Enhancing Diversity of LLM-Generated Educational Tasks

Manh Hung Nguyen
MPI-SWS
manguyen@mpi-sws.org

Sebastian Tschiatschek
University of Vienna
sebastian.tschiatschek@univie.ac.at

Adish Singla
MPI-SWS
adishs@mpi-sws.org

ABSTRACT

Large language models (LLMs) have shown the potential for generating educational content at scale, assisting educators in creating practice tasks or synthesizing data for training educational models. However, LLMs suffer from the "Artificial Hivemind" effect, where they produce homogeneous content. This homogeneity limits the diversity of LLM-generated tasks, a crucial factor in these educational settings. In this paper, we investigate how to increase the diversity of generated tasks while keeping their utility high. Inspired by the divergent–convergent thinking stages in creativity literature, we propose a prompting framework with two reasoning stages: (1) exploring the creative space, and (2) satisfying the input requirements. We evaluate CreativeDC, a method instantiated from this framework in the domain of Python programming, using both automated metrics and expert evaluation. Results show that CreativeDC produces significantly more distinct high-utility tasks (about 1.6×) than baselines. Our work offers an effective approach for generating and evaluating more diverse tasks at scale.

Keywords

Large Language Models, Task Generation, Programming Education, Diversity, Divergent–Convergent Thinking

1. INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable capabilities, leading to their rapid adoption across important domains such as education [1–14]. In particular, prior work has used LLMs to generate programming tasks [15–18], enabling active learning strategies such as the "doer effect" through task generation at scale [19, 20]. Moreover, LLMs can also be used to generate synthetic data, for example, for training tutor models [12, 14, 21]. In such settings, maintaining task diversity is crucial to adequately cover learning objectives and different difficulty levels. Yet LLMs have been shown to exhibit an "Artificial Hivemind" effect [22], where they tend to produce similar and repetitive outputs within a single model and across different models, resulting in limited output diversity.

Existing work has proposed various methods to address the limited diversity and homogeneity of LLM outputs. These include fine-tuning through preference optimization with creativity signals [23] and using multi-agent discussion or debate frameworks [24]. However, these approaches are typically designed and evaluated for benchmarks such as creative writing or general instruction following, rather than the structured requirements of educational task generation. Another line of work has incorporated thematic topics and learning concepts as input to LLMs to generate a wide range of tasks, e.g., in programming education [15–17]. However, these efforts do not primarily focus on optimizing for task diversity. This gap motivates our central question: *How can we generate a diverse set of high-utility practice tasks*?

In this work, we tackle this question in the context of programming task generation. Drawing inspiration from the creativity literature and the divergent-convergent thinking stages [25–27], we propose a two-stage prompting framework and instantiate CreativeDC, a method for generating diverse contextualized programming tasks. Given a theme and a programming concept as context, CreativeDC instructs the LLM to first go through a *divergent thinking stage*, where it explores the creative space of ideas with respect to the input theme. Then, in the following *convergent thinking stage*, the model refines promising ideas into a valid programming task using the given programming concept. This decoupling encourages the LLM to explore a broader ideation space before committing to a final task. Our main contributions are:

1. We propose a novel prompting framework with divergent-convergent thinking stages and develop CreativeDC, a method for generating diverse programming tasks.
2. We define and propose approaches to evaluate effective diversity, which measures the number of distinct tasks among high-utility tasks.
3. We show through automated and expert evaluations that CreativeDC achieves significantly higher effective diversity than baseline methods.

Figure 1 shows an overview of our method. We release our code and data publicly to support future research.[1]

[1]github.com/machine-teaching-group/edm2026-creativedc.

Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. Enhancing Diversity of LLM-Generated Educational Tasks. In Anthony Botelho, Maria Mercedes T. Rodrigo, Adish Singla, Hiroaki Ogata, Hyojeong So, and Young Hoan Cho (eds.) Proceedings of the 19th International Conference on Educational Data Mining, Seoul, Republic of Korea, June, 2026, pp. 657–663. International Educational Data Mining Society (2026).

https://doi.org/10.5281/zenodo.21039728

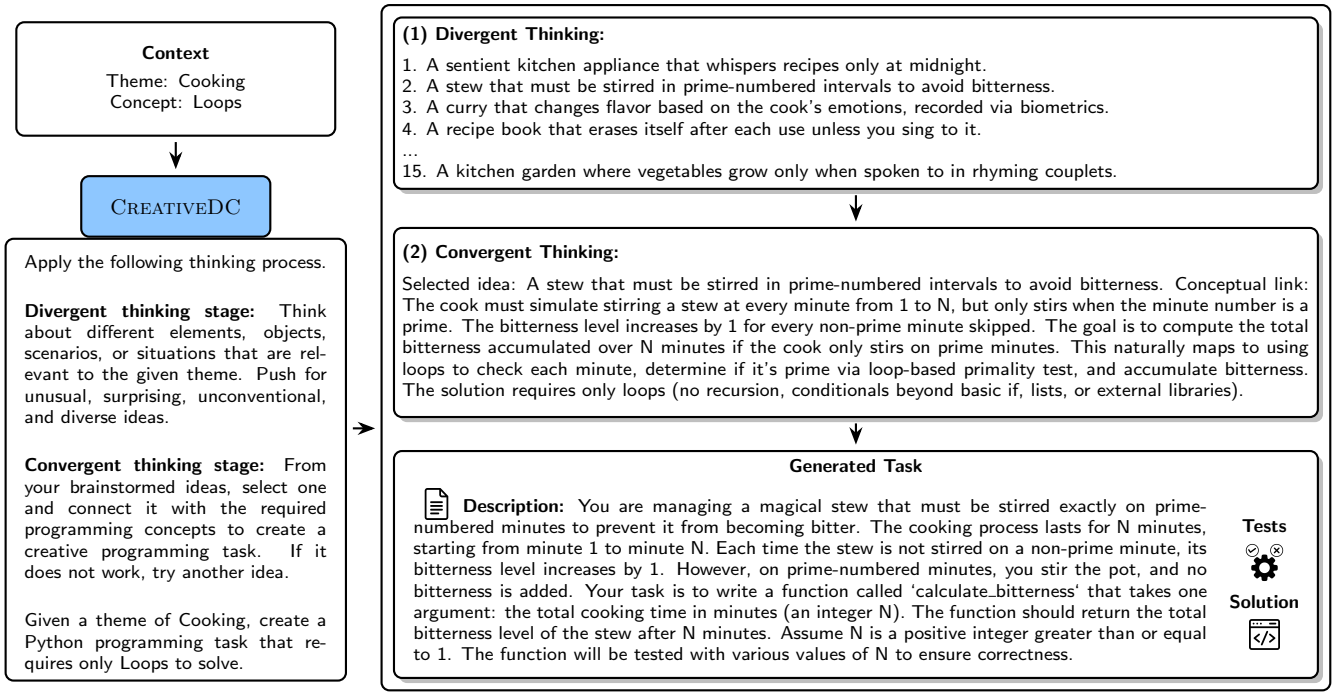

Figure 1: Overview of CreativeDC for programming task generation. The input is a context containing a theme and a programming concept. Our method instructs an LLM to apply a two-stage reasoning process: *(1) Divergent thinking stage* and *(2) Convergent thinking stage*. The generated task consists of a description, a test suite, and a solution.

2. RELATED WORK

LLMs for Task Generation. LLMs have been used to generate educational tasks across domains, including topic-specific questions, mathematics word problems, and programming exercises [2–9]. In programming education, recent work has generated multiple-choice items and contextualized tasks [15–18]. However, these works primarily optimize task correctness or difficulty alignment, whereas we also consider the diversity of generated tasks.

Diversity Collapse of LLMs. LLMs exhibit an "Artificial Hivemind" effect, characterized by high intra-model and inter-model homogeneity [22]. Post-training alignment further amplifies this convergence [28] and reduces LLM creativity and output novelty [29]. Moreover, LLMs have been shown to lack novelty [23, 29] and suffer from "functional fixedness", a bias that limits unconventional thinking [30]. These findings motivate methods that explicitly scaffold exploration rather than direct prompting alone.

