ASCILITE 2026

The Future is Connection:

Collaboration, Creativity and Compassion in a Digital Society

Large language model generation of plausible semantic distractors for data science Parsons problems

Leying Zou, Eduardo Oliveira, Geela Venise Fabic Chee

The University of Melbourne

Parsons problems have been widely recognised to support novice learners by reducing the cognitive demands of writing code from scratch through rearrangement of provided code blocks. A common variation includes distractors – erroneous blocks of code which do not belong to the correct solution. While Parsons problems have been widely studied for introductory programming, they remain underexplored in various domains including in data science education, where students may possess statistical knowledge but limited programming experience. Developing distractors is also time-consuming for educators. This study investigated the feasibility of using a large language model (LLM) to generate semantic distractors for data science Parsons problems. A set of distractors was generated for 315 Python-based data science Parsons problems and was evaluated for pedagogical plausibility, compilability, and output divergence. Validated distractors were then incorporated as exemplars in a refined prompt to generate a second set of distractors, which were evaluated using the same criteria. The findings demonstrate that LLMs can be used to develop semantic distractors for data science Parsons problems and refining the prompts using validated exemplars further improved the quality. This study contributes to the limited literature on data science Parsons problems, distractor design and LLM-supported development of these problems.

Keywords: Parsons problems, distractors, data science, LLM, quantitative

Introduction and background

The ability to read, reason about, and construct code has become a foundational capability underpinning data analysis and algorithmic thinking across diverse disciplines (Grover & Pea, 2017; Wing, 2006). In data science (DS) education, this is challenging because many students come from non-computing backgrounds and must learn computational concepts with programming syntax (Liu et al., 2023; Robins et al., 2003). Furthermore, DS programming often involves logical errors that do not trigger run-time exceptions (Singh et al., 2024). Common DS programming errors, such as incorrect data aggregation, are considered semantic rather than syntactic, representing reasoning gaps that are difficult to detect and address at scale, especially for novice learners.

As a pedagogical tool, Parsons problems (Parsons & Haden, 2006) offer an accessible, learner-centred response. By providing learners with the scrambled lines or blocks of a correct solution, this format removes syntactic barriers, allowing students to concentrate on structure and logic (Bender et al., 2023). While well-established in introductory programming (Denny et al., 2008), their application in several domains, such as in DS, remains comparatively underexplored (Ericson et al., 2022). Unlike block-based environments like Scratch, which require students to eventually transition to a text-based syntax (Liao, 2023), Parsons problems can be authored directly in target languages like Python – a programming language commonly used in DS.

There are various types of Parsons problems, one of them is Parsons problems with distractors (Parsons & Haden, 2006), which contains incorrect blocks of code that do not belong to the correct solution. However, the pedagogical power of a Parsons problem with distractors depends heavily on the characteristics of the distractors rather than simply their inclusion. Distractors can be used as the primary lever for adjusting difficulty and highlighting learner misconceptions (Gierl et al., 2017; Smith & Zilles, 2023). Well-designed distractors prompt reflection and productive struggle (Kapur, 2008), while poorly designed ones merely add

cognitive load (Harms et al., 2016). Drawing on research into designing distractors for multiple-choice questions, effective semantic distractors can exploit method confusion or superficially consistent logic (Lee et al., 2025). Syntactically incorrect distractors may be appropriate for introductory programming when the objective is to target misconceptions about fundamental aspects such as syntax. However, in DS education, a distractor should focus on higher-order understanding rather than superficial syntactic errors. Thus, a plausible distractor in DS education must be "compilable" (runs without syntax errors) but functionally incorrect (produces an incorrect output), thereby mirroring authentic student mistakes.

