Automated Feedback on Student-Generated UML and ER Diagrams Using Large Language Models

Sebastian Gürtl
sebastian.guertl@tugraz.at
Graz University of Technology
Graz, Austria

Gloria Schimetta
gloria.schimetta@student.tugraz.at
Graz University of Technology
Graz, Austria

David Kerschbaumer
david.kerschbaumer@tugraz.at
Graz University of Technology
Graz, Austria

Michael Liut
michael.liut@utoronto.ca
University of Toronto Mississauga
Mississauga, Canada

Alexander Steinmaurer
alexander.steinmaurer@it-u.at
Interdisciplinary Transformation
University Austria
Linz, Austria

Abstract

UML and ER diagrams are foundational in computer science education but come with challenges for learners due to the need for abstract thinking, contextual understanding, and mastery of both syntax and semantics. These complexities are difficult to address through traditional teaching methods, which often struggle to provide scalable, personalized feedback, especially in large classes. We introduce DUET (Diagrammatic UML & ER Tutor), a prototype of an LLM-based tool, which converts a reference diagram and a student-submitted diagram into a textual representation and provides structured feedback based on the differences. It uses a multi-stage LLM pipeline to compare diagrams and generate reflective feedback. Furthermore, the tool enables analytical insights for educators, aiming to foster self-directed learning and inform instructional strategies. We evaluated DUET through semi-structured interviews with six participants, including two educators and four teaching assistants. They identified strengths such as accessibility, scalability, and learning support alongside limitations, including reliability and potential misuse. Participants also suggested potential improvements, such as bulk upload functionality and interactive clarification features. DUET presents a promising direction for integrating LLMs into modeling education and offers a foundation for future classroom integration and empirical evaluation.

CCS Concepts

• **Applied computing → Computer-assisted instruction**; • **Social and professional topics → Computing education**; • **Human-centered computing → Interactive systems and tools**.

Keywords

Large Language Models, Automated Feedback, Unified Modeling Language, Entity-Relationship Diagram, CS Education, Student Support

1 Introduction

Software modeling is an integral part of software development, system design, or business process modeling. It requires skills in system analysis, abstraction, and problem solving, but also knowledge of modeling languages. Two of these widely used notations are the *Unified Modeling Language* (UML) and *Entity Relationship* (ER) diagrams.

Although UML is treated as standard for software engineering, its application in the computer science (CS) industry has dwindled and only selected components, such as class and sequence diagrams being used for high-level conceptualization, have remained [11, 14].

Despite this lack of use in the industry, UML is believed to be beneficial in CS education, as its concepts and notation provide a helpful framework for novice students [11]. However, learners often struggle with the complexity of UML and frequently make mistakes in notation, scope, and diagram interpretation [2, 5, 9, 12]. Ideally, these challenges require guidance and feedback from educators and teaching assistants (TAs) while students learn.

Traditionally, undergraduate CS classes are characterized by a large number of students, which stands in contrast to the need for individual feedback. For this reason, feedback and assessment often rely on rigid, rule-based methods that lack personalized guidance. Large Language Models (LLMs), on the other hand, excel in natural language understanding, pattern recognition, and adaptive feedback generation, making them well-suited for analyzing student diagrams and offering targeted suggestions [13]. In addition, LLMs can engage in interactive dialogues to clarify design goals and constraints, ultimately improving both productivity and design quality [1, 7]. The growing popularity of prompt engineering further enhances the potential of LLMs, leading to more effective and tailored results in software design feedback [15].

To address this, we propose an LLM-based tool that transforms UML and ER diagrams into PlantUML – a tool that textually represents diagrams. The tool subsequently analyzes these diagrams by comparing them against a reference solution provided by the instructor. Based on this, it generates hints and improvement suggestions for the students while the instructors can view common issues and teaching recommendations.

