# Improving LLM-Generated Educational Content: A Case Study on Prototyping, Prompt Engineering, and Evaluating a Tool for Generating Programming Problems for Data Science

Jiaen Yu*
University of California San Diego
La Jolla, USA
jiy037@ucsd.edu

Ylesia Wu*
University of California San Diego
La Jolla, USA
xw001@ucsd.edu

Gabriel Cha
University of California San Diego
La Jolla, USA
gcha@ucsd.edu

Ayush Shah
University of California San Diego
La Jolla, USA
ajshah@ucsd.edu

Sam Lau
University of California San Diego
La Jolla, USA
lau@ucsd.edu

## Abstract

One key challenge for instructors is creating high-quality educational content, such as programming practice questions for introductory programming courses. While Large Language Models (LLMs) show promise for this task, their output quality can be inconsistent, and it is often unclear how to systematically improve their performance. In this experience report, we present the development process for ContentGen, an open-source tool that generates programming questions within the context of data science instructional materials. We describe our process of designing the tool and iteratively improving the tool through prompt engineering. To evaluate our changes, we designed and open-sourced a dataset of 91 test cases based on our course materials and developed three metrics to assess the generated questions: Correctness, Contextual Fit, and Coherence. We compare three prompting strategies and find that providing detailed instructions and an automatically generated summary of recently covered instructional materials to the LLM substantially improves the quality of the generated questions across our metrics. A usability study with six data science instructors further suggests that our final prototype is perceived as usable and effective. Our work contributes a case study of evidence-based prompt engineering for an educational tool and offers a practical approach for instructors and tool designers to evaluate and enhance LLM-based content generation.

## CCS Concepts

• **Human-centered computing → Empirical studies in HCI**.

## Keywords

Large Language Models, Prompt Engineering, LLM Tool Evaluation, Data Science

*Jiaen Yu and Ylesia Wu contributed equally to this work as co-first authors.

**ACM Reference Format:**
Jiaen Yu, Ylesia Wu, Gabriel Cha, Ayush Shah, and Sam Lau. 2026. Improving LLM-Generated Educational Content: A Case Study on Prototyping, Prompt Engineering, and Evaluating a Tool for Generating Programming Problems for Data Science. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2026), February 18–21, 2026, St. Louis, MO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770762.3772619

## 1 Introduction

One of the main challenges for instructors is creating and revising educational content, such as practice questions for programming courses. For programming learners, one common type of practice question asks students to write short snippets of code to practice basic skills. Recent advances in Large Language Models (LLMs) have shown promising results for generating this type of content [24]. However, LLM-generated content varies in quality; while LLMs can sometimes generate useful material, they often produce code with subtle errors, hallucinations, or questions that are pedagogically weak. This inconsistency prevents instructors from trusting LLM-based tools because of the additional work required to verify and fix the generated content.

One common strategy for improving LLM outputs is prompt engineering, the process of designing and refining inputs (prompts) to elicit desired outputs [23, 30]. However, it is often unclear how to systematically evaluate whether changes to a prompt actually lead to better results, especially in an educational context – what does "better" mean when an LLM generates a practice problem? To address this, we present an experience report on the design, implementation, and iterative improvement of ContentGen, a prototype tool that generates simple programming questions within an instructor's existing workflow. As data science instructors ourselves, we built ContentGen as a JupyterLab extension that generates practice problems based on existing instructional material. However, our initial version of the tool performed poorly, which motivated us to use an evidence-based approach to improving it.

To guide our development, we constructed a dataset of 91 examples from data science courses we teach. We then developed three binary metrics tailored to our pedagogical goals: Correctness (is the solution code valid?), Contextual Fit (does the question use the lecture's context?), and Coherence (does the question assess the relevant topic?). Using this evaluation framework, we compared three different prompting strategies: Baseline (generic instructions with entire notebook as context), Detailed (step-by-step instructions with variable information), and Structured (detailed instructions plus an automatically generated summary of the lecture content organized by pedagogical topics). We found that the Structured prompt substantially improved the quality of generated questions across all three metrics compared to the Baseline prompt. A usability study with data science instructors further suggests that our final prototype is perceived as useful and effective.

This paper contributes a case study of evidence-based prompt engineering for an educational tool. We offer our process as a practical model for how instructors and tool designers can prototype, evaluate, and iteratively enhance LLM-based content generation tools. Our contributions are:

(1) The design and implementation of ContentGen, an open-source tool for generating programming questions within JupyterLab.
(2) A dataset of course materials from two data science courses, along with our complete evaluation results, to enable replication and future work.
(3) A case study of our iterative prompt engineering process, demonstrating how a lightweight quantitative evaluation can lead to meaningful improvements in an LLM-based tool for education.

