Kashihara, A. et al. (Eds.) (2024). Proceedings of the 32nd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Designing an LLM-Based Dialogue Tutoring System for Novice Programming

Julieto PEREZ[ab*] & Ethel ONG[b]
[a]*MSU-Iligan Institute of Technology, Philippines*
[b]*De La Salle University, Philippines*
*julieto_perez@dlsu.edu.ph

Abstract: In this paper, we present the design of a dialogue-based tutoring system called DT4-Coding to help novice programmers understand their misconceptions about programming concepts and work towards correcting them. The error or misconception detection and correction are based on Ohlsson's Theory of Learning from Performance Error, which suggests that a mentor (in this case, a software tutor) is necessary to help learners detect these errors and provide feedback to repair faulty knowledge. We utilized the constraint-based model (CBM) for domain and knowledge modeling, which allows human experts to represent knowledge as a constraint. These constraints were used to assess student answers and to generate LLM-based feedback via a question-answer form of dialogue. By incorporating human-written domain knowledge, the system could potentially reduce the problem of LLMs in generating solutions that may be factually incorrect.

Keywords: dialogue-based tutoring systems, constraint-based tutor, LLM-based tutoring system

1. Introduction

Large Language Models (LLMs) have significantly transformed teaching and learning, including programming education, by providing easy access to code generation. However, researchers have raised concerns about LLMs' possible detrimental effects on programming instruction since they were introduced (Prather et al., 2024). For instance, LLMs may offer misleading feedback on logic and semantic errors, necessitating teacher involvement to guide students effectively (Kiesler et al., 2023). Moreover, studies have shown that LLMs can be unreliable, sometimes making the same mistakes as students (Hellas et al., 2023). Despite these issues, LLMs' zero-shot language understanding and generation capabilities make them appealing to educational systems (Zhu et al., 2024). However, these systems often rely on prompts that lack a theoretical basis, leading to problems like hallucinations (Chowdhury et al., 2024). Although LLMs can provide personalized and immediate feedback, they may produce incorrect solutions and fail to reveal students' understanding of programming principles.

We proposed a dialogue-based approach to programming instruction to maximize the potential benefits while addressing the limitations of LLMs as a software tutor. This method encourages natural language dialogue, promoting activities such as self-reflection and self-explanation, which help students construct knowledge and identify misconceptions (Chi et al., 1994; Nye et al., 2014; Weerasinghe & Mitrovic, 2006). Currently, we rely on LLMs' zero-shot language understanding capability for assessment. While LLMs have resolved scalability issues in feedback generation, Stamper et al. (2024) argue that many systems overlook crucial aspects of instructional design in the design process. They recommend incorporating insights from learning science into LLM-based intelligent tutoring systems (ITS). To address this, our system uses a Constraint-based Model (CBM) for student and domain modeling, employing expert-encoded knowledge as constraints. This approach may reduce the likelihood of LLMs generating incorrect information by guiding prompts with expert knowledge.

2. Related Studies

Programming is a complex activity that requires various skills, such as problem-solving, abstraction, mathematical logic, testing, and debugging (Ben-Ari, 1998). Dialogue-based tutoring (DTS) has proven to be an effective approach to assisting learners with these challenges (VanLEHN, 2011), as evidenced by systems like the AutoTutor family of software tutors (Nye et al., 2014). Early dialogue systems, such as PROPL (Lane & VanLehn, 2004), relied on handcrafted rules for natural language understanding (NLU) and generation, but this changed with the rise of machine learning, which became the main method for handling natural language processing (NLP) tasks.

With recent advances in deep learning and the introduction of Large Language Models (LLMs), the approach to assessment and feedback generation in programming education has shifted. LLMs' in-context learning capabilities allow feedback generation without the traditional assessment methods, simply by prompting (Matelsky et al., 2023). While LLMs have been increasingly used for assessing programming assignments and helping learners write code (Liffiton et al., 2023; Reeves et al., 2023), few studies have explored dialogue-based approaches that truly understand student learning in programming (Ym et al., 2023).

