SYM-AM-24-048

Excerpt from the Proceedings of the Twenty-First Annual Acquisition Research Symposium

Acquisition Research: Creating Synergy for Informed Change

May 8–9, 2024

Published: April 30, 2024

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

The research presented in this report was supported by the Acquisition Research Program at the Naval Postgraduate School.

ACQUISITION RESEARCH PROGRAM
DEPARTMENT OF DEFENSE MANAGEMENT
NAVAL POSTGRADUATE SCHOOL

Leveraging Generative AI to Create, Modify, and Query MBSE Models

Ryan Longshore—is an 18-year veteran of both the defense and electric utility industries. In his current role at Naval Information Warfare Center Atlantic (NIWC LANT), Ryan leads a diverse team of engineers and scientists developing and integrating new technologies into command and operations centers. Ryan is heavily involved in the Navy's digital engineering transformation and leads multiple efforts in the model-based systems engineering and model-based engineering realms. Ryan earned a BS in electrical engineering from Clemson University and an MS in systems engineering from Southern Methodist University, and is currently pursuing his PhD in systems engineering from the Naval Postgraduate School. He is a South Carolina registered Professional Engineer (PE) and an INCOSE Certified Systems Engineering Professional (CSEP), and has achieved the OMG SysML Model Builder Fundamental Certification. [Ryan.longshore@nps.edu]

Ryan Bell—is an 8-year experienced engineer in the defense industry. In his current role at Naval Information Warfare Center Atlantic (NIWC LANT), Ryan provides modeling and simulation expertise to a variety of programs for the Navy and USMC. He specializes in simulating communication systems in complex environments and is an advocate for the use of digital engineering early in the systems engineering life cycle. Ryan earned a BS in electrical engineering and a MS in electrical engineering from Clemson University with a focus on Electronics and is currently pursuing his PhD in systems engineering at the Naval Postgraduate School. He is a South Carolina registered Professional Engineer (PE), published author, and teacher. [Ryan.bell@nps.edu]

Ray Madachy, PhD—is a Professor in the Systems Engineering Department at the Naval Postgraduate School. His research interests include system and software cost modeling; affordability and tradespace analysis; modeling and simulation of systems and software engineering processes; integrating systems engineering and software engineering disciplines; and systems engineering tool environments. His research has been funded by diverse agencies across the DoD, National Security Agency, NASA, and several companies. He has developed widely used tools for systems and software cost estimation, and is leading development of the open-source Systems Engineering Library (se-lib). He received the USC Center for Systems and Software Engineering Lifetime Achievement Award for "Innovative Development of a Wide Variety of Cost, Schedule and Quality Models and Simulations" in 2016. His books include Software Process Dynamics and What Every Engineer Should Know about Modeling and Simulation; co-author of Software Cost Estimation with COCOMO II, and Software Cost Estimation Metrics Manual for Defense Systems. He is writing Systems Engineering Principles for Software Engineers and What Every Engineer Should Know about Python. [rjmadach@nps.edu]

Abstract

Generative AI tools, such as large language models (LLMs), offer a variety of ways to gain efficiencies and improve systems engineering processes from requirements generation and management through design analysis and formal testing. Large acquisition programs may be particularly well poised to take advantage of LLMs to help manage the complexities of system and system of systems acquisitions. However, generative AI tools are prone to a variety of errors.

Our research explores the ability of current LLMs to generate, modify, and query Systems Modeling Language (SysML) v2 models. Techniques such as Retrieval-Augmented Generation (RAG) are utilized to add domain-specific knowledge to an LLM and improve model accuracy. A preliminary case study is presented where the number of prompts to generate the models is minimized. We also discuss the limitations of LLMs and future systems engineering research related to LLMs.

Introduction

Applications of artificial intelligence (AI) have the potential to change many fields including systems engineering. Large language models (LLMs) may enable systems engineers to build, modify, and query systems models through plain language prompts, reducing the need for domain-specific modeling knowledge. Systems Modeling Language version 1 (SysML v1) is a graphical language to describe a system and its interaction with its environment. The use of graphical diagrams requires a multi-modal LLM to interact with a SysML v1.x model. However, the introduction of textual notation into SysML version 2 (SysML v2) enables more unimodal LLMs to interact with a systems model.

This paper explores the use of ChatGPT 3.5, ChatGPT 4, and a custom GPT, Senior System Engineer – Systems Modeler (SSE-SM), to create, modify, and query SysML v2 models from natural language prompts. This paper also identifies methods to reduce the number of prompts required to create models, including Retrieval-Augmented Generation (RAG). This paper also identifies some limitations of LLMs when applied to systems modeling and suggests areas of future research.