## 2 Related Work

This section reviews prior work on generating and evaluating educational content with LLMs. Our work is situated within the broader context of programming education, which is rapidly evolving with the introduction of LLMs. These tools are changing student behaviors [10] and raising pedagogical concerns about cognitive offloading and skill development [15, 26]. In response, instructors are currently adjusting their course materials and policies in various ways [32, 33], including adapting to the use of LLMs [18], and are also exploring methods of using LLMs productively to produce educational material [22].

*Generating Educational Content with LLMs.* Recent advancements in LLMs have opened potential applications for generating educational materials [11, 24]. However, studies indicate that the quality of LLM-generated content can be inconsistent. Researchers have found that questions produced by LLMs in domains like machine learning and biology are often too simple or generic [7]. In other cases, LLM-generated questions fail to capture the nuances of complex topics like program execution [8], or have difficulty creating effective distractors for multiple-choice questions [27]. A systematic review of automatic question generation also found a historical focus on fact-based questions over higher-order thinking [16].

To address these shortcomings, researchers have proposed techniques to improve the quality and relevance of generated content. These include creating human-in-the-loop systems [3, 21] and improving the generation process itself by incorporating principles from educational psychology [29], focusing on problem-solving steps [28], planning the question structure in advance [19], and personalizing content to be more contextually relevant [20]. This body of work highlights a key challenge: while automated content generation is feasible, ensuring its pedagogical value requires careful design. This paper extends prior work by presenting, to our knowledge, the first real-world case study of improving LLM-generated content for introductory data science programming.

*Evaluating LLM-Generated Content.* Since LLMs are both capable yet imperfect, researchers have explored many methods of evaluating them [1]. In programming, benchmarks have been developed to assess code generation on tasks ranging from functional correctness [2] to real-world software engineering problems [12, 13].

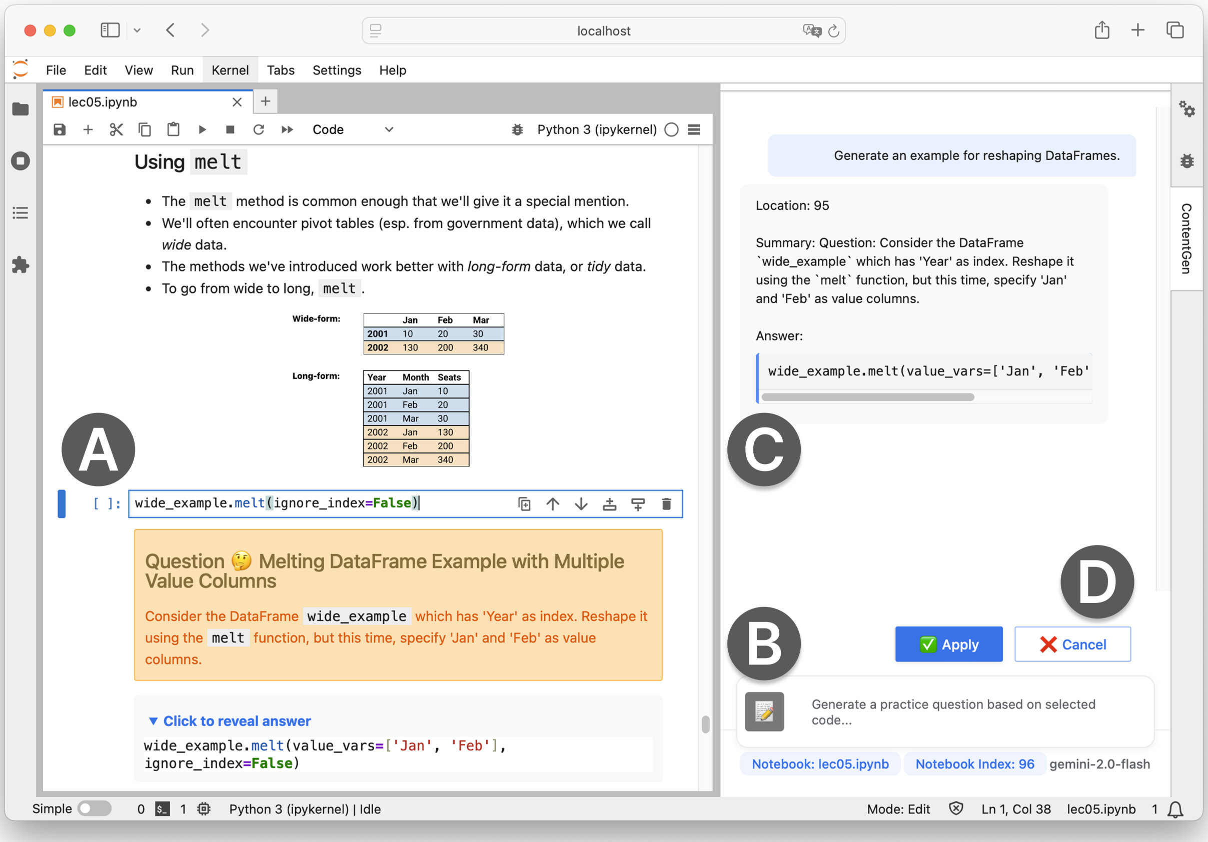
Beyond code generation, a growing body of work evaluates the pedagogical quality of LLM-generated educational content for programming education. A common finding is that while AI-generated materials are often factually correct, they may lack the richness of human-authored content [5] or contain subtle errors [6, 14], highlighting the need for robust evaluation methods for LLM-generated content. In this paper, we extend to metrics beyond just correctness in order to better align LLM outputs with instructors' pedagogical goals.