2 Background and Related Work

2.1 UML and ER Diagrams in CS Education

While practitioners in the industry have mostly abandoned UML, it is still highly relevant in introductory CS and software engineering classes [11]. The process of visualizing system architecture and data structures is challenging for novice software engineers. UML and ER diagrams enforce the principles of software abstraction and guide developers in structuring complex systems. Students, however, struggle with the complexity of UML and ER notation, leading to frequent mistakes [2]. These mistakes have been extensively discussed in literature [5, 12], with the following being prominent in UML diagrams:

- General Difficulties: Diagram construction and symbol use.
- Class Diagram: Attributes, inheritance confusion.
- Sequence Diagram: Lifeline naming, object interaction.

The most common issues of ER diagrams include:

- Inconsistency with the class diagram.
- Missing or redundant relationships.
- Wrong multiplicities.

Due to the variety of tools available, students can take different approaches to creating a UML or ER diagram, including textual representations.

2.2 PlantUML

PlantUML[1] is a popular tool that allows the creation of various types of diagrams, including most UML diagrams, as well as ER diagrams. Romeo et al. [14] analyzed 13,000 open-source repositories on GitHub and found that PlantUML has been the most used text-based UML tool since UML's resurgence in 2016. Since LLMs still lack the ability to extract high-detail information from images effectively, text representations of UML diagrams, such as PlantUML, can be more accurately processed and analyzed.

2.3 LLMs for UML and ER Diagram Generation

LLMs are trained on vast datasets for various tasks. Given the advantages of text-based UML representations, recent research has explored how LLMs can assist in generating, interpreting, and optimizing PlantUML diagrams. While not explicitly trained for diagram generation, they can still assist students in UML modeling by providing examples and naming suggestions. However, LLMs have inherent limitations as shown in various studies [4, 6, 16, 17].

Wang et al. [16] conducted a study analyzing UML models created by students who used OpenAI's ChatGPT as a support tool for their modeling tasks. The study found that it correctly identified 66% of classes, 75% of operations, and 91% of attributes. However, it struggled with class relationships, achieving only 25% accuracy. LLMs, particularly GPT-4, perform notably better with sequence diagrams, correctly identifying 74% of objects, 68% of messages, and maintaining the correct message order 82% of the time.

Multimodal LLMs still face challenges in analyzing diagrams, particularly UML and ER diagrams. In case studies by Conrardy et al. and Camara et al. [4, 6], GPT-4 significantly outperformed both open-source and commercial multimodal LLMs in converting hand-drawn UML diagrams to PlantUML. In one experiment, GPT-4 did not produce a single syntax error after 36 generations, while other models produced multiple. Nonetheless, all models, including GPT-4, made minor mistakes requiring human correction. GPT-4 outperformed all the other LLMs, often producing near-perfect results in a single iteration.

[1]https://plantuml.com/en/

Another study published by Wang et al. [17] compared GPT-4's UML grading to that of a human instructor. Its assessments matched the instructor's in approximately 50% of cases. Discrepancies were attributed to *misunderstanding* (e.g., the grading schema), *overstrictness* (e.g., rejecting similar valid alternatives), and *wrong identification* (e.g., GPT-4 failed to recognize a relationship). The study also confirmed that sequence diagrams are less prone to errors in GPT-4's assessments. Across 40 UML diagrams (40 each), `GPT-4o` correctly evaluated approximately 35 class and sequence diagrams, making three incorrect identifications for class diagrams but none for sequence diagrams.

As seen in the studies above, LLMs show promise in analyzing and generating UML and ER diagrams. However, automated evaluation tools (even without the use of LLMs) can provide additional structured feedback that can further support students in their learning process.

2.4 Automated Evaluation of UML and ER Diagrams

To address the challenges students have experienced while working with modeling languages, several evaluation tools have been developed to assist with the learning process.

UML Miner [3] applies process mining to study student behavior in Visual Paradigm[2], performing conformance checking against expected behavior and similar solutions. This approach, however, relies on manual feedback from educators, and the students do not receive direct feedback from the tool.

UML Mentor [8] introduces a peer-review system where students create UML diagrams for software design challenges and receive feedback from peers. While it eases instructor workload, peer-feedback may lack expertise and is limited to predefined tasks.

