responses. Accordingly, throughout our project, we tried to integrate logical inference into our model using multiple inference tools.

a) ***Using Pyke****:* As a first step, we used Pyke [13] in Python to build a customized rule-based agent. We created

our rules from scratch, and started the agent by asking the user the following questions:

- Are you familiar with any programming language? (if the answer is yes, we move to the next question, if no we stop asking any questions)
- Are you familiar with Python?
- Rate your level of knowledge in Python.

Based on the user's responses, the following rules will be applied:

- If the user has zero knowledge of programming, assign a beginner CoCoT tutor.
- If the user has an intermediate level, assign an intermediate CoCoT tutor.
- If the user has an expert level, assign an advanced CoCoT tutor.

Our original plan was to use the information from our rule-based agent (the tutor level) as input for our language model alongside the prompts provided by the student. This will enable CoCoT to offer responses based on the user's level of knowledge, making the learning process more efficient and somehow customized. However, due to some challenges that we will discuss in Section VI, we transitioned from using Pyke to employing LangChain-defined agents.

*b) **Using LangChain Agents**:* The goal is to build an LLM-agent which integrates our fine tuned language model with other tools which provide the agent with reasoning and decision-making capabilities. We considered two types of tools: predefined tools available in LangChain and customized tools that we can develop to build personalized solutions. For predefined tools, we used:

- **Wikipedia:** This tool is essential for providing our model with knowledge from various domains. It enables our chatbot to answer questions using information from Wikipedia.
- **llm-math:** This tool leverages the power of our model to perform complex mathematical reasoning and calculations.
- **PythonREPLTool:** This tool provides the agent with the ability to execute Python code in a real-time environment. It is essential in case of CoCoT as it allows students to see immediate results and receive instant feedback on their programming effort.

For the customized tools, we created a tool which returns the date in real time when asked by the user, making our model helpful for tasks like scheduling, planning, and keeping track of time.

First, we created and tested an agent for each tool independently. After making sure that all tools are functioning properly, we combined all the tools into a single LLM-agent.

Regarding the language model, our original plan was to integrate these tools into our fine-tuned model as mentioned earlier. However, due to core challenges that we will discuss in Section VI, we had to select another LLM model compatible with these tools. We attempted to use OpenAI models such as GPT, but access issues prevented us from utilizing them. As a result, we opted to work with Mistral [14]. Consequently, we successfully developed a Mistral-based LLM-agent incorporating all the tools mentioned, and we integrated this agent into a chatbot interface using Gradio similar to the one described in section IV-C.2.

V. Results and Discussions

To evaluate CoCoT, we conducted two types of assessments. First, our team performed internal tests. Then, we carried out human-subject tests involving a group of novice programmers.

A. Internal Evaluation

In this evaluation, we conducted internal tests through comparing CoCoT to its core model, LLaMA-3 8b. To facilitate this comparison, we integrated LLaMA-3 8b into the same interface used for CoCoT, enabling us to test both models using their chatbots.

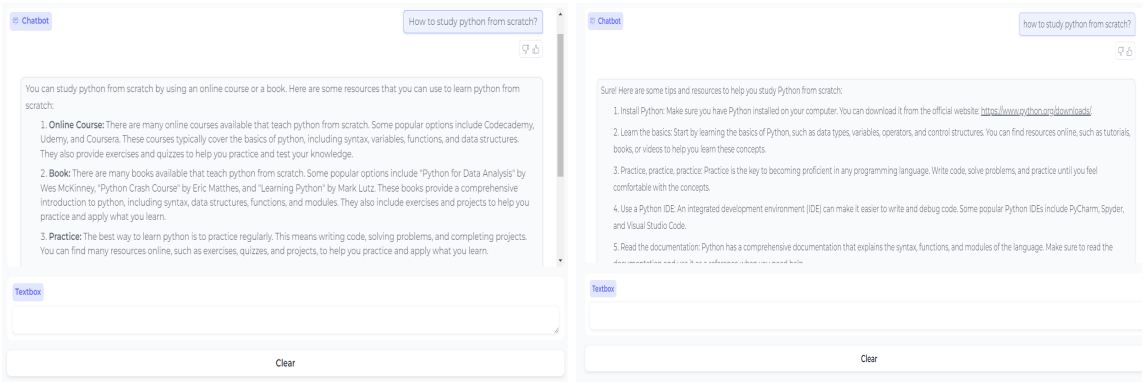
During testing, we sent a set of five prompts to each of the chatbots for evaluation and analyzed their responses. The prompts primarily focused on advice for learning Python or addressing technical python programming questions. This type of assessment was chosen to highlight the differences between the original model (before fine-tuning) and CoCoT (after fine-tuning).