While past work documents flaws in LLM-generated content, programming and data science instructors still lack a clear process to reduce the frequency and impact of these flaws. To address this gap, we present a detailed case study of the iterative design, prompt engineering, and evaluation of an LLM-based tool for generating educational content, offering a practical model for other instructors to systematically improve LLM-generated materials.

## 3 Instructional Setting and Design Goals

As instructors of programming-heavy data science courses, we were inspired by the potential of LLM tools to generate programming exercises [11, 24] and sought to apply these techniques in our own teaching. We primarily teach first- and early second-year data science students at a large, public, PhD-granting university in North America. These students learn Python basics and then use the pandas library to manipulate data tables for exploratory data analysis and data cleaning. In our courses, we introduce Python and pandas fundamentals with "finger exercises": short programming problems presented during lectures and office hours. These exercises are close variants of material students have already seen in lecture, designed to build confidence before they tackle more challenging assignment problems.

Our initial attempts to generate these exercises using off-the-shelf LLM tools like the ChatGPT and Claude web apps revealed several challenges. First, providing the necessary context to these tools was cumbersome. Our instructional materials are primarily in Jupyter notebooks [17] which often focus on one or two specific datasets. It is pedagogically important for generated questions to reuse these datasets to avoid the cognitive overhead of students having to understand a new domain. However, this required manually providing information about the dataset in each prompt. Second, it was difficult to persuade the LLM to adhere to our course-specific programming idioms. For example, in one of our courses, we aim to ease the learning curve of `pandas` by teaching novices to use more verbose methods like `df.get('salary')` instead of the more standard `df['salary']`. However, we found it difficult to convince the



Using `melt`

- The `melt` method is common enough that we'll give it a special mention.
- We'll often encounter pivot tables (esp. from government data), which we call *wide* data.
- The methods we've introduced work better with *long-form* data, or *tidy* data.
- To go from wide to long, `melt`.

Wide-form:

| | Jan | Feb | Mar |
|---|---|---|---|
| **2001** | 10 | 20 | 30 |
| **2002** | 130 | 200 | 340 |

Long-form:

| Year | Month | Seats |
|---|---|---|
| 2001 | Jan | 10 |
| 2001 | Feb | 20 |
| 2001 | Mar | 30 |
| 2002 | Jan | 130 |
| 2002 | Feb | 200 |
| 2002 | Mar | 340 |

```
[ ]: wide_example.melt(ignore_index=False)
```

Question 🤔 Melting DataFrame Example with Multiple Value Columns

Consider the DataFrame `wide_example` which has 'Year' as index. Reshape it using the `melt` function, but this time, specify 'Jan' and 'Feb' as value columns.

▼ Click to reveal answer

```
wide_example.melt(value_vars=['Jan', 'Feb'],
ignore_index=False)
```

Generate an example for reshaping DataFrames.

Location: 95

Summary: Question: Consider the DataFrame `wide_example` which has 'Year' as index. Reshape it using the `melt` function, but this time, specify 'Jan' and 'Feb' as value columns.

Answer:

```
wide_example.melt(value_vars=['Jan', 'Feb'
```

Generate a practice question based on selected code...

Notebook: lec05.ipynb   Notebook Index: 96   gemini-2.0-flash

**Figure 1: An overview of ContentGen's interface. A) Instructors can select a notebook cell as inspiration for a practice question. B) They can specify the input prompt to clarify their intent. C) ContentGen generates a question and corresponding answer, shown both in the sidebar and embedded below the original cell. D) Instructors can accept the suggestion, cancel it, or submit a followup prompt for a new one.**

LLM to adhere to our course's guidelines. Finally, we found it challenging to systematically evaluate whether changes to our prompts were actually improving the quality of the generated questions.

These challenges led us to define the following design goals for a system to generate programming questions for data science:

*D1.* **In-workflow:** The tool should be embedded within the interface that instructors are already using to develop educational content (in our case, JupyterLab), rather than requiring them to switch to a separate web application.