Improving Diversity of LLM Outputs. Methods have been proposed to promote diversity in LLM outputs, including fine-tuning with creativity signals [23], multi-agent frameworks [24, 31], and persona simulation for diverse perspectives [21]. However, these methods are primarily designed for open-ended generation settings such as creative writing. Our work instead targets educational task generation, where outputs must balance diversity with utility.

3. PROBLEM SETUP

We formalize the problem of generating diverse tasks, discuss the metrics for evaluation, and state our objective.

Preliminaries. We denote a programming task as a tuple $\mathcal{P} = (\mathcal{P}_{\text{desc}}, \mathcal{P}_{\text{eval}}, \mathcal{P}_{\text{sol}})$, where $\mathcal{P}_{\text{desc}}$ is a natural-language task description, $\mathcal{P}_{\text{eval}}$ is an evaluation guide or final answer for checking the correctness of any answer, and $\mathcal{P}_{\text{sol}}$ is a reference solution. Given a context $\mathcal{C}$, which specifies the requirements for task generation, and an integer K, we seek to generate a set of tasks that satisfy $\mathcal{C}$. We denote the generated set of K tasks as $\mathcal{S}(\mathcal{C}) = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K\}$. For example, if $\mathcal{C}$ corresponds to a learning concept, we aim to generate K tasks that help students practice that concept.

Metrics. We evaluate a task set $\mathcal{S}(\mathcal{C})$ along the following metrics. *Diversity* captures the degree of task variation and is defined as the number of semantically distinct tasks in $\mathcal{S}(\mathcal{C})$. *Utility* captures whether a task $\mathcal{P} \in \mathcal{S}(\mathcal{C})$ is valid, comprehensible, and relevant to the given context. We assign $\text{Utility}(\mathcal{P}) = 1$ if it meets these criteria and 0 otherwise. We define the subset of high-utility tasks as $\mathcal{S}_{\text{util}=1}(\mathcal{C}) = \{\mathcal{P} \in \mathcal{S}(\mathcal{C}) \mid \text{Utility}(\mathcal{P}) = 1\}$. We define *Effective Diversity* as the diversity of this high-utility subset. Intuitively, it measures the number of distinct high-utility tasks. We detail the measurement of these metrics in Section 5.

Objective. Our goal is to design a method that, given a context $\mathcal{C}$, produces a set of K tasks $\mathcal{S}(\mathcal{C})$ with high *effective diversity*. This objective is challenging because encouraging diverse outputs can conflict with the utility constraints [32].

In this work, we instantiate this setup for the generation of programming tasks given a context, following prior work on contextualized programming tasks [15–17]. Figure 1 illustrates an example context and a generated task.

Table 1: Prompts for all methods. {placeholder} is replaced with the actual theme and concept for each context.

BASE
Task Instruction Given a theme of {theme}, create a Python programming problem that requires only {concept} to solve. The problem should include a problem description, a test suite, and a solution program. Below are the requirements for the problem: • The problem must be clearly relevant to the given theme of {theme} and the theme is explicitly used throughout. It requires only {concept} to solve the problem. • The problem description must be sensible and sound natural. It must provide comprehensive information required to solve the problem and pass the test suite (e.g., how the program will be tested, function signatures). Do not use type hints. Do not mention the required programming concepts in the problem description. • The test suite must consist of at least 5 comprehensive test cases written in the Pytest framework format. The testsuite must be correct and cover both base and corner cases. If the test suite involves handling files and I/O, the related files should be created using 'setup_module()' and removed using 'teardown_module()' functions in the Pytest framework. Everything from the solution program will be imported manually; do not import anything else except 'pytest' and 'os'. Do not use multiple assert statements in a single test case. • The solution program must use only {concept}. Do not include any comments, usage examples, or tests in the solution program. The solution program must pass the test suite. # Output Format Output a JSON object with the following keys: `'description'`, `'test_suite'`, and `'solution'`.
COT
Think step by step to generate a problem. # Task Instruction (same as in BASE) # Output Format Output a JSON object with the following keys: `'chain_of_thought'`, `'description'`, `'test_suite'`, and `'solution'`.
CREATIVEDC
Apply the following thinking process to generate a problem. *Divergent thinking phase:* Think about only the given theme and list down wildly different and underexplored elements, objects, scenarios, or situations that are relevant. Ignore the required programming concepts in this phase. Push for unusual, surprising, unconventional, and diverse ideas. Explore the creative space related to the theme as much as possible. *Convergent thinking phase:* From your brainstormed ideas, select one and connect it with the required programming concepts to create a creative programming problem. Make sure the problem does not require any other programming concepts other than the given programming concepts. If it does not work, feel free to go back to select another idea and try again. # Task Instruction (same as in BASE) # Output Format Output a JSON object with the following keys: `'divergent_thinking'`, `'convergent_thinking'`, `'description'`, `'test_suite'`, and `'solution'`.

