NORDDESIGN CONFERENCE
REYKJAVIK, ICELAND, 12-14 AUGUST 2024

Towards Automated Design: Automatically Generating Modeling Elements with Prompt Engineering and Generative Artificial Intelligence

Pranav Jayant Kulkarni, Denis Tissen, Ruslan Bernijazov, Roman Dumitrescu

Heinz Nixdorf Institute, Advanced Systems Engineering Research Group, Paderborn University

Abstract: Developing Intelligent Technical Systems (ITS) involves a complex process encompassing planning, analysis, design, production, and maintenance. Model-Based Systems Engineering (MBSE) is a key methodology for systematic systems engineering. Designing models for ITS requires harmonious interaction of various elements, posing a challenge in MBSE. Leveraging Generative Artificial Intelligence, we generated a dataset for modeling, using prompt engineering on large language models. The generated artifacts can aid engineers in MBSE design or serve as synthetic training data for AI assistants.

Keywords: Data-Driven Model-Based Systems Engineering, Automated Design, Large Language Models, Design With AI

1 Introduction

The development of *intelligent technical systems (ITS)* requires advanced approaches to meet the increasing system complexity and variety of stakeholder requirements. *Model-Based Systems Engineering (MBSE)* has proven to be a promising development approach to deal with growing system complexity and increased enterprise agility (Friedenthal 2023). In general, *Systems Engineering (SE)* focuses on developing holistic solutions and integrating system components to meet customer needs and functions (Hitchins 2007). SE starts by defining the system requirements, followed by designing system elements, synthesis, and validation of the complex system (Walden 2023). MBSE is an extension of document-based SE, where information about the system is formalized in a system model. This model-centered approach can provide a consistent and traceable system design necessary for interdisciplinary system development (Friedenthal 2023). System models help gain a deeper understanding of the connection between the system requirements and the system's emergent properties, internal structure, and behaviors. Modeling makes the complexity of integrating different perspectives manageable. System models are either designed in workshops, in which the models are subsequently digitized, or directly in digital form using a modeling tool (Tschirner 2016). Formal modeling languages, like SysML (Delligatti 2014), are used to capture the system design in a formal way.

However, integrating MBSE is still challenging for companies today (Bretz 2021). For instance, organizations need to comply with changing regulations, and the organization's employees lack information regarding the evolving regulations. In such cases, organizations need to invest sufficient time in employee training or obtain the subject matter expertise externally before integrating it into their development process and system models (Tissen et al. 2023). Tissen et al. introduced the iQBuddy model recommendation system, which addresses this problem by providing system engineers with suitable, ready-made model elements. iQBuddy aims to reduce redundant modeling work and allow system engineers to focus more on value addition and creative development activities (Tissen et al. 2023).

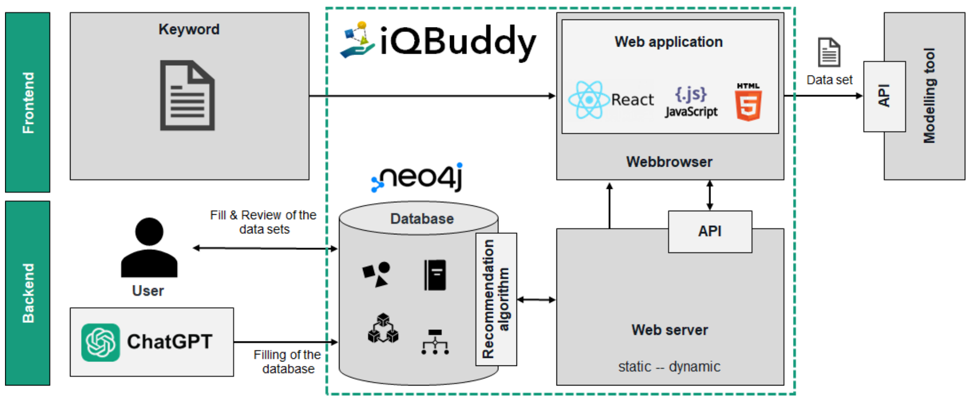


Figure 1: Basic architecture concept of iQBuddy (Tissen et al. 2023).

Figure 1 presents the implementation concept of iQBuddy. The system comprises a web application through which engineers interact with the backend to obtain their modeling elements. The elements are stored in a database and connected to the web application through a web server. This paper presents our approach for the automated generation of modeling elements data with *generative artificial intelligence (GenAI)* and its implementation details. There exist many ways to build a database. Techniques such as manual data gathering from various sources, importing (Biswas et al. 2020), or mining data (Lethbridge et al. 2005) from existing sources, and utilizing crowdsourcing platforms (Afshan et al. 2013) for data collection are widely used. The decision to adopt GenAI as our data generation technique stems from readily accessible resources within the project, technical capabilities of the team along with time and budgetary limitations. The following paper further describes our research approach (section 2), our solution idea (section 3), and detailed implementation of how GenAI is used to create the system elements (section 4), results (section 5), observations, and concluding statement (section 6).