*D2.* **Automatic Context Discovery:** The tool should automatically gather the necessary information from the instructional materials to generate a useful practice question, rather than forcing instructors to manually select and provide context in the prompt.

*D3.* **Effective for Instruction:** Generated questions should be *correct* (the solution code is valid), have *contextual fit* (the question uses the lecture's context, like its datasets), and be *coherent* (the question assesses the relevant topic).

## 4 System Design and Implementation

To instantiate our design goals from Section 3, we prototyped ContentGen, a tool that generates in-context questions and examples within JupyterLab. We direct readers to Figure 1 to see how the tool appears to users.

### 4.1 User Interface and Workflow

To satisfy our goal of creating an in-workflow (*D1*) tool, instructors interact with ContentGen through a sidebar chat interface within JupyterLab. To generate a question, they select the notebook cell with existing content that they want to use to generate practice question variants. Then, they can enter a short prompt with desired characteristics of their question, or leave the prompt blank to use the system defaults. ContentGen will then generate a question, display it in the sidebar, and also insert the question into the notebook itself below the user's currently selected notebook cell. Instructors have the option to apply or cancel the generated cell, or provide follow-up input to make changes. The first time instructors use ContentGen after installation, they will be asked to enter an API key for the Gemini LLM, although future versions of the tool will support other LLMs.

## 4.2 System Architecture and Implementation

ContentGen is implemented using standard Python, JavaScript, the JupyterLab Extension API, and an LLM API. All code, prompts, and datasets used for evaluating the tool are open-source on GitHub[1]. The current version of the prototype exclusively uses Gemini 2.0 Flash because we wanted to see whether we could still produce high-quality content using a smaller and cheaper model[2].

To address our goal of automatic context discovery (*D2*), ContentGen's backend analyzes the contents of the current Jupyter notebook when a user requests a question. The backend combines the user's prompt with contextual information extracted from the notebook to construct a single prompt for an LLM. The specifics of what context is extracted and how it is used in the prompt depend on the prompting strategy, which we evolved over our iterative design process and describe next. The result from the LLM is returned to the frontend, where the chat interface is updated and the new cell is inserted into the notebook.

*4.2.1 LLM Prompt Design.* To generate practice questions that are effective for instruction (*D3*) we compared three different prompting approaches that used different amounts of notebook context and preprocessing.

The first prompt version (Baseline) contains one paragraph of instructions to the LLM to generate both practice question and solution code. As context, it includes the entire notebook's content verbatim as a JSON string, the user's currently selected cell, and the user's instructions. This prompt was designed to mimic the scenario where an instructor copy-pastes their entire lecture as context for an LLM prompt.

The second version (Detailed) contains detailed, step-by-step instructions to the LLM that first asks the LLM to generate a code snippet that is a slight variation of the user's selected cell and only then generate a question to match the code snippet. As context, this prompt includes the same context as the Baseline prompt but also includes the names of the currently defined variables in the notebook.

The third and final prompt version (Structured) adds to the Detailed prompt by including extra information about the notebook's structure and organization. Specifically, before the user makes a request, ContentGen proactively processes the notebook with an LLM call to group cells into pedagogical topics, and lists out subtopics, functions defined, and datasets used within each topic. This additional information is provided as a JSON string at the end of the Structured prompt.

## 5 Quantitative Evaluation

To explore how prompt strategies can influence the quality of LLM-generated content, we conducted a quantitative evaluation of the questions and answers generated by ContentGen.

[1]https://github.com/dstl-lab/ContentGen-demo
[2]At the time of writing, Gemini 2.0 Flash is more than 95% cheaper per token than flagship models like GPT-4o, Claude 3.7, and Gemini 2.5 Pro.

## 5.1 Methodology

We conducted a structured evaluation of contents generated by ContentGen using a dataset of Jupyter notebook cells from real course materials. Our methodology focuses on the three components: (1) the construction of test cases, (2) the definition of evaluation metrics, and (3) a reliable evaluation process.

*5.1.1 Test Case Construction.* We purposively selected 91 test cases from the lecture notebooks of two foundational undergraduate data science courses that we teach. To ensure comprehensive coverage, we selected cells that span all major programming concepts taught in these courses, including Python basics, NumPy basics, pandas basics, intermediate pandas, data visualization, and machine learning using scikit-learn.

Our selection strategy was informed by our teaching experience, prioritizing cells that cover concepts students typically struggle with and topics essential for completing homework assignments and exams. Each test case consists of one notebook cell (either code or markdown) paired with a brief user prompt designed to guide question generation. These prompts are tailored to the educational goals of each cell, such as "*Generate a similar example or question based on...*", and reflect how we would naturally request practice questions as instructors.