4. OUR METHOD: CREATIVEDC

We draw inspiration from creativity literature and staged models [25–27] to propose a two-stage prompting framework that structures the model's reasoning into *divergent thinking* and *convergent thinking* stages. Divergent thinking aims to generate many diverse ideas without judgment, while convergent thinking selects and refines the best ideas into a concrete solution. We instantiate this framework as CREATIVEDC (**Creative D**ivergent-**C**onvergent) (cf. Figure 1).

Stage 1: Divergent Thinking. In this stage, CREATIVEDC instructs the LLM to explore the thematic space of context $\mathcal{C}$ without imposing any constraints from the target programming concept in $\mathcal{C}$. By focusing only on the theme, the model is encouraged to brainstorm varied and underexplored elements, objects, scenarios, or situations. Pushing for unusual, surprising, and unconventional ideas helps the model traverse a broader semantic space. The goal of this stage is to avoid the premature convergence often seen in direct prompting, where the model settles too quickly on ideas that satisfy the technical requirements but offer limited variety.

Stage 2: Convergent Thinking. In the second stage, CREATIVEDC instructs the model to select one promising idea from the brainstormed list and refine it into a valid programming task $\mathcal{P}$. The output must satisfy all requirements of $\mathcal{C}$ and, crucially, primarily test the specified programming concept. If the selected idea cannot be developed into a valid task, the model is encouraged to return to the brainstormed list and try another idea. In this way, the convergent stage uses the broader idea pool produced by Stage 1 while enforcing the constraints required for utility.

After going through the two stages, the model outputs the final task $\mathcal{P} = (\mathcal{P}_{\text{desc}}, \mathcal{P}_{\text{eval}}, \mathcal{P}_{\text{sol}})$. Figure 1 illustrates the two stages of our method with an example generated task.

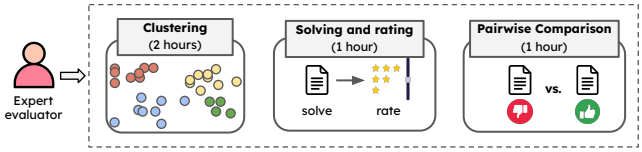

Figure 2: Overview of Expert Evaluation.

5. EVALUATION SETUP

In this section, we present an evaluation setup designed to answer the following questions: **(i)** How does effective diversity scale with the number of generated tasks? **(ii)** How do expert evaluators perceive the generated tasks?

Task Contexts. We use a set of 20 task contexts, each of which is a pair of a theme and a target programming concept following prior work on contextualized tasks [15–17]. We use 4 themes ("Cooking", "Science Fiction", "Superheroes", and "Board Games") and 5 concepts ("Variables", "Selection Statements", "Loops", "Lists", and "Strings"). For each context, we generate K tasks per method. We report the results for $K \in \{10, 20, 30, 40, 50\}$ in Section 6.

Methods. We compare CreativeDC (described in Section 4) against two baselines: Base and CoT. Base directly prompts the model to generate a task from the given theme and concept. CoT prepends a "Think step by step" instruction to the Base prompt to elicit intermediate reasoning [33]. Table 1 shows the prompts for all methods. For all methods, we use Qwen3-235B-A22B [34], a strong open-weight model, enabling reproducible generation. We set the temperature to 1.0.