UMLegend [10] incorporates gamification, creating a sandbox environment for students to solve provided exercises. Live feedback notifies the student of errors. This technique proved to increase diagram correctness 11% in an experiment. UMLegend, however, limits students in the creation of the diagram – hand-drawn diagrams as well as diagrams drawn in other tools cannot be evaluated. The evaluation engine is restricted in its assessment capabilities, as it relies on a single reference solution for comparison.

In summary, despite UML's declining industry use, it remains an essential educational tool. Students often struggle with UML and ER diagrams, necessitating automated evaluation solutions. While LLMs, particularly GPT-4, demonstrate promising results in analyzing and generating UML and ER diagrams, existing evaluation tools demonstrate that human oversight remains essential in the near future to guide LLM-based assessments. Meanwhile, tools that do not leverage LLMs provide alternative feedback approaches, offering valuable insights for alternative implementations.

3 Diagrammatic UML & ER Tutor

Building on the motivation, challenges, and related work, this section introduces the proposed automated feedback tool *Diagrammatic UML & ER Tutor* (DUET) designed to address educational gaps. In particular, we present the tool's design, system architecture, workflow, and user interaction. It utilizes advanced LLMs, such as

[2]https://www.visual-paradigm.com

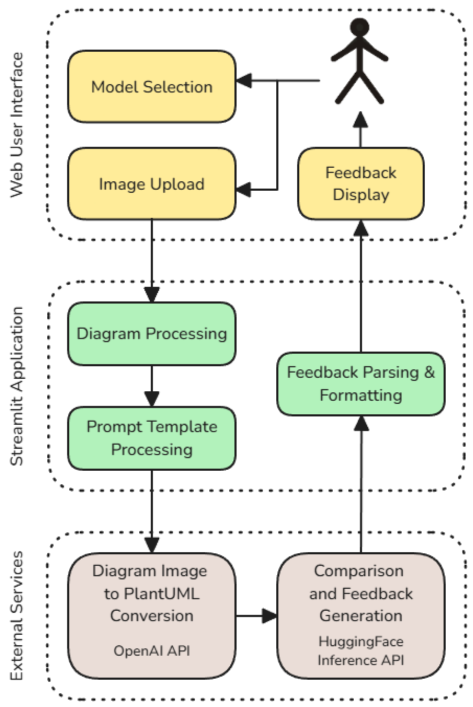

Web User Interface
Model Selection
Image Upload
Feedback Display
Streamlit Application
Diagram Processing
Prompt Template Processing
Feedback Parsing & Formatting
External Services
Diagram Image to PlantUML Conversion
OpenAI API
Comparison and Feedback Generation
HuggingFace Inference API

Figure 1: High-level system architecture of the automated UML and ER diagram feedback tool DUET. The web interface handles LLM selections and diagram uploads. GPT-4o converts diagram images into PlantUML code. A smaller LLM compares the representations and generates structured feedback for students and educators.

GPT-4o, and smaller complementary LLMs to provide meaningful automated feedback on student-generated UML and ER diagrams. Students upload self-drawn diagrams, which are compared with an instructor-based reference diagram. The source code of DUET and supplementary materials are available in a GitHub repository[3].

3.1 Architecture

Figure 1 depicts an overview of DUET's system architecture. The automated feedback tool is developed using Python and Streamlit[4], which provides a web-based graphical user interface. The service is hosted on HuggingFace Spaces[5].

External LLMs are integrated via APIs and form the backbone of the tool's pipeline. OpenAI's `GPT-4o` initially converts the uploaded diagrams into PlantUML code. Users authenticate their access by entering a personal OpenAI API key. Subsequently, a smaller LLM, such as `Mistral-7B-Instruct-v0.3`, analyzes and compares the PlantUML codes against an instructor-provided reference solution. The tool currently accesses this LLM through HuggingFace's inference API, authenticated by a personal API key. However, the architecture also allows easy adaptation to locally hosted LLMs via frameworks such as Ollama[6].

The system uses predefined prompt templates for UML and ER diagrams. This approach allows for easy adaptation of prompts without modifications to the underlying application logic.

3.2 Workflow and User Interaction

Building upon the system architecture, we illustrate the pipeline in Figure 2, showing how users can interact with DUET and how diagrams are processed.

