Cracking the Code: Evaluating Zero-Shot Prompting Methods for Providing Programming Feedback

Niklas Ippisch & Markus Herklotz
LMU Munich, Germany

Anna-Carolina Haensch
LMU Munich, Germany
University of Maryland, College Park, MD, USA

Jan Simson, Jacob Beck & Malte Schierholz
LMU Munich, Germany
Munich Center for Machine Learning (MCML), Germany

Abstract

We introduce an evaluation framework to assess the feedback given by large language models (LLMs) under different prompt engineering techniques and conduct a case study, systematically varying prompts to examine their influence on feedback quality for common programming errors in R. Our findings suggest that prompts recommending a stepwise approach improve precision, whereas omitting explicit details on which data to analyze can bolster error identification.

1 Background

Even though large language models (LLMs) are increasingly used to support students and developers with programming tasks (Pankiewicz & Baker, 2023), how to elicit optimal feedback from these models remains underexplored in research (Dai et al., 2023) and results of prompt engineering evaluations are often contradictory regarding possible differences between prompts (Jacobsen & Weber, 2023; Mungoli, 2023; Sahoo et al., 2024; Tang et al., 2024; Wang et al., 2024). Moreover, it remains uncertain whether the quality of responses to prompt engineering strategies is context-sensitive. In this work, we are interested in how to best elicit feedback on basic errors typically made by beginners when learning the R programming language (R Core Team, 2021). R is well-known for its steep learning curve (Çetinkaya Rundel & Rundel, 2018; Rode & Ringel, 2019; Tucker et al., 2022), especially as it may be students' first programming experience. When equipping students with an LLM to provide feedback, the feedback-quality is crucial since it can significantly influence the individual as well as their future behavior (Hattie & Timperley, 2007) and response reliability varies (Mozannar et al., 2023). To assess LLM feedback quality across different prompts, we created a structured evaluation framework and systematically varied different prompt engineering techniques (Marvin et al., 2024). Our contribution is twofold: First, we present our evaluation framework, which utilizes the feedback frame of Ryan et al. (2020) and can be applied beyond our study to similar contexts; second, we apply this framework to a focused use case, analyzing feedback on common beginner errors in R programming.

2 Methods and Data

To reduce costs and response times, we focused our evaluation on different zero-shot prompting methods (see Appendix C for exact wordings): In zero-shot *Chain of Thought* prompting (Wei et al., 2023) a stepwise process is suggested; these steps, in the form of sub-tasks, are made explicit in *Prompt Chaining* (Wu et al., 2022). *Tree of Thought* prompting (Yao et al., 2023a) does not divide the task into sub-tasks but rather into different areas, which the LLM processes sequentially while discarding irrelevant ones. Lastly, *ReAct* prompting (Yao et al., 2023b) enforces reasoning by dividing the task into different thoughts and actions. We also included a *baseline* prompt for comparison, which we developed iteratively by trial and error (the criteria here were comparable to the criteria below, but were not formalized at that point). Prompts were sent to GPT-3.5-turbo through the Microsoft Azure API directly in R.

We identified five errors commonly encountered by beginners in introductory courses to statistical programming and used these to develop test cases, deliberately crafted to include specific errors (see Appendix B for details). Ryan et al. (2020) suggest that next to just stating if something is right or wrong, feedback should also focus on the response as well as on the underlying concept in order to be meaningful. Hence, we also investigated besides whether (a) the problem has been detected or not, (b) if the concrete error has been described and (c) if there is an explanation as to why that error occurred. Additionally for our case of users troubleshooting problems, the response should suggest (d) how to resolve the error, (e) be concise, and (f) avoid unrelated suggestions that may confuse beginners. One author verified these criteria awarding one point per fulfillment. Thus, with ten iterations per problem, five problems and six criteria a maximum score of 300 points per prompt was possible.

3 RESULTS

Table 1 shows the summarized results. Overall, all prompts performed well, with overall ratings above 0.85, but there are subtle differences between the prompts (see also table 2 in Appendix A for detailed results). All responses detected the error, and all but one (*Chain of Thought*) were within 200 tokens.

Table 1: Summary of evaluation results. Cells indicate the percentage of fulfilled criteria per prompting technique. Best performing prompting techniques per criteria are highlighted in bold.