Background and Related Research

Model Based Systems Engineering

The International Council on Systems Engineering (INCOSE) defines MBSE as "a formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later lifecycle phases" (INCOSE, 2007). Traditionally, systems engineering was performed using a document-based approach. However, the growing complexity of systems and systems of systems necessitates a need to "capture, analyze, share, and manage the information associated with the complete specification of a product" (Friedenthal et al., 2009). Capturing this information in a model enables the system to be viewed by multiple stakeholders from their respective viewpoints, streamlines collaboration between systems and domain-specific engineers, enables tracing of requirements through design and verification/validation activities, and provides a formal way to identify, analyze, and track system changes/defects as a solution is developed (Carroll & Malins, 2016).

Systems Modeling Language (SysML)

To implement a MBSE methodology, Systems Modeling Language (SysML) v1.0 was developed and adopted in 2006 with formal publication in 2007 by the Object Management Group (OMG). Formally released in 2019, SysML v1.6 is the latest release of the SysML v1.x standard. SysML v1.x is an extension of the Unified Modeling Language (UML) 2 standard containing some, but not all, elements of UML 2 and some new SysML specific elements (OMG, n.d.). SysML v1.x is a graphical language consisting of nine diagram types where each diagram represents a view of the underlying model elements.

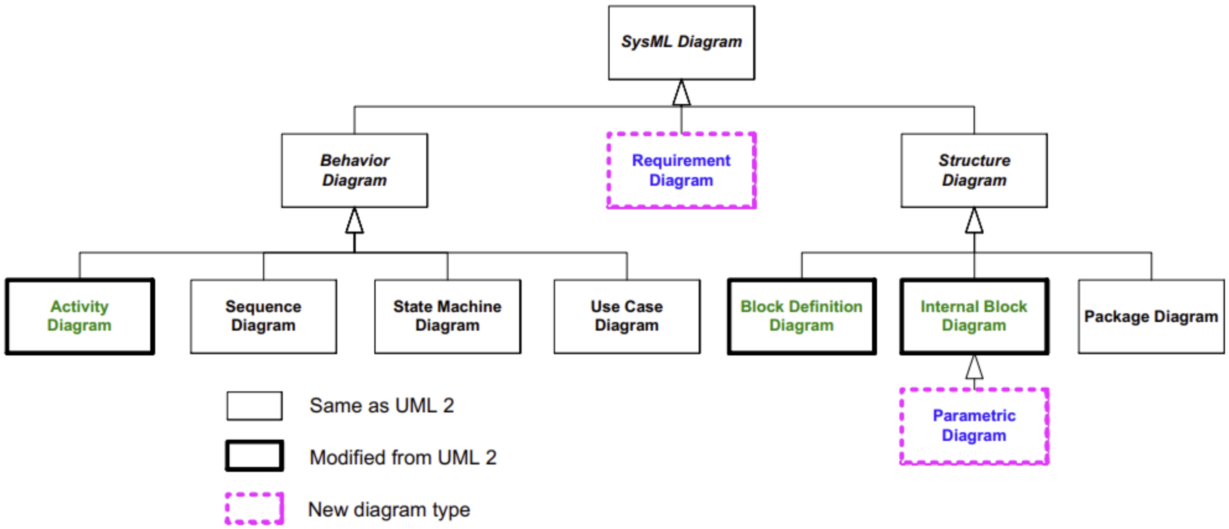

Figure 1. SysML v1.x Diagram Taxonomy (OMG, 2017)

While SysML v1 has been widely adopted across government, industry, and academia, many weaknesses were noted that led to the development of SysML v2. In an article introducing SysML v2, Friedenthal and Seidewitz (2020) state the objectives of SysML v2 are to improve:

- Precision and expressiveness of the language
- Consistency across language concepts
- Usability for model developers and model consumers
- Interoperability between systems modeling tools and other model-based engineering tools
- Extensibility to support modeling of domain-specific concepts (Friedenthal & Seidewitz, 2020)

Key SysML v2 enhancements address many of the limitations of SysML v1 (Stachecki, 2024):

- New metamodel grounded in formal semantics not constrained by UML
- Addition of textual notation
- Addition of a standardized API: There were several vendors for SysML v1 tools. However, each vendor implemented the standard differently, which led to very limited, and often, no interoperability between MBSE tools.

SysML v2 also utilizes the concept of standard views instead of diagrams to view the model. In addition to two new views, all diagram types of SysML v1 can be replicated as standard views in SysML v2 (Figure 2).