2 Research Approach

This section describes the design principles followed to develop the iQBuddy demonstrator. Our research is influenced by the framework proposed by Blessing and Chakrabarti (Blessing and Chakrabarti 2009). It involves a comprehensive methodology consisting of four key steps: researching the existing system, understanding the stakeholders' needs, developing an initial prototype system, and finally, getting feedback, validating, and improving the system.

MBSE is a challenging approach, especially modeling systems that emerge as a hurdle (Proper and Bjekovic 2019). We can provide engineers with an assistance system (Ricci et al. 2010; Mussbacher et al. 2020). Such an assistance system can benefit non-experts who want to adopt MBSE practices better. A system element recommendation system will enhance the user's experience and throughput. Existing tools do not offer such support or features to the user. To test our hypothesis, we conducted a workshop with participants from research, industry professionals, and management. From the short literature analysis and the workshop feedback, the goal of building iQBuddy as a system modeling element recommender was derived. These correspond to the first two key steps of our research methodology.

After identifying the necessary stakeholder needs, we developed a prototype. GenAI was used to create a large modeling element database to meet the time constraints. This forms the paper's core and is further explained in section 4. We presented our system at a conference and received positive participant feedback, further solidifying the need for such a database. These tasks will be worked on later and described in detail in section 6.

3 Solution Idea

The solution idea is twofold, addressing the desired database structure, foundations of GenAI, and the conceptual idea for its creation. Section 3.1 elaborates on the structural design considerations, while section 3.2 delves into GenAI fundamentals necessary to create this data structure. Section 3.3 summarizes the conceptual idea for data generation.

3.1 FLP Database Structure

Figure 2 describes the structure of the database following a *function*, *logical element*, and *physical element* structure by the familiar (R)FLP (Requirement, Function, Logical Architecture, Physical Architecture) approach (Götz and Donges 2016). The functional elements outline general functions in a 'noun + verb' format, such as 'transform energy.' These functions relate to logical elements, like 'solar panel'. Logical elements are concretized with attributes, like 'electricity yield [kWh/year]' Logical elements are included at different system abstraction levels, such as 'photovoltaic system,' is on a higher abstraction level than 'solar panel,' 'battery', 'cables,' and 'inverter'. These elements are related to each other as a 'photovoltaic system' consists of the other elements. Physical elements list manufacturer-specific solutions links, offering examples for follow-up processes like purchasing. We generated this FLP database containing the modeling elements using generative artificial intelligence.

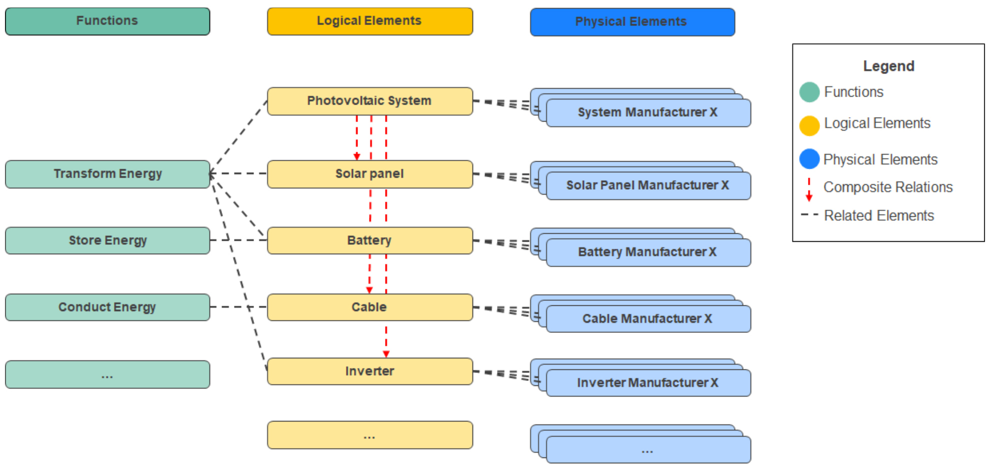


Figure 2: Structure and Relation of Elements in the Database.

3.2 Generative AI Background

GenAI models have gained significant attention due to their ability to produce diverse content, including text, images, music, conversations, code, or video. The breakthrough of GenAI models lies in their use of massive datasets and advanced techniques to generate impressively high-accuracy results. This has garnered significant attention and enthusiasm from people. For instance, consider OpenAI's ChatGPT, which amassed a user base of 100 million within just two months of its launch. To put this into perspective, TikTok took nine months to achieve a similar milestone, while Instagram took more than two years (Guardian 2023). ChatGPT is an example of a large language model (LLM), a GenAI model class that generates textual output based on user input. The general working of an LLM is given in Figure 3.

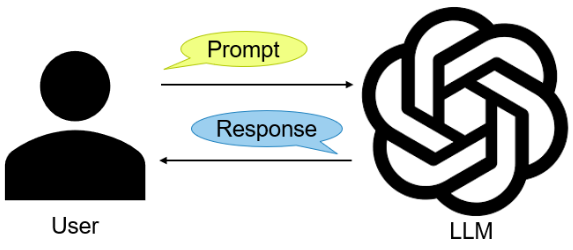


Figure 3: Prompting an LLM for appropriate response.