Stamper et al. (2024) noted that many of these systems overlook key aspects of instructional design and the evaluation of feedback's impact on learning. Traditional intelligent tutors use two main learner modeling approaches: the constraint-based model (CBM), which is suited for ill-defined problems like programming and is based on Ohlsson's theory of learning from performance error (Ohlsson, 1996), and the production rules model, inspired by Anderson's ACT-R theory (Anderson, 1996). CBM is particularly advantageous because it doesn't require a bug library; instead, it checks a student's solution against constraints, with any violation indicating a misconception.

3. System Design

To attain the tutoring system's instructional purpose, it is designed to utilize CBM for student and domain modeling. CBM is based on the Theory of Learning from Performance Errors, which holds that learning undergoes two phases*: error detection* and *correction* (Ohlsson, 1996). To help students understand their errors, we employ a *Question-Answer* strategy. This strategy will expose the student's misconceptions through their reasoning in an answer to a question. Feedback is given to repair the faulty knowledge, allowing the students to integrate the new knowledge, thereby correcting their errors or misconceptions. Graesser's five-step tutoring frame facilitates such a strategy (Graesser et al., 2000.). Figure 1 shows the key components of the dialogue system necessary to facilitate interaction. Each of these components is described in the succeeding sections.

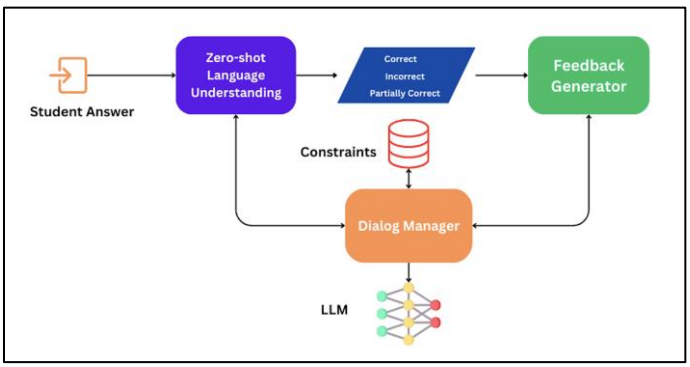

Figure 1. Components of the Dialogue Tutor

3.1 Constraints

The Theory of Learning from Performance Error paved the way for developing a Constraint-based Model (CBM) for student and domain modeling in intelligent tutoring systems (Ohlsson,

1994; Mitrovic & Ohlsson, 1999). A constraint can be considered a unit of declarative knowledge that evaluates the outcomes of tentative actions taken without sufficient reasons, which means it encodes prescriptive rather than descriptive knowledge. The type of constraint used in CBM has the general form: *When such-and-such conditions are the case, then such-and-such other conditions ought to be the case as well (or else something has gone wrong)* (Ohlsson, 2016). Two types of constraints in this work were adopted from the work of Holland et al. (2009): *syntactic* and *semantic* constraints. The syntactic constraints capture the rules of syntax in a certain programming language. For example, "*Each assignment must contain a valid expression on the right-hand side."* The semantic constraints contain the encoded logical component of the solution. An example is, *"If the problem requires a function to be applied to a range of values, the solution must contain a loop."*

This work differs from previous works utilizing CBM, emphasizing conversational tutoring through a question-answer approach. The evaluative nature of constraints makes it well-suited for assessing students' responses to questions, providing an opportunity to gauge their understanding of the domain. This is why, when creating a problem, the constraint is used as a reference answer. This allows corresponding feedback to be given to align faulty student knowledge manifested in their incorrect responses.

3.2 Dialog Manager

To decide on what and how to respond to a student's answer to a question, the dialog manager controls the flow of the dialogue by prompting the LLM. There are four types of prompts: *a prompt for assessing a student's answer*, *a prompt to generate feedback for a correct answer*, *a prompt to produce feedback for an incorrect answer*, and a *prompt for generating the question for a partially correct answer*.