Expert-based Evaluation Overview. We use a mix of automated and expert-based evaluation as further discussed in Section 5.1 and Section 5.2. We present an overview of our expert evaluation for assessing diversity and multiple task aspects in Figure 2. More concretely, we recruited three experts with experience in programming education and tutoring in programming-related courses. All three are non-native English speakers (one female, two males; mean age 28.7 years), and none are authors of this paper. Participation was voluntary, and no personally identifiable data was collected. Each expert evaluator was assigned the same programming concept ("Variables") and a randomly sampled theme. We use small subsets of tasks generated during the automated evaluation to reduce the workload. We exclude Base from expert evaluation due to similar performance to CoT in automated evaluations (cf. Section 6). To maintain objectivity, the source of each task remained undisclosed throughout the study.

5.1 Measuring Effective Diversity

We measure the effective diversity of a task set $\mathcal{S}$ (defined in Section 3), which requires evaluating both task diversity and task utility. Below, we present two approaches we used to evaluate the set diversity: automated and expert-based.

- **Automated Vendi Score.** We use the Vendi Score [35] to measure task diversity automatically. Given a set of tasks, it ranges from 1 (all tasks identical) to $|\mathcal{S}|$ (all tasks dissimilar), quantifying the number of distinct tasks in $\mathcal{S}$. To begin, each task $\mathcal{P}$ is represented by an embedding obtained from a model (Qwen3-Embedding-0.6B [36]). Given the task embeddings, we construct a cosine similarity matrix and compute the Vendi Score as the exponential of the Shannon entropy of its eigenvalues.
- **Expert-based Clustering.** We involve expert evaluators to assess task diversity through clustering the generated tasks. Given a set of tasks, an evaluator either assigns each task to a cluster based on its semantic similarity to other tasks in the set or creates a new cluster for the task. The number of final clusters provides an expert-annotated measure of task diversity, corresponding to the number of distinct tasks as judged by experts.

Next, we describe how we measure effective diversity which is defined as the diversity (number of distinct tasks) of the high-utility subset $\mathcal{S}_{\text{util}=1}(\mathcal{C})$ (cf. Section 3). To this end, we need to evaluate the utility of generated tasks. For scalability, we use an LLM-as-a-judge approach, which has been used in prior work including task utility evaluation [15, 22, 32]. Given a task $\mathcal{P}$, the judge first generates a solution, which is executed against $\mathcal{P}_{\text{eval}}$ to verify *Task Validity*. The judge then assesses *Context Relevance* (whether the task is relevant to the given context) and *Comprehensibility* (whether sufficient information is provided). Utility($\mathcal{P}$) = 1 if the task $\mathcal{P}$ satisfies all three criteria. We use Gemini 2.5 Flash-Lite [37] as the judge, chosen for its cost-effectiveness. Using a model from a different family than the generator also avoids self-preference bias.

5.2 Measuring Other Task Aspects

Beyond diversity metrics, we asked experts to perform additional evaluations on small subsets of generated tasks. More specifically, each evaluator solved and rated 6 tasks (3 from CoT and 3 from CreativeDC) via a web interface (providing a task description, code editor, and console). After completing a task or reaching a 10-minute limit, evaluators rated *Theme Relevance*, *Concept Relevance*, *Task Validity*, and *Comprehensibility* on 3-point Likert scales (e.g., Relevant / Partially Relevant / Irrelevant). Supplementary dimensions included *Difficulty* (3-point) and *Interestingness* (5-point). After finishing the task ratings, we asked each evaluator to perform pairwise comparison in terms of novelty. Each pair comprised one task from CoT and one task from CreativeDC. We randomly sampled 20 pairs of tasks. For each pair, the evaluator selected the more novel task and wrote a brief justification.

6. RESULTS

We present the effective diversity with respect to the number of tasks K in Figure 3 and ratings for other task aspects in Table 2.