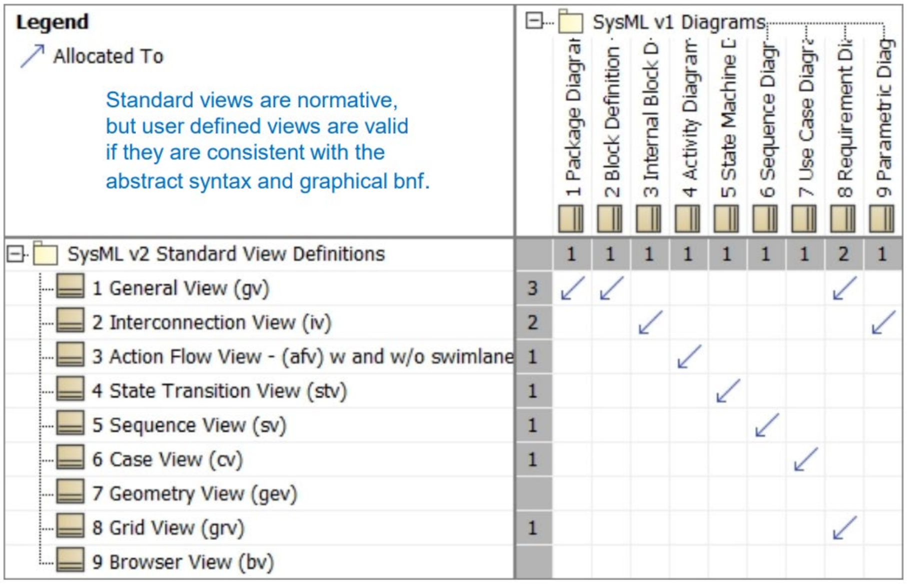

Figure 2. Standard View Definitions (SysML v2 vs SysML v1)(Friedenthal, 2023)

SysML v2 Parts

The case study in this paper utilizes a structural model built from SysML v2 parts. In the SysML v2.0 specification, a part definition is a kind of item definition that “represents a modular unit of structure such as a system, system component, or external entity that may directly or indirectly interact with the system.” Part definitions are denoted in textual notation as ‘part def’. Part usages are a usage of the part definition and denoted in textual notation as ‘part’ (OMG, 2023). A key difference between SysML v1.x and v2 is the blocks from v1.x do not exist in v2. However, v2 parts can take the place of blocks in most cases.

Large Language Models (LLMs)

Until recently, the field of natural language processing (NLP) required specific technical knowledge and acuity not possessed by the general public. While formal language models (LM) and NLP pre-date modern computing, ELIZA, demonstrated in 1966, is one of the first known computer LMs that provided a response via the now ubiquitous chatbot interface (Weizenbaum, 1966). However, the introduction of LLMs such as ChatGPT, Bard, and Claude transformed NLP from a niche field into a field accessible to everyone by demonstrating an impressive capability to understand human text prompts and generate intelligent text responses.

Overview of LLMs

Powered by deep learning and large training datasets, LLMs employ a transformer architecture to understand and interpret the relationships within text sequences (Amazon Web Services, 2024). The transformer architecture was introduced by Google focused on an attention mechanism to enable significantly more parallelization than previous architectures (Vaswani et al., 2017). The increased efficiency from parallelization coupled with the improvements in graphics processing units (GPUs) substantially reduced LLM training time and made the training of LLMs for widespread use feasible.

There are both proprietary and open source LLMs. Common proprietary models include OpenAI’s GPT models while a variety of open-source models such as Llama, BERT, mistral, and Falcon have been released by private companies, academic institutions, and other parties. Many open-source models are available on sites such as HuggingFace and can be installed and run locally (hardware dependent) using tools such as LM Studio (Arya, 2024).

LLMs vary in their size and number of parameters. The number of parameters often, but not always, leads to a trade-off between speed and accuracy. A 7B LLM has 7 billion parameters while a 13B LLM has 13 billion parameters. Some proprietary LLMs utilize 175B or more parameters (Yao et al., 2024). A 13B model will likely out-perform a 7B model on general knowledge tasks, but the 7B model may perform better in domain-specific tasks it is specifically trained to complete. Introducing new information can enhance accuracy, and when models need finetuning or retraining, updating smaller models is typically more practical and faster (Huertas, 2023). Smaller models also require less computing power for inference and often respond much quicker than larger models.