When the student responds to a question, the software tutor will prompt the LLM to compare the student's answer to the reference answer (which is a constraint). The question referred to was created during the authoring stage. It is designed to cover a constraint and should elicit an answer indicating whether it is entailed or violated. Given an example presented in Listing 1, although the student's answer is incomplete, this can be entailed from the constraint. The student answer checker works for this purpose, as discussed in Section 3.3. The answer is considered correct when the student's answer is an entailment. Thus, a prompt for producing positive feedback is executed. If the answer is incorrect, the LLM will be prompted to generate feedback addressing the misconception following the format presented in Listing 2. After an entailment or violation is detected, the dialog manager will prompt the LLM to retrieve the list of concepts not covered by the student's answer and generate a question focusing on the identified concept, thereby initiating a multi-turn conversation. This is further discussed in Section 3.4.

A pre-defined question, a constraint, and a student's answer are given as follows:

- **Question:** Why does the error occur?
- **Reference Answer (Constraint):** "A variable must be declared with a specific data type before it can be used".
- **Student's answer:** "The variable was not declared".

Listing 1. Example list of inputs in a tutoring session.

3.3 Student Answer Checker

When the dialog manager calls for assessment, a zero-shot prompting to LLM is initiated to check whether a student's answer is an entailment or a violation with respect to the constraint. For example, given the scenario in Listing 1, the student's answer is correct with respect to the question. The answer is in a negative statement form and entails a violation of the constraint. A prompt is written to check the student's answer using the format presented in Listing 2. The result of the assessment determines the next action to be taken.

- **Description**: Determine whether a student's answer is either an entailment (positive statement form), a violation (negative statement form), or an out-of-topic with respect to the constraint.
- **Input indicator:** Student answer: "Input text", Constraint: "input text",
- **Output indicator:** Assertion, Classification, Reason

Listing 2. Prompt format for incorrect feedback.

3.4 Feedback for Correct and Incorrect Answers

Feedback is a mechanism used to repair students' errors or misconceptions when detected during reasoning in a dialogue. Aligned with the CBM approach, the format used by popular constraint-based tutors was adopted because it has been reported to help learners. The approach suggested that when an error or misconception is detected, effective feedback should tell the student where the error is and what constitutes the error and re-iterate the domain principle violated by the student. For example, the feedback message might say, "*The error lies in the condition of the if-statement, and the comparison operator causes a logical error. Remember that the condition in an if-statement must evaluate to true for the block statement to be executed*". When the assessment is completely correct, positive feedback is given by first generating positive backchannel feedback (e.g., That's right!), followed by the same format described above.

3.5 Feedback for Partially Correct Answer

When positive feedback is generated after a correct answer, another prompt will be initiated to check whether the student's answer has covered all the information in the constraint. Feedback in the form of a question will be initiated to cover the missing information in a student's answer. For example, given an example described in Listing 1, although the student's answer can be entailed from the reference answer, it is incomplete because of the missing phrase "*with a specific data type*". To elicit this information, a sub-question will be automatically generated by prompting the LLM. The same process of checking the answer to this sub-question is used, as described in Section 3.3. When a student's answer is correct, the same feedback will be provided; otherwise, the system will generate a pump, and the loop continues. Positive feedback will be generated if the correct answer is provided after the pump. Otherwise, the system will provide the correct solution and end the dialogue.

It is possible to have multiple missing information in a student's answer. The tutor will tackle each of these, making a multi-turn conversation.

3.6 *Software Interface*

Figure 2 shows the interface of the system, which we call *DT4-Coding*. This view has three sections: the problem description, the conversation, and the code area. Teachers author the problems; thus, they must be predefined along with the main questions and the corresponding code. The tutor initiates the conversation by posing a question. The student then responds by typing an answer, which the tutor evaluates. Based on this assessment, the tutor provides appropriate feedback, as outlined in the previous sections.

In Graesser's tutoring frame, the fifth step is assessing the student's understanding after receiving help. In this case, an assessment is done through code modification to verify if students understand or learn from their previous mistakes. The student must have corrected the faulty knowledge about the domain by incorporating knowledge gained from feedback. Hence, the system must allow students to modify a given code. A log of all interactions will be

captured for each session, which will be used further to assess the system's impact on students' learning experiences.