By comparing the responses, we were able to observe the impact of our fine-tuning. CoCoT delivered more accurate and detailed answers, reflecting the features typically desired for coding tutor. For example, in Fig. 2, we asked both models, "How to study Python from scratch?" LLaMA-3 8b suggested using online courses or books, among other options. In contrast, CoCoT provided more specific guidance, including detailed methods and references to helpful websites. We then asked, "How to implement a while loop in Python?" As shown in Fig. 3, LLaMA-3 8b gave a brief and direct response, whereas CoCoT offered a more detailed explanation of the concept, along with an example to aid the student's understanding. Other examples are shown in Figures 4, 5, and 6, where the differences between CoCoT's and LLaMA-3 8b responses can be directly observed. Our internal evaluation allows us to conclude that CoCoT is better suited for Python tutoring tasks compared to LLaMA-3 8b.

B. Human-Subject Evaluation

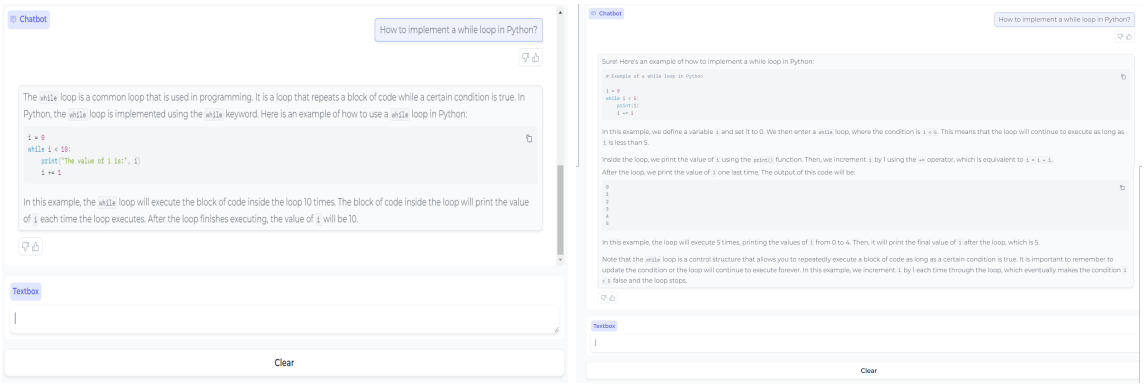
To evaluate CoCoT fairly, we conducted human-subject tests to determine whether CoCoT made a noticeable difference for novice users. To do so, we selected three Python exercises, presented in the Appendix, and asked five novice users to solve them with the help of CoCoT. We made sure that users won't ask CoCoT for a direct answer, but instead learned how to solve the exercises themselves with CoCoT guidance. Additionally, to ensure consistency, all users were given the same exercises.

After completing the exercises with CoCoT, the users were asked to fill out a survey. The survey was designed based on Bloom's Revised Taxonomy [15] to incorporate educational