6.1 Effective Diversity

Effective Diversity via Automated Vendi Score. Figure 3a shows that all methods generate more distinct tasks as the number of tasks K increases. Notably, CreativeDC significantly outperforms both CoT and Base across all K values ($p < 0.001$, Bonferroni-corrected paired Wilcoxon signed-rank tests [38]), while CoT and Base are statistically indistinguishable. At $K = 50$, CreativeDC generates about 13

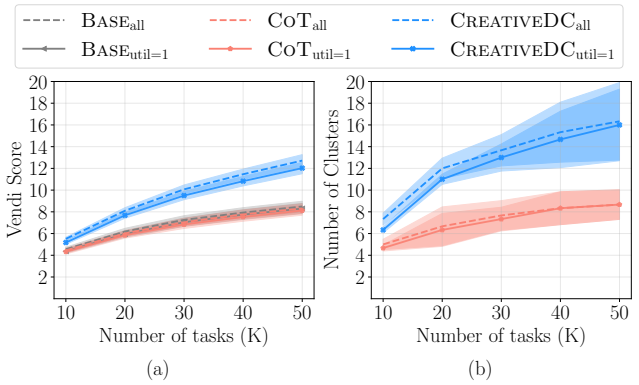

Figure 3: Effective diversity with respect to number of tasks K via (a) automated Vendi scores and (b) expert evaluation. Solid lines denote effective diversity (diversity of high-utility tasks only) and dashed lines denote diversity over all tasks. Shaded bands show SE across contexts. CreativeDC consistently outperforms baselines across both evaluations.

distinct tasks (dashed blue line), and about 12 distinct tasks when restricted to high-utility tasks only (solid blue line). In contrast, both CoT and BASE generate about 8 distinct high-utility tasks (solid red and grey lines).

Effective Diversity via Expert Evaluation. Figure 3b shows that effective diversity perceived by experts has a similar pattern but is generally higher than the automated Vendi Score. For instance, evaluators created about 16 clusters at $K = 50$ for CREATIVEDC when restricted to high-utility tasks, compared to about 9 clusters for CoT. While CoT shows signs of saturation, CREATIVEDC maintains an upward trend. Because each expert evaluated a uniquely themed subset of tasks, inter-rater agreement measures are not applicable. This evaluation provides evidence that the automated metric we used can serve as a proxy for expert-perceived diversity.

6.2 Expert Ratings for Other Task Aspects

We compute the average expert ratings (cf. Section 5.2) by mapping 3-point scales to $\{1, 0.5, 0\}$ and 5-point scales to $\{1, 0.75, 0.5, 0.25, 0\}$. Table 2 shows that tasks generated by CREATIVEDC were judged higher than those from CoT on the utility-related submetrics. Tasks generated by both methods were perceived as between easy and medium in difficulty. Notably, CREATIVEDC tasks were rated higher in interestingness (0.81 vs. 0.50) and were preferred in 96.67% of pairwise novelty comparisons. Justifications for experts' decisions favoring tasks generated by CREATIVEDC include: "It makes the cooking theme and the question very life-like"; "Trading recorded personal experiences as currency is definitely novel to me–the problem even opens up my mind for new possibilities that may really happen in the future".

7. CONCLUDING DISCUSSION

We proposed CREATIVEDC, a method for generating diverse and high-utility tasks in programming education. Through automated and expert evaluations, CREATIVEDC has been shown to produce more distinct high-utility programming tasks, improving the effective diversity of generated tasks.

Table 2: Expert evaluator ratings (means ± SE) for various task aspects: utility, novelty, and difficulty. Highest scores are bolded for metrics where higher is better (↑).

Metric	CoT	CREATIVEDC
Utility:TaskValidity ↑	0.78 ± 0.15	**1.00 ± 0.00**
Utility:ThemeRelevance ↑	0.94 ± 0.06	**1.00 ± 0.00**
Utility:ConceptRelevance ↑	0.89 ± 0.07	**0.94 ± 0.06**
Utility:Comprehensibility ↑	0.72 ± 0.15	**0.83 ± 0.08**
Novelty:Interestingness ↑	0.50 ± 0.04	**0.81 ± 0.04**
Novelty:Win Rate (%) ↑	3.33 ± 2.32	**96.67 ± 2.32**
Difficulty:Self-rated	0.17 ± 0.08	0.17 ± 0.08
Difficulty:Pass Rate (%)	88.89 ± 11.11	100.00 ± 0.00
Difficulty:Time Taken (mins)	3.84 ± 0.55	3.38 ± 0.61

