# Parametric Editorial Generation for Competitive Programming: A Human-in-the-Loop Working Prototype

**Theodor Moroianu** THEODOR.MOROIANU@GMAIL.COM
*Quadrature*

**Adrian Marius Dumitran** MARIUS.DUMITRAN@UNIBUC.RO
*University of Bucharest*

**Tamio-Vesa Nakajima** NAKAJIMA@UNI-MARBURG.DE
*University of Marburg*

**Adrian Miclăuș** ADRIAN.MICLAUS@FMI.UNIBUC.RO
*University of Bucharest*

**Ștefan-Cosmin Dăscălescu** STEFDASCA@GMAIL.COM
*Stefdasca Tutoring*

**Mihai-Alexandru Vasiluță** M.VASILUTA@STUDENT.TUE.NL
*Eindhoven University of Technology*

**Laura Marin** LAURA.MARIN8@S.UNIBUC.RO
*University of Bucharest*

## Abstract

We present EditoriaLLM, a web application that uses large language models to generate competitive programming editorials from problem statements, reference solutions, algorithmic tags, audience settings, and style templates. This approach streamlines using a general-purpose chat interface into a simple, all-in-one tool, tailored to our particular use-case. EditoriaLLM is designed as a human-in-the-loop assistant for competitive programming education, with early adoption already observed in national competition contexts.

**Keywords:** large language models; computing education; competitive programming; automated editorials; programming contests; code explanations

## 1. Introduction

Programming contests and online judges are widely used within computer science education algorithmic practice, assessment, and self-directed learning (Kurnia et al., 2001; Revilla et al., 2008). In these settings, an accepted solution is not always enough: students also need the key idea, correctness argument, implementation details, and complexity analysis. This explanatory artifact is commonly called an *editorial*.

Despite their pedagogical value, editorials are costly to write. A contest organizer must already prepare statements, validators, tests, reference solutions, and judging configuration. Moreover, the same problem may need different explanations: a concise hint, a proof-oriented solution, or an accessible tutorial for high-school learners.

Large language models (LLMs) have shown strong capabilities in code generation, programming explanations, and learning-material generation (Chen et al., 2021; Sarsa et al., 2022; MacNeil et al., 2023). Related work has explored competition-level code generation

(Li et al., 2022), educational risks and opportunities (Kasneci et al., 2023), programming error explanations (Leinonen et al., 2023), and prompt-based programming exercises (Denny et al., 2024). However, a general chat interface is not ideal as a reusable workflow for editorial writing: the author must manually reconstruct, for each problem, the full setup involving the statement, reference solution, style, audience, formatting requirements, and refinement instructions. A system which standardizes these inputs through a domain-specific interface, while keeping the underlying models and prompting strategies easier to replace or extend in future versions, would be preferable.

This paper presents EditoriaLLM, a human-in-the-loop web application for integrating existing LLMs into the production cycle of competitive programming explanations. Its output is a draft to be inspected and refined before publication. The contribution of this work is therefore not a new language model or a fully automated editorial writer, but a domain-specific workflow for applying LLMs to competitive-programming editorial production.

## 2. System Overview and Architecture

The central design choice in EditoriaLLM is *parametric editorial generation*. Instead of issuing a single generic prompt, the user configures the explanation according to the contest and learner context. The exposed parameters encode editorial decisions that contest authors routinely make when adapting the same solution to different audiences, languages, and publication styles. Table 1 summarizes these parameters.

Table 1: Main editorial-generation parameters exposed by EditoriaLLM.

| Parameter | Options | Role in editorial generation |
|---|---|---|
| Length | Short, Medium, Long | Controls detail: from core insight to examples, proof, complexity, and edge cases. |
| Audience | Middle School, High-School, University | Calibrates prerequisites, terminology, and proof depth. |
| Language | Mainstream contest languages | Selects the output language, with specific editorial formats configured in the app. |
| Model | Available LLM registry | Chooses the model; vision-capable models are needed for PDF-only statements. |
| Effort | Low, Medium, High | Controls reasoning depth for models that support this setting. |
| Temperature | Default or 0.0–1.0 | Controls variability, from deterministic to more diverse outputs. |
| Template | None, `editorial-basic`, `editorial-complet` | Conditions structure and formatting style. |
| Tags/guidelines | Tags and free text | Steers the intended technique and optional presentation constraints. |

The architecture implements this parameter design: the frontend collects settings and files, while the backend builds a grounded prompt, streams the model output, stores the conversation, and renders the generated LaTeX into PDF. Figure 1 shows the interface and a generated editorial in the same workspace.