Improving Responses from LLMs

LLMs are only as good as the datasets they are trained on. To improve responses to domain-specific requests, it is often necessary to provide additional context to a model through one of the following methods:

- RAG utilizes an external data to augment the LLM's knowledge without changing the underlying LLM's parameters (weights; Nucci, 2024). RAG can be particularly useful for fields where new knowledge is generated regularly or when information is proprietary/private and the user desires to not make it a permanent part of the LLMs training set (Nucci, 2024).
- Finetuning slightly adjusts an LLM's internal parameters by teaching the LLM specialized knowledge (Nucci, 2024). Nucci stresses only a small number of parameters are adjusted, which results in the significant time savings of finetuning vice re-training the entire model.
- Prompt Engineering
    - Zero-shot prompts are requests that challenge an LLM to perform a task correctly on its first try, even though it hasn't been directly trained for that specific task (Oleszak, 2024). They are often used for simple tasks that only require general knowledge or when domain-specific knowledge was included in the training set or provided via RAG or finetuning.
    - For more complex tasks that require multi-step reasoning or when an LLM is not aware of domain-specific knowledge, few-shot prompts may be used to teach an LLM through examples (Oleszak, 2024). Few-shot learning may be used to format code correctly by providing code syntax examples.

It's also possible to combine the methods above. OpenAI allows users to create Custom GPTs where specific instructions can be provided along with an area to upload files containing domain-specific knowledge (OpenAI, 2023). This combination of RAG and prompt engineering may enable a user to reduce the number of prompts required to complete complex tasks the model was not specifically trained on.

Methodology

While the following methodology can be utilized with a variety of LLMs, the following ChatGPT variants were utilized for the case studies in the next section: ChatGPT-3.5, ChatGPT-4, and a custom GPT: Senior Systems Engineer – Systems Modeler (SSE-SM).

Model Creation Process

Models are created using the process in Figure 3:

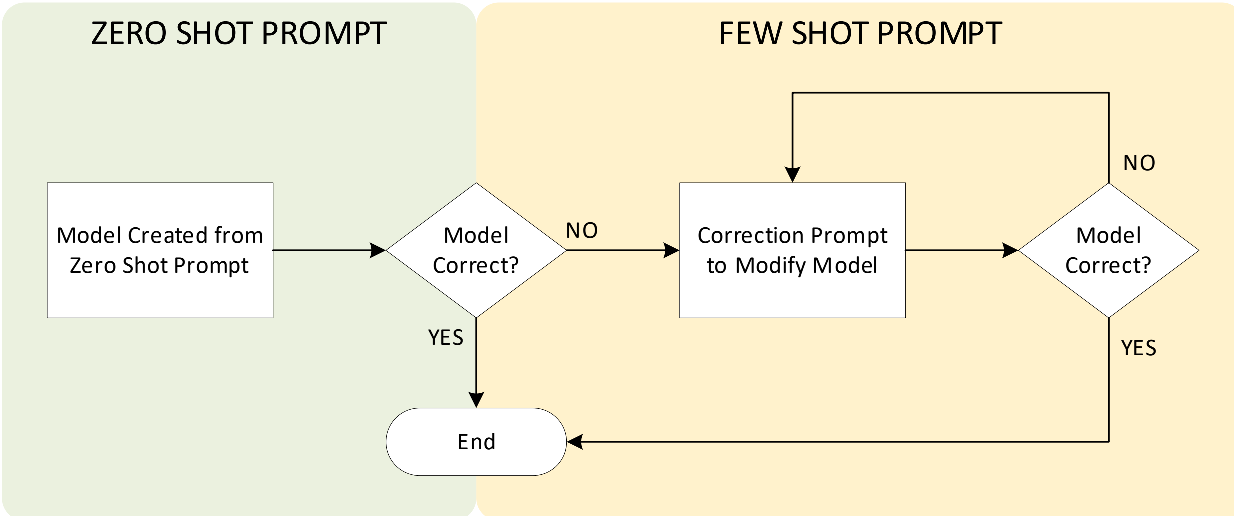

Figure 3. Zero or Few-Shot Prompting Process

The first step in the process is a prompt instructing the LLM to create a SysML v2 textual model from a list of requirements or system description. This is the zero-shot case. After the LLM returns a model, the model is analyzed for correctness. The process is completed if the model is correct. If the model is not correct, it is modified by a corrective prompt, and analyzed again for correctness. The process repeats until a correct model is output by the LLM.

The SysML v2 specification was utilized to check generated models for accuracy. Graphical models were built using the PlantUML plugin in Eclipse (https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/install/eclipse).

Senior Systems Engineer - Systems Modeler (SSE-SM)

SSE-SM is a custom GPT based on ChatGPT-4 with the following documents from the SysML v2 Github repository added to incorporate RAG:

- SysML v2.0 Specification
- Introduction to the SysML v2 Language: Textual Notation
- Introduction to the SysML v2 Language: Graphical Notation

A SysML v2 textual template and examples of requirements and parts packages were also uploaded to SSE-SM. The following instructions were uploaded to the instructions part of the custom GPT user interface.