Next, we discuss limitations of our work and suggest future directions. First, our evaluation focused on introductory programming concepts; extending this framework to other educational domains, such as mathematics or science, would demonstrate its generalizability. Second, while our expert study provides initial validation, it is limited by its sample size and demographic diversity; it would be useful to conduct large-scale studies to evaluate the tasks from the perspective of students. Third, the two stages in our method can be iterated in a loop to improve the quality of the generated tasks; exploring this iterative process is a promising direction. Fourth, our experiments focus on introductory Python programming concepts; it would be useful to check whether this diversity holds in harder problems with advanced concepts and whether diversity across difficulty levels can be enforced while maintaining the effective validity and utility of the generated tasks. Finally, our evaluation focused on theme and concept as a task context; expanding the context to include learner-state contextualization (e.g., difficulty, student profiles) is a natural next step for generating diverse and personalized tasks.

Acknowledgement

Funded/Co-funded by the European Union (ERC, TOPS, 101039090). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Sabina Elkins, Ekaterina Kochmar, Jackie C. K. Cheung, and Iulian Serban. How Teachers Can Use Large Language Models and Bloom's Taxonomy to Create Educational Quizzes. In *AAAI*, 2024.

[2] Jaewook Lee, Digory Smith, Simon Woodhead, and Andrew S. Lan. Math Multiple Choice Question Generation via Human-Large Language Model Collaboration. In *EDM*, 2024.

[3] Ziqing Li, Mutlu Cukurova, and Sahan Bulathwela. A Novel Approach to Scalable and Automatic Topic-Controlled Question Generation in Education. In *LAK*, 2025.

[4] Sahan Bulathwela, Hamze Muse, and Emine Yilmaz. Scalable Educational Question Generation with Pre-trained Language Models. In *AIED*, 2023.

[5] Zichao Wang, Andrew S. Lan, and Richard G. Baraniuk. Math Word Problem Generation with Mathematical Consistency and Problem Context Constraints. In *EMNLP*, 2021.

[6] Hai Li, Rui Guo, Chenglu Li, and Wanli Xing. Automated Quality Assessment of Multimodal Mathematical Stories Generated by Generative Artificial Intelligence. In *L@S*, 2024.

[7] Ying Jiao, Kumar Shridhar, Peng Cui, Wangchunshu Zhou, and Mrinmaya Sachan. Automatic Educational Question Generation with Difficulty Level Controls. In *AIED*, 2023.

[8] Yongan Yu, Alexandre Krantz, and Nikki G. Lobczowski. From Recall to Reasoning: Automated Question Generation for Deeper Math Learning Through Large Language Models. In *AIED*, 2025.

[9] Kia Karbasi, Kevin Hong, Mohammad Amin Samadi, and Gregory J. Pottie. Multi-Agent Collaborative Framework For Math Problem Generation. In *EDM*, 2025.

[10] Niklas Scholz, Manh Hung Nguyen, Adish Singla, and Tomohiro Nagashima. Partnering with AI: A Pedagogical Feedback System for LLM Integration Into Programming Education. In *ECTEL*, 2025.

[11] Mehmet Arif Demirtas, Claire Zheng, Max Fowler, and Kathryn I. Cunningham. Generating Planning Feedback for Open-Ended Programming Exercises with LLMs. In *AIED*, 2025.

[12] Alexander Scarlatos, Naiming Liu, Jaewook Lee, Richard G. Baraniuk, and Andrew S. Lan. Training LLM-Based Tutors to Improve Student Learning Outcomes in Dialogues. In *AIED*, 2025.

[13] Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. Large Language Models for In-Context Student Modeling: Synthesizing Student's Behavior in Visual Programming. In *EDM*, 2024.

[14] Lorenzo Lee Solano, Charles Koutcheme, Juho Leinonen, Alexandra Vassar, and Jake Renzella. Fine-Tuning Open-Source Models as a Viable Alternative to Proprietary LLMs for Explaining Compiler Messages. In *SIGCSE*, 2026.

[15] Manh Hung Nguyen, Victor-Alexandru Padurean, Alkis Gotovos, Sebastian Tschiatschek, and Adish Singla. Synthesizing High-Quality Programming Tasks with LLM-Based Expert and Student Agents. In *AIED*, 2025.