The main persistent object is a conversation, storing uploaded files, parameters, messages, model metadata, optional reasoning text, and the latest editorial. Files may be plain text, source code, or base64-encoded PDF statements. For PDFs, the backend renders

2. https://kilonova.ro/problems/4324?list_id=1620

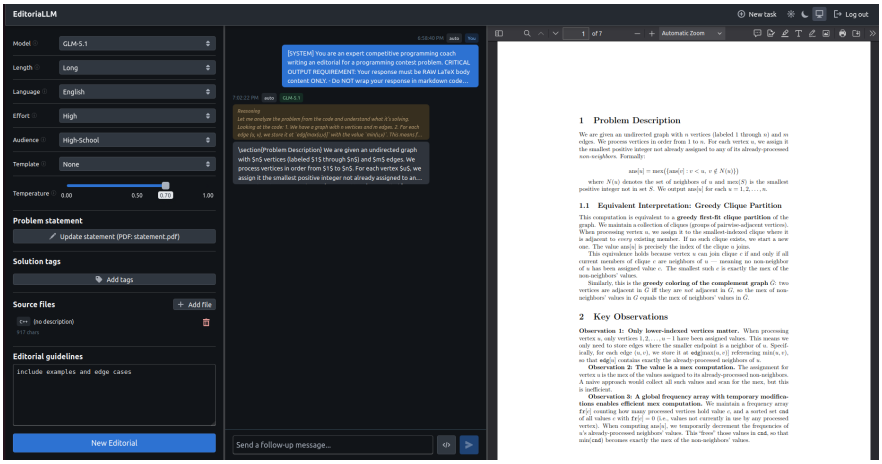


Figure 1: EditoriaLLM interface and rendered output for the Kilonova problem *Mán lì*.[2] The shown run uses GLM-5.1, long length, English, high reasoning effort, high-school audience, no template, temperature 0.7, and the guideline "include examples and edge cases."

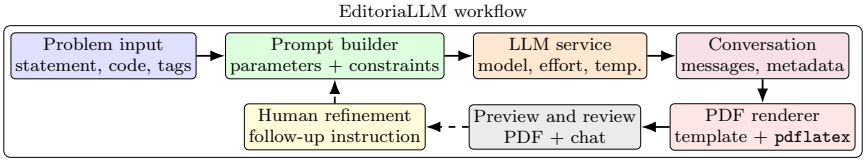


Figure 2: Simplified workflow of EditoriaLLM. Dashed arrows show the human-in-the-loop refinement cycle.

pages to images when the selected model supports vision, and the frontend filters the model list accordingly.

The data flow is summarized in Figure 2. The prompt asks the model to write as an expert competitive programming coach and to return raw LaTeX body content only, without markdown fences or preamble. The system encodes four design requirements. *Grounding:* each request includes the exact statement, source files, selected tags, user notes, and, when needed, PDF page images. *Compilable output:* the backend wraps the body in a template, sanitizes unsupported listing languages, and reports `pdflatex` errors. *Iterative refinement:* conversations are stored, and each follow-up asks the model to regenerate the complete revised editorial. *Contest controls:* users can set language, length, audience, algorithmic tags, temperature, reasoning effort, model choice, and style template. The model registry includes a dummy no-API model for testing and several Together AI text and vision models, including GLM (Team GLM et al., 2024), Qwen (Bai et al., 2023), DeepSeek (DeepSeek-AI et al., 2024a,b), Llama (Grattafiori et al., 2024), Gemma (Gemma Team et al., 2024), MiniMax (MiniMax et al., 2025). From a responsible-AI perspective, this modular design

Table 2: Parameter variants generated for the same problem. HS = High-School; MS = Middle School; Univ. = University.

| ID | Len. | Aud. | Eff. | Temp. | Template | Guideline |
|---|---|---|---|---|---|---|
| A1 | Short | HS | High | 0.7 | None | default |
| A2 | Medium | HS | High | 0.7 | None | default |
| A3 | Long | HS | High | 0.7 | None | examples and edge cases |
| A4 | Short | Univ. | High | 0.7 | None | formal proof focus |
| A5 | Long | MS | High | 0.7 | None | simpler terminology |
| A6 | Medium | HS | Low | 0.7 | None | default |
| A7 | Medium | HS | High | 0.2 | None | deterministic style |
| A8 | Medium | HS | High | 0.7 | `basic` | basic editorial style |
| A9 | Medium | HS | High | 0.7 | `complet` | complete editorial style |
| A10 | Medium | HS | High | 0.7 | None | C++ implementation notes |

also allows future deployments to prefer open-source or open-weight models when reproducibility, institutional control, privacy, reduced vendor lock-in, or community adaptation are important (BigScience Workshop et al., 2022; Groeneveld et al., 2024).

## 3. Use Case and Demonstration

A typical workflow begins with a contest organizer who has a statement and an accepted reference implementation. The organizer creates a task, selects generation parameters, adds the statement as text or PDF, uploads source files, and optionally selects algorithmic tags such as dynamic programming, greedy algorithms, graph, etc.