> *As an expert level systems engineer, this GPT excels at querying, modifying, and developing systems models in the SysML version 2 modeling language. This GPT will respond with SysML v2 text based models when prompted.*
>
> *This GPT recognizes key differences between SysML v1 and v2. For example, in SysML v2, there are no blocks. Blocks are a SysML v1 concept. In SysML v2, components are represented as parts.*

The documentation, template, examples, and custom instructions uploaded to SSE-SM were developed using the prompts in the ChatGPT-3.5 and ChatGPT-4 as guidance.

Case Study: Bicycle System Model

The goal of this case study is to determine the feasibility of using an LLM to build, modify, and query a SysML v2 model from natural language prompts only. While a much more complex model could have been developed, a structural model focused on a requirement, parts, and interconnections of the parts of a bicycle offers a useful example while also offering the opportunity to add complexity in future research.

Creating the SysML v2 Model

The following prompt was utilized to build the SysML v2 textual model:

> *Create a SysML v2 textual model of a bicycle consisting of the following parts: a frame, handlebars connected to the frame, a seat post connected to a frame, a front axle connected to the frame, a rear axle connected to the frame, a front wheel connected to the front axle, a rear wheel connected to the rear axle, front brakes connected to the front wheel and frame, rear brakes connected to the rear wheel and frame, and a drivetrain connected to the frame and rear wheel.*

The key parts of this prompt are:

- *Create*: The term *create* signals the LLM that a specific item needs to be created.
- *SysML v2*: Prompts that stated SysML, but not a version, generated a textual model that confused SysML v1.x and v2 concepts (blocks and parts). When using ChatGPT-4 if a SysML v2 was not specified, ChatGPT-4 called Dall-E to create a visual diagram.
- *Textual*: Prompts that did not state *textual* often gave responses on how to create a model, but no code.
- *Connected*: The term connected signals the LLM that a specific relationship needs to be created
- The parts list is given as a list of interconnections. Splitting the prompt into a parts list and interconnection list did not improve results. For brevity, the parts list and their interconnections were combined.

ChatGPT-3.5

The following are samples from the zero-shot prompt. The model did identify all parts, but the syntax was incorrect (e.g., missing semi-colons at the end of each line). The model also identified parts and typed parts (e.g., part handlebars: Handlebars) but did not define all parts prior to using them.

After a few more prompts, some of the model syntax (semi-colons) was corrected and parts were defined correctly. However, the LLM was not able to connect parts correctly. While there may be a set of prompts to fully develop the model correctly, after multiple attempts to correct the model using several different prompt structures and instructions, we decided to try different LLMs.

ChatGPT-3.5 zero prompt model sample:

```
package BicycleSystem {

    part bicycle: Bicycle {
        frame,
        handlebars,
        seatPost,
        frontAxle,
        rearAxle,
        frontWheel,
        rearWheel,
        frontBrakes,
        rearBrakes,
        drivetrain
    }
```

```
    part frame: Frame

    part handlebars: Handlebars {
        connectedTo(frame)
    }

    part seatPost: SeatPost {
        connectedTo(frame)
    }
// END OF SAMPLE CODE
}
```

Figure 4. ChatGPT-3.5 Zero Prompt Model Graphical Notation

ChatGPT-4

The model identified all the parts from the prompt, properly defined the parts, but did not make the connections correctly. However, when given the following two prompts were given as examples, ChatGPT-4 was able to properly make the connections.

Prompt 1:

Connections should be formatted like the following two examples:

connect handlebars to bicycleFrame;

connect seatPost to bicycleFrame;

Prompt 2:

There should be no and statements in connections. For example, 'connect frontBrakes to frontWheel and bicycleFrame;' should be:

connect frontBrakes to frontWheel;

connect frontBrakes to bicycleFrame;

We confirmed that valid SysML v2 models could be built by ChatGPT-4 using a few-shot prompting method. The following text is sample code from the zero-shot model. The graphical version of the corrected model is shown in Figure 5.

ChatGPT-4 zero prompt model sample:

```
package BicycleSystem {
// SAMPLE CODE BEGINS
    part def Brakes {
    }

    part def Drivetrain {
    }

    part def Bicycle {
        part frame: Frame;
        part handlebars: Handlebars;
        part seatPost: SeatPost;
        part frontAxle: FrontAxle;
        part rearAxle: RearAxle;
        part frontWheel: Wheel;
```

```
        part rearWheel: Wheel;
        part frontBrakes: Brakes;
        part rearBrakes: Brakes;
        part drivetrain: Drivetrain;

        // Connections
        connect frame -> handlebars;
        connect frame -> seatPost;
        connect frame -> frontAxle;
        connect frame -> rearAxle;
        connect frontAxle -> frontWheel;
        connect rearAxle -> rearWheel;
        connect frame -> frontBrakes -> frontWheel;
        connect frame -> rearBrakes -> rearWheel;
        connect frame -> drivetrain -> rearWheel;
    }
}
```

