Generating Effective Distractors for Introductory Programming Challenges: LLMs vs Humans

Mohammad Hassany
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
moh70@pitt.edu

Peter Brusilovsky
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
peterb@pitt.edu

Jaromir Savelka
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jsavelka@cs.cmu.edu

Arun Balajiee Lekshmi Narayanan
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
arl122@pitt.edu

Kamil Akhuseyinoglu
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
kaa108@pitt.edu

Arav Agarwal
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
arava@andrew.cmu.edu

Rully Agus Hendrawan
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
ruhendrawan@pitt.edu

Abstract

As large language models (LLMs) show great promise in generating a wide spectrum of educational materials, robust yet cost-effective assessment of the quality and effectiveness of such materials becomes an important challenge. Traditional approaches, including expert-based quality assessment and student-centered evaluation, are resource-consuming, and do not scale efficiently. In this work, we explored the use of pre-existing student learning data as a promising approach to evaluate LLM-generated learning materials. Specifically, we used a dataset where students were completing the program construction challenges by picking the correct answers among human-authored distractors to evaluate the quality of LLM-generated distractors for the same challenges. The dataset included responses from 1,071 students across 22 classes taught from Fall 2017 to Spring 2023. We evaluated five prominent LLMs (OpenAI-o1, GPT-4, GPT-4o, GPT-4o-mini, and Llama-3.1-8b) across three different prompts to see which combinations result in more effective distractors, i.e., those that are plausible (often picked by students), and potentially based on common misconceptions. Our results suggest that GPT-4o was the most effective model, matching close to 50% of the functional distractors originally authored by humans. At the same time, all of the evaluated LLMs generated many novel distractors, i.e., those that did not match the pre-existing human-authored ones. Our preliminary analysis shows that those appear to be promising. Establishing their effectiveness in real-world classroom settings is left for future work.

LAK 2025, Dublin, Ireland

ACM ISBN 979-8-4007-0701-8/25/03
https://doi.org/10.1145/3706468.3706529

CCS Concepts

• **Applied computing** → **E-learning**; • **Computing methodologies** → **Artificial intelligence**; • **Human-centered computing** → **Empirical studies in interaction design**.

Keywords

Large Language Models (LLMs), Distractor Generation and Evaluation, Student Learning Data, Introductory Programming, GPT, LLaMA

ACM Reference Format:
Mohammad Hassany, Peter Brusilovsky, Jaromir Savelka, Arun Balajiee Lekshmi Narayanan, Kamil Akhuseyinoglu, Arav Agarwal, and Rully Agus Hendrawan. 2025. Generating Effective Distractors for Introductory Programming Challenges: LLMs vs Humans. In *LAK25: The 15th International Learning Analytics and Knowledge Conference (LAK 2025), March 03–07, 2025, Dublin, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3706468.3706529

1 Introduction

Over the last three years, the use of Large Language Models (LLMs) for supporting computer science education and, more specifically, introductory programming has been extensively explored. Different projects explored the use of LLMs to generate solutions to programming problems [8], provide feedback and corrections for student solutions [4, 20, 21], explain compiler error messages [12] or existing program code [19], and generate various types of learning content such as worked examples [16], multiple-choice questions [5], and programming problems [30]. The rapidly increasing volume of research activity in this direction has stressed an important challenge that most of the research encountered: how to assess the quality of LLM-generated artifacts. This question is important not only for demonstrating that LLMs could be valuable in the learning process but also for comparing multiple alternative ways to achieve the same goals, such as using different prompts and different types of LLMs.

Traditional evaluation approaches range from expert-based quality assessment [12, 27] to student-centered evaluation through classroom studies [39]. However, neither of these approaches can scale sufficiently well to potentially compare many generation options that modern LLMs can offer. Both types of evaluation are costly and cannot be used to run assessments on a sufficiently large scale. Moreover, the judgment of domain experts could produce biased results in cases where it differs from the judgment of students who are the target audience in the learning process. In turn, classroom evaluation of LLM-based approaches could be considered as the ultimate proof of their value, but the need to run a classroom study to compare LLM alternatives significantly slows down the design-evaluation loop. In this paper, we explore the use of pre-existing learning data as an alternative approach to evaluating the quality of LLM-generated artifacts in the educational process. The use of real learning data retains the benefit of evaluating LLM artifacts from the student perspective, while also significantly reducing the time required for the evaluation of alternatives. Most importantly, it could be performed on a large scale and allows us to compare many design alternatives in a single study.

This paper explores the use of large learning data to generate distractors for programming "challenges", small multiple-choice problems used to assess student's understanding of worked code examples. Effective distractors should be plausible while unambiguously wrong options [10]. Their goal is to distract students from easily picking the correct answer [1, 22]. The generation of high-quality distractors is a time-consuming task [10]. Hence, in practice, the distractors may often be left ineffective, letting the correct solution stand out. As a result, students miss a learning opportunity by identifying the correct answer without much thought. Recent pioneer works [7, 33] provided the first evidence that LLMs might be able to help instructors by generating plausible distractors. In this paper, we explore this opportunity more extensively by engaging past data on learner problem-solving and their use of distractors. The learner data enables us to produce a more reliable assessment of LLM potential in generating distractors as well as to compare a much broader range of approaches and models.

2 Related Work

2.1 Distractors

Manually authoring high-quality distractors is difficult and time-consuming [10]. Hence, the appeal of (semi-)automatic generation of distractors. For MCQs, researchers have explored countless approaches [24, 29]. In some cases, only a limited number of distractors is feasible [7]. Students' common misconceptions can serve as a good source of high-quality distractors. An experienced instructor may be aware of these and therefore can anticipate misconceptions directly [7, 9, 26, 32, 33, 38, 40]. Gierl et al. [9] described six guidelines for developing high-quality distractors. The most important property of a high-quality distractor is its plausibility while being unambiguously incorrect [7, 9, 10, 22, 24, 29, 32]. A plausible distractor must be related to the problems [9, 22]. Distractors should be similar to (but not confused with) a correct answer [15]. As to the ideal number of distractors and their ordering, there is no consensus. Two distractors along with the correct answer might be all that is needed [9, 22]. The ordering can be random, but the final position of the correct answer should be carefully reviewed [9]. In this work, we adopted a modified version of the MCQ Distractors Development Guidelines [9].