Step 1: User Input and Diagram Upload. Users select a language model for diagram recognition and another model for comparison and feedback generation. Once the model preferences are set, two diagrams can be uploaded: one serving as the instructor-provided reference solution and one representing the student's solution. The tool accepts both hand-drawn and digitally created UML and ER diagrams. In the current version, only a single student solution can be uploaded at a time, as bulk uploads are not yet supported.

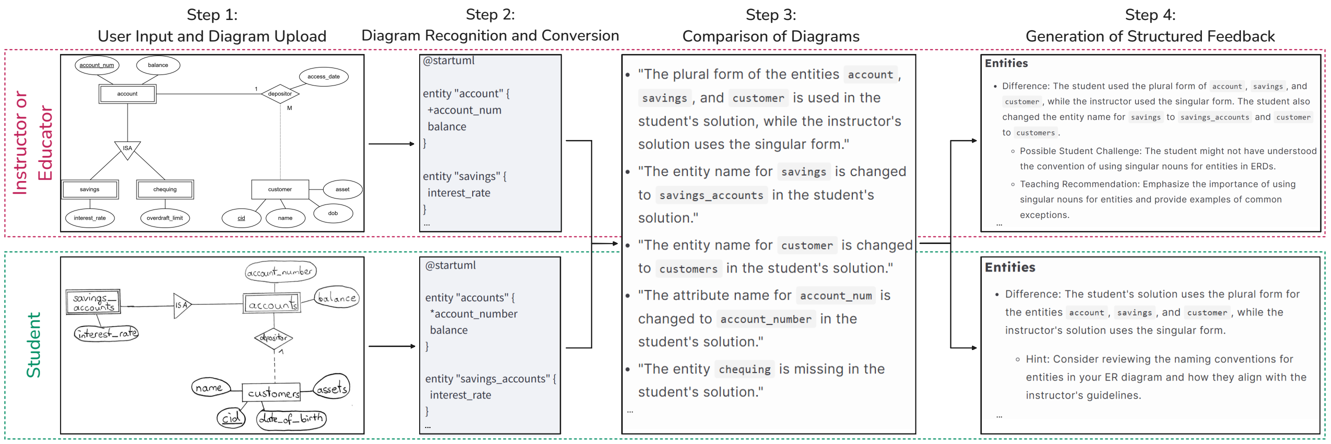
Once both diagrams are submitted, users are guided to the next step.

Step 2: Diagram Recognition and Conversion. The tool processes both uploaded diagrams to extract their structural representations. OpenAI's `GPT-4o` converts the diagram images into PlantUML code. This step involves recognizing and interpreting visual elements such as classes, attributes, operations, and relationships. Since the recognition process depends on an LLM, the quality of the extracted PlantUML code may vary based on image quality, diagram clarity, handwriting legibility (for hand-drawn diagrams), and adherence to standard notations. The generated PlantUML code is the basis for the subsequent comparison and feedback generation.

Step 3: Comparison of Diagrams. After the conversion step, the tool compares the student's version against the instructor's reference solution. In this step, a smaller LLM, such as `Mistral-7B`, analyzes both representations to identify structural differences. The comparison focuses on objective elements such as missing or additional classes or entities, modified attributes, relationships, or changes in visibility. However, this step does not include any judgment or adjustments – rather, it generates a list of differences that serves as the foundation for feedback generation in the next step.

Step 4: Generation of Structured Feedback. Based on the identified differences, the tool generates two types of structured feedback, again using a smaller LLM. The first type is directed at students and consists of reflective, neutral hints that encourage them to revisit specific aspects of their diagram without explicitly labeling elements as correct or incorrect. The second type targets educators and offers insights into potential student misunderstandings, misconceptions, and possible instructional implications. Both types of feedback are organized into predefined categories such as classes, attributes, and relationships. The generated feedback is displayed in the web interface as structured Markdown text, organized by predefined categories.