For each test case, we used the three different prompt versions of ContentGen (described in Section 4) to generate a question and its paired answer. We generated an initial 273 question-answer pairs. To simulate the realistic scenario where instructors iterate on generated questions, we provided follow-up prompts for an additional 24 pairs to attempt to fix clear issues in the initially generated questions, resulting in a total of 297 question-answer pairs across all prompt versions for evaluation.

*5.1.2 Evaluation Criteria.* We evaluated the quality of the generated questions and the corresponding answers using three metrics developed based on our early empirical observations in pilot studies and the design goals of ContentGen (Section 3). For data science, a good question is not sufficient to be just correct, but also needs to be presented in a clear and coherent way based on the current teaching context. Therefore, we define three binary evaluation metrics in Table 1.

*5.1.3 Calibration and Evaluation Process.* To ensure reliability and consistency, we adopted a structured process with two rounds of evaluation involving three authors of this paper. We began with a calibration round, where all three authors independently rated a randomly sampled subset of 55 question-answer pairs (approximately 18.5% of the 297 generated pairs) with the defined metrics. After the independent ratings, the three raters discussed discrepancies and refined shared interpretations. The agreement scores in percentage and the Fleiss' Kappa, summarized in Table 2, show that the raters achieved high agreement across all three metrics. The $\kappa$ scores indicate substantial agreement for Correctness and Coherence, and almost perfect agreement for Contextual Fit.

In the main evaluation round, each rater was assigned a distinct version of ContentGen (one rater evaluated all Baseline questions, another evaluated all Detailed questions, and the third evaluated all Structured questions) and independently evaluated all questions generated with that version. This assignment strategy ensured

**Table 1: Evaluation metrics and scoring rubric used to assess the quality of generated contents.**

| Metric | Definition (must satisfy all) |
|---|---|
| Correctness | The question is logically valid and includes all necessary information to answer it. The solution is accurate with respect to the concepts, and the code is runnable. |
| Contextual Fit | The question reuses the same dataset or variables for instructional continuity. The question aligns with the topic of the original cell and avoids unfamiliar APIs or concepts not yet introduced. |
| Coherence | The question is clear and focuses on the most relevant educational topics or concepts. The question is different from the existing notebook cells and worth teaching as an independent example. |

**Table 2: Calibration results demonstrated high agreement across all metrics.**

| Metric | Agreement Percentage | Fleiss' Kappa $\kappa$ |
|---|---|---|
| Correctness | 90.9% | 0.772 |
| Contextual Fit | 90.9% | 0.867 |
| Coherence | 87.3% | 0.746 |

consistent evaluation standards within each prompt version while the calibration process ensured comparable standards across versions. Each generated question was independently evaluated on the three binary metrics described in Section 5.1.2, with raters assigning a score of 1 if the question met the standard and 0 otherwise.

To facilitate comparison across different versions of prompt strategies, we normalized the scores by computing the percentage of questions that received a score of 1 in each dimension. The detailed analysis of the results is presented in Section 5.2.

## 5.2 Results

After confirming the reliability of our metrics through calibration (Section 5.1.3), we conducted the main evaluation across all generated question-answer pairs.

The evaluation results (Figure 2) revealed substantial quality differences across the three prompt versions. The Baseline version performed poorly across all metrics, with Contextual Fit scoring only 23%. Analysis of the failures revealed that Baseline questions frequently created new datasets rather than reusing existing ones, used pandas operations too advanced for the current point in the course, or made incorrect assumptions about dataset contents (e.g., wrong column names).

The addition of detailed instructions in the Detailed version led to dramatic improvements, particularly in Contextual Fit, which jumped from 23% to 76%. This improvement was largely attributed to explicit instructions on variable reuse and scope alignment.

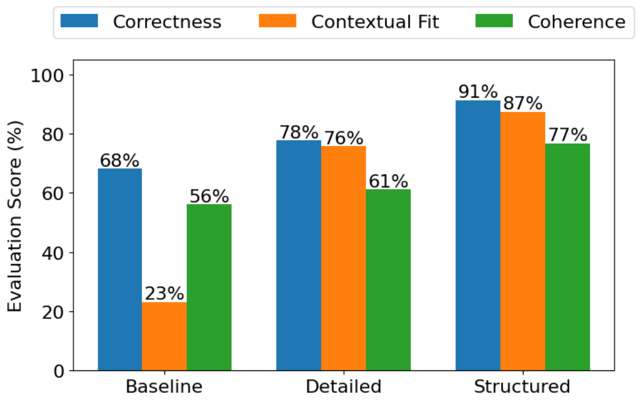


**Figure 2: Iteration of prompt versions led to significant improvement across all metrics, with the Structured prompt achieving the best performance overall.**

However, Correctness failures still occurred when generated code would error out or produce incorrect results, and Coherence issues persisted when questions were either verbatim copies of existing notebook cells or were pedagogically inappropriate (too simple or too challenging).