LLMs have recently attracted a lot of attention due to their natural language and code-generating capabilities. These powerful models may potentially enable the large-scale generation of novel high-quality learning materials in many disciplines, including computing education. In computing education, the automatic generation of distractors has been explored in the context of automated drafting of multiple-choice questions (MCQs). Doughty et al. [5] proposed an LLM-powered pipeline for the generation of introductory programming MCQs, including distractors, from high-level course context and fine-grained learning objectives (LOs). They compared the quality of the generated distractors with distractors authored by human instructors, finding that the automatically generated distractors are comparable but somewhat lower quality. The main issue identified by the authors is that GPT-4 often generates a correct answer as a distractor (4.9%). Tran et al. [37] evaluated the LLM-powered generation of isomorphic MCQs from an existing question bank and reported the same issue. The discussed work explored the effects of prompt engineering on the quality of the generated distractors and provided valuable insights. For example, [37] reports that requesting the distractors to be "plausible" improves their quality. In our work, we also explore the effects of prompt engineering and report novel findings. Hou at al. [14] explored the use of LLMs to automatically generate distractors for Parson's problems. Feng et al. [7] reported that state-of-the-art LLMs do not generate distractors based on students' common errors and misconceptions in the context of math MCQs. In our work, we experiment with incorporating common misconceptions in the prompt supplied to an LLM.

2.2 Automatic Generation of Learning Artifacts in Computing Education

There are many other examples of using LLMs to generate learning artifacts for computing education. Sarsa et al. [30] focused on programming exercises, including sample solutions and test cases. They reported that in most cases the automatically generated content was not ready to be used as-is, but it was at least novel and sensible. Furthermore, most of the time only minor tweaks would have been required to fix the issues. Del Carpio Gutierrez et al. [2] evaluated a more recent GPT-4 model, observing that the quality of automatically generated exercises was high. Hou et al. [14] proposed an LLM-powered learning environment that automatically generates personalized Parson's puzzles. Sridhar et al. [35] explored the automatic generation of learning objectives for a course on artificial intelligence. Other examples of LLM-generated artifacts in computing education include code explanations [23, 30], model solutions [3, 31], feedback [25, 28], and responses to help requests [17, 36]. The automatically generated learning artifacts, while promising, may often exhibit certain limitations that make them unsuitable for use in real-world settings. At the same time, a rigorous traditional evaluation might be difficult or prohibitively expensive. In our work, we suggest a novel evaluation approach that relies on pre-existing large-scale student data.

2.3 Evaluating Large Language Models in Educational Context

The first pioneer research on LLM in computer science education focused on evaluating LLM–generated solutions to programming problems, which could be evaluated automatically using test cases [8, 30]. As the research progressed to exploring other LLM–generated artifacts, such as learner-focused questions, programming problems, and explanations, it became important to develop human-centered evaluation approaches. Currently, we can distinguish three types of these approaches: expert-based, student-based, and classroom evaluation. In *expert-based* approaches, LLM–generated artifacts are evaluated by domain experts (researchers, instructors, content authors) [6, 18, 27, 30]. In the evaluation process, the artifacts (explanations, problem solutions, program corrections, etc.) are presented independently to several expert evaluators to be rated numerically by different aspects such as quality, completeness, and readiness. To eliminate bias when several approaches are compared, the evaluators are not aware of the approach that produced each artifact [11]. The *student-based* evaluation uses a similar rating-based evaluation strategy but involves students (i.e., target users) as the evaluators [4, 21]. The weak side of both rating-based is taking this evaluation out of the use context, where human judgment could be less reliable. To address this problem, an increasing number of studies use full-scale *classroom experiments* to evaluate the quality of generated artifacts [12]. Recently published studies use classroom experiments to evaluate worked examples [16] and feedback for student solutions [20].

3 Dataset

In this work, we used a dataset of student responses to a set of 38 program construction "challenges" offered by the PCEX system [13]. A "challenge" is a special form of multiple-choice problem that assesses students' knowledge of program construction. It displays a statement of a programming problem along with a code that solves this problem. The code has one missing (masked) line. The student's goal is to complete the program by selecting the missing line from a list of options consisting of one correct answer and four or more distractors (Figure 1). Most challenges in the selected set have 4 distractors, except two with 5 and one with 6 distractors (a total of 156 unique distractors). The students were able to make several attempts to solve the challenge until it was solved correctly. After each attempt, the PCEX system executed the completed program and displayed the obtained result. If a distractor was selected, the system also contrasted the wrong result produced by the program with this distractor with the expected correct result. The challenges are openly available and were used in multiple programming courses; however, the dataset of student responses is not publicly available and was provided by the tool's developer upon our request. Student responses include the student identifier, course ID, challenge ID, selected line, timestamp, nth_attempt, and the result. The "nth_attempt" resets to 1 when the student opens the challenge and is increased by 1 after each continuous attempt to solve it.

The dataset includes 28,871 responses from 1,071 students who interacted with 38 selected challenges (Table 1). The data were collected from students in 22 courses delivered from Fall 2017 to

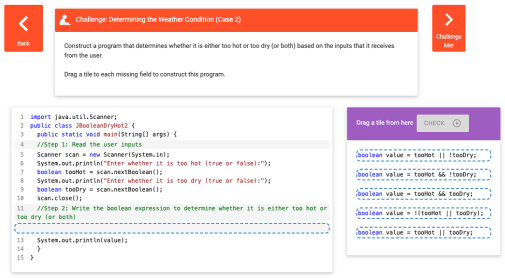

Figure 1: Program Construction Challenge: Students are given a problem statement and solution, with one line masked. They are asked to fill in the masked line by picking the correct answer from the list of available options.

Spring 2023. On average, there were 52.45 (SD=28) students per course (Figure 2) attempting 31.68 challenges out of 38 on average (SD=8.24, min=18, max=38). The distribution of students and attempts across classes is shown in Figure 2. The initial data analysis revealed a considerable number of cases in which the student selected a distractor (i.e., incorrect answer) even after solving the challenge (i.e., selecting the correct answer), apparently being interested in which program outcome is produced by each distractor. To distinguish these explorations from true attempts to solve the challenge, we excluded all attempts to select a distractor after the first correct attempt. The final data set included 13,419 failed attempts. On average, 157.16 students (SD=97.55, min=38, max=457) failed each challenge at least once. Distractors received an average of 86.02 attempts (SD=68.71), and students failed each challenge with an average of 2.25 times (SD=1.69).