[3]https://github.com/sguertl/learnersourcing2025-duet
[4]https://streamlit.io/
[5]https://huggingface.co/spaces
[6]https://ollama.com/



```
@startuml

entity "account" {
 +account_num
 balance
}

entity "savings" {
 interest_rate
}
...
```

```
@startuml

entity "accounts" {
 *account_number
 balance
}

entity "savings_accounts" {
 interest_rate
}
...
```

- "The plural form of the entities `account`, `savings`, and `customer` is used in the student's solution, while the instructor's solution uses the singular form."
- "The entity name for `savings` is changed to `savings_accounts` in the student's solution."
- "The entity name for `customer` is changed to `customers` in the student's solution."
- "The attribute name for `account_num` is changed to `account_number` in the student's solution."
- "The entity `chequing` is missing in the student's solution."

...

Entities

- Difference: The student used the plural form of `account`, `savings`, and `customer`, while the instructor used the singular form. The student also changed the entity name for `savings` to `savings_accounts` and `customer` to `customers`.
  - Possible Student Challenge: The student might not have understood the convention of using singular nouns for entities in ERDs.
  - Teaching Recommendation: Emphasize the importance of using singular nouns for entities and provide examples of common exceptions.

...

Entities

- Difference: The student's solution uses the plural form for the entities `account`, `savings`, and `customer`, while the instructor's solution uses the singular form.
  - Hint: Consider reviewing the naming conventions for entities in your ER diagram and how they align with the instructor's guidelines.

...

Figure 2: Illustration of DUET's workflow using ER diagrams, showing the process from diagram upload and conversion to PlantUML to comparison and structured feedback generation.

The current version of DUET is intended as a functional prototype for initial pilot testing in educational settings. It provides a foundation for iterative refinement and further adaptation based on feedback from both students and educators.

4 Evaluation

4.1 Methods

We conducted semi-structured interviews with six participants actively engaged in teaching courses frequently using UML and ER diagrams in the winter semester of 2024. Two participants are researchers in computer science education and university lecturers. We refer to them as L1 and L2. L1 teaches a third-year *Introduction to Databases* course for undergraduate computing majors and minors ($N = 300$) at a research-focused North American university, while L2 teaches a second-semester *Introduction to Object-Oriented Programming* course for undergraduate computing majors ($N = 500$) at a research-focused European university. Furthermore, interviews were conducted with four senior graduate student TAs, with two TAs affiliated with each educator. These participants are referred to as TA1–TA4.

Each interview session took approximately 30 minutes and began with an introduction to the tool and a demonstration of its workflow, detailed in Section 3.2. Subsequently, a discussion was held based on the questions in Table 1.

Two interviewers recorded notes in bullet-point format for each question and participants verified the notes for accuracy. The collected interview notes were anonymized. We followed the thematic analysis approach, whereby segments of the notes were labeled to capture their essence, grouped into themes, and summarized. All data can be found in the project's GitHub repository.

4.2 Evaluation Results

Four major categories emerged from the participants' responses in the summarized interview data.

Benefits. The six participants emphasized the tool's ability to provide instant and iterative feedback as a personalized, real-time learning aid, allowing students to correct errors and improve their understanding. L2 highlighted that "*students can use the system whenever they want and are not limited by the availability of TAs or lectures*". Compared to traditional fixed-time lectures or labs, students can work on their assignments when it fits their schedule and receive immediate support. Its availability and scalability offer easy adoption to large-scale courses without increasing the workload for tutors. TA1 and TA2 even saw the potential of reducing the workload for TAs and increasing fairness by enabling faster assessment using the tool's standardized feedback.

Limitations. L1, TA1, TA3, and TA4 reported concerns about the accuracy and reliability of the tool. Specifically, the tool's non-deterministic output, resulting in variable and potentially inaccurate feedback across executions, can potentially lead to student confusion and avoidance of using the tool. Two participants (TA3, TA4) identified a possible way to cheat by generating solutions rather than elaborating on them. The possibility of tool misuse must be considered when implementing it in classroom settings.

Table 1: Questions asked at the evaluation interviews to six participants to identify strengths and weaknesses of DUET.

Q#	Question
Q1	List and describe potential use cases for this tool?
Q2	What features do you believe need to be refined?
Q3	What features do you believe need to be added?
Q4	In terms of using this tool in an introduction to databases or introduction to software engineering course, so think of an educational setting, what are 2-3 *pros* of this tool?
Q5	In terms of using this tool in an introduction to databases or introduction to software engineering course, so think of an educational setting, what are 2-3 *cons* of this tool?
Q6	Is there anything else you want to add or say?