Prompting Technique	**Evaluation Criteria**						**Total**
	(a)	(b)	(c)	(d)	(e)	(f)	
Baseline prompt	**1.00**	0.86	**0.86**	0.86	**1.00**	0.66	0.873
Chain of Thought	**1.00**	**0.88**	0.84	**0.88**	0.98	0.76	**0.89**
Prompt Chaining	**1.00**	0.82	0.82	0.82	**1.00**	0.64	0.85
Tree of Thought	**1.00**	0.82	0.82	0.84	**1.00**	0.76	0.873
ReAct	**1.00**	0.82	0.76	**0.88**	**1.00**	**0.78**	0.873

Regarding typos and not loaded code, however, the performance varied more. *Chain of Thought* prompting and *ReAct* had bigger problems in providing a correct reason for errors resulting from typos. Despite the required line of code being present, all prompts struggled to identify cases where the column required for executing the code was missing in the data. *Chain of Thought* prompting and the *baseline* prompt detected missing columns more frequently, suggesting better incorporation of provided information. Lastly, *Chain of Thought* prompting, *Tree of Thought* prompting, and *ReAct* more effectively avoided irrelevant suggestions.

To summarize, it seems like there is a trade-off between higher precision (i.e., no irrelevant suggestions) and better detection of the underlying error (i.e., higher scores in terms of error identification and reason for error). Regarding **precision**, prompts that enforce a stepwise process (*Chain of Thought, Tree of Thought,* and *ReAct*) seem to perform better. Prompts, in which the information to be analyzed (e.g., script, data, etc.) are not specifically mentioned (*baseline* prompt and *Chain of Thought*), seem to perform better regarding the **error identification and reason**. Two exceptions stand out: *Prompt Chaining*, where a stepwise process is enforced and the data to be analyzed is mentioned, performs worse both in terms of precision and error identification. On the other hand, *Chain of Thought* prompting excelled in precision despite not explicitly specifying information to analyze. That prompts omitting explicit data references perform better in error identification seems contra-intuitive. Apparently, the LLM appears to verify column existence more reliably when data is not explicitly mentioned, suggesting that excessive information might cause confusion.

4 Concluding remarks

Our proposed evaluation framework provides a structured and replicable method to assess prompt engineering strategies for programming assistance. While our study focuses on errors commonly made by beginners in R programming, the framework is designed to be applicable across various use cases. We encourage others to build on our approach, refining and extending it to further enhance the effectiveness of LLMs in educational and professional settings.

URM Statement

Authors NI and ACH meet the URM criteria of the ICLR 2025 Tiny Papers Track.

References

Wei Dai, Jionghao Lin, Flora Jin, Tongguang Li, Yi-Shan Tsai, Dragan Gaševic, and Guanliang Chen. Can large language models provide feedback to students? a case study on ChatGPT. 2023. doi: 10.1109/ICALT58122.2023.00100.

John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1): 81–112, 2007. doi: 10.3102/003465430298487.

Lucas Jasper Jacobsen and Kira Elena Weber. The promises and pitfalls of LLMs as feedback providers: A study of prompt engineering and the quality of AI-driven feedback. 2023. doi: 10.31219/osf.io/cr257.

Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. Prompt engineering in large language models. In I. Jeena Jacob, Selwyn Piramuthu, and Przemyslaw Falkowski-Gilski (eds.), *Data Intelligence and Cognitive Informatics*, pp. 387–402. Springer Nature Singapore, 2024. doi: 10.1007/978-981-99-7962-2_30.

Hussein Mozannar, Jimin J Lee, Dennis Wei, Prasanna Sattigeri, Subhro Das, and David Sontag. Effective human-AI teams via learned natural language rules and onboarding. 2023.

Neelesh Mungoli. Exploring the synergy of prompt engineering and reinforcement learning for enhanced control and responsiveness in chat GPT. *Journal of Electrical Electronics Engineering*, 2(3), 2023. doi: 10.33140/JEEE.02.03.02.

Maciej Pankiewicz and Ryan S. Baker. Large language models (GPT) for automating feedback on programming assignments, 2023.

R Core Team. R: A language and environment for statistical computing, 2021. URL `https://www.R-project.org/`.