4 Method

The key motivation of this paper is to use a large body of pre-existing student interactions with programming "challenges" as a source of data to compare the ability of different LLMs and different prompt strategies to generate good distractors. It is the power of existing data that enables us to identify good distractors. Indeed, a good distractor should have a good chance of being picked up by students, especially on their first attempt. Using the data of student distractor selection, we were able to separate 4-6 distractors provided by the challenge authors for each challenge into strong (functional) and weak (non-functional). These data were used as the "gold standard" to compare 5 LLM models and 3 prompt strategies reviewed below according to their ability to generate good distractors and replicate all human-authored distractors for the 38 challenges. This approach has an obvious limitation: if a distractor generated by LLM matches one of the human-authored distractors, we can judge it as functional or non-functional, however, we cannot provide any data-driven judgment for non-matched distractors. To complete this study, the authors examined the characteristics of non-matched distractors and labeled them to provide an additional source of data for comparing the models.

Topic	Challenges	Students	Attempts
arrays	6	411	3,050
bool_exps	10	712	6,747
if-else	3	675	2,247
loops	14	564	9,412
strings	2	612	1,926
vars_ops	3	1,029	5,489
Total:	38	1071*	28,871

Table 1: Number of challenges, students, and attempts per topic. *Students attempted multiple challenges across different topics.

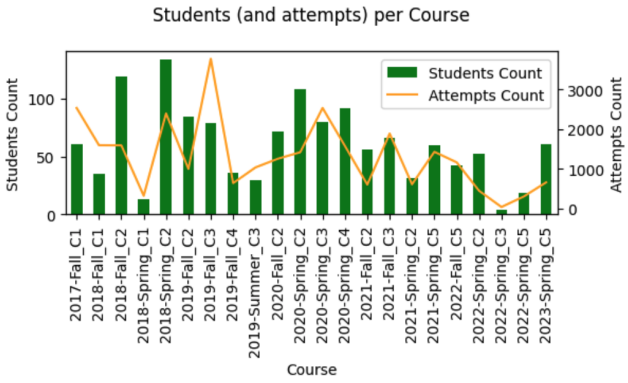

Figure 2: Number of students and their attempts per course.

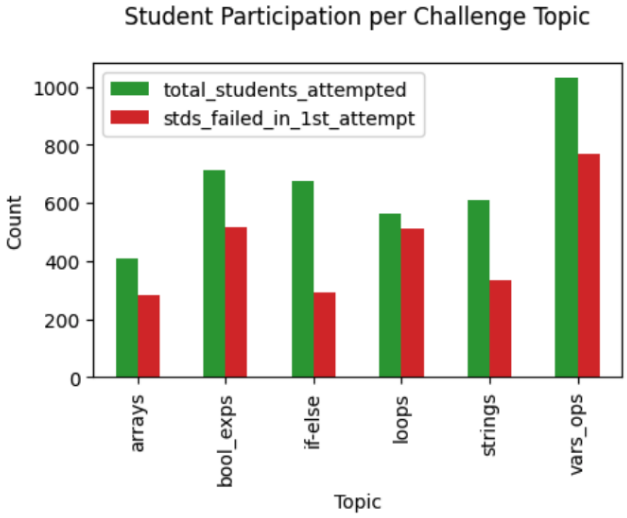

Figure 3: Number of unique students attempting challenges and failing on their 1st attempt.

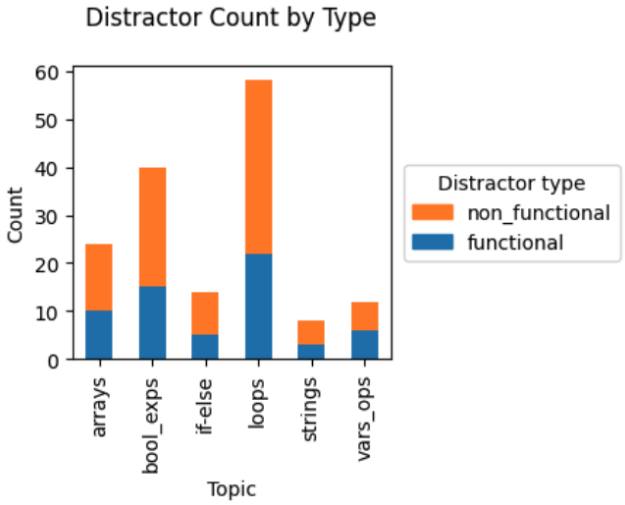

Figure 4: Number of distractors per each type (functional and non-functional).

4.1 Functional Distractors

A functional distractor is an incorrect option in a multiple-choice question that is sufficiently plausible to distract students from choosing the correct answer [22]. Although the existing literature [32] suggests that a distractor picked up by at least 5% of the examinees should be considered functional, it does not apply to our data set. PCEX challenges were designed for self-study rather than for a test and each challenge can be attempted multiple times. Given that almost all challenges included 4 distractors, each option (correct answer or distractor) has a 20% chance of being randomly picked at the first attempt, making the 5% thresholds too low for this distinction. In this study, we set the threshold at 25%, defining a functional distractor as one that is picked at a rate higher than the random chance by a student on the first attempt (see Figure 3). This threshold splits the 156 distractors into 61 functional and 95 non-functional distractors (see Figure 4 for this distribution per topic).

4.2 Model Specifics

To compare different models by their ability to generate distractors, we selected a representative sample of 5 LLMs, which included 4 models from OpenAI and one from Meta. The models were chosen primarily for their capability, ease of integration, and ease of use. The models were used with their default configurations; except for the temperature, which was set to zero.

- ***OpenAI-o1****-preview-2024-09-12*, is a new model from OpenAI, designed to automatically perform extra steps before responding, enhancing reasoning in complex tasks.
- ***GPT-4****-0613*, is a highly accurate model with broader general knowledge and advanced reasoning capabilities.
- ***GPT-4o****-2024-08-06*, is an efficient model comparable to the popular GPT-4 Turbo.
- ***GPT-4o-mini****-2024-07-18*, is a smaller and more cost-friendly model than GPT-4o and GPT-4.
- ***Meta-Llama****-3.1-8b-instruct-turbo*, quantized version of open source generative-AI flagship model from Meta.

The JSON schema of the output format specified in the prompt (Figure 5.7) was used only with GPT-4o and GPT-4o-mini as they

support "structured output". A few outputs generated by GPT-4 and some by Meta-Llama required manual adjustments to conform to a valid JSON object. OpenAI-o1-preview-2024-09-12 model does not support the "system" role, so it was added at the beginning of the prompt.

4.3 Prompting LLMs

Since the quality of an LLM-generated artifact frequently depends on the nature of the prompt used to generate it, we explored several versions of the prompt for each model. The key factor that distinguishes these prompts is the use of misconceptions. Since good distractors usually correspond to specific misconceptions (Section 2.1), we attempted to guide the model in generating plausible distractors by using its internal knowledge (P2) or external sources (P3) of common student misconceptions. We also attempted to guide LLM by providing distractors generation guidelines.