While pedagogically valuable, manually authoring high-quality semantic distractors is a slow process demanding deep domain knowledge and pedagogical insight (Ericson et al., 2022). This limits their use, particularly in fast-moving fields like DS where evolving libraries continually generate new misconceptions. Large language models (LLMs) present a scalable alternative. Recent models demonstrate strong code comprehension and generation capabilities (OpenAI et al., 2024), and early applications, such as CodeTailor, have successfully used LLMs to generate personalised Parsons problems for introductory programming (Hou et al., 2024, 2025). Guided by prompting techniques like chain-of-thought (CoT) and few-shot (Brown et al., 2020; Wei et al., 2022), an LLM can act as a collaborative design assistant. CoT prompting is a technique that specifically encourages the AI model to break down its pedagogical reasoning behind the task before producing its final response. Few-shot prompting involves providing the AI with a small number of worked examples so that it can infer the desired pattern. Rather than replacing the educator's judgement, the AI generates candidate distractors that the educator validates. However, there is limited evidence on whether LLMs can reliably produce semantically meaningful distractors for DS rather than superficial syntactic variations.

This study investigates this gap by designing and evaluating a workflow that uses an LLM to generate plausible semantic distractors for DS Parsons problems, positioning the educator as the human-in-the-loop. This contributes to three underexplored areas in DS education: the application of Parsons problems, the development of semantic distractors for these problems, and LLM-supported development. In this workflow, the educator retains pedagogical control, validating the AI-generated outputs and making the final decision on whether a distractor is appropriate for deployment. We are guided by two research questions:

RQ1. Can an LLM be used to support the generation of plausible semantic distractors for data science Parsons problems?
RQ2. Does incorporating validated distractors improve the quality of generated distractors for data science Parsons problems?

By answering these questions, we aim to demonstrate how an LLM can be steered as a practical partner for educators that enables them to create high-quality distractors at scale while substantially reducing the manual effort required for their development.

Methodology

Study design, materials and ethics

This study employed a three-stage descriptive design to systematically generate and evaluate semantic distractors for DS Parsons problems. The process was divided into three distinct phases: (1) initial generation of distractors using an LLM; (2) evaluation of those distractors for structural and functional plausibility; and (3) refinement, which used distractors from Stage 2, validated by two of the authors serving as domain experts to guide the AI in generating a refined batch.

The dataset for this study was sourced from an existing automated Parsons problem platform developed at the University of Melbourne. All Parsons problems in the platform contain Python 3 programming code. To ensure the coding tasks reflected authentic data science practices, the platform combined seven core DS programming topics (e.g., Correlation, Linear Regression, Pandas DataFrames) with nine real-world application contexts (e.g.,

Sales Forecasting, Traffic Flow Analysis). These topics were extracted from the curriculum of a second-year undergraduate data science subject in the same university. Applying abstract programming concepts to realistic scenarios is essential; it mirrors how DS can be taught to students from non-computing backgrounds. This also allowed us to evaluate whether the LLM could generate distractors that were contextually relevant, rather than just technically incorrect.

Within this framework, five unique Parsons problem exercises, referred to throughout this paper as "puzzles," were created for each topic-context combination, resulting in a core dataset of 315 puzzles. For clarity, a single puzzle represents one complete Parsons problem task. Each puzzle contains all the foundational elements a student would need: a problem description, the correct lines of code (which would eventually be scrambled), the expected program output, and associated Comma Separated Values (CSV) data where applicable. These 315 puzzles served as the consistent baseline across all three stages of the study. By feeding the exact same 315 puzzles to the LLM in Stage 1 and Stage 3, we were able to conduct a direct comparison of the AI's distractor-generation performance before and after refinement.

This study used a GenAI-assisted methodology and did not involve human participants, student data, personal data, or identifiable information. Human involvement was limited to the authors evaluating the generated outputs. In accordance with the institutional requirements for human research, formal ethics approval was not required.