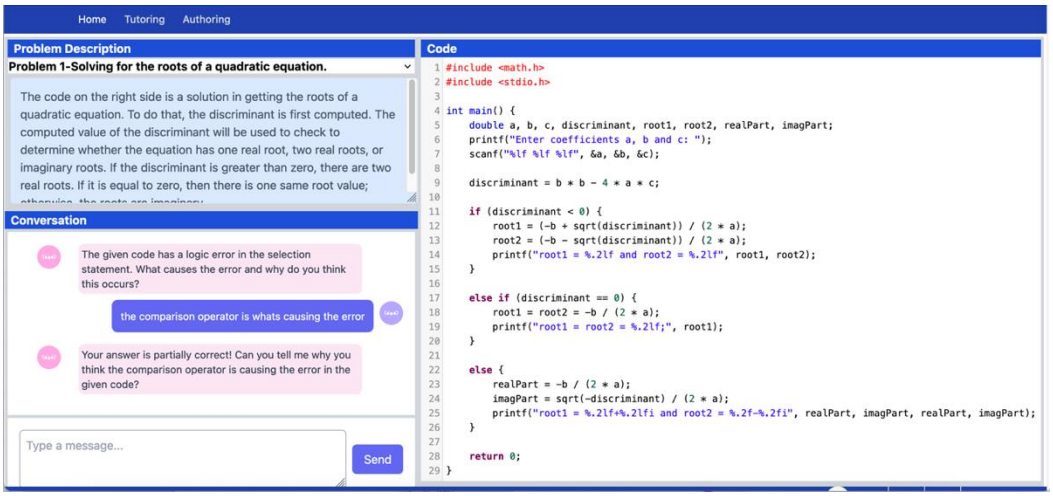

Figure 2. Interface of DT4-Coding

4. Further Work

Hallucination remains a significant problem in LLM-based applications, impacting the quality of both language understanding and generation tasks. To address this problem, the system needs to have a certain level of controllability. In the context of this study, the quality of the feedback generated depends on the quality of the student's answer being assessed. When utilizing zero-shot language understanding capabilities, LLMs must accurately break down entailments for each student's answer to ensure correct classification results. During the initial exploration, zero-shot prompting can yield acceptable results after several iterations and careful crafting of the prompts. However, this process requires manual effort and may lead to suboptimal solutions. Thus, it is important to utilize automatic prompting techniques like Automatic Chain of Thought (Auto-CoT) to achieve the best results for this intricate task. This will entail comparing other automatic prompting techniques to determine the most effective solution.

The *Retrieval-Augmented Generation (RAG)* technique influences the *constraint-augmented feedback generation* method. In retrieving information from the collected documents about selected tutoring topics, the human written constraint is used as a query to retrieve patterns from the LLM. Hence, a method for evaluating the quality of feedback must follow *RAG metrics*. In the context of RAG, the quality of LLM outputs relies on the quality of the two components: the *retriever* and the *generator*. In this study, the evaluation must include both the quality of the assessment output and the quality of the generated feedback given the constraint. The quality of feedback must be checked against the following criteria: *faithfulness*, *answer relevancy*, *contextual precision*, *contextual recall*, and *contextual relevancy*.

To properly understand the implications of this LLM-based software tutor, in addition to the direct assessment following the fifth step of the tutoring frame, an acid test will be conducted to assess further the potential benefits and impact of DT4-Coding on students' learning experiences. This will include identifying patterns emerging in students' usage of the software tutor, which can be observed in the dialogue history, and a semester-long field evaluation study involving students enrolled in introductory programming courses. Furthermore, participants will undergo a pre-test assessment to form the basis for dividing them into two groups. The control group will utilize traditional learning methods without the software tutor, while the experimental group will utilize the software tutor as their primary learning tool. While recognizing that various factors influence student performance, a comparative analysis of the final grade to assess learning outcomes can offer insights into the potential benefits of the software tutor.