- *Prompt 1 (P1)* is a baseline prompt that asks the model to generate distractors for the target line without mentioning misconceptions or guidelines.
- *Prompt 2 (P2)* asks the model to use internal knowledge of common student misconceptions to generate distractors for the target line.
- *Prompt 3 (P3),* asks the model to use the list of common student misconceptions provided by [34] along with a modified version of the distractor development guidelines [9] to generate distractors for the target line.

As shown in Figure 5, these prompts have several common parts: 1) provide the context, explaining how the generated distractors will be used in a program construction challenge, 2) specify the task each model is expected to perform, 3) nudge the model to avoid generating correct answers as distractors, and 5) includes the program description which the model should consider when generating distractors for the specific line in the program solution (note that the target line is not masked in the prompt). Finally, 7) specifies the expected output. The system role, defined as "You are a learning support bot focused on introductory programming," has been used in all three prompts. P2 and P3 use a similar output format, as they intend to generate misconception-based distractors, while P1 does not. P3 contains a modified version of the distractor development guidelines [9] and the list of common misconceptions [34].

As part of the prompt tuning, we explored several variances of our prompts. Using the GPT-4o model, we varied the following prompt aspects to determine which option would generate more distractors that are similar to good human-authored distractors:

- *Hinting the model to avoid generating correct answers*, we tried different instructions in the prompt to ensure the model didn't generate correct answers as a distractor, but we were unsuccessful. In some cases, the model even generated the correct answer with an explanation highlighting that it is a correct answer and shouldn't be used as a distractor. We believe that ensuring this through prompting is extremely challenging. It would be more effective to implement a post-generation filtering mechanism for this purpose.
- *Changing temperatures (temp=0, 0.3, and 0.6)*, we observed a slight decline of ~5% in the number of matched distractors generated by the model. However, this was not consistent. For this work, we used temp=0 to have a deterministic list of distractors, allowing us to compare these models more consistently with their default configurations. These models can be fine-tuned, which is beyond the scope of this work.
- *Masking the target line*, we observed that masking the target line resulted in approximately ~50% fewer matching distractors generated by the model. One possible explanation may be that masking the target line allows the model to generate more diverse distractors. In addition, the program description for the challenges is mostly vague and open-ended; we found that the challenges can have many possible correct answers. So, we decided not to mask the target line, helping the model get a better context about the distractor generation task.
- *Specifying the number of distractors that LLM should generate (n=None and n=10)*, we observed a slightly higher number of matched distractors with n=10 compared to n=None. Across different models and prompts, we also observed duplicate distractors (within the same generation). Based on our experience with avoiding the generation of correct answers, we didn't address this in our prompt tuning, as filtering them out after generation would be an easier solution. Refer to the results section for a similar comparison of matched distractors using various n-values.

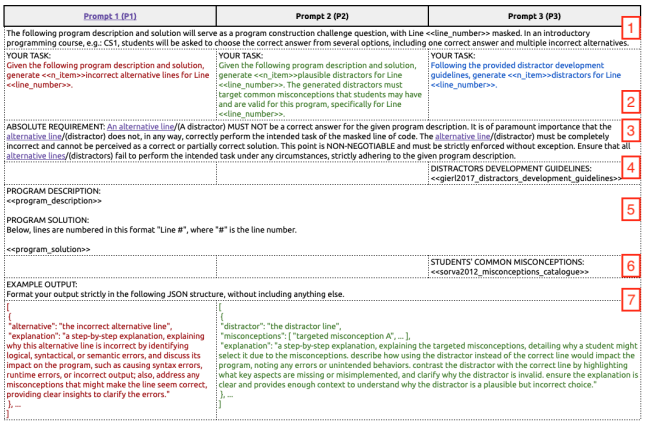

Figure 5: Prompts used to generate distractors: 1) provides context, 2) specifies the task, 3) hints to avoid generating a correct answer, 4) and 6) only appears in P3, 5) program description and solution, and 7) expected output format.

4.4 Assessing LLM-Generated Distractors

Using the designed prompts, each model was tasked with generating 10 distractors for each of the 38 challenges. To assess the quality of LLM-generated distractors, we attempted to match all distractors generated by different models for a specific challenge to distractors provided by human authors for the same challenge. Since all model-prompt combinations were requested to generate the same number of distractors, the more model-generated distractors that can be matched to human-authored distractors, the better

the model performs in generating distractors that were considered plausible by the human authors.

After the generated distractors were split into matched and non-matched groups, we evaluated each group separately using different approaches. The matched distractors generated by each approach were further split into functional and non-functional distractors. This split allowed us to compare approaches by their ability to replicate functional distractors authored by humans. We also compared the models by their ability to generate "top distractors". For this analysis, human-authored distractors were ranked based on the total number of attempts by students and then matched with generated distractors. The overlap with functional and top distractors was also averaged across challenges per topic to provide a holistic view of how each model and prompt performed.

To evaluate non-matched distractors, which can't be assessed using student interaction data, two authors reviewed and labeled all 2511 unique LLM-generated distractors that did not match human-authored distractors. The inter-rater agreement and the distribution of these labels are reviewed in the results section.

5 Results

In this section, the results and analysis of distractors generated by various models and prompts matching human-authored distractors are presented by topics and distractor types (functional, non-functional, and top-performing). GPT-4o, the best among these models, could generate 50% of all human-authored functional distractors, while Meta-Llama had the lowest match rate. OpenAI-o1 and GPT-4 performed slightly similar. Although a portion of these human-authored distractors were matched by these models, there is still a noticeable number of non-matched distractors. In comparing distractors generated by different n-values (asking the model different numbers of distractors to generate), matching all human-authored distractors may not be feasible or at least efficient; this should not be the ultimate goal of such generation tasks; rather, the focus should be on generating more functional distractors. We labeled the non-matched generations and revealed that a big portion of them can be potentially used as distractors but require further evaluation.