Jacob B. Rode and Megan M. Ringel. Statistical software output in the classroom: A comparison of r and SPSS. *Teaching of Psychology*, 46(4):319–327, 2019. doi: 10.1177/0098628319872605.

Anna Ryan, Terry Judd, David Swanson, Douglas P. Larsen, Simone Elliott, Katina Tzanetos, and Kulamakan Kulasegaram. Beyond right or wrong: More effective feedback for formative multiple-choice tests. *Perspectives on Medical Education*, 9(5):307–313, 2020. doi: 10.1007/s40037-020-00606-z.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. 2024. doi: 10.48550/arXiv.2402.07927.

Yiyi Tang, Ziyan Xiao, Xue Li, Qingpeng Zhang, Esther W Chan, Ian Ck Wong, and Research Data Collaboration Task Force. Large language model in medical information extraction from titles and abstracts with prompt engineering strategies: A comparative study of GPT-3.5 and GPT-4. 2024. doi: 10.1101/2024.03.20.24304572.

Mary C. Tucker, T. Shaw Stacy, Ji Y. Son, and James W. Stigler. Teaching statistics and data analysis with r. *Journal of Statistics and Data Science Education*, 31(1), 2022. doi: 10.1080/26939169.2022.2089410.

Li Wang, Xi Chen, XiangWen Deng, Hao Wen, MingKe You, WeiZhi Liu, Qi Li, and Jian Li. Prompt engineering in consistency and reliability with the evidence-based guideline for LLMs. *NPJ Digital Medicine*, 7:41, 2024. doi: 10.1038/s41746-024-01029-4.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL `http://arxiv.org/abs/2201.11903`.

Tongshuang Wu, Michael Terry, and Carrie Jun Cai. AI chains: Transparent and controllable human-AI interaction by chaining large language model prompts. In *CHI Conference on Human Factors in Computing Systems*, pp. 1–22, 2022. doi: 10.1145/3491102.3517582.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL `http://arxiv.org/abs/2305.10601`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2023b. URL `http://arxiv.org/abs/2210.03629`.

Mine Çetinkaya Rundel and Colin Rundel. Infrastructure and tools for teaching computing throughout the statistical curriculum. *The American Statistician*, 72(1):58–65, 2018. doi: 10.1080/00031305.2017.1397549.

Appendix

A Detailed results of the evaluation

Table 2: Detailed evaluation results for different prompting techniques. Cells indicate the percentage of fulfilled criteria per prompting technique, problem and evaluation criteria. Best performing prompting techniques per criteria are highlighted in bold.

Prompting Technique	**Problem**	**Evaluation Criteria**						**Total**
		(a)	(b)	(c)	(d)	(e)	(f)	
Baseline prompt	Working directory	1.00	0.90	0.90	0.90	1.00	0.70	0.90
	Package not loaded	1.00	1.00	1.00	1.00	1.00	0.90	0.98
	Code not loaded	1.00	0.40	0.40	0.40	1.00	0.40	0.60
	Typos	1.00	1.00	1.00	1.00	1.00	0.60	0.93
	Variable naming	1.00	1.00	1.00	1.00	1.00	0.70	0.95
	Total	**1.00**	**0.86**	**0.86**	**0.86**	**1.00**	**0.66**	**0.873**
Chain of Thought	Working directory	1.00	1.00	0.90	1.00	1.00	0.80	0.95
	Package not loaded	1.00	1.00	1.00	1.00	1.00	0.90	0.98
	Code not loaded	1.00	0.60	0.60	0.60	0.90	0.50	0.70
	Typos	1.00	0.80	0.70	0.80	1.00	1.00	0.88
	Variable naming	1.00	1.00	1.00	1.00	1.00	0.60	0.93
	Total	**1.00**	**0.88**	**0.84**	**0.88**	**0.98**	**0.76**	**0.89**
Prompt Chaining	Working directory	1.00	1.00	1.00	1.00	1.00	0.60	0.93
	Package not loaded	1.00	1.00	1.00	1.00	1.00	0.90	0.98
	Code not loaded	1.00	0.10	0.10	0.10	1.00	0.10	0.40
	Typos	1.00	1.00	1.00	1.00	1.00	0.60	0.93
	Variable naming	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Total	**1.00**	**0.82**	**0.82**	**0.82**	**1.00**	**0.64**	**0.85**
Tree of Thought	Working directory	1.00	1.00	1.00	1.00	1.00	0.80	0.97
	Package not loaded	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Code not loaded	1.00	0.20	0.20	0.20	1.00	0.20	0.47
	Typos	1.00	0.90	0.90	1.00	1.00	0.90	0.95
	Variable naming	1.00	1.00	1.00	1.00	1.00	0.90	0.98
	Total	**1.00**	**0.82**	**0.82**	**0.84**	**1.00**	**0.76**	**0.873**
ReAct	Working directory	1.00	1.00	1.00	1.00	1.00	0.90	0.98
	Package not loaded	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Code not loaded	1.00	0.10	0.10	0.40	1.00	0.10	0.45
	Typos	1.00	1.00	0.70	1.00	1.00	0.90	0.93
	Variable naming	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Total	**1.00**	**0.82**	**0.76**	**0.88**	**1.00**	**0.78**	**0.873**