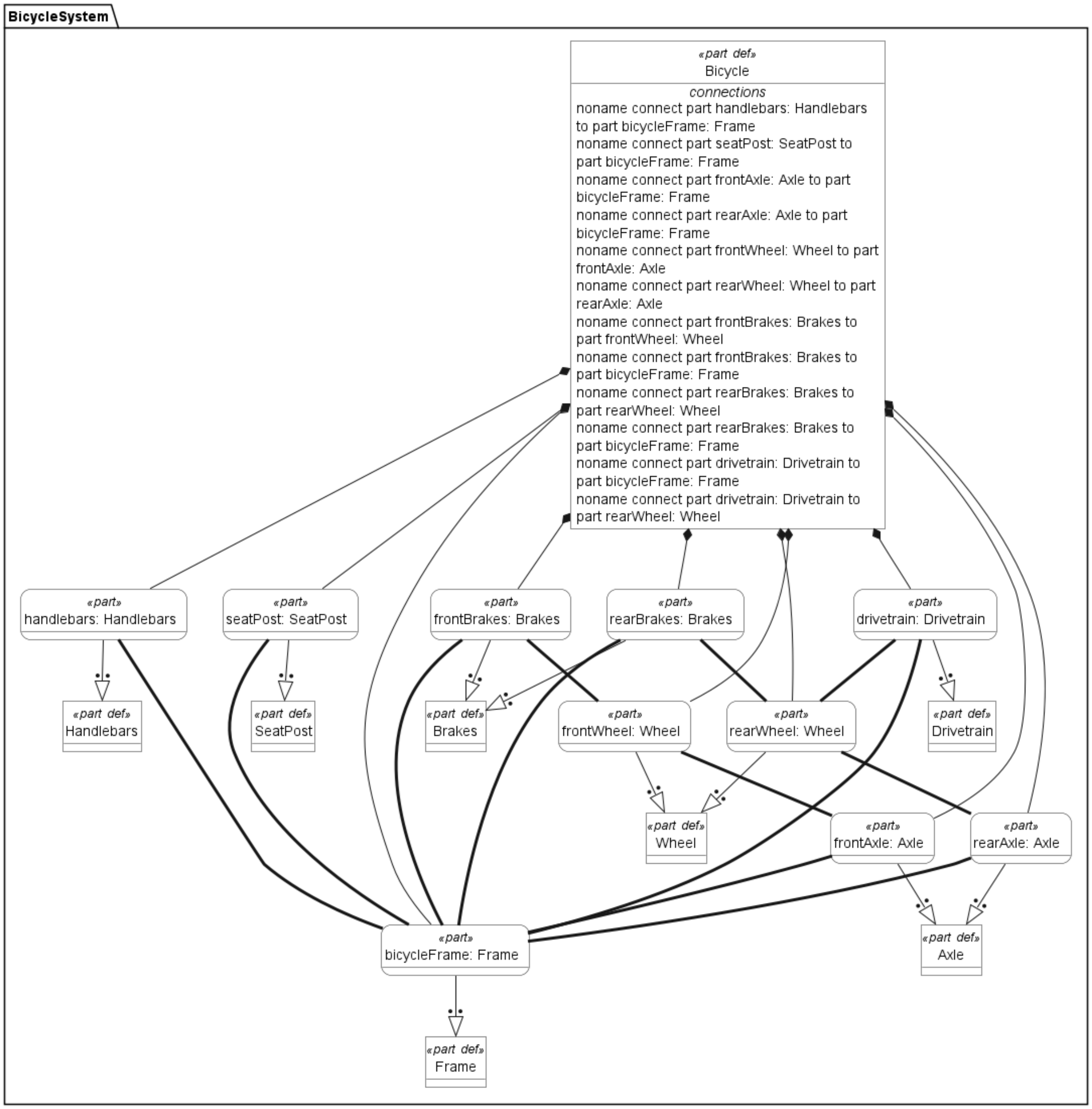

Figure 5. ChatGPT-4 Model after Examples Provided (Few-Shot Prompt)

SSE-SM

SSE-SM was developed using the few-shot prompt example instructions from the ChatGPT-4 prompts and providing additional knowledge via RAG (see under heading “Senior Systems Engineer - Systems Modeler (SSE-SM)” for full details). By building a correct model, we confirmed that SSE-SM can build a fully valid model with one prompt. While the initial prompt was enough to create a valid model, due to the inclusion of additional knowledge, this is not a zero-shot prompt.

The graphical version of the corrected model is shown in Figure 5.