5.1 Matched Distractors

In the first step of our analysis, the generated distractors were matched to the human-authored distractors for each of the 38 challenges. To analyze the differences between models and prompts, we averaged the match per topic (Figure 6). The results show that the Meta-Llama model produced the lowest fraction of matched distractors, while GPT-4o (specifically P2) was the largest. GPT-4 and GPT-4o-mini compared to GPT-4o replicated a lower but comparable fraction of human-authored distractors. Model performance varied between the challenges, which produced performance outliers for small topics like *strings* and *var_ops*. OpenAI-o1 outperformed other models for the topic *var_ops* (variables and operations), which was the most challenging overall, with the average fraction of matched human-authored distractors lower than 10%. GPT-4o-mini outperformed others on the topic *strings*. We also observed that some prompts generated duplicated distractors in the same generation, so we also compared the models in this respect (Figure 7). The data show that OpenAI-o1 generated the least duplicates and Meta-Llama the most. This could partially explain the poor matching performance of Meta-Llama mentioned above: more duplicate distractors means fewer unique distractors that could match the human-authored distractors. Other models generated a noticeable number of duplicates just for 1-2 topics.

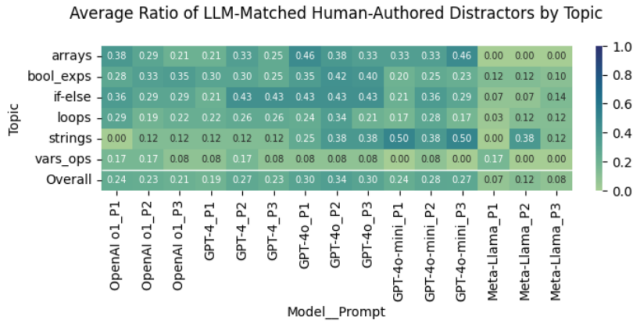

Figure 6: Average ratio (grouped by topic) of human-authored distractors that LLMs could match.

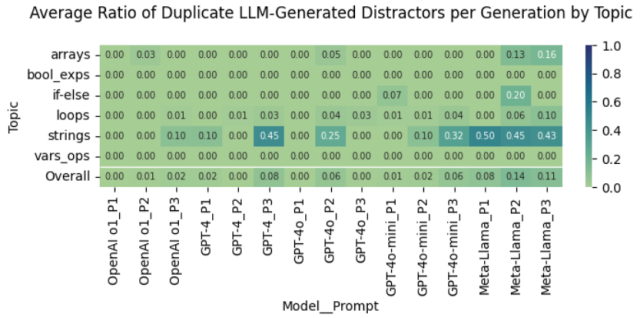

Figure 7: Average ratio (grouped by topic) of LLM-generated duplicate distractors within each generation.

The second step of model performance analysis focused on how the generated and matched distractors were split between functional and non-functional (Figure 8). In this analysis, a larger number of matched functional distractors indicates better performance. The data shows that GPT-4o with P2 produced both the highest total number of matched distractors and the highest number of functional distractors (31 functional and 25 non-functional), while Meta-Llama_P1 has the fewest (4 functional and 6 non-functional). At the model level, Meta-Llama has the lowest, while GPT-4o has the highest number of matched functional distractors. The remaining 3 models showed comparable solid performance, slightly lower than GPT-4o. At the prompt level, P2 generated the highest number of functional distractors with most models, with P3 being the second best. This data points to the value of guiding LLMs to misconception-related distractors. A better performance of P2 hints that LLMs' "own" perception of misconceptions was more helpful than a human-produced catalog of misconceptions [34]. However, in OpenAI-o1, misconception-oriented P2 and P3 produced fewer functional distractors than the simplest prompt P1. An interesting observation across all models and prompts is that the majority of pairs (including

all pairs for GPT-4o) generated a larger (or equal) number of functional than nonfunctional distractors. In contrast, with the same 25% thresholds for differentiating functional and non-functional distractors, humans authored noticeably more non-functional than functional distractors (Figure 4).

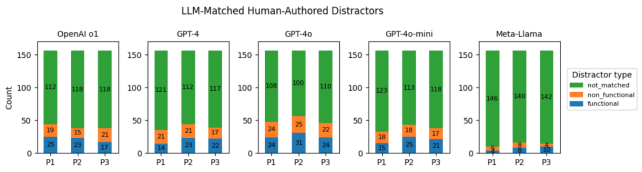

Figure 8: Number of human-authored functional and non-functional distractors that were matched by LLMs.

In addition to checking how many of the human-authored functional and non-functional distractors these models can generate, we also checked which model and prompt can match the top distractors. We ranked human-authored distractors based on the number of students attempting them on their first attempt. The very top distractor for each challenge is the one selected by the highest number of students. Figure 9 (the y-axis max value is 38, the total number of challenges) shows the number of distractors matched by each model and prompt for the best distractors with the rank 1 to 4. The results are consistent with the previous analysis. GPT-4o with P2 shows the best performance and GPT-4o performs best on the model level. On the prompt level, P2 performs best in most cases (for both functional and non-functional distractors) except for OpenAI-o1 where P1 shows the best performance.

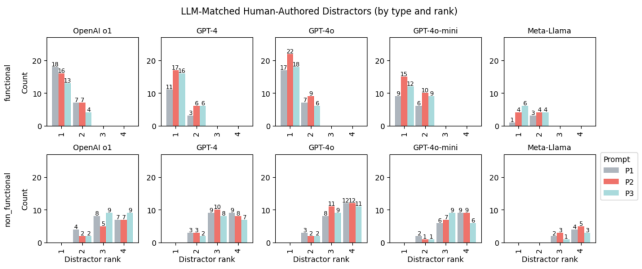

Figure 9: Number of human-authored distractors matched by LLMs at different distractor ranks. Distractor rank is defined by the number of students attempting it on the first challenge attempt, with 1 being the highest and 4 being the lowest rank.

5.2 Should We Ask LLM to Generate More Distractors?

As mentioned in Section 4.3, during our prompt tuning experiments, we explored whether we should ask LLM to generate a specific number of distractors. At that point, 10 distractors were selected as the best choice for the experiment across models. The emergence of the best-performing model (GPT-4o) encouraged us to explore this issue more extensively. We hypothesized that asking a model to generate a specific number of distractors (hereafter referred to as "n") would affect the number of matching distractors as well. Using GPT-4o, we generated distractors with different values of n (n=None, n=sameaschallenge, n=10, and n=20). In "n=None", we let the model decide the number of distractors to generate, while in "n=sameaschallenge", we asked for the same number of distractors that the human author provided for the challenge (typically, 4 distractors). Figure 10 shows that there is little difference between n=None and n=sameaschallenge for both total and functional matched distractors, however, requesting more distractors increases both numbers. We can see the most noticeable difference between asking for 4 (n=sameaschallenge) and 10 distractors, however, asking for 10 more (n=20) produces a much smaller additional gain. It seems that with n=20 we are close to hitting the maximum number of matching distractors GPT-4o can generate (all prompts generated 59 distractors, functional and non-functional combined). This data hints that n=10 selected for the main study might be the optimal number of distractors to request. It produces a good number of functional distractors (especially for P2) while not overburdening the human author in a human-AI collaborative authoring process by offering 20 candidate distractors to choose from.