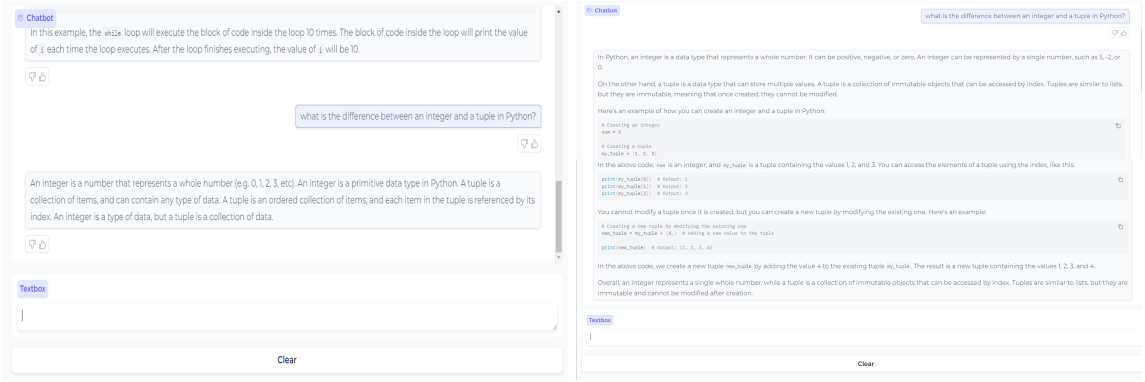
(a) LLaMA-3 8b (b) CoCoT

Fig. 2: Generated Response using LLaMA-3 8b Versus CoCoT for prompt-1



(a) LLaMA-3 8b (b) CoCoT

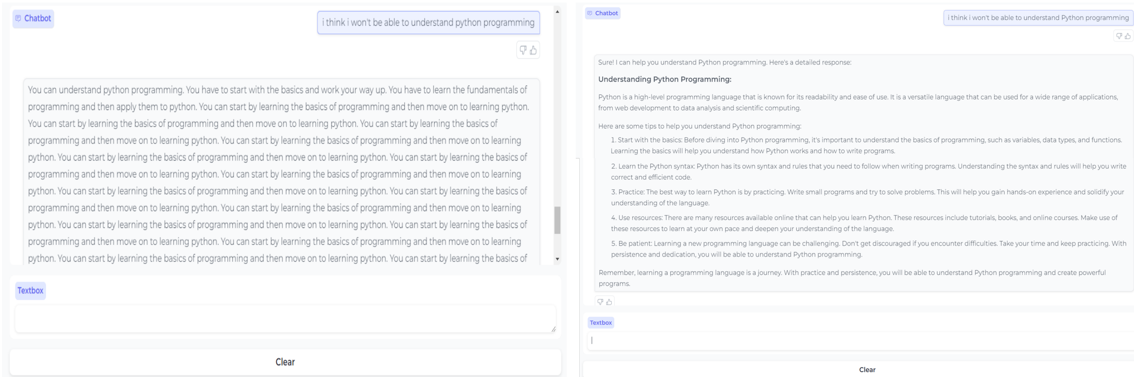
Fig. 3: Generated Response using LLaMA-3 8b Versus CoCoT for prompt-2



(a) LLaMA-3 8b (b) CoCoT

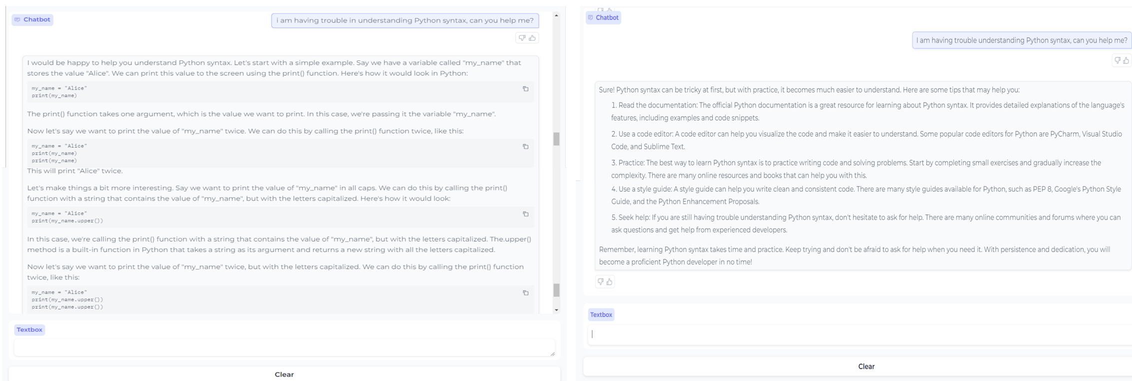
Fig. 4: Generated Response using LLaMA-3 8b Versus CoCoT for prompt-3

perspectives, in addition to subjective questions to rate users' experiences. Bloom's Taxonomy distinguishes four dimensions of knowledge: factual, conceptual, procedural, and metacognitive. Additionally, it includes six categories in the cognitive process dimension: remember, understand, apply, analyze, evaluate, and create. According to [16], coding tutoring involves various aspects of these dimensions. In the knowledge dimensions, it requires factual knowledge (understanding syntax), conceptual knowledge (understanding algorithms), and procedural knowledge (building complete source code). In the cognitive process dimension, students need to understand, analyze, and apply concepts. Therefore, we created the survey in a way to assess how well CoCoT provides these features to students. The survey questions are presented in the Appendix, where users can rate their responses on a scale from 0 to 10. Questions 1 to 7 assess CoCoT's performance, while questions 8 to 10 evaluate the student's ability to benefit from CoCoT.