The process of 'asking the LLM' for a response is called prompting, as shown in Figure 3. Depending on the prompt, the LLM will give a specific response in a desired format or a vague and useless answer. The act of designing useful, meaningful prompts from the LLMs that yield the desired outcome in the desired format is called Prompt Engineering (PE) (McKinsey & Company 2023). Activities in PE include adding relevant details to help the model gain a more nuanced understanding of the user demands. Another technique to improve LLM's output is fine-tuning of models. Fine-tuning uses additional task-relevant data to improve model performance (Amazon Web Services Inc. 2023). Two popular approaches for fine-tuning LLMs are instruction finetuning (IFT) and parameter-efficient finetuning (PEFT). Instruction fine-tuning involves training LLMs with specific command-like inputs, encouraging the model to generate specific outputs aligned with those instructions (Chung et al. 2022). PEFT aims to improve the efficiency of model fine-tuning by modifying fewer parameters during the adaptation process. This technique reduces the computational load by selectively updating a smaller subset of the model's parameters, optimizing performance with minimal adjustments, and making the fine-tuning process more resource-efficient (Ding et al. 2023).

3.3 Data Generation Conceptual Idea

To leverage the capabilities of LLMs, good prompts are needed. These prompts should be reusable patterns, or templates so that they can be used for more than one type of system or artifact type. Due to resource and time constraints, we do not delve into fine-tuning LLMs, or PEFT, with documents related to MBSE. Thus, our solution idea was primarily focused on prompt engineering, without any relying on local documents or other sources. We also planned the set of inputs to LLMs which would output the desired artifact group in the desired format. To generate system functions, we planned to provide a system description and an output format. Next, we'll use the generated functions and prompt templates to create logical elements. We further extend this chain by providing the generated logical elements to physical elements. After generating all elements, we conduct post-processing to clean and extract the data into the desired format using a Python script. Additionally, we leverage a LLM to establish relationships among potentially associated elements. The implementation details for this conceptual idea follow in section 4.3.

4 Implementation Detail

The implementation contains three aspects, starting with the database setup (section 4.1), the model selection, deployment, and initialization (section 4.2), and the data generation (section 4.3). This contains the generation of the dataset, the model set up for prompting, and utilization for generating modeling artifacts.

4.1 Database Overview

We generate the above-described FLP data set according to the general schema of our database, which is detailed in Figure 4. Each element should have its own identifier and name. On top of that, we plan to add additional information about the logical and physical elements for the end users' benefits. Logical elements contain the description, inputs accepted by the elements, outputs provided by them, and some characteristic parameters. Physical elements contain additional product-specific information and a URL that can be used to view the web page associated with the product.

FLP ensures a trace from system requirements to corresponding system functions, further to logical elements, and finally, the actual real-world components or physical elements (Walden 2023). In a database, we must create meaningful relationships or connections between system functions, logical elements, and physical elements. We call these relations *cross-group* or *inter-group relations* as they trace related elements across different artifact groups. In the schema (Figure 4), they are traced by using *foreign key (FK)* dependencies among different artifact groups. Engineers and stakeholders, in general, may use similar words to describe system functions or logical elements depending on their requirements or preferences. For example, 'convert energy' and 'transform energy' are similar system functions. To accommodate such cases, we created relations within the same artifact group or *intra-group relations*. These are traced using *self-referencing keys (SRK)* in our schema (Figure 4). Logical elements can be composed of many different logical elements, like an automated vehicle consisting of wheels, an engine, etc. To address such cases, we created another form of intra-group relations called composite relationships. Composite relations are only traced in the logical elements group. Unlike synonymous and inter-group relations, these composite relations are directional. For example, it is necessary to symbolize that an autonomous vehicle consists of an engine, not vice versa.