References

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *Am. Psychol.*, *51*(4), 355–365. https://doi.org/10.1037/0003-066x.51.4.355

Ben-Ari, M. (1998). Constructivism in computer science education. *Technical Symposium on Computer Science Education*. https://api.semanticscholar.org/CorpusID:60449159

Chi, M. T. H., Leeuw, N. D., Chiu, M.-H., & Lavancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science*, *18*(3), 439–477. https://doi.org/10.1016/0364-0213(94)90016-7

Chowdhury, S. P., Zouhar, V., & Sachan, M. (2024). *Scaling the authoring of AutoTutors with Large Language Models*.

Graesser, A. C., Person, N., & Harter, D. (n.d.). *Teaching Tactics in AutoTutor*.

Hellas, A., Leinonen, J., Sarsa, S., Koutcheme, C., Kujanpää, L., & Sorva, J. (2023). Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, 93–105. https://doi.org/10.1145/3568813.3600139

Holland, J., Mitrovic, A., & Martin, B. (2009). *J-Latte: A Constraint-Based Tutor for Java*.

Kiesler, N., Lohr, D., & Keuning, H. (2023). *Exploring the Potential of Large Language Models to Generate Formative Programming Feedback*. https://doi.org/10.48550/arXiv.2309.00029

Lane, H. C., & VanLehn, K. (2004). A Dialogue-Based Tutoring System for Beginning Programming. *The Florida AI Research Society*. https://api.semanticscholar.org/CorpusID:262370016

Liffiton, M., Sheese, B., Savelka, J., & Denny, P. (2023). *CodeHelp: Using large language models with guardrails for scalable support in programming classes*.

Matelsky, J. K., Parodi, F., Liu, T., Lange, R. D., & Kording, K. P. (2023). *A large language model-assisted education tool to provide feedback on open-ended responses*.

Nye, B., Graesser, A., & Hu, X. (2014). AutoTutor and Family: A Review of 17 Years of Natural Language Tutoring. *International Journal of Artificial Intelligence in Education*, *24*. https://doi.org/10.1007/s40593-014-0029-5

Ohlsson, S. (1996). Learning from performance errors. *Psychol. Rev.*, *103*(2), 241–262. https://doi.org/10.1037/0033-295x.103.2.241

Ohlsson, S. (2016). Constraint-Based Modeling: From Cognitive Theory to Computer Tutoring – and Back Again. *International Journal of Artificial Intelligence in Education*, *26*(1), 457–473. https://doi.org/10.1007/s40593-015-0075-7

Prather, J., Denny, P., Leinonen, J., Smith IV, D. H., Reeves, B. N., MacNeil, S., Becker, B. A., Luxton-Reilly, A., Amarouche, T., & Kimmel, B. (2024). Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models. *ArXiv*, *1*.

Reeves, B., Sarsa, S., Prather, J., Denny, P., Becker, B. A., Hellas, A., Kimmel, B., Powell, G., & Leinonen, J. (2023, June 29). *Evaluating the performance of code generation models for solving parsons problems with small prompt variations*. Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, New York, NY, USA. https://doi.org/10.1145/3587102.3588805

Stamper, J., Xiao, R., & Hou, X. (2024). *Enhancing LLM-based feedback: Insights from Intelligent Tutoring Systems and the learning sciences*.

VanLEHN, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educ. Psychol.*, *46*(4), 197–221. https://doi.org/10.1080/00461520.2011.611369

Weerasinghe, A., & Mitrovic, A. (2006). Facilitating deep learning through self-explanation in an open-ended domain. *KES Journal*, *10*, 3–19. https://doi.org/10.3233/KES-2006-10101

Ym, P., Ganesan, V., Arumugam, D. K., Gupta, M., Shadagopan, N., Dixit, T., Segal, S., Kumar, P., Jain, M., & Rajamani, S. (2023). *PwR: Exploring the role of representations in conversational programming*.

Zhu, Z., Cheng, X., An, H., Wang, Z., Chen, D., & Huang, Z. (2024). Zero-Shot Spoken Language Understanding via Large Language Models: A Preliminary Study. In N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, & N. Xue (Eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)* (pp. 17877–17883). ELRA and ICCL. https://aclanthology.org/2024.lrec-main.1554