(a) LLaMA-3 8b (b) CoCoT

Fig. 5: Generated Response using LLaMA-3 8b Versus CoCoT for prompt-4



(a) LLaMA-3 8b (b) CoCoT

Fig. 6: Generated Response using LLaMA-3 8b Versus CoCoT for prompt-5

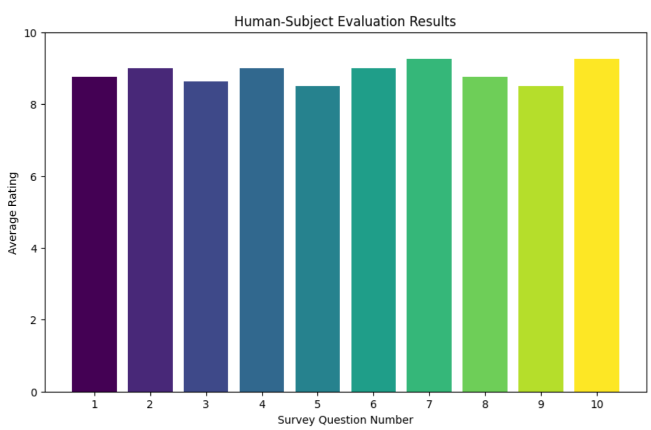


Fig. 7: Average results obtained from human-subject evaluation over 5 subjects

For each question in the survey, average results over the five human subjects were collected and are presented in Fig. 7. The obtained results align with our expectations, showing a high level of user satisfaction with CoCoT as a Python companion tutor. The positive responses indicate that CoCoT successfully achieves the desired features, fulfilling its role as an educational tool in Python programming.

VI. Insights and Challenges

Throughout the project, we encountered several challenges that impacted our approach and outcomes:

1) GPU Resource Limitations: For running the fine-tuning and evaluation processes, we used 'Google Colab' [17] and the 'Lightning' [18] framework, both of which provided limited GPU resources. Due to the large size of the both the base model "LLaMA3-8b" and the combined dataset, we faced constraints with GPU resources. This limitation hindered our ability to implement training with early stopping while monitoring validation loss on validation data. Consequently, we ended up only monitoring the training loss during the training process. This limitation also prevented us from calculating key evaluation metrics, such as accuracy, precision, recall, and F1-score, on the test set using 'trainer.evaluate()' from 'SFTTrainer' after completing the training, which hindered our ability to fully evaluate the fine-tuned model.

2) Integrating Language Models with Rule-Based Systems: We faced challenges integrating the rule-based system developed using Pyke, as described

in section IV-D.0.a, with the language model. The lack of similar projects or references online made it difficult to find guidance or best practices for combining rule-based logic with language models, which further constrained our ability to enhance the model's reasoning capabilities as we originally planned.

3) Integrating Language Models with Inference Tools: We also struggled finding online references that specifically address the integration of fine-tuned language models with inference tools for building LLM agents. Most available examples focused on using pre-trained models from sources like OpenAI or Mistral AI. As a result, we had to adapt our approach and build an LLM agent with Mistral AI instead of using our fine-tuned model (CoCoT). This adaptation limited our ability to integrate inference tools into our model, thereby restricting its reasoning capabilities.

VII. Conclusion and Future Work

In this project, we introduced CoCoT as a Python coding tutor chatbot. CoCoT is designed to assist students in learning Python, even from scratch. To build CoCoT, we fine-tuned the base model LLaMA-3 8b using three different datasets, each capturing essential aspects of a Python tutor, such as teaching features and strong programming skills. Additionally, we integrated logical inference and developed a Mistral-based chatbot equipped with various tools, including a Wikipedia search engine and a Python code executor. Finally, we evaluated CoCoT through both internal and human-subject tests. The internal evaluation demonstrated that CoCoT effectively acquired Python tutoring capabilities after fine-tuning, showing a significant improvement compared to the original LLaMA-3 8b model. Additionally, the human-subject evaluation yielded similar results, revealing a high level of user satisfaction with CoCoT as a Python companion tutor.