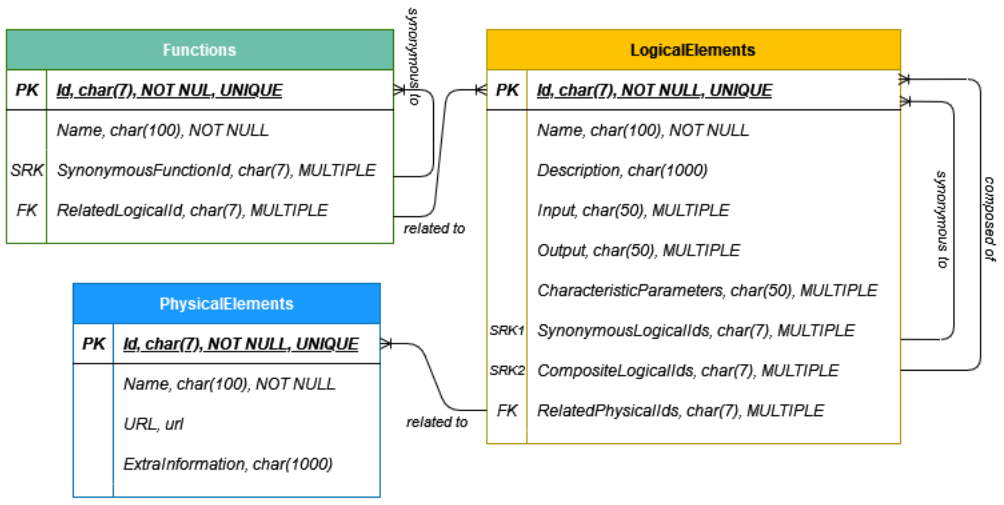

Functions
PK
Id, char(7), NOT NUL, UNIQUE
Name, char(100), NOT NULL
SRK
SynonymousFunctionId, char(7), MULTIPLE
FK
RelatedLogicalId, char(7), MULTIPLE
synonymous to
related to
LogicalElements
PK
Id, char(7), NOT NULL, UNIQUE
Name, char(100), NOT NULL
Description, char(1000)
Input, char(50), MULTIPLE
Output, char(50), MULTIPLE
CharacteristicParameters, char(50), MULTIPLE
SRK1
SynonymousLogicalIds, char(7), MULTIPLE
SRK2
CompositeLogicalIds, char(7), MULTIPLE
FK
RelatedPhysicalIds, char(7), MULTIPLE
synonymous to
composed of
PhysicalElements
PK
Id, char(7), NOT NULL, UNIQUE
Name, char(100), NOT NULL
URL, url
ExtraInformation, char(1000)
related to

Figure 4: Database Schema.

4.2 Model Selection, Deployment, and Initialization

To generate the above-defined database, we used LLMs from Azure OpenAI Service (Microsoft Azure 2023). We deployed the GPT-3.5 Turbo-03-01 model on Azure AI Studio, leveraging its powerful capabilities. This model has a maximum token capacity and rate limit of 200,000 tokens per minute, which is sufficient for our use case requirements. Due to resource constraints, we chose the baseline GPT-3.5 version over the more advanced GPT-4.0. GPT-3.5 is more cost-effective than its counterpart. Thus, we set up an LLM on Azure for frequent querying. We used only one model to generate all three artifact groups but updated the hyperparameters used during prompting. We used a fixed set of hyperparameters for one artifact group. We wrote Python scripts to query this model and used its LangChain library. LangChain facilitates the development of context-aware LLM applications by seamlessly connecting models with various context sources, such as prompt instructions and few-shot examples (LangChain 2023). We use these LLMs and Python language capabilities to query or prompt the deployed model and generate the required database.

4.3 Data Generation Process

As described in Figure 3, the first step for generating data from LLM is defining a good prompt template. We tried three different prompt structures. The first template was a simple prompt string stating the requirements for an artifact group. For this template, we asked the LLM to generate the output as a JSON. In the second template, we added a one-shot learning example to the earlier template. Finally, in the third template, we used few-shot learning by providing three examples to the LLM. The model can use these examples provided in the prompt and learn about the specific requirements, for instance, output format. This technique is called in-context learning in LLMs and is a valuable prompt engineering technique that can achieve better results (Kirk et al. 2022).

Two MBSE experts from our team examined the generated output from the three templates. The one-shot learning template was selected as it yielded the best outcome. The output generated with the first template was vague and not suitable for generating large scale data. We use one-shot inference over the few-shot inference technique because we observed that the deployed LLM produces a more coherent answer with one-shot for this use case. Using few-shot learning, the model generated multiple duplicate elements as the output. Table 1 describes the selected prompt template which yielded the best solution. Although LLMs are more than capable of generating output in a structured manner (Chung et al. 2022), for instance, in a JSON or a CSV format. However, we found that some additional characters are needed to generate output in these formats (for example, curly brackets in JSONs and commas in CSV). These additional characters use up token space from the allocated rate limit of the model. Also, we needed additional flexibility in assigning IDs to the output to ensure relationships among the elements from different artifact groups. For these reasons, we decided to generate output as a table and add a single character as a separator (we chose pipe ['|'] as the separator).

The prompt templates in Table 1 contain a static and a variable component. The variable component is highlighted within curly brackets. Before querying the model, the variable string is replaced with an actual value in the Python script. For instance, to generate functions for an ITS, one needs to provide the system name, a short description of how it works, and the number of functions that should be generated. An example set of actual values that can be substituted for the carriable component is: (a digital camera, a device that captures and records visual images and scenes by bouncing and collecting light reflected from surfaces, 10). Each template also contains an example instruction-response set.

Table 1: One-shot learning prompts used to generate data in various artifact groups.