Use Cases in Learning. Instant feedback and an arbitrary number of interactions make the tool an effective learning asset. However, TA3 raised concerns about using it for assessment because of its non-deterministic output: "*Learners and educators can never be sure about the correctness, which makes it not suitable for assessment*". To mitigate this concern, the tool should not be used to assess students' submissions automatically but rather as a learning tool to improve and learn interactively, with the information that results might differ for each run. L2, TA3, and TA4 emphasized that the tool should not replace TAs but offer another possibility to practice when no TA is available.

Additional Feature Ideas. In addition to the benefits of the system for students, five participants (L1, L2, TA1, TA2, TA4) suggested that educators could gain additional insights into the learning process. L1 expressed: "*The tool could store all submission data to improve understanding of misconceptions through aggregated student submissions*". Teaching methods could be adopted, and common errors could be explicitly discussed. Furthermore, TA3 and TA4 proposed a more interactive component, allowing students to ask questions after receiving feedback and to resolve misconceptions or misinterpretations.

5 Conclusion

In this paper, we explore the use of LLMs to provide structured and automated feedback on UML and ER diagrams from the perspective of introductory software engineering and databases courses. The approach leverages LLM-based analysis to identify differences between student-created diagrams and instructors' reference solutions. Consequently, it enables reflective learning and targeted instructional interventions. To demonstrate this approach, we developed DUET, a prototype tool that applies LLMs to the analysis of student and instructor diagrams. The system integrates sophisticated models such as OpenAI's `GPT-4o` to convert diagram images into PlantUML code, and smaller models such as `Mistral-7B` to identify structural differences and generate feedback presented separately for students and educators. Preliminary insights from semi-structured interviews with two educators and four TAs highlight initial use cases, perceived advantages, and potential limitations in applying this approach.

The use of LLMs to generate structured, formative feedback has the potential to support reflective learning and iterative improvement in introductory modeling courses in computing disciplines. As emphasized by the interviewees, constructive feedback could help students reconsider their design decisions without judging them as correct or incorrect, which could encourage deeper engagement with core concepts. In addition, immediate feedback may reduce students' reliance on instructor availability and enable them to receive and refine their work more independently. However, this approach and the tool are not intended to replace instructors or TAs, but rather serve as a supplementary component that supports and enhances existing teaching practices. For educators, this approach offers an opportunity to identify common misconceptions in student submissions, which are often challenging to detect systematically through manual review, and to inform targeted instructional interventions. In particular, in large-scale courses, automated feedback could help scale formative assessment while reducing instructor workload. For example, students might complete a supportive self-check with DUET before submitting their final diagram. This would allow students to reflect on possible structural issues and revise their work without interacting with instructors and thus depend on their availability. Alternatively, instructors could encourage students to use the tool during a preparatory activity before lab or tutorial sessions. This may help students arrive with more refined questions and a clearer understanding of their design choices. In addition, DUET may serve as a starting point for peer learning activities, where students collaboratively discuss the feedback received and subsequently compare alternative design strategies.

The results raise concerns about the accuracy and interpretability of LLM-generated feedback, particularly when dealing with complex or ambiguous diagrams. Since the feedback generation process remains largely a black box, it may be difficult for students and educators to fully understand or evaluate the reasoning behind the tool's hints and suggestions. The interviewees also noted the risk that students might rely too heavily on automated feedback and subsequent guidance, which could hinder the development of their modeling skills. Without opportunities to engage in problem-solving and self-correction, students may become passive recipients of feedback rather than actively developing their conceptual understanding and design reasoning. From a methodological perspective, the study involved a small number of participants and hypothetical scenarios, limiting the generalizability of the findings to real classroom contexts.

Future work will focus on improving DUET and deploying this approach in actual educational situations to evaluate its impact on student learning, instructional practices, and feedback effectiveness. Further development of the tool will explore improved system and user prompts, feedback presentation, and support for multiple student submissions and reusable reference solutions. Additionally, measures will be implemented to reduce the risk of misuse, such as using the tool for solution generation.