The Structured version achieved the highest overall scores by incorporating a structured summary of the notebook. This version substantially improved Correctness and Contextual Fit, with questions more successfully reusing existing datasets in meaningful ways. Coherence also improved compared to earlier versions, but remained the lowest among all metrics, suggesting that generating content clearly aligned with instructional goals remains the most challenging aspect of automated question generation.

In addition, we examined the average number of tokens used in each prompt version. The majority of tokens came from the lecture notebooks themselves, which were inserted verbatim into the prompt. Since notebook lengths vary considerably across courses and lectures, total token usage also differed significantly. In our dataset, the Baseline and Detailed versions used approximately 6,000–9,000 tokens per generation. The Structured version required about 12,000–16,000 tokens because it involved two separate LLM calls: one to summarize the notebook structure and another to generate a question. Despite the increased token usage, our tool remains cost-efficient: we estimate that roughly 700 questions can be generated per U.S. dollar using the latest Structured version of ContentGen with Gemini 2.0 Flash.

These results suggest that systematic prompt engineering can substantially improve LLM-generated educational content. The progression from Baseline to Detailed to Structured prompts shows that explicit instructions about variable reuse and context-aware preprocessing are more effective than simply providing more raw context. While all three metrics improved with better prompting, Coherence remained the most challenging metric, suggesting that generating pedagogically appropriate questions requires careful attention to instructional goals beyond technical correctness.

## 6 Qualitative Evaluation

Our quantitative evaluation (Section 5) was based on the metrics of Correctness, Contextual Fit, and Coherence that we created based on our experience as instructors. To externally validate our prototype's usefulness for other instructors and to identify areas for improvement, we conducted usability studies with data science instructors.

### 6.1 Methodology

We conducted 50-minute, one-on-one, think-aloud usability studies with six participants (P1-P6, three faculty and three teaching assistants) from introductory and intermediate data science courses in our department. All participants regularly used Jupyter notebooks for teaching and had some experience using LLMs for content generation. Participants installed ContentGen with the highest-performing Structured prompt on their personal computers and used ContentGen to generate practice questions based on a lecture they gave recently. We asked participants interview questions derived from the Technology Acceptance Model framework [9] and analyzed the recordings using the thematic analysis framework [4].

### 6.2 Results

Instructors found ContentGen easy to install and intuitive to use with minimal instruction (P1-P6). They reported that the generated questions were generally useful and that they could see themselves using the generated questions in their courses (P1-P6). Since this positive feedback is consistent with our quantitative evaluation, we focus the remainder of this section on pointing out the cases where the tool did *not* perform well.

Our analysis of cases where instructors disliked ContentGen's output validated our three quantitative metrics. For example, instructors almost always ran the generated solution to check for Correctness; when it was inaccurate, they noted that such errors would "be misleading" for students (P2, P4). Similarly, issues with Contextual Fit arose when a generated question failed to integrate into the lecture's flow. One instructor, for instance, pointed out that a generated question used a function they didn't "teach . . . until a week later" (P1), disrupting the pedagogical sequence. Finally, instructors flagged issues with Coherence when a question was not pedagogically sound, such as when it was "very complicated" (P1) or did not test the intended concept. As one instructor noted, the goal is to "test the concept, not necessarily the syntax" (P2), and questions failed when they did not meet this goal.

Although we originally imagined that a tool like ContentGen could be useful during a live lecture to provide real-time practice, instructors generally felt more confident using ContentGen while preparing materials than using the tool live. Several also saw potential for using it in less formal settings like office hours, where they could "check the correctness" (P1, P4, P5) in a lower-stakes environment. This reluctance to use the tool live stemmed from a lack of trust in the LLM's reliability: "you don't know if it's going to give you a good question" (P1). Taken together, these findings validate our evaluation metrics and suggest that LLM-based tools could move from preparation aids to live, in-class assistants, if future development prioritizes improving Correctness, Contextual Fit, and Coherence to build instructor trust.

## 7 Discussion

In this section, we reflect on the process of prototyping ContentGen and discuss the implications of our work for future research.

*Engineering Prompts, Context, and Metrics.* One of the most surprising aspects of building ContentGen was the challenge of improving LLM outputs. We initially anticipated that the main technical hurdle would be integrating our tool into JupyterLab, and that working with an LLM API would be relatively straightforward. In practice, the opposite was true. The most difficult part of our process was not the implementation, but rather defining what "better" meant for an LLM-generated practice problem. Our metrics of Correctness, Contextual Fit, and Coherence emerged from our experiences as data science instructors, who have different pedagogical needs than instructors of more "traditional" computer science courses. For example, ensuring that a generated question reused a lecture's dataset (Contextual Fit) was just as important as the question being answerable (Correctness).