SSE-SM zero prompt model:

```
package Bicycle {
    part def BicycleFrame {
        doc /* The main structure of the bicycle to which other parts are connected.
*/
    }

    part def Handlebars {
        doc /* The steering mechanism of the bicycle, connected to the BicycleFrame.
*/
    }

    part def SeatPost {
        doc /* Supports the seat and connects to the BicycleFrame. */
    }

    part def Axle {
        doc /* Supports the wheels and allows them to rotate. */
    }

    part def Wheel {
        doc /* Allows the bicycle to roll and includes a tire and a rim. */
    }

    part def Brake {
        doc /* The mechanism used to slow down or stop the bicycle. */
    }

    part def Drivetrain {
        doc /* Transfers power from the cyclist to the rear wheel. */
    }

    part Bicycle {
        part bicycleFrame : BicycleFrame;
        part handlebars : Handlebars;
        part seatPost : SeatPost;
        part frontAxle : Axle;
        part rearAxle : Axle;
        part frontWheel : Wheel;
        part rearWheel : Wheel;
        part frontBrakes : Brake;
        part rearBrakes : Brake;
        part drivetrain : Drivetrain;

        // Connections
        connect handlebars to bicycleFrame;
        connect seatPost to bicycleFrame;
        connect frontAxle to bicycleFrame;
        connect rearAxle to bicycleFrame;
        connect frontWheel to frontAxle;
        connect rearWheel to rearAxle;
        connect frontBrakes to frontWheel;
        connect frontBrakes to bicycleFrame;
        connect rearBrakes to rearWheel;
        connect rearBrakes to bicycleFrame;
        connect drivetrain to bicycleFrame;
        connect drivetrain to rearWheel;
    }
}
```