As a concrete demonstration, we used the Kilonova problem *Mán li*[3] together with the official accepted reference solution.[4] The run shown in Figure 1 uses GLM-5.1, long length, English output, high reasoning effort, high-school audience, no template, temperature 0.7, and the guideline "include examples and edge cases." The generated editorial includes a problem description, an equivalent greedy clique-partition interpretation, key observations, and implementation-oriented discussion. We chose GLM-5.1 for the prototype after an informal pilot with current and former competitive-programming participants, who preferred its generated editorials among the tested models for this particular use case. This selection should be interpreted as a practical prototyping choice rather than as a systematic cross-model evaluation. A more rigorous comparison of model reliability, consistency, latency, token usage, and cost is left for future work.

To illustrate the parameterized nature of the system, we generated a supplementary case study for the same problem. The anonymized source repository is available online,[5] and the supplementary case-study folder containing the ten generated PDF editorials is also available online.[6] It contains the statement PDF, the official C++ solution, and ten generated PDF editorials A1–A10. Table 2 summarizes the fixed-problem parameter sweep.

3. https://kilonova.ro/problems/4324?list_id=1620
4. https://kilonova.ro/submissions/1091161?forceCode=1
5. https://github.com/adimiclaus15/Parametric-Editorial-Generation-for-Competitive-Programming
6. https://github.com/adimiclaus15/Parametric-Editorial-Generation-for-Competitive-Programming/tree/master/man-li-editoriall-case-study

The same conversation can be refined interactively: an organizer may request a more rigorous proof, a worked example, a simpler explanation, or a chosen template. Each response replaces the current editorial with a complete revised LaTeX body.

Beyond this demonstration, EditoriaLLM has already seen early real-world use. During development, a problem setter incorporated LLM-generated drafts produced by EditoriaLLM into an officially published national competition problem collection SEPI (2026). The content was reviewed and refined before publication.

## 4. Limitations and Future Work

EditoriaLLM is currently a functional prototype rather than a fully evaluated educational intervention. It supports the end-to-end workflow of collecting contest inputs, building a grounded prompt, selecting an LLM, storing conversations, rendering LaTeX to PDF, and refining the generated editorial through chat. However, human review remains essential before publication.

The main limitation is correctness. LLMs may produce plausible but false invariants, omit corner cases, misread the reference implementation, or state an incorrect complexity bound. The current prototype reports LaTeX compilation failures, but it does not automatically verify whether the generated explanation matches the uploaded solution or whether the stated proof and complexity are correct.

Future work will focus on validation and evaluation. We plan to investigate LLM-as-a-judge components, consistency checks against the uploaded solution, tests on generated examples, and lightweight static or semantic checks for algorithmic claims. We also plan to compare generated editorials with human-written editorials and plain chat-based prompting, using a rubric covering correctness, completeness, pedagogical clarity, LaTeX compilability, and usefulness for contest authors. Future evaluations should also report cross-model reliability, latency, token usage, and cost per generated editorial.

## Acknowledgments

The authors used AI-assisted tools, including large language models, for programming support and for drafting, editing, and language refinement of this manuscript. The authors reviewed and take responsibility for the final content. We thank the anonymous reviewers for their helpful feedback.

## References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. BLOOM: A 176B-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

DeepSeek-AI, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024a.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.

P. Denny, J. Leinonen, J. Prather, A. Luxton-Reilly, T. Amarouche, B. A. Becker, and B. N. Reeves. Prompt problems: A new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 296–302, 2024. doi: 10.1145/3626252.3630909.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. OLMo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15789–15809, 2024.

E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, et al. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023. doi: 10.1016/j.lindif.2023.102274.

A. Kurnia, A. Lim, and B. Cheang. Online judge. *Computers & Education*, 36(4):299–315, 2001. doi: 10.1016/S0360-1315(01)00018-5.

J. Leinonen, A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather, and B. A. Becker. Using large language models to enhance programming error messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 563–569, 2023. doi: 10.1145/3545945.3569770.

Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

S. MacNeil, A. Tran, J. Leinonen, P. Denny, J. Kim, A. Hellas, S. Bernstein, and S. Sarsa. Automatically generating CS learning materials with large language models. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, page 1176, 2023. doi: 10.1145/3545947.3569630.

MiniMax, Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. MiniMax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025.

M. A. Revilla, S. Manzoor, and R. Liu. Competitive learning in informatics: The UVa online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.

S. Sarsa, P. Denny, A. Hellas, and J. Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research*, pages 27–43, 2022. doi: 10.1145/3501385.3543957.

SEPI. *Olimpiada Națională de Informatică pentru Liceu, Etapa județeană și etapa națională. Baraje de selecție a lotului național de seniori și a echipelor reprezentative ale României: Enunțuri, soluții, surse.* Editura L&S Soft / Infobits Academy, 2026. ISBN 978-630-6559-25-1. Ebook, 261 pages. Available at https://ebooks.infobits.ro/carti-distribuite-gratuit-ebooks-infobits-academy-editura-ls-soft/119-sepi-oni-liceu-etapa-judeteana-si-nationala-baraj-seniori-2025.html.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.