Once we defined these metrics, it became much easier to navigate the large design space of possible prompt and context variations to provide the LLM. We recommend that future tool-builders work closely with instructors to understand their specific needs and identify where off-the-shelf LLM outputs fall short before beginning the engineering process. Although improving our prompt itself (from Baseline to Detailed) generated an initial substantial improvement across our metrics, we found that further iterations on the prompt alone didn't produce similar gains. Instead, much of our engineering effort to produce the most effective Structured prompt focused on curating the right context to provide the LLM. This aligns with recent views that *context engineering* is perhaps a more appropriate term than *prompt engineering* for the work required to make LLM-based tools effective in practice [25, 31].

*Using LLMs in Practice as Educators.* Our findings offer several practical implications for data science instructors and curriculum designers seeking to leverage LLMs. First, our work suggests that the most effective way to improve LLM-generated content is not necessarily through intricate prompt design, but through careful context curation. Instead of providing an LLM with an entire lecture notebook, instructors may achieve better results by first using an LLM to generate a structured summary of the notebook's topics, functions, and datasets, and then using that summary as the context for a second generation request. This two-step process, while more involved, provides a concrete strategy for improving the quality of generated content.

As LLMs become increasingly powerful and inexpensive, we can imagine that instructors will less frequently create educational content from start to finish, and more frequently provide creative initial ideas for content that an LLM will generate. To be effective in this new role, instructors may require new skills in prompting, evaluating, and integrating AI-generated content into their courses. This points to a potential need for professional development and new tools that move beyond content generation to support instructors in this validation workflow, helping them quickly assess the pedagogical quality of an output and build the necessary trust to use these tools effectively.

## References

[1] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. 15, 3 (2024), 1–45. doi:10.1145/3641289

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. *Evaluating Large Language Models Trained on Code.* doi:10.48550/arXiv.2107.03374 arXiv:2107.03374 [cs]

[3] Zirui Cheng, Jingfei Xu, and Haojian Jin. 2024. TreeQuestion: Assessing Conceptual Learning Outcomes with LLM-Generated Multiple-Choice Questions. 8 (2024), 1–29. Issue CSCW2. doi:10.1145/3686970

[4] Victoria Clarke and Virginia Braun. 2014. Thematic Analysis. In *Encyclopedia of Critical Psychology*, Thomas Teo (Ed.). Springer New York, 1947–1952. doi:10.1007/978-1-4614-5583-7_311

[5] Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. 2023. *Can We Trust AI-Generated Educational Content? Comparative Analysis of Human and AI-Generated Learning Resources.* doi:10.48550/arXiv.2306.10509 arXiv:2306.10509 [cs]

[6] Jacob Doughty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, Arav Agarwal, Christopher Bogart, Eric Keylor, Can Kultur, Jaromir Savelka, and Majd Sakr. 2024. A Comparative Study of AI-Generated (GPT-4) and Human-crafted MCQs in Programming Education. In *Proceedings of the 26th Australasian Computing Education Conference* (2024-01-29). ACM, 114–123. doi:10.1145/3636243.3636256

[7] Sabina Elkins, Ekaterina Kochmar, Iulian Serban, and Jackie C. K. Cheung. 2023. How Useful Are Educational Questions Generated by Large Language Models?. In *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky* (Cham, 2023), Ning Wang, Genaro Rebolledo-Mendez, Vania Dimitrova, Noboru Matsuda, and Olga C. Santos (Eds.). Springer Nature Switzerland, 536–542. doi:10.1007/978-3-031-36336-8_83

[8] Aysa Xuemo Fan, Ranran Haoran Zhang, Luc Paquette, and Rui Zhang. 2023. *Exploring the Potential of Large Language Models in Generating Code-Tracing Questions for Introductory Programming Courses.* doi:10.48550/arXiv.2310.15317 arXiv:2310.15317 [cs]

[9] Andrina Granić and Nikola Marangunić. 2019. Technology Acceptance Model in Educational Context: A Systematic Literature Review. 50, 5 (2019), 2572–2593. doi:10.1111/bjet.12864

[10] Irene Hou, Sophia Mettille, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The Effects of Generative AI on Computing Students' Help-Seeking Preferences. In *Proceedings of the 26th Australasian Computing Education Conference* (2024-01-29). ACM, 39–48. doi:10.1145/3636243.3636248

[11] Sven Jacobs, Henning Peters, Steffen Jaschke, and Natalie Kiesler. 2025. *Unlimited Practice Opportunities: Automated Generation of Comprehensive, Personalized Programming Tasks.* doi:10.48550/arXiv.2503.11704 arXiv:2503.11704 [cs]

[12] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. *LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code.* doi:10.48550/arXiv.2403.07974 arXiv:2403.07974 [cs]

[13] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* doi:10.48550/arXiv.2310.06770 arXiv:2310.06770 [cs]