[16] Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. Evaluating Contextually Personalized Programming Exercises Created with Generative AI. In *ICER*, 2024.

[17] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *ICER*, 2022.

[18] Mohammad Hassany, Peter Brusilovsky, Jaromír Savelka, Arun Balajiee Lekshmi Narayanan, Kamil Akhuseyinoglu, Arav Agarwal, and Rully Agus Hendrawan. Generating Effective Distractors for Introductory Programming Challenges: LLMs vs Humans. In *LAK*, 2025.

[19] Paul Denny, Sumit Gulwani, Neil T. Heffernan, Tanja Käser, Steven Moore, Anna N. Rafferty, and Adish Singla. Generative AI for Education (GAIED): Advances, Opportunities, and Challenges. *CoRR*, abs/2402.01580, 2024.

[20] Rachel Van Campenhout, Kevin S. Autry, Michelle W. Clark, and Benny G. Johnson. Scaling the Doer Effect: A Replication Analysis Using AI-Generated Questions. In *L@S*, 2025.

[21] Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling Synthetic Data Creation with 1,000,000,000 Personas. *CoRR*, abs/2406.20094, 2024.

[22] Liwei Jiang, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, Alon Albalak, and Yejin Choi. Artificial Hivemind: The Open-Ended Homogeneity of Language Models (and Beyond). In *NeurIPS*, 2025.

[23] Mete Ismayilzada, Antonio Laverghetta Jr., Simone A. Luchini, Reet Patel, Antoine Bosselut, Lonneke Van Der Plas, and Roger E. Beaty. Creative Preference Optimization. In *EMNLP*, 2025.

[24] Li-Chun Lu, Shou-Jen Chen, Tsung-Min Pai, Chan-Hung Yu, Hung-Yi Lee, and Shao-Hua Sun. LLM Discussion: Enhancing the Creativity of Large Language Models via Discussion Framework and Role-Play. In *COLM*, 2024.

[25] Graham Wallas. *The Art of Thought*. 1926.

[26] Ronald A. Finke, Thomas B. Ward, and Steven M. Smith. *Creative Cognition: Theory, Research, and Applications*. The MIT Press, 1992.

[27] Joy Paul Guilford. The Structure of Intellect. *Psychological Bulletin*, 53, 1956.

[28] Yiming Zhang, Harshita Diddee, Susan Holm, Hanchen Liu, Xinyue Liu, Vinay Samuel, Barry Wang, and Daphne Ippolito. NoveltyBench: Evaluating Language Models for Humanlike Diversity. In *COLM*, 2025.

[29] Ximing Lu, Melanie Sclar, Skyler Hallinan, Niloofar Mireshghallah, Jiacheng Liu, Seungju Han, Allyson Ettinger, Liwei Jiang, Khyathi Raghavi Chandu, Nouha Dziri, and Yejin Choi. AI as Humanity's Salieri: Quantifying Linguistic Creativity of Language Models via Systematic Attribution of Machine Text against Web Text. In *ICLR*, 2025.

[30] Yufei Tian, Abhilasha Ravichander, Lianhui Qin, Ronan Le Bras, Raja Marjieh, Nanyun Peng, Yejin Choi, Thomas L. Griffiths, and Faeze Brahman. MacGyver: Are Large Language Models Creative Problem Solvers? In *NAACL*, 2024.

[31] Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. Prompt Optimization Across Multiple Agents for Representing Diverse Human Populations. *CoRR*, abs/2510.07064, 2025.

[32] Vishakh Padmakumar, Chen Yueh-Han, Jane Pan, Valerie Chen, and He He. Measuring LLM Novelty As The Frontier Of Original And High-Quality Output. *CoRR*, abs/2504.09389, 2025.

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*, 2022.

[34] An Yang et al. Qwen3 Technical Report. *CoRR*, abs/2505.09388, 2025.

[35] Dan Friedman and Adji Bousso Dieng. The Vendi Score: A Diversity Evaluation Metric for Machine Learning. *TMLR*, 2023.

[36] Yanzhao Zhang et al. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *CoRR*, abs/2506.05176, 2025.

[37] Gemini Team. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. *CoRR*, abs/2507.06261, 2025.

[38] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.