B CODE USED FOR EVALUATION

B.1 FALSE WORKING DIRECTORY

Code[1]**:**

```
read.csv("data/testfile")
```

Error message:

```
Error in file(file, "rt") : cannot open the connection
```

Problem: No folder `data` and no file `testfile`

B.2 PACKAGE NOT LOADED

Code:

```
plot = ggplot(data = mtcars, mapping = aes(x = hp, y = cyl)) +
geom_point()
```

Error message:

```
Error in ggplot(data = mtcars, mapping = aes(x = hp, y = cyl)) : could
not find function "ggplot"
```

Problem: Package `ggplot2` not loaded

B.3 CODE NOT LOADED

Code:

```
data = mtcars
data$weight_kg = data$wt * 0.454
data$weight_kgsq = data$weight_kg ^2
```

Error message:

```
Error in '$<-.data.frame'('*tmp*, weight_kgsq, value = numeric(0)) :
replacement has 0 rows, data has 32
```

Problem: The middle line is not executed, hence the column `weight_kg` does not exist

B.4 TYPOS

Code:

```
data = mtcars
summary[data$hp]
```

Error message:

```
Error in summary[data$hp] : object of type 'closure' is not subsettable
```

Problem: Brackets instead of parentheses

[1]The introduced noise, the used test codes and prompt wordings can also be found on OSF: osf.io/cgzk9

B.5 Variable Naming

Code:

```
var1 = 5
var2 = 3
var3 = var_1 * var2
```

Error message:

```
Error: object 'var_1' not found
```

Problem: In the third line, `var_1` instead of `var1` is used

C Prompts

C.1 System Prompt

Kept constant for all of the prompts below.

"You are helping students in an R programming course for beginners and give feedback on why it is wrong, how to correct it and how to improve in the future."

C.2 Evaluated Prompts

(1) Baseline prompt
"Identify the errors (there might be multiple) and give me feedback on how to correct the issue in maximum three sentences."

(2) Chain of Thought
"Analyze the information provided step by step and afterwards give feedback on how to correct the issue in maximum three sentences."

(3) Prompt Chaining
"In the first step, analyze the script provided, loaded data and their structure, existing variables and functions and the packages loaded. In the second step, use the error message and the relevant parts of your analysis of the information and give feedback on how to correct the issue in maximum three sentences."

(4) Tree of Thought
"Assume, the error is in one of the following areas: written script and code, loaded data, loaded variables and functions, packages. Stepwise, check whether there is an error in each of the areas connected to the error message provided. If not, drop that area. When just one area is left, provide feedback on how to correct the issue in maximum three sentences."

(5) ReAct
"Thought 1 I first have to analyze the information provided and screen briefly the script, the loaded data including the structure, the loaded packages, and the variables and functions created.
Action 1 Analyze information
Thought 2 An error message is mentioned. I should check the error message and identify, based on the prior analysis, the relevant parts of the information provided which are linked to the error message.
Action 2 Check error message
Thought 3 After the analysis of the information and error message, I am now able to concisely explain why the error message occurred and how to correct it.
Action 3 End [Explain]
Question Give feedback on how to correct the issue in maximum three sentences."