[14] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Proceedings of the 26th Australasian Computing Education Conference* (2024-01-29). ACM, 77–86. doi:10.1145/3636243.3636252

[15] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. 2023. ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education. 103 (2023), 102274. doi:10.1016/j.lindif.2023.102274

[16] Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler, and Salam Al-Emari. 2020. A Systematic Review of Automatic Question Generation for Educational Purposes. 30, 1 (2020), 121–204. doi:10.1007/s40593-019-00186-y

[17] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2020). IEEE, 1–11. https://ieeexplore.ieee.org/abstract/document/9127201/

[18] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance Is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools Such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1* (Chicago IL USA, 2023-08-07). ACM, 106–121. doi:10.1145/3568813.3600138

[19] Kunze Li and Yu Zhang. 2024. Planning First, Question Second: An LLM-Guided Method for Controllable Question Generation. In *Findings of the Association for Computational Linguistics: ACL 2024* (Bangkok, Thailand, 2024-08), Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 4715–4729. doi:10.18653/v1/2024.findings-acl.280

[20] Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. 2024. Evaluating Contextually Personalized Programming Exercises Created with Generative AI. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (Melbourne VIC Australia, 2024-08-12). ACM, 95–113. doi:10.1145/3632620.3671103

[21] Xinyi Lu, Simin Fan, Jessica Houghton, Lu Wang, and Xu Wang. 2023. ReadingQuizMaker: A Human-NLP Collaborative System That Supports Instructors to Design High-Quality Reading Quiz Questions. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2023-04-19) *(CHI '23)*. Association for Computing Machinery, 1–18. doi:10.1145/3544548.3580957

[22] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Vee Pettit, Leo Porter, Brent N. Reeves, Jaromir Savelka, David H. Smith, Sven Strickroth, and Daniel Zingaro. 2025. Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. In *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (Milan Italy, 2025-01-22). ACM, 300–338. doi:10.1145/3689187.3709614

[23] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2025. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications.* doi:10.48550/arXiv.2402.07927 arXiv:2402.07927 [cs]

[24] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (New York, NY, USA, 2022-08-03) *(ICER '22, Vol. 1)*. Association for Computing Machinery, 27–43. doi:10.1145/3501385.3543957

[25] Philipp Schmid. 2025. *The New Skill in AI Is Not Prompting, It's Context Engineering.* https://www.philschmid.de/context-engineering

[26] Yiyin Shen, Xinyi Ai, Adalbert Gerald Soosai Raj, Rogers Jeffrey Leo John, and Meenakshi Syamkumar. 2024. Implications of ChatGPT for Data Science Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland OR USA, 2024-03-07). ACM, 1230–1236. doi:10.1145/3626252.3630874

[27] Andrew Tran, Kenneth Angelikas, Egi Rama, Chiku Okechukwu, David H. Smith, and Stephen MacNeil. 2023. Generating Multiple Choice Questions for Computing Courses Using Large Language Models. In *2023 IEEE Frontiers in Education Conference (FIE)* (2023). IEEE, 1–8. https://ieeexplore.ieee.org/abstract/document/10342898/

[28] Xu Wang, Simin Fan, Jessica Houghton, and Lu Wang. 2022. *Towards Process-Oriented, Modular, and Versatile Question Generation That Meets Educational Needs.* doi:10.48550/arXiv.2205.00355 arXiv:2205.00355 [cs]

[29] Zichao Wang, Jakob Valdez, Debshila Basu Mallick, and Richard G. Baraniuk. 2022. Towards Human-Like Educational Question Generation with Large Language Models. In *Artificial Intelligence in Education* (Cham, 2022), Maria Mercedes Rodrigo, Noburu Matsuda, Alexandra I. Cristea, and Vania Dimitrova (Eds.). Springer International Publishing, 153–166. doi:10.1007/978-3-031-11644-5_13

[30] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with Chatgpt. (2023). arXiv:2302.11382 https://file.mixpaper.cn/paper_store/2023/681177f8-cd15-4e0f-a23b-997c6b9f9dd2.pdf

[31] Simon Willison. 2025. *Context Engineering.* Simon Willison's Weblog. https://simonwillison.net/2025/Jun/27/context-engineering/

[32] Jiaen Yu, Anshul Shah, John Driscoll, Yandong Xiang, Xingyin Xu, Sophia Krause-Levy, and Soohyun Nam Liao. 2025. Engagement with Metacognition-promoting Web-based Interventions and its Relationship with Learning Outcomes. In *2025 ASEE Annual Conference & Exposition.*

[33] Jiaen Yu, Anshul Shah, John Driscoll, Yandong Xiang, Xingyin Xu, Sophia Krause-Levy, and Soohyun Nam Liao. 2025. Student Usage of Metacognition-Promoting Tool in a CS2 Course and its Relationship with Performance. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2.* 1671–1672.