

LLM-Based Specification Generation for Automated Table Authoring in Periodic Safety Review Reports

Seung-Hyeok Yang^{a*}, Sang-Beom Kang^a, Dae-Young Lee^a

^aFNC Technology Co., Ltd., 13 Heungdeok 1-ro, 32F, Giheung-gu, Yongin-si, Gyeonggi-do, 16954, Korea

*Corresponding author: shyang@fnctech.com

***Keywords :** Periodic Safety Review, Large Language Model, Declarative Specification, Document Automation

1. Introduction

A Periodic Safety Review (PSR) report runs to some 5,000 to 6,000 pages for a single site and covers all fourteen safety factors [1,2]. Many of its tables are produced by transferring or aggregating values from supporting data that record operating, maintenance, and testing history. The work itself is simple, but the format and origin of the data differ from table to table, so it takes considerable effort.

Filling such tables with a large language model (LLM) raises a problem. A document submitted to a regulatory body must yield the same result from the same input, yet having an LLM produce the table directly gives a different result on every run.

This study had the LLM produce a specification rather than the table itself [3,4]. A specification records how to fill one table in a fixed set of fields: which column of the supporting data goes to which column of the table, what conditions select the entries, and what the aggregation is based on. Filling in the table is done by code that reads this specification. Once the author reviews the specification, amends it if necessary, and approves it, the specification is fixed and only the code needs to be run thereafter.

2. Method

The study targeted tables in the safety performance section of a PSR report. These divide into transcription-type tables, which transfer records as they are, and aggregation-type tables, which count occurrences. Some tables require combining several sources or transposing rows and columns, but this study addressed only the two types above.

Each table was filled by two approaches and the results compared. In the specification approach, the LLM reads the supporting data and a reference report to produce a specification, which code then executes. In the code-generation approach, the LLM produces the processing code and executes it. Results were compared cell by cell against the corresponding table in an already published PSR report.

The author selects a reference report from the documents at hand. Since each plant produces a report every ten years, the most recent document is usually from another plant. The experiments were therefore conducted with the reference and the target belonging to different plants.

PSR reports and supporting data are classified material and cannot be sent to external services, so inference has to take place on in-house systems. An open-weight model (Qwen3.6:35b-a3b-bf16) was run with Ollama, and temperature, which governs the randomness of the output, was set low at 0.1 to keep results as consistent as possible [5].

3. Results and Discussion

3.1 Comparison of the Two Approaches

With the specification approach, the aggregation-type tables matched the published report in all ten runs. The transcription-type table matched the report in row count, with 96.8% of cells in agreement. The runs were also stable: every run of the aggregation-type tables produced identical output.

The code-generation approach gave a different result on each run from the same input. Of fifteen runs, only four - all aggregation-type - matched the report; the rest either generated far too many rows or halted partway through.

Table I: Results by approach

	Specification approach	Code-generation approach
Runs	15	15
Match with report	Aggregation 10/10 Transcription 96.8%	Aggregation 4/10 Transcription 0/5
Time per table	16-60 s	70-270 s

Both approaches used the same model with the same settings. What differed was what the LLM was asked to produce. A specification has a fixed set of fields to complete, so the correct answer narrows to nearly one, whereas code performing the same processing can be written in many ways.

The difference in time was also large. Moreover, re-running an approved specification requires no further call to the LLM and takes about 1.8 seconds per table. The code-generation approach has no mechanism for storing and reusing what it produces, so it builds everything from scratch each time and takes correspondingly longer.

3.2 Analysis of the Results

The 3.2% of cells in the transcription-type table that did not match are passages the author had reworded. The automated output transfers the wording of the supporting data as it stands, while the published report contains the author's edits. No value was incorrect and no row was missing.

The experiment was repeated with a different reference report, and the values entered in the table did not change. Values come from the supporting data; the reference report indicates only the shape of the table. What did change was formatting, such as how empty cells are marked, and this was resolved by extracting the formatting from a report of the target plant.

A specification is not always correct, however. In the course of the experiments the LLM sometimes filled a particular field incorrectly, and verification that only checks whether the generated values exist in the supporting data does not catch this. This is why the author must review the specification. Generating a table directly means checking several hundred cells, whereas a specification has few fields, and correcting a faulty field and re-running propagates the fix across the whole table.

3.3 Limitations

This study covered one section — safety performance — among the many sections of a PSR report. Whether the results obtained here extend to tables under other safety factors, such as design or ageing, was not examined. The type of supporting data and the composition of tables differ by safety factor, so the range that a specification can express may differ as well. Tables that require combining several sources or transposing rows and columns were excluded from this study. The time an author spends reviewing a specification, and how many errors that review actually catches, were also not measured.

4. Conclusion

This study had the LLM produce a specification for filling a table rather than the table itself, leaving the actual filling to code.

Two types of table were run through the specification approach and the code-generation approach and compared against an already published report. The specification approach matched on every aggregation-type table and reached 96.8% cell agreement on the transcription-type table. The code-generation approach gave a different result on each run and matched in only four of fifteen runs. Changing the reference report did not alter the values entered in the table.

This difference stems not from the capability of the LLM but from what it was asked to produce. A specification has a fixed set of fields to complete, so the same conditions yield the same result, and where the

result does depart from the report the cause narrows to a particular field. Rather than checking several hundred cells, the author reviews a handful of fields, and correcting a faulty field and re-running propagates the fix across the whole table. In this way the reproducibility and reviewability required of documents submitted to a regulatory body can be secured.

Future work will widen the range of tables the specification approach can handle and extend it to other safety factors.

ACKNOWLEDGMENTS

This work was supported by Korea Hydro & Nuclear Power Co., Ltd. (No. 2025-AI-001-01).

REFERENCES

- [1] IAEA, *Periodic Safety Review for Nuclear Power Plants*, Specific Safety Guide No. SSG-25, 2013.
- [2] Nuclear Safety Act, Article 23 (Periodic Safety Review), and Enforcement Decree Articles 36–39, Republic of Korea.
- [3] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "PAL: Program-aided Language Models," *Proc. ICML*, 2023.
- [4] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks," *TMLR*, 2023.
- [5] Qwen Team, "Qwen3.6-35B-A3B," Hugging Face model card, Alibaba Cloud, 2026.
<https://huggingface.co/Qwen/Qwen3.6-35B-A3B>.