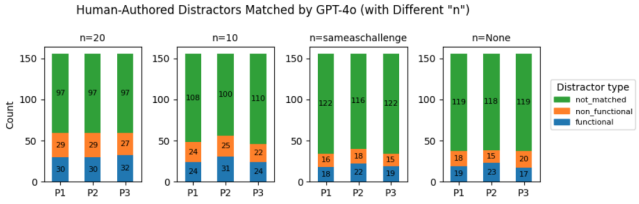

Figure 10: Number of human-authored distractors matched by GPT-4o (with different values of "n" – the number of distractors LLM is asked to generate).

5.3 Distractors Similarity

Using Levenshtein ratio (ranging from 0 to 1) as the similarity metric, we assessed how similar the generated distractors are to the correct answer (Figure 11) and to each other (Figure 12) within each challenge and generation. The data show that human-authored distractors are highly similar to the correct answer with little variation and also quite close to each other. One explanation for this is that instructors create new distractors by slightly modifying the correct answers or previous distractors, which stresses again that distractor generation is a challenging task. We also observe that OpenAI-o1 (P1), GPT-4 (P1, P2), and GP4-4o (P1, P2, P3) produce distractors that are considerably similar to the correct answer, with slightly more variation. P3, probably through its reference to a diverse misconception catalog, encourages all models to produce distractors that venture farthest from the correct answer on average and with larger variability (i.e., generated distractors range from very similar to very different from the correct answer). In terms of within-challenge distractor similarity, all models produce distractors that are much more diverse (dissimilar to each other) than human-authored distractors.

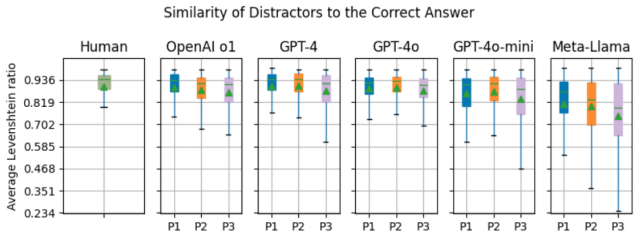

Figure 11: Average similarity of human-authored and LLM-generated distractors with the correct answer.

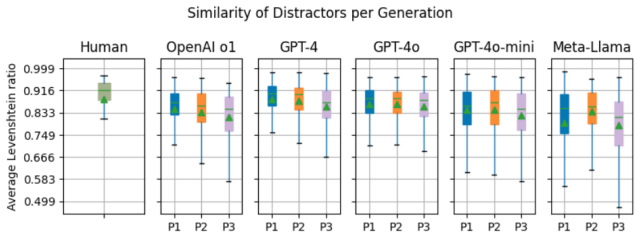

Figure 12: Average similarity (within each challenge) of human-authored and LLM-generated distractors.

5.4 Non-matched Distractors

A weak spot of our distractor-assessment approach based on existing data of student interaction with distractors is that we can't use it to assess LLM-generated distractors that do not match human-authored distractors. At the same time, Figure 13 shows that these non-matched distractors make up the majority of LLM-generated distractors. This "hidden part of the iceberg" can include great potential distractors as well as complete nonsense, so a reliable comparison of LLM distractor-generation ability can't be completed without at least some analysis of this part. To perform this analysis, all LLM-generated distractors that do not match human-authored distractors were labeled by one of the authors based on the following labeling code book:

- *Nonsense*, is a line that has obvious syntactical errors, disqualifying it as a distractor e.g.: "for (int i = 0; i < 8; i++"
- *Unnecessary statements*, is a line that contains obvious unnecessary statements making it stand out among other distractors and easily ignored, e.g.: "matrix[0][2] = 10; matrix[0][0] = 10;".
- *Correct answer*, is a line that correctly answers the challenge and should not be generated as a distractor.
- *Distractor*, is a line that is a valid potential distractor (anything that doesn't fall under the above categories).

To validate the labeling, a sample of 375 distractors was also labeled by a different author using this codebook. The inter-rater agreement between the two authors, as measured in Cohen's Kappa, is 0.934 indicating an almost perfect agreement. Figure 13 shows the full distribution of the generated distractors for all combinations, including matched functional and non-functional distractors and the non-matched distractors across these four labels. As the figure shows, the majority of the non-matched distractors can be used as potential distractors. However, their quality cannot be reliably assessed without a sizeable study with the target users. This "big picture" table shows that GPT-4o-mini and specifically Meta-Llama perform weaker than other models in most aspects. In particular, they generate a noticeable fraction of distractors with unnecessary statements that make them an obvious option for students to disregard. OpenAI-o1, while lagging behind GPT-4o in replicating functional distractors, generates the lowest number of correct answers and duplicates leaving the largest number of generated lines to be potential distractors. Interestingly, only OpenAI-o1 and GPT-4, unlike other models generated a few nonsense lines. We think this may be the result of the statement "*identifying logical, syntactical, or semantic errors*" in the expected output section of P1. GPT-4o, the leading model of the matching analysis, performs competitively overall with no unnecessary statements and nonsense lines generated. However, it has two weak spots: It generates a noticeable number of duplicates and correct answers as distractors. These distractors, however, could be easily filtered out in the authoring process giving us reasons to tentatively recommend this model for its excellent performance in generating top-matched distractors.