	Prompts
Functions	I want to build an **{system}** that **{short_description}**. Please follow the instructions below to generate all the required system functions for this complex system: 1. Generate **{func_count}** different system functions for the **{system}** in the 'verb + noun' form. Restrict the functions to two words as far as possible. 2. Provide the output as a table of IDs, function names, alternate names, or synonyms in the sentence case. 3. Please ensure the functions are clearly defined and cover various actions related to the **{system}**. Do not give anything else other than the system functions. System: automatic photovoltaic system Functions: | id | name | synonym | | 1 | Charge Battery | Power up Battery; Charge Cell | | 2 | Store Energy | Save Energy; Capture Energy | | 3 | Convert Energy | Transform Energy | | 4 | Monitor Weather | Check Climate | : : : System: **{system}** Functions:
Logical elements	A logical element for a system is an electro-mechanical component that performs a certain function. Please follow the instructions below to generate the required logical elements for the given system function: 1. Generate at most **{log_count}** different logical elements for the given function. 2. Provide the output as a table with six columns for the Name of the logical element, a one-line description of the element, synonymous elements, input, output, and characteristic parameters. 3. Please ensure that the logical elements are clearly described and can efficiently perform the functions. 4. Do not generate more functions. 5. Give only the tabular output. Function: Store Energy Logical: | Name | Synonyms | Description | Inputs | Outputs | Characteristic Parameters | | Battery | Cell; Rechargeable battery| A source of electric power consisting of one or more electrochemical cells with external connections for powering electrical devices. | battery status (information) electric charge(energy) | battery status (information), electric charge(energy) | Capacity (kWh), efficiency (%), cycle life, charge/discharge rate (kW), voltage (V) | | Wire | Cable; Cables; Wires; Connecting Wires | The electro-technical component transporting electricity to transmit energy and information. | Data (information); Electricity (energy) | Data (information); Electricity (energy) | Length (m); cross-section(mm^2); resistance(Ω); current carrying capacity (Amps); Temperature (°C) | : : : Function: **{function}** Logical:

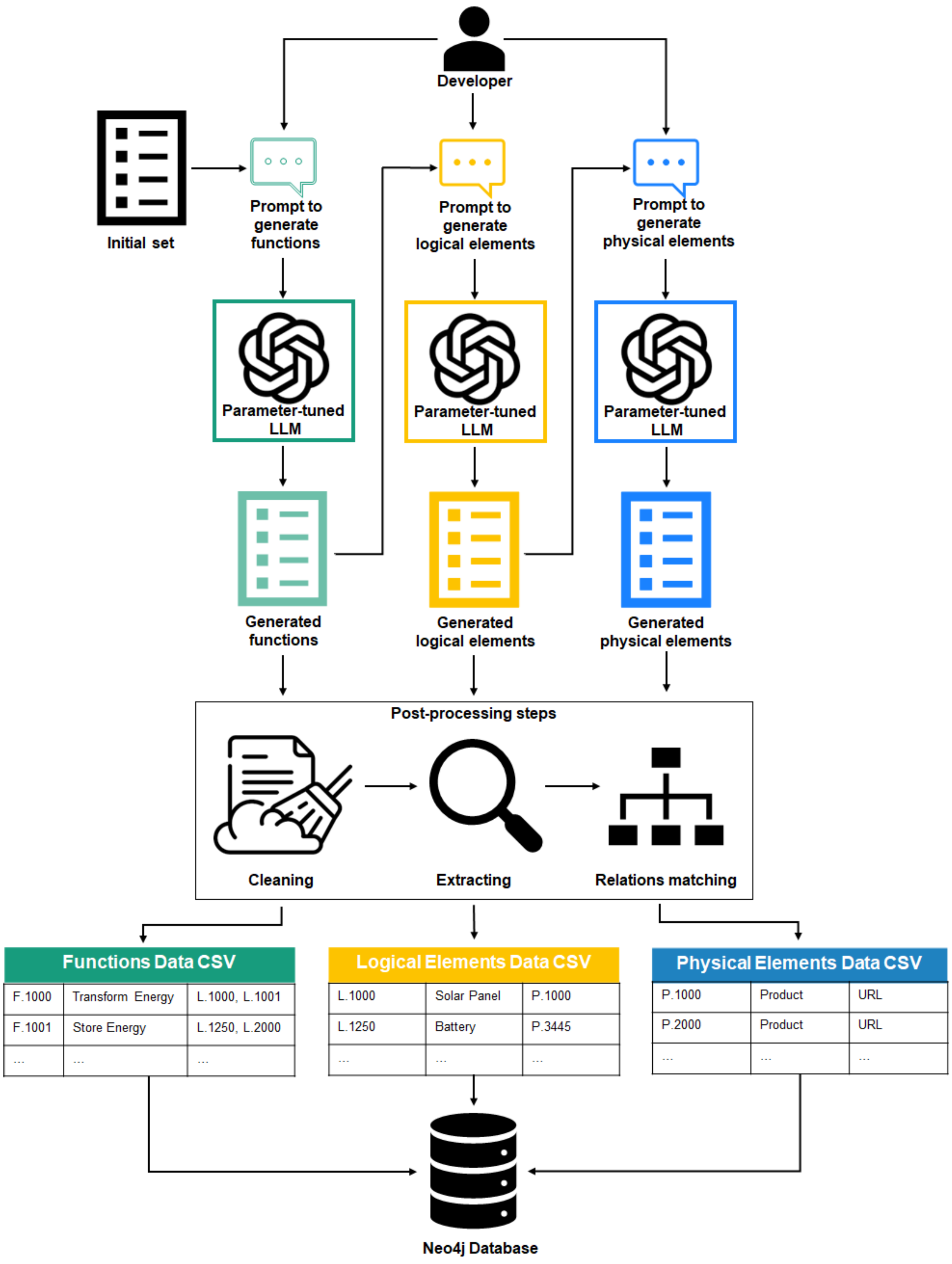
Physical elements	I want to buy a **{component}** for industrial and commercial use. Please follow the instructions below to find at most five real-world components 1. Crawl the web (Google, Amazon, retail suppliers, wholesale suppliers) to find out at most **{phys_count}** real-world electrical components that I can buy from the seller. 2. Provide the output as a table of names, URLs, special information 3. Do not provide URLs that lead to a page not being found. 4. Do not give anything else other than the table. component: Buzzer Physical: | Name | URL | Extra Information | | Buzzer1 | https://www.LinkToBuyProduct1.de | Operating Voltage = 24 V/DC; Color = Red; Flash Frequency= 84 flashes/min; | | Buzzer2 | https://www.linkToBuyProduct2.com | 80 dB 50 Hz Electromagnetic buzzer, AC 220V 25 mA industrial high-power alarm buzzer, for electronic products (AC 220V) | : : : component: **{component}** Physical:

The data generation process begins with an initial set of system names and descriptions, which were identified by our experts in MBSE. System functions are generated iteratively for each input system and the prompt template. Subsequently, the generated system functions and prompt template for logical elements are utilized to generate logical elements. Similarly, the process is repeated for generating physical elements, with the additional challenge of generating real URLs for physical products. Afterward, post-processing steps are performed to remove blanks, garbage strings, and duplicates. Finally, the generated data is formatted into CSV files and imported into the Neo4j database using *cypher queries*, a database language to read, create, update data in a graphical structure.

Figure 5 describes the overall data generation process. We use an initial set of system names and their descriptions to generate functions. We iterate over them and prompt the model by replacing the variable text with the names and descriptions from the initial set. We aim to generate 20 functions for each input system. In every iteration, we saved the intermediate results and the functions generated for one system in text files on our local machine. After completing this process, we consolidate the intermediate results to obtain the final generated function set.

We use these generated system functions and the prompt template for logical elements to generate the logical elements from the model. As mentioned above, we use the same model for generating all elements from the three artifact groups. We only change the model's hyperparameters and clear the model's memory to provide a context-free environment for further generation. Generating logical elements follows a similar workflow to that of the system functions. We iterate over the generated functions and prompt the model for logical elements related to it using the prompt template. Once all logical elements are generated, we prompt the model again for composite logical relationships among the generated logical elements. We have generated the logical elements from the system function generated in the prior step using LLM. This makes a logical correlation between the functions and the generated logical elements. We create trace links in the database to link a function to the system elements generated based on it.

Finally, we use the same process to generate physical elements. In physical elements, the additional challenge is to generate real URLs to the physical products available in the market. In the prompts, we provided real-world examples of products and their links to check if the LLM can generate links. Owing to legal issues, we replaced the actual product and URLs with dummy values in Table 1 and Figure 5. After generating the physical elements from logical elements, we map the physical and logical elements as a relationship in the logical elements. After generating all the data, we perform post-processing steps of removing blanks, garbage strings, and duplicates. The generated data is then formatted into three CSV files, one each for each artifact type. These CSV files are in the same schema as described in Figure 4. These CSV files are then easily imported into the Neo4j database.



Functions Data CSV		
F.1000	Transform Energy	L.1000, L.1001
F.1001	Store Energy	L.1250, L.2000
...	...	...

Logical Elements Data CSV		
L.1000	Solar Panel	P.1000
L.1250	Battery	P.3445
...	...	...

Physical Elements Data CSV		
P.1000	Product	URL
P.2000	Product	URL
...	...	...

Figure 5: Data Generation Process.

5 Results

Our study showcases that GenAI can be used to create engineering data. In this section, we describe the quantity and quality of the generated data in detail. Two experts in the data engineering field assessed the generated data. According to their findings, the overall quality of the data was satisfactory. However, the specificity and effectiveness of the data generation can be improved by providing more concrete inputs to the LLM. There is no doubt regarding the quantity of data that can be generated using LLMs. We created 624 functional elements, 3376 logical elements, and 12611 physical elements. The quantity of different kinds of relations, mentioned in Figure 4, are presented in Table 2. Using LLMs, we generated all parameters except the IDs for each artifact type explained in Figure 4. Another interesting observation through the generated data is that LLMs can give us synonymous relations among elements from the same group (function to function and logical to logical). For instance, in the generated data, 'validate transaction' and 'confirm payment' are synonymous functions, and logical elements 'network cables' are synonymous with 'LAN cables' This diversity in the generated output provides flexibility and choice for the end users to select a suitable element specific to their requirements. For logical elements especially, the LLMs generated composite relations for most elements. For instance, the logical element 'laser distance sensor' comprises 'lens,' 'transistor,' and 'photodiode' in the generated data. Apart from such intra-group relations, LLMs can generate appropriate relations between elements from two groups (function to logical and logical to physical relations). For example, the function 'drive car' is related to the logical element 'car engine', and this logical element is related to the physical element '6.5 HP Horizontal Shaft Gas Engine' (a 212cc gas engine by Predator).