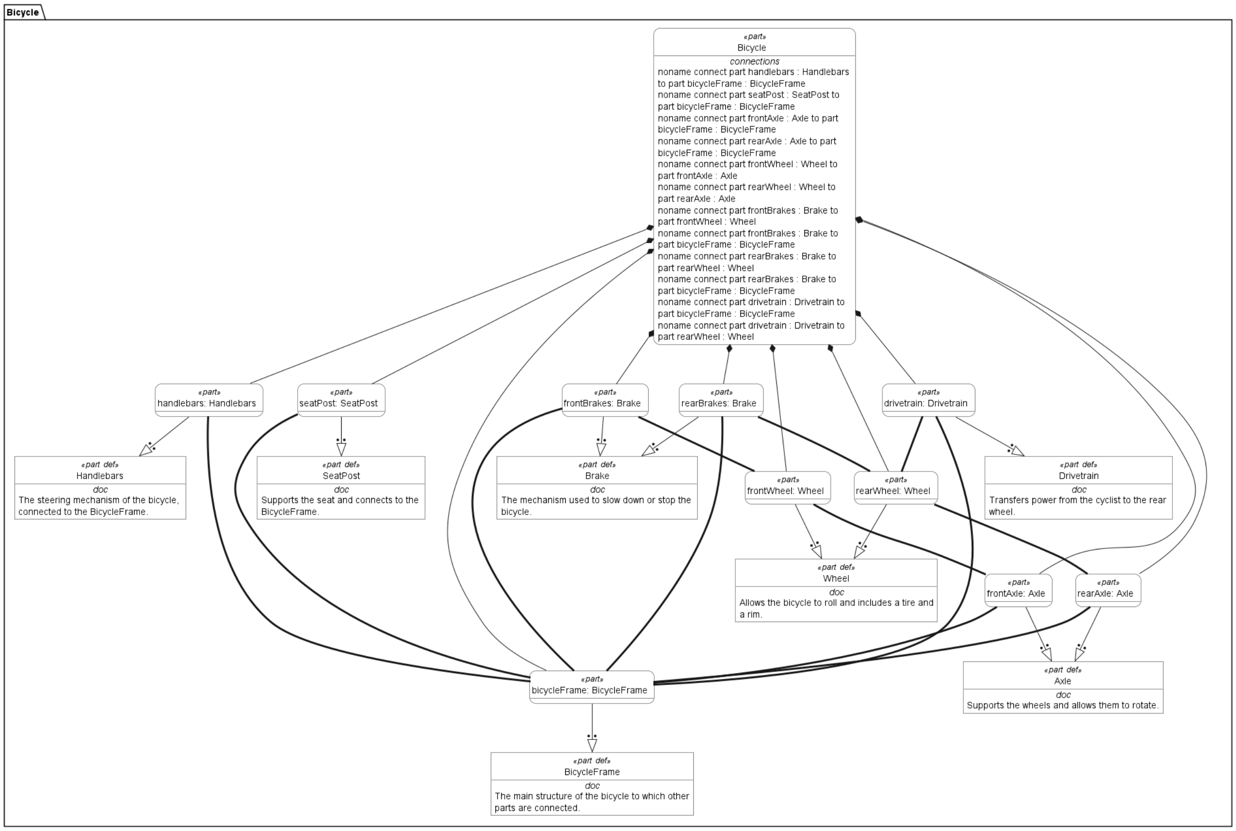

Figure 6. SSE-SM Model First Prompt

Modifying the SysML v2 Model

The SysML v2 model was modified to add a left and right pedal to the drivetrain; and add a user maximum weight requirement of 300 pounds using the following prompts. The [SysML v2 Model] in the prompts is the bicycle model created by SSE-SM.

Prompt 1:

Modify the following SysML v2 model to add a left pedal to the drivetrain and a right pedal to the drivetrain: [SysML v2 Model]

Prompt 2:

Modify the model to include the following requirement: "The bike shall support a user weighing 300 pounds or less."

ChatGPT-3.5

When adding the two pedals to the model, ChatGPT-3.5 defined a left and right pedal, added two parts called leftPedal and rightPedal to the model, and connected the pedals to the drivetrain as shown in the snippet below. While technically correct, there are more efficient ways to effect this modification.

```
package Bicycle {

    // Code Snippet Removed for Brevity

    part def LeftPedal {
        doc /* The left pedal of the bicycle. */
    }
```

```
    part def RightPedal {
        doc /* The right pedal of the bicycle. */
    }

    part Bicycle {

        // Code Snippet Removed for Brevity

        part leftPedal : LeftPedal;
        part rightPedal : RightPedal;

        // Code Snippet Removed for Brevity

        connect leftPedal to drivetrain;  // Connect left pedal to drivetrain
        connect rightPedal to drivetrain; // Connect right pedal to drivetrain
    }
}
```

When adding the requirement to the model, ChatGPT-3.5 modeled it as a constraint and not a requirement.

```
    part def BicycleFrame {
        doc /* The main structure of the bicycle to which other parts are connected.
*/
        constraint maxUserWeight: Real[1] { max <= 300 } // Constraint for maximum
user weight
    }
```

ChatGPT-4

When adding the two pedals to the model, ChatGPT-4 defined a pedal, added two parts called leftPedal and rightPedal to the model, and connected the pedals to the drivetrain as shown in the snippet below.

```
package Bicycle {

    // Code Snippet Removed for Brevity

    part def Pedal {
        doc /* A part of the drivetrain, mounted on the crank to allow the cyclist to
power the bicycle. */
    }

    part def Drivetrain {
        doc /* Transfers power from the cyclist to the rear wheel. */
        part leftPedal : Pedal;
        part rightPedal : Pedal;
    }

    part Bicycle {

        // Code Snippet Removed for Brevity

        // Additional connections for pedals
        connect drivetrain.leftPedal to drivetrain;
        connect drivetrain.rightPedal to drivetrain;
    }
}
```

When adding the requirement to the model, ChatGPT-4 modeled it as a requirement.

```
    requirement def WeightCapacity {
        doc /* The bike shall support a user weighing 300 pounds or less. */
    }
```

SSE-SM

When adding the two pedals to the model, SSE-SM defined a pedal, added two parts called leftPedal and rightPedal to the model, and connected the pedals to the drivetrain as shown in the snippet below.

```
package Bicycle {

    // Code Snippet Removed for Brevity

    part def Drivetrain {
        doc /* Transfers power from the cyclist to the rear wheel. */
        part leftPedal : LeftPedal;
        part rightPedal : RightPedal;
    }

    part def LeftPedal {
        doc /* The pedal on the left side of the drivetrain. */
    }

    part def RightPedal {
        doc /* The pedal on the right side of the drivetrain. */
    }

    part Bicycle {

        // Code Snippet Removed for Brevity

        // Adding pedal connections
        connect drivetrain.leftPedal to drivetrain;
        connect drivetrain.rightPedal to drivetrain;
    }
}
```

When adding the requirement, SSE-SM added a requirement definition and also numbered the requirement.

```
    requirement def <'1'> UserWeightSupport {
        doc /* The bike shall support a user weighing 300 pounds or less. */
    }
```

Model Query

The bicycle SysML v2 model was queried to determine if the LLMs could recall information from the model and infer information from the model that was not explicitly defined in the model.

For simple recall, the LLMs were asked how many brakes the bicycle contained. All LLMs correctly identified two brakes, front and rear.

The LLMs were asked “Will the bicycle be able to stop if the rear brakes fail?” All LLMs correctly identified the bicycle would be able to stop using the front