In future work, we aim to integrate the tools we have developed into CoCoT, despite the current lack of resources for such integration. Additionally, we plan to incorporate the user's knowledge level into CoCoT, as mentioned in subsection IV-D.0.a.

References

[1] S. Khoirom, M. Sonia, B. Laikhuram, J. Laishram, and T. D. Singh, "Comparative analysis of python and java for beginners," *Int. Res. J. Eng. Technol*, vol. 7, no. 8, pp. 4384–4407, 2020.

[2] "https://leftronic.com/blog/python-statistics: :text=python

[3] C. W. Okonkwo and A. Ade-Ibijola, "Python-bot: A chatbot for teaching python programming." *Engineering Letters*, vol. 29, no. 1, 2020.

[4] R. Zhang, J. Han, C. Liu, A. Zhou, P. Lu, Y. Qiao, H. Li, and P. Gao, "Llama-adapter: Efficient fine-tuning of large language models with zero-initialized attention," in *The Twelfth International Conference on Learning Representations*, 2024.

[5] Q. Xie, Q. Chen, A. Chen, C. Peng, Y. Hu, F. Lin, X. Peng, J. Huang, J. Zhang, V. Keloth *et al.*, "Me llama: Foundation large language models for medical applications," *arXiv preprint arXiv:2402.12749*, 2024.

[6] P. Denny, S. MacNeil, J. Savelka, L. Porter, and A. Luxton-Reilly, "Desirable characteristics for ai teaching assistants in programming education," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 2024, pp. 408–414.

[7] "Huggingface." [Online]. Available: https://huggingface.co/docs/transformers/index

[8] "codefuse-ai," 2024. [Online]. Available: https://huggingface.co/datasets/codefuse-ai/CodeExercise-Python-27k

[9] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for" mind" exploration of large language model society," *Advances in Neural Information Processing Systems*, vol. 36, pp. 51 991–52 008, 2023.

[10] "Best practices instructor dataset," 2024. [Online]. Available: https://huggingface.co/datasets/iblai/ibl-best-practices-instructor-dataset

[11] "Pythontutor-evol-1k-dpo-gpt4-vs-35," 2024. [Online]. Available: https://huggingface.co/datasets/KrisPi/PythonTutor-Evol-1k-DPO-GPT4v-s-35?row=4

[12] "Evol-instruct-code-80k-v1," 2024. [Online]. Available: https://huggingface.co/datasets/nickrosh/Evol-Instruct-Code-80k-v1

[13] "https://pyke.sourceforge.net/."

[14] "Mistral ai." [Online]. Available: https://mistral.ai/

[15] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory into practice*, vol. 41, no. 4, pp. 212–218, 2002.

[16] S. Hobert, "Say hello to 'coding tutor'! design and evaluation of a chatbot-based learning system supporting students to learn to program," 2019.

[17] "Google colaboratory." [Online]. Available: https://colab.research.google.com/

[18] "Lightning ai." [Online]. Available: https://lightning.ai/

Appendix

Survey Questions

1) How effectively did CoCoT explain the assigned exercises?
2) Did CoCoT provide and clarify the necessary syntax for the tasks?
3) How well did CoCoT explain the theoretical background and algorithms?
4) How effectively did CoCoT answer questions about the learning content?
5) Did CoCoT provide personalized instruction based on your needs?
6) How accurately did CoCoT identify and correct your programming mistakes?
7) Did CoCoT give constructive feedback on your work?
8) How well did you understand the problem statement after using CoCoT?
9) Were you able to apply the algorithm effectively?
10) Were you able to build the final source code independently?

Python Exercises

1) Write a Python program that prompts the user to enter two numbers and then prints the sum, difference, product, and quotient of those numbers.
2) Write a Python program that asks the user to enter their name and age, then prints a message addressed to them, stating the year they will turn 100 years old.
3) Write a Python program to calculate the factorial of a given number.