Table 2: Number of Relations Established by LLM

Relation	Related Entities	Count
synonyous to	Function-Function	145
related to	Function-Logical	3873
synonyous to	Logical-Logical	4402
composed of	Logical-Logical	6065
related to	Logical-Physical	21312
Total		**35797**

Owing to the heavy volume, we are still in the process of a complete quality analysis. However, our initial observations are presented in this paragraph. Although we generated a heavy volume of data, the quality of data can still be improved in the future. Some logical elements are missing descriptions, input, or characteristic parameters. Some fields show garbage values generated by LLMs, for instance, a sequence of the same character (----- or ….) or words like 'name' and 'description' instead of generating the name or description of an element. We found some elements with incorrect descriptions or characteristic parameters due to hallucination tendencies from the LLM. Some data is redundant. For example, 'Solar cell,' 'Solar Cell,' and 'Solar cells' all point to one logical element: a solar cell. However, these were encountered multiple times in the LLM-generated output. Some composite relationships do not make any sense. For example, we discovered composite relations that are impossible in the real world, like 'wire' is composed of a 'solar panel' instead of the other. A major problem lies with the purchase URLs in the physical elements, where the LLM has generated many fake URLs. To tackle these challenges, we can refine the prompt templates by fine-tuning and implementing a robust post-processing pipeline to detect and rectify erroneous data entries. Our further research endeavors will investigate these strategies to enhance the quality and reliability of the data generation process.

6 Discussion

We generated a database of modeling elements using LLMs, showcasing the potential of GenAI for generating artificial data in the engineering domain. GenAI tools are widely used today. However, their application in the engineering field is limited. This is predominantly because GenAI models can generate a large quantity of data but not necessarily good quality. There is also a lack of guidelines for engineers wanting to leverage these tools for their use case, given the abundance of GenAI tools available online (Baciu 2023). Addressing these challenges will be crucial in unlocking the full potential of GenAI and LLMs in engineering endeavors.

We acknowledge the limitations in our approach to generating this database. The simplicity of our approach, relying solely on prompt engineering, leads to some poor-quality data being generated by the LLM. Due to resource constraints, heavy fine-tuning of the models was not extensively pursued, potentially limiting the models' performance. Another aspect of improvement could be the choice of model. A comparable process could be done with other LLMs, like GPT-4.0, and a comparative study of the quality and quantity of output could be performed. We also did not incorporate a document QA-retrieval approach, which could leverage reference documents to enhance the relevance and accuracy of the generated output. Addressing all these limitations remains a task for us in the future. Another interesting task is to integrate the LLM model with a crawling mechanism that facilitates the 'Search + LLM approach' (Kirk et al. 2022). In terms of physical elements, where there is little scope for abstraction and necessity for real-world data, a crawler can add to the knowledge base of the LLM. It may also be useful in verifying the output generated by it.

Another crucial area for research is the verifiability of the generated output. LLMs and other GenAI models tend to hallucinate, i.e., generate false or illogical output. A crawler may also be used to validate the generated data. Additionally, we may also look in the direction of existing factual knowledge graphs like WikiData and ConceptNet as a source to fine-tune our LLM or for verifying the generated data. We also plan to make this database available for usage for a limited audience of systems engineers and give an option to them to rate the quality of generated entities and the relations between them. This will give us an idea regarding the quality of the generated data and also guide us to develop better prompts or post-processing techniques to improve the output.

References

Afshan, Sheeva; McMinn, Phil; Stevenson, Mark (2013): Evolving Readable String Test Inputs Using a Natural Language Model to Reduce Human Oracle Cost. In : 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation: IEEE.

Amazon Web Services Inc. (2023): Fine-tune a foundation model. AWS. Available online at https://docs.aws.amazon.com/sagemaker/latest/dg/jumpstart-foundation-models-fine-tuning.html.

Baciu, Assaf (2023): Tackling the Fragmented Nature of Multiple GenAI Tools. Forbes. Available online at https://www.forbes.com/sites/forbestechcouncil/2023/12/26/tackling-the-fragmented-nature-of-multiple-genai-tools/.

Biswas, Neepa; Sarkar, Anamitra; Mondal, Kartick Chandra (2020): Efficient incremental loading in ETL processing for real-time data integration. In *Innovations Syst Softw Eng* 16 (1), pp. 53–61. DOI: 10.1007/s11334-019-00344-4.