Distribution of Generated Distractors by Type

Model	Prompt	Matched Distractors		Non-Matched Distractors					Total
		Functional	Non-functional	Potential-distractors	Correct Answer	Unnecessary-statement	Nonsense	Duplicates	
OpenAI o1	P1	25 (6.53%)	19 (5.0%)	322 (84.1%)	12 (3.1%)	1 (0.3%)	4 (1.0%)	0 (0.0%)	383
	P2	23 (5.88%)	15 (3.8%)	336 (85.9%)	9 (2.3%)	0 (0.0%)	4 (1.0%)	4 (1.0%)	391
	P3	17 (4.35%)	21 (5.4%)	337 (86.2%)	10 (2.6%)	0 (0.0%)	0 (0.0%)	6 (1.5%)	391
GPT-4	P1	14 (3.67%)	21 (5.5%)	317 (83.2%)	16 (4.2%)	3 (0.8%)	8 (2.1%)	2 (0.5%)	381
	P2	23 (6.01%)	21 (5.5%)	307 (80.2%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	6 (1.6%)	383
	P3	22 (5.60%)	17 (4.3%)	283 (72.0%)	40 (10.2%)	3 (0.8%)	1 (0.3%)	27 (6.9%)	393
GPT-4o	P1	24 (6.32%)	24 (6.3%)	307 (80.8%)	25 (6.6%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	380
	P2	31 (7.99%)	25 (6.4%)	275 (70.9%)	37 (9.5%)	0 (0.0%)	0 (0.0%)	20 (5.2%)	388
	P3	24 (6.23%)	22 (5.7%)	301 (78.2%)	30 (7.8%)	0 (0.0%)	0 (0.0%)	8 (2.1%)	385
GPT-4o-mini	P1	15 (3.91%)	18 (4.7%)	306 (79.7%)	22 (5.7%)	15 (3.9%)	0 (0.0%)	8 (2.1%)	384
	P2	25 (6.46%)	18 (4.7%)	289 (74.7%)	35 (9.0%)	11 (2.8%)	0 (0.0%)	9 (2.3%)	387
	P3	21 (5.34%)	17 (4.3%)	297 (75.6%)	33 (8.4%)	8 (2.0%)	0 (0.0%)	17 (4.3%)	393
Meta-Llama	P1	4 (1.04%)	6 (1.6%)	298 (77.2%)	34 (8.8%)	28 (7.3%)	0 (0.0%)	16 (4.1%)	386
	P2	8 (2.04%)	8 (2.0%)	238 (60.6%)	30 (7.6%)	71 (18.1%)	0 (0.0%)	38 (9.7%)	393
	P3	10 (2.43%)	4 (1.0%)	209 (50.9%)	38 (9.2%)	93 (22.6%)	0 (0.0%)	57 (13.9%)	411

Figure 13: Distribution of generated distractors by their types; the percentage is calculated for each row (model and prompt). Each column is color-coded from red (worst) to green (best) except the "Total".

6 Discussion

Evaluating the quality of LLM output has become critical, yet it is challenging and expensive to perform at scale. In this work, we demonstrated a novel approach for evaluating LLM-generated artifacts based on student learning data. The approach potentially enables large-scale comparison of alternative models and prompting approaches. In our study, we focused on program construction challenges, simple coding problems where one line of the code was masked and students were asked to identify the correct code line from the given list of options (1 correct answer and 4 distractors). As student learning data, we used students' responses to the challenges, i.e., their selections of code lines to fill in the missing blanks. We used several LLMs to generate distractors for the same challenges. Then, LLM-generated distractors were matched to human-authored distractors. The matching enabled us to use data on the effectiveness of the distractors (i.e., functional vs. non-functional) extracted from student logs to assess a subset of LLM-generated distractors.

Our analysis revealed that human-authored distractors for these challenges are very similar to the correct answer and to each other.

While the tested LLMs and prompting approaches resulted in a subset of distractors comparable to the human ones in terms of similarity, the rest of the generated distractors had a higher variability – from very similar to very different distractors. Based on our analysis of the generated distractors, we hypothesize that there may be a limited number of human-authored distractors that these models can match. One possible explanation of this can be that instructors create distractors by small modifications of the correct answer and already created distractors. Although we found considerable overlap between human and LLM-generated distractors, a large portion is left non-matched. Our labeling revealed that most of these non-matched distractors are good distractor candidates, however, their quality requires empirical evaluation. We also found that LLMs sometimes generate correct answers as distractors despite our attempts to eliminate this behavior through prompt engineering. Interestingly, flagship models such as OpenAI-o1 or GPT-4 also occasionally generated very poor distractors that students could easily distinguish and disregard. It includes distractors with obvious syntax errors (mostly by the Meta-Llama) or with unnecessary statements distinguishable by their length. These observations point out that the best way to leverage LLM's ability to generate distractors is not through fully automatic generation but through a human-AI collaborative authoring process [11]. In this process, humans can delegate distractor generation to the LLM and, once generated, review, refine, and choose the best distractors for the programming challenge.

To have a clear comparison of the tested LLMs and prompting approaches, we categorized the human-authored distractors from the dataset into functional, non-functional, and top-performing distractors. Despite what was suggested by the literature [32], we used a more stringent threshold for this distinction; a functional distractor should be attempted by at least 25% of the examinees (higher than the chance of being picked randomly). Unlike human-authored distractors, where the number of non-functional distractors is higher than functional distractors, these models could match more of the functional distractors. At its best, GPT-4o_P2 could generate 50% human-authored functional distractors. Overall, GPT-4o_P2 was also the best-performing of all.

Since misconception is one of several factors negatively impacting a student's performance, we also designed a version of a prompt to encourage the LLM to generate misconception-based distractors. In P1, as the baseline, we asked the models to generate incorrect alternative lines. Except for OpenAI-o1, in all other selected models, considering students' common misconceptions for generating distractors resulted in more matched distractors. This number was higher in the prompt where the internal knowledge of the LLM was used (P2).

7 Limitations and Future Work

This work demonstrated an approach to evaluate the quality of LLM-generated artifacts and compare multiple alternative ways to generate them using student learning data. However, the approach has its limitations: it can only be applied to evaluate and compare LLM artifacts that match the student learning data. The distractors for which historical data is not available should be evaluated and compared using traditional approaches. In our future work, we plan to do a classroom study. Another limitation is that our dataset was limited to 38 challenges distributed unevenly across topics. This affects the generalizability of the results indicating the need for further studies. The data we used has an additional issue - many problem statements in this dataset were found to be vague and open-ended, allowing more than one correct answer. We think that instructors designed the problems with more specific goals in mind but neglected to include important details in the problem statement. These vague problem statements negatively affected LLM's ability to generate good distractors. We plan to address these limitations in our future work.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number 2213789.

References

[1] Giorgio Biancini, Alessio Ferrato, and Carla Limongelli. 2024. Multiple-Choice Question Generation Using Large Language Models: Methodology and Educator Insights. In *Adjunct Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization*. 584–590.

[2] Andre Del Carpio Gutierrez, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating Automatically Generated Contextualised Programming Exercises. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Portland OR USA, 289–295.

[3] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada). 1136–1142.

[4] Paul Denny, David H Smith IV, Max Fowler, James Prather, Brett A Becker, and Juho Leinonen. 2024. Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 283–289.

[5] 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 *26th Australasian Computing Education Conference*. ACM, Sydney NSW Australia, 114–123.