6 Acknowledgments

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery Grant #RGPIN-2024-04348, for their financial support.

References

[1] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards human-bot collaborative software architecting with chatgpt. In *Proceedings of the 27th international conference on evaluation and assessment in software engineering*. 279–285.

[2] Seiko Akayama, Birgit Demuth, Timothy C Lethbridge, Marion Scholz, Perdita Stevens, and Dave R Stikkolorum. 2013. Tool use in software modelling education. In *Proceedings of the Educators' Symposium co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*.

[3] Pasquale Ardimento, Lerina Aversano, Mario Luca Bernardi, Vito Alessandro Carella, Marta Cimitile, and Michele Scalera. 2023. UML Miner: A Tool for Mining UML Diagrams. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 30–34. doi:10.1109/MODELS-C59198.2023.00014

[4] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling* 22, 3 (2023), 781–793.

[5] Stanislav Chren, Barbora Buhnova, Martin Macak, Lukas Daubner, and Bruno Rossi. 2019. Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course. In *2019 IEEE/ACM 41st International Conference on*

Software Engineering: Software Engineering Education and Training (ICSE-SEET). 100–109. doi:10.1109/ICSE-SEET.2019.00019

[6] Aaron Conrardy and Jordi Cabot. 2024. From image to uml: first results of image based uml diagram generation using llms. *arXiv preprint arXiv:2404.11376* (2024).

[7] Gabriele De Vito, Fabio Palomba, Carmine Gravino, Sergio Di Martino, and Filomena Ferrucci. 2023. Echo: An approach to enhance use case quality exploiting large language models. In *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 53–60.

[8] Rutwa Engineer, Volodymyr Yaremchuk, Eren Suner, Omar Khamis, Alex Apostolu, and Arthur Ng. 2024. UML Mentor: A Tool for Interactive and Collaborative Software Design Education. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 2* (Virtual Event, NC, USA) *(SIGCSE Virtual 2024)*. Association for Computing Machinery, New York, NY, USA, 309. doi:10.1145/3649409.3691078

[9] John Erickson and Keng Siau. 2007. Can UML be simplified? Practitioner use of UML in separate domains. (2007).

[10] Giacomo Garaccione, Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2025. Gamification of conceptual modeling education: an analysis of productivity and students' perception. *Software Quality Journal* 33, 1 (2025), 1–19.

[11] Marian Petre. 2013. UML in practice. In *2013 35th International Conference on Software Engineering (ICSE)*. 722–731. doi:10.1109/ICSE.2013.6606618

[12] Rebecca Reuter, Theresa Stark, Yvonne Sedelmaier, Dieter Landes, Jürgen Mottok, and Christian Wolff. 2020. Insights in Students' Problems during UML Modeling. In *2020 IEEE Global Engineering Education Conference (EDUCON)*. 592–600. doi:10.1109/EDUCON45650.2020.9125110

[13] Elijah Rivera, Alexander Steinmaurer, Kathi Fisler, and Shriram Krishnamurthi. 2024. Iterative Student Program Planning using Transformer-Driven Feedback. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 45–51. doi:10.1145/3649217.3653607

[14] Joseph Romeo, Marco Raglianti, Nagy Csaba, and Michele Lanza. 2025. UML is back. Or is it?. In *ICSE 2025 47th International Conference on Software Engineering*.

[15] Cigdem Sengul, Rumyana Neykova, and Giuseppe Destefanis. 2024. Software engineering education in the era of conversational AI: current trends and future directions. *Frontiers in Artificial Intelligence* 7 (2024), 1436350.

[16] Beian Wang, Chong Wang, Peng Liang, Bing Li, and Cheng Zeng. 2024. How LLMs aid in UML modeling: an exploratory study with novice analysts. In *2024 IEEE International Conference on Software Services Engineering (SSE)*. IEEE, 249–257.

[17] Chong Wang, Beian Wang, Peng Liang, and Jie Liang. 2024. Assessing UML Models by ChatGPT: Implications for Education. *arXiv preprint arXiv:2412.17200* (2024).