Blessing, Lucienne T. M.; Chakrabarti, Amaresh (2009): DRM, a Design Research Methodology. London: Springer London.

Bretz, Lukas Helmut (2021): Rahmenwerk zur Planung und Einführung von Systems Engineering und Model-Based Systems Engineering. With assistance of Roman Dumitrescu, Jürgen Gausemeier, Maarten Bonnema, Universitätsbibliothek Paderborn. Paderborn.

Chung, Hyung Won; Le Hou; Longpre, Shayne; Zoph, Barret; Tay, Yi; Fedus, William et al. (2022): Scaling Instruction-Finetuned Language Models. Available online at https://arxiv.org/pdf/2210.11416.pdf.

Delligatti, Lenny (2014): SysML distilled. A brief guide to the systems modeling language. Upper Saddle River, NJ: Addison-Wesley. Available online at https://learning.oreilly.com/library/view/-/9780133430356/?ar.

Ding, Ning; Qin, Yujia; Yang, Guang; Wei, Fuchao; Yang, Zonghan; Su, Yusheng et al. (2023): Parameter-efficient fine-tuning of large-scale pre-trained language models. In *Nat Mach Intell* 5 (3), pp. 220–235. DOI: 10.1038/s42256-023-00626-4.

Friedenthal, Sanford (2023): Future Directions for MBSE with SysML v2. In *Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2023)*, pp. 5–9. DOI: 10.5220/0011946300003402.

Götz, Adriana; Donges, Christian (2016): Automatisierter Übergang vom dokumenten-zum modell-zentrierten Requirements Engineering als Ausgangsbasis für MBSE. München: Carl Hanser Verlag GmbH & Co. KG.

Guardian (2023): ChatGPT reaches 100 million users two months after launch. Unprecedented take-up may make AI chatbot the fastest-growing consumer internet app ever, analysts say. Available online at https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app.

Hitchins, Derek K. (2007): Systems engineering. A 21st century systems methodology. Chichester, England, Hoboken, NJ: John Wiley (Wiley series in systems engineering and management). Available online at https://onlinelibrary.wiley.com/doi/book/10.1002/9780470518762.

Kirk, James R.; Wray, Robert E.; Lindes, Peter; Laird, John E. (2022): Integrating Diverse Knowledge Sources for Online One-shot Learning of Novel Tasks. Available online at http://arxiv.org/pdf/2208.09554v3.

LangChain (2023): Introduction. Available online at https://python.langchain.com/docs/get_started/introduction.

Lethbridge, Timothy C.; Sim, Susan Elliott; Singer, Janice (2005): Studying Software Engineers: Data Collection Techniques for Software Field Studies. In *Empir Software Eng* 10 (3), pp. 311–341. DOI: 10.1007/s10664-005-1290-x.

McKinsey & Company (2023): What is prompt engineering? Available online at https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-prompt-engineering.

Microsoft Azure (2023): Azure OpenAI Services. Build your own copilot and generative AI applications. Available online at https://azure.microsoft.com/en-us/products/ai-services/openai-service.

Mussbacher, Gunter; Combemale, Benoit; Kienzle, Jörg; Abrahão, Silvia; Ali, Hyacinth; Bencomo, Nelly et al. (2020): Opportunities in intelligent modeling assistance. In *Softw Syst Model* 19 (5), pp. 1045–1053. DOI: 10.1007/s10270-020-00814-5.

Proper, Hendrik A.; Bjekovic, Marija (2019): Fundamental challenges in systems modelling. In *Lecture Notes in Informatics (LNI), Gesellschaft für Informatik*. Available online at https://dl.gi.de/bitstreams/8d495bf2-26b8-4233-825d-401d9c6df453/download.

Ricci, Francesco; Rokach, Lior; Shapira, Bracha (2010): Recommender Systems Handbook: Scholars Portal (Recommender Systems Handbook).

Tissen, Denis; Tschirner, Christian; Koldewey, Christian; Dumitrescu, Roman (2023): Nachhaltigkeit durch Daten im MBSE: Einblick in die Entwicklung eines Modell-Empfehlungssystems. In Walter Hoch, Daria Wilke, Stefan Dreiseitel, Rüdiger Kaffenberger (Eds.): Tag des Systems Engineering: Gesellschaft für Systems Engineering e. V.

Tschirner, Christian (2016): Rahmenwerk zur Integration des modellbasierten Systems Engineering in die Produktentstehung mechatronischer Systeme. Fakultät für Maschinenbau. Veröffentlichungen der Universität, Paderborn. Available online at https://nbn-resolving.org/urn:nbn:de:hbz:466:2-27332.

Walden, David D. (Ed.) (2023): Systems engineering handbook. A guide for system life cycle processes and activities. International Council on Systems Engineering. Fifth edition. Hoboken, NJ: Wiley. Available online at https://learning.oreilly.com/library/view/-/9781119814290/?ar.

Contact: Pranav Jayant Kulkarni, Paderborn University, pranav.jayant.kulkarni@hni.uni-paderborn.de