[6] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating large language models in class-level code generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[7] Wanyong Feng, Jaewook Lee, Hunter McNichols, Alexander Scarlatos, Digory Smith, Simon Woodhead, Nancy Otero Ornelas, and Andrew Lan. 2024. Exploring Automated Distractor Generation for Math Multiple-choice Questions via Large Language Models. arXiv:2404.02124 [cs.CL]

[8] James Finnie-Ansley, Paul Denny, Brett A Becker, Andrew Luxton-Reilly, and James Prather. 2022. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*. 10–19.

[9] Mark J. Gierl, Okan Bulut, Qi Guo, and Xinxin Zhang. 2017. Developing, Analyzing, and Using Distractors for Multiple-Choice Tests in Education: A Comprehensive Review. *Review of Educational Research* 87, 6 (Dec. 2017), 1082–1116.

[10] Christian Grévisse. 2024. Comparative Quality Analysis of GPT-Based Multiple Choice Question Generation. In *Applied Informatics*, Hector Florez and Marcelo Leon (Eds.). Springer Nature Switzerland, Cham, 435–447.

[11] Mohammad Hassany, Jiaze Ke, Peter Brusilovsky, Arun Balajiee Lekshmi Narayanan, and Kamil Akhuseyinoglu. 2024. Authoring Worked Examples for JAVA Programming with Human AI Collaboration. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*. ACM, 101–103.

[12] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the responses of large language models to beginner programmers' help requests. In *2023 ACM Conference on International Computing Education Research, Volume 1*. 93–105.

[13] Roya Hosseini, Kamil Akhuseyinoglu, Peter Brusilovsky, Lauri Malmi, Kerttu Pollari-Malmi, Christian Schunn, and Teemu Sirkiä. 2020. Improving Engagement in Program Construction Examples for Learning Python Programming. *International Journal of Artificial Intelligence in Education* 30, 2 (2020), 299–336.

[14] Xinying Hou, Zihan Wu, Xu Wang, and Barbara J. Ericson. 2024. CodeTailor: LLM-Powered Personalized Parsons Puzzles for Engaging Support While Learning

Programming. In *Eleventh ACM Conference on Learning @ Scale*. New York, NY, USA, 51–62.

[15] Shu Jiang and John Lee. 2017. Distractor Generation for Chinese Fill-in-the-blank Items. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, Joel Tetreault, Jill Burstein, Claudia Leacock, and Helen Yannakoudakis (Eds.). Association for Computational Linguistics, Copenhagen, Denmark, 143–148.

[16] 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*. 77–86.

[17] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *2023 CHI Conference on Human Factors in Computing Systems*. 1–23.

[18] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes de Oliveira Neto. 2024. Beyond code generation: An observational study of chatgpt usage in software engineering practice. *Proceedings of the ACM on Software Engineering* (2024), 1819–1840.

[19] Charles Koutcheme, Nicola Dainese, and Arto Hellas. 2024. Using Program Repair as a Proxy for Language Models' Feedback Ability in Programming Education. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*. 165–181.

[20] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024. Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 52–58.

[21] Charles Koutcheme and Arto Hellas. 2024. Propagating Large Language Models Programming Feedback. In *Proceedings of the Eleventh ACM Conference on Learning at Scale*. 366–370.

[22] Archana Praveen Kumar, Ashalatha Nayak, Manjula Shenoy K., Shashank Goyal, and Chaitanya. 2023. A novel approach to generate distractors for Multiple Choice Questions. *Expert Systems with Applications* 225 (Sept. 2023), 120022.

[23] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education*. 124–130.

[24] Chen Liang, Xiao Yang, Neisarg Dave, Drew Wham, Bart Pursel, and C. Lee Giles. 2018. Distractor Generation for Multiple Choice Questions Using Learning to Rank. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, New Orleans, Louisiana, 284–290.

[25] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. 1–11.

[26] Hunter McNichols, Wanyong Feng, Jaewook Lee, Alexander Scarlatos, Digory Smith, Simon Woodhead, and Andrew S. Lan. 2023. Automated Distractor and Feedback Generation for Math Multiple-choice Questions via In-context Learning.

[27] Arun Balajiee Lekshmi Narayanan, Priti Oli, Jeevan Chapagain, Mohammad Hassany, Rabin Banjade, Peter Brusilovsky, and Vasile Rus. 2024. Anonymized for blind review. In *AI for Education: Bridging Innovation and Responsibility at the 38th AAAI Annual Conference on AI*.

[28] Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. In *Proceedings of the 16th International Conference on Educational Data Mining*. International Educational Data Mining Society, 370–377.

[29] Siyu Ren and Kenny Q. Zhu. 2021. Knowledge-Driven Distractor Generation for Cloze-Style Multiple Choice Questions. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 5 (May 2021), 4339–4347.

[30] 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*. 27–43.

[31] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. ACM, New York, NY, USA, 78–92.

[32] Jinnie Shin, Qi Guo, and Mark J. Gierl. 2019. Multiple-Choice Item Distractor Development Using Topic Modeling Approaches. *Frontiers in Psychology* 10 (April 2019).

[33] David H. Smith and Craig Zilles. 2023. Discovering, Autogenerating, and Evaluating Distractors for Python Parsons Problems in CS1. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Toronto ON Canada, 924–930.

[34] Juha Sorva. 2012. *Visual Program Simulation in Introductory Programming Education*. Ph. D. Dissertation.

[35] Pragnya Sridhar, Aidan Doyle, Arav Agarwal, Christopher Bogart, Jaromir Savelka, and Majd Sakr. 2023. Harnessing LLMs in Curricular Design: Using GPT-4 to Support Authoring of Learning Objectives. *arXiv preprint arXiv:2306.17459* (2023).

[36] Kehui Tan, Tianqi Pang, and Chenyou Fan. 2023. Towards Applying Powerful Large AI Models in Classroom Teaching: Opportunities, Challenges and Prospects. arXiv:2305.03433 [cs.AI]

[37] 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)*. IEEE, College Station, TX, USA, 1–8.

[38] Des Traynor and J. Paul Gibson. 2005. Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education (SIGCSE '05)*. ACM, New York, NY, USA, 495–499.

[39] Annapurna Vadaparty, Daniel Zingaro, David H Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: Integrating LLMs into CS1 Instruction. In *2024 Conference on Innovation and Technology in Computer Science Education V. 1*. 297–303.

[40] Zihan Wu and David H. Smith. 2024. Evaluating Micro Parsons Problems as Exam Questions. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*. ACM, New York, NY, USA, 674–680.