Stage 1: Initial generation of distractors

To address RQ1, a prompt containing the topic, context (e.g., Traffic Flow Analysis), question description, correct code, detailed distractor generation instructions, output constraints, and a predefined JSON output format for each puzzle was provided to Gemini-2.5-flash. A temperature of 0.7 was used to allow sufficient variation in the generated distractors while maintaining reasonable output consistency. The prompt incorporated zero-shot CoT prompting which required the AI model to specify its pedagogical reasoning behind each task before producing the final output. For each puzzle, the model was instructed to generate two logic-based distractors and two method-based distractors. A logic-based distractor changes the reasoning within a computational step while preserving the overall procedure. For example, the target procedure is mean error – specifically, Mean Absolute Error (MAE) – but a distractor used Mean Squared Error – a similar procedure but logically incorrect. In contrast, a method-based distractor replaces the computational method with a different one, such as instead of calculating sum, median is used. In total, 315 puzzles with four distractors each resulted in 1,260 initial distractors. The model was also prompted to generate a corresponding pedagogical rationale for each distractor, articulating the specific student misconception it intended to assess.

Stage 2: Evaluation of generated distractors

Stage 2 evaluated the functional plausibility of the generated distractors. First, each puzzle was verified whether it contained four semantically plausible distractors: two logic-based distractors and two method-based distractors. Second, distractors of the same type had to target the exact same line of correct code. For example, if the AI generated two logic distractors, both distractors should offer incorrect variations of the same correct line, instead of entirely different parts of the program.

To evaluate each distractor, we replaced the corresponding correct line with one generated distractor, producing a modified version of the original program. Then, it was further assessed through two program-level checks: compilability and output divergence. The first examined whether the modified program could still be compiled in a Python environment without any errors. The second assessed output divergence by comparing the modified program's output against the output of the correct solution. A distractor was considered output-divergent when the modified program produced a different result, indicating a notable change in the behaviour of the program, rather than a surface-level change. These checks ensure distractors were not superficially created but introduce a meaningful change in program behaviour that mirrors a genuine student error.

Stage 3: Refinement

To address RQ2, which is to determine whether incorporating validated distractors improves the development of distractors, a second round of generation was conducted. This was to assess whether distractor quality can be improved by providing the LLM with the same 315 puzzles provided in the LLM during Stage 1 combined with concrete examples of successful distractors from Stage 2. The purpose of Stage 3 was not to ask the AI to correct failed distractors from Stage 2, but to test whether validated exemplars from Stage 2 can help produce a completely new batch of 1,260 distractors with a higher success rate.

To achieve this, four highly plausible distractors (two logic-based and two method-based) that passed all checks in Stage 2 were manually selected as exemplars. The first exemplar showed an aggregation-related logic error, where a counting operation was used instead of a summation operation. The second example represented a method-selection error, where covariance was used instead of correlation. The third example also reflected method selection, but in a different context, where tokenisation was used instead of lemmatisation in a natural language processing task. The fourth example showed a logic-based parameter-order error, where the order of ground-truth and predicted values was reversed. These provided useful coverage of the main distractor patterns examined in this study, across different semantic error types relevant to data science programming tasks. Based on prior research, it is suggested that three to five examples is sufficient to guide LLM response patterns without unnecessarily inflating prompt length (Brown et al., 2020). These four were appended to the original Stage 1 prompt to act as “few-shot” exemplars to guide the AI in inferring the target pattern.

Quality control and validity

To rigorously measure whether the new results from Stage 3 truly improved the AI’s performance, the new distractors were extracted and evaluated using the exact same inclusion criteria and automated checks applied in Stage 2. Each refined distractor was paired with its corresponding correct code and subjected to identical inclusion criteria. Maintaining these strict guardrails ensured a fair comparison and guarantees that both datasets were structurally equivalent before undergoing functional testing. Additionally, to evaluate pedagogical plausibility (RQ1), two authors with relevant expertise independently reviewed a representative sample (approximately 20%) of the generated distractors and their accompanying objectives to assess whether they plausibly reflected common data science misconceptions. To ensure that RQ2 is carefully addressed to check whether any improvement was truly attributable to the validated few-shot exemplar distractors provided to the AI in the refinement stage, the exact same 315 puzzles were provided to the LLM in both Stages 1 and 3. This allowed a direct comparison of the AI’s distractor-generation performance before and after refinement. In the context of AI educational research where LLM outputs are inherently unpredictable, this level of experimental control isolates the contribution of the validated exemplar distractors in the refinement stage within the workflow, which provides a more rigorous way to answer our second research question.

Data analysis

To ensure a fair comparison, the same 315 puzzles used for initial generation during Stage 1 were also provided to the LLM in Stage 3 (refinement). Both Stage 2 (evaluation) and Stage 3 (refinement) also used the same calculations below to assess each distractor:

The proportion of distractors that compiled successfully, calculated as:

$$\textit{Compilability Rate} = \frac{\text{Number of distractors that compiled successfully}}{\text{Total number of generated distractors}} \times 100\%$$

The proportion of distractors that produced a divergent output, calculated as:

$$\textit{Output Divergence Rate} = \frac{\text{Number of output-divergent distractors}}{\text{Total number of generated distractors}} \times 100\%$$

In this study, a higher compilability rate indicates that more distractors structurally fit the original program, while a higher output divergence rate suggests more distractors produced an observable change in the program's result.

The sole difference between Stages 2 and 3 is the additional refinement in Stage 3 using few-shot prompting with validated distractors manually selected from Stage 2 results. To evaluate whether providing exemplar distractors using few-shot prompting resulted in an improvement, we compared the two stages by calculating the relative change for each metric:

$$Relative\ Change = \left(\frac{\text{Stage 3 Rate} - \text{Stage 2 Rate}}{\text{Stage 2 Rate}}\right) \times 100\%$$

Findings

Stage 1: Initial generation of distractors

To address RQ1, Stage 1 evaluated the LLM's baseline capacity to generate plausible distractors for DS Parsons problems. Specifically, the LLM was instructed to generate logic-based and method-based semantic distractors using zero-shot CoT prompting. This resulted in 1,260 initial distractors (315 puzzles each with two logic-based and two method-based distractors) with the specific pedagogical objectives. Beyond structural compliance, answering RQ1 requires assessing the pedagogical plausibility of these distractors within DS puzzles.

Table 1 shows examples of distractors produced by the LLM. In this example, the logic-based distractors alter the reasoning used to define the error term (e.g., removing the absolute value allows positive and negative errors to cancel out). Similarly, the method-based distractors alter the aggregation technique, changing the target metric from MAE to a Mean Squared Error or Root Mean Squared Error calculation. Because these distractors preserve a close surface relationship with the correct code while fundamentally changing the mathematical meaning, they are more meaningful for educators, instead of syntactically incorrect distractors.

Table 1.
An example of distractors and the corresponding correct code lines

Correct code line	Distractor	Type
errors = abs(data['actual'] - data['predicted'])	errors = data['actual'] - data['predicted']	Logic
errors = abs(data['actual'] - data['predicted'])	errors = (data['actual'] - data['predicted'])**2	Logic
mae = np.mean(errors)	mae = np.sum(errors)	Method
mae = np.mean(errors)	mae = np.sqrt(np.mean(errors**2))	Method

Table 2 further highlights the capability of the LLM across diverse data science contexts, showcasing the model's self-generated pedagogical objectives. Two authors independently reviewed these objectives and judged them to be pedagogically plausible representations of common novice misconceptions. For example, the distinction between the correct line and the distractor in the first row of Table 2 appears trivial: the correct line is to append the data at the end of the list, but the distractor is inserting it at the start. However, this alters the sequence of the resulting data, which is vital to maintain across all processing stages in a data science pipeline. The second row in Table 2 shows a common student mistake. The aim is to generate predictions using a model, but the input provided in the distractor is fundamentally different. The correct line only targets specific values, [13] – [15], while X likely represents an entire dataset which can significantly change the output. Overall, the Stage 1 findings confirm that an LLM can be used to successfully author plausible semantic distractors for DS puzzles (RQ1).

Table 2.
Examples of generated distractors and their objectives

Distractor	Correct line	Objectives
relevant_diseases.insert(0, disease)	relevant_diseases.append(disease)	Understand correct method selection for list manipulation (e.g., 'append' vs. 'insert') and ensure the correct data type/structure is being collected for subsequent processing.
predictions = model.predict(X)	predictions = model.predict(np.array([[13], [14], [15]]))	Understand correct method selection and parameter usage in applied data science contexts.; Recognize and correctly format input data (e.g., 2D arrays) for machine learning model prediction.

Stage 2: Evaluation of generated distractors

Of the 1,260 initial distractors generated in Stage 1, 750 (59.52%) successfully met all inclusion criteria for Stage 2 (Table 3). Notably, the model demonstrated highly balanced performance across both logic-based and method-based categories, retaining 59.05% of logic-based and 60.00% of method-based distractors. This indicates that zero-shot CoT prompting is moderately effective at generating structurally compliant distractors for Parsons problems, without inherently biasing one error type over the other.

Table 3.
Semantic filtering of results by distractor type

Distractor type	Total	Semantically valid	Rate (%)
Logic	630	372	59.05%
Method	630	378	60.00%
Total	1,260	750	59.52%

Stage 2 also evaluated the functional plausibility of the 750 filtered distractors. This automated phase tested the distractors strictly on their ability to integrate into the original codebase without errors (compilability) and their ability to fundamentally alter the program's behaviour (output divergence). Slightly more than half (57.87%) of the zero-shot generated distractors compiled successfully when inserted into the original code. For output divergence rate, it resulted in a similar rate (52.53%) where the modified puzzle which included the distractors produced an output different to the original expected output. However, the most critical finding emerges from the divergence-to-compile ratio: from the distractors that survived the compilation check, 90.78% successfully produced an output divergent from the correct solution.

These results reveal a distinct pattern in the LLM's zero-shot generation capabilities: its semantic potential is significantly stronger than its syntactic stability. The retained distractors reflect common areas of novice difficulty such as confusing related evaluation metrics or selecting plausible but incorrect Pandas functions, which are essential characteristics of high-quality Parsons problems. The primary attrition in the workflow occurred at the compilability stage, where a substantial proportion of distractors failed to run within the strict structural confines of the existing puzzle code. Yet, once a distractor cleared this structural hurdle, it was highly likely to introduce a meaningful, observable change in program logic rather than a superficial variation. Consequently, while zero-shot generation requires rigorous structural filtering, the distractors that survive are functionally highly viable for deployment in DS Parsons problems. From the valid distractors from this stage, four plausible distractors were manually selected as exemplars for the next stage – refinement.

Stage 3: Refinement

To directly answer RQ2, Stage 3 examined whether providing the LLM with manually validated, exemplar distractors from Stage 2 using few-shot prompting has improved the overall quality of the generated distractors. The zero-shot CoT outcomes from Stage 2 which used the same metrics (compilability rate and output divergence rate) served as the baseline for this comparison.

Table 4 summarises the overall performance shift between the two prompting strategies: zero-shot compared to few-shot. To capture the true impact of the refinement, the relative gain was calculated by dividing the absolute difference between the stages by the Stage 2 baseline. The inclusion of few-shot examples yielded a clear, positive impact on general usability. The compilability rate saw a notable relative gain of 10.91%, indicating that the exemplars successfully guided the LLM to produce distractors that better adhered to the strict structural and syntactic constraints of the original puzzle code. The output divergence rate also improved, though by a more modest relative margin of 5.78%.

Table 4.
Comparison of distractor evaluation results between Stage 2 and Stage 3

Dimension	Stage 2	Stage 3	Gain (%)
Compilability Rate	0.5787	0.6417	10.91%
Output Divergence Rate	0.5253	0.5557	5.78%

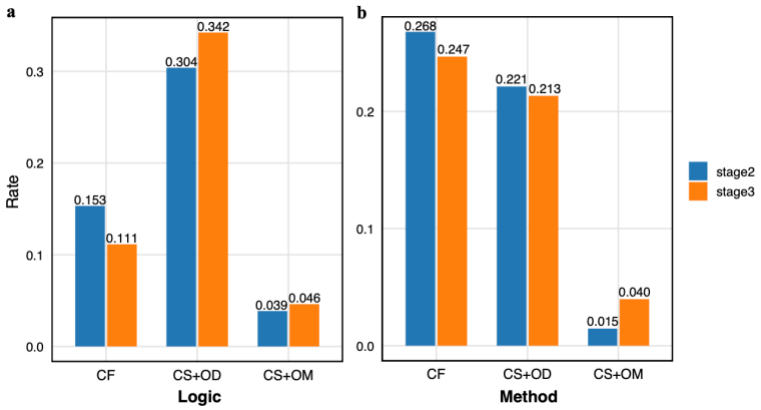


Figure 1. Comparison of distractor plausibility outcomes between Stage 2 and Stage 3 for (a) logic-based and (b) method-based distractors. Categories include Compile Failed (CF), Compile Success with Output Divergence (CS+OD), and Compile Success with Output Match (CS+OM)

However, a deeper analysis of the evaluation outcomes by distractor type (Figure 1) reveals that this improvement was not uniform when logic-based and method-based distractors were compared. Distractor outcomes were categorised into three distinct groups: Compile Failed (CF), Compile Success with Output Divergence (CS+OD; the ideal diagnostic outcome), and Compile Success with Output Match (CS+OM; where the code runs but fails to alter the final result, rendering it ineffective as a distractor). For logic-based distractors (Figure 1a), the few-shot refinement was highly effective. The proportion of compile failures (CF) decreased from 15.3% in Stage 2 to 11.1% in Stage 3. More importantly, this structural improvement translated directly into functional pedagogical value: successful, output-divergent cases (CS+OD) increased from 30.4% to 34.2%. This indicates that the few-shot examples successfully taught the model how to alter the underlying program reasoning without breaking the syntax.

Conversely, method-based distractors (Figure 1b) exhibited a different pattern. While few-shot prompting successfully reduced compile failures (CF dropping from 26.8% to 24.7%), this newly gained syntactic stability

did not translate into functional divergence. In fact, ideal CS+OD distractors decreased slightly from 22.1% to 21.3%, while ineffective CS+OM distractors more than doubled (from 1.5% to 4.0%). This suggests that when guided by method-based exemplars, the model learned to produce highly syntactically valid code, but frequently substituted methods that ultimately resulted in the exact same mathematical output (e.g., substituting one valid scaling function for another that achieves the same net result).

In summary, regarding RQ2, including validated few-shot examples reliably improves the syntactic validity (compilability) of all generated distractors. However, it only improves the semantic plausibility (output divergence) of logic-based distractors. For method-based generation, zero-shot CoT prompting remains marginally more effective at producing functionally divergent errors than few-shot.

Discussion and future work

This study investigated the feasibility of using an LLM to generate semantic distractors for data science Parsons problems (RQ1). In doing so, it contributes to three underexplored areas in the literature: the use of Parsons problems in DS education (Ericson et al., 2022), the design of semantic distractors for these problems, and AI-supported development specifically for DS Parsons problems. Prior work has examined block-based programming with Scratch in DS education (Liao, 2023), while Hou et al. (2024, 2025) used LLMs to generate Parsons problems for introductory programming. However, limited attention has been given to the use of LLMs to generate semantic distractors specifically for DS Parsons problems. Although this study does not establish whether Parsons problems are as effective in the DS domains as they are in other domains, it addresses an earlier design challenge by demonstrating the feasibility of using LLMs to develop semantic distractors for these problems. This provides a practical foundation for future studies evaluating their use with learners.

The findings demonstrate that an LLM is highly capable of generating structurally viable and functionally divergent distractors when using zero-shot CoT prompting. However, our investigation into including manually validated distractors as exemplars in few-shot prompting (RQ2) revealed a critical nuance in AI-assisted learning design. Providing the LLM with validated pedagogical exemplars universally improved the syntactic validity (compilability) of the output, but its impact on semantic plausibility (output divergence) depended entirely on the distractor type. The improvement in compilability is broadly consistent with research showing that worked examples and reasoning prompts can guide model outputs towards a desired task pattern (Brown et al., 2020; Wei et al., 2022). Few-shot prompting successfully taught the model to produce better logic-based errors, but it inadvertently caused the model to over-optimise for syntax in method-based tasks, frequently resulting in code that compiled but failed to change the mathematical output. This highlights that GenAI cannot be treated as a monolithic solution; prompt engineering must be carefully tailored to the specific cognitive skill being assessed.

For educators, these findings offer a scalable solution to a significant pedagogical bottleneck. Authoring high-quality semantic distractors manually is time-intensive and requires deep domain expertise (Ericson et al., 2022). By using an LLM to generate plausible initial candidates for educator validation, this human-in-the-loop approach preserves human pedagogical control while reducing authoring effort. This partnership makes it practical to deploy distractor-rich Parsons problems widely, even in fast-moving fields like DS where new libraries and functions continually emerge.

For learners, output-divergent distractors may support productive struggle described by Kapur (2008), by requiring them to determine why plausible-looking code produces an incorrect result. When an AI generates a distractor that runs cleanly but calculates a mathematically incorrect result, it mirrors the exact, silent semantic errors that frustrate novice DS students. Engaging with these carefully curated distractors encourages vital metacognitive reflection, allowing students to focus on why a specific analytical method or logical sequence is flawed without being derailed by superficial errors such as missing commas or bracket errors.

This study is constrained by its scope. The dataset, while spanning multiple contexts, was limited to 315 puzzles derived from a single automated platform and reflected the curriculum of one specific DS course. Consequently, the findings may not generalise to more advanced, open-ended machine learning workflows or different programming languages used in DS (e.g., R). Furthermore, the evaluation relied primarily on automated proxy metrics (compilability and output divergence) alongside review by two authors serving as evaluators, rather than on observing actual student interactions with the generated distractors.

The natural progression of this research is to move from pipeline evaluation to in-vivo classroom deployment. Future classroom studies should examine whether these distractors support productive struggle, improve learning outcomes, and reduce attention to syntactic errors without creating unnecessary cognitive load.

References

Bender, J., Zhao, B., Dziena, A., & Kaiser, G. (2023). Integrating Parsons puzzles within Scratch enables efficient computational thinking learning. *Research and Practice in Technology Enhanced Learning*, *18*, 022–022. https://doi.org/10.58459/rptel.2023.18022

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. *Proceedings of the Fourth International Workshop on Computing Education Research*, 113–124. https://doi.org/10.1145/1404520.1404532

Ericson, B. J., Denny, P., Prather, J., Duran, R., Hellas, A., Leinonen, J., Miller, C. S., Morrison, B. B., Pearce, J. L., & Rodger, S. H. (2022). Parsons problems and beyond: Systematic literature review and empirical study designs. *Proceedings of the 2022 working group reports on innovation and technology in computer science education*, 191–234. https://doi.org/10.1145/3571785.3574127

Gierl, M. J., Bulut, O., Guo, Q., & Zhang, X. (2017). Developing, analyzing, and using distractors for multiple-choice tests in education: A comprehensive review. *Review of Educational Research*, *87*(6), 1082–1116. https://doi.org/10.3102/0034654317726529

Grover, S., & Pea, R. (2017). *Computational thinking: A competency whose time has come*.Bloomsbury. https://doi.org/10.5040/9781350057142.ch-003

Harms, K. J., Chen, J., & Kelleher, C. L. (2016). Distractors in Parsons problems decrease learning efficiency for young novice programmers. *Proceedings of the 2016 ACM conference on international computing education research*, 241–250. https://doi.org/10.1145/2960310.2960314

Hou, X., Wu, Z., Wang, X., & Ericson, B. J. (2024). CodeTailor: Personalized Parsons puzzles are preferred over AI-generated solutions to support learning. *arXiv*: 2401.12125; Version 2. https://doi.org/10.48550/arXiv.2401.12125

Hou, X., Wu, Z., Wang, X., & Ericson, B. J. (2025). Personalized Parsons puzzles as scaffolding enhance practice engagement over just showing LLM-powered solutions. *arXiv*, 2501.09210; Version 1. https://doi.org/10.48550/arXiv.2501.09210

Kapur, M. (2008). Productive failure. *Cognition and Instruction*, *26*(3), 379–424. https://doi.org/10.1080/07370000802212669

Lee, Y., Kim, S., & Jo, Y. (2025). Generating plausible distractors for multiple-choice questions via student choice prediction. *Proceedings of the 63rd annual meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 23669–23692. https://aclanthology.org/2025.acl-long.1154/

Liao, S.-M. (2023). Scratch to R: Toward an inclusive pedagogy in teaching coding. *Journal of Statistics and Data Science Education*, *31*(1), 45–56. https://doi.org/10.1080/26939169.2022.2090467

Liu, X., Golen, E., Raj, R. K., & Fluet, K. (2023). Offering data science coursework to non-computing majors. *Proceedings of the 2nd international workshop on data systems education: Bridging education practice with education research*, (pp. 44–49). https://doi.org/10.1145/3596673.3596971

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., … Zoph, B. (2024). GPT-4 technical report. *arXiv*, 2303.08774. https://doi.org/10.48550/arXiv.2303.08774

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian conference on computing education - Volume 52, ACE '06*, *52*, 157–163. https://dl.acm.org/doi/10.5555/1151869.1151890

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Singh, A., Fariha, A., Brooks, C., Soares, G., Henley, A. Z., Tiwari, A., M, C., Choi, H., & Gulwani, S. (2024). Investigating student mistakes in introductory data science programming. *Proceedings of the 55th ACM technical symposium on computer science education V. 1*, 1258–1264. https://doi.org/10.1145/3626252.3630884

Smith, D. H., & Zilles, C. (2023). Discovering, autogenerating, and evaluating distractors for Python Parsons problems in CS1. *Proceedings of the 54th ACM technical symposium on computer science education V. 1*, 924–930. https://doi.org/10.1145/3545945.3569801

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837. https://doi.org/10.48550/arXiv.2201.11903

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Zou, L., Oliveira, E.[†] & Chee, G. V. F.[†] (2026). Large language model generation of plausible semantic distractors for data science Parsons problems. In Cochrane, T., Narayan, V., Poole, T., Lai, P., Lake, P., Moore, O., & Li, L. (Eds.), *The Future is Connection: Collaboration, Creativity and Compassion in a Digital Society*. Proceedings ASCILITE 2026. Brisbane (pp. 441–450). https://doi.org/10.65106/apubs.2026.3251

[†]These authors contributed equally to this work.

Note: All published papers are refereed, having undergone a double-blind peer-review process.
The author(s) assign a Creative Commons by attribution license enabling others to distribute, remix, tweak, and build upon their work, even commercially, as long as credit is given to the author(s) for the original creation.

© Zou, L., Oliveira, E, & Chee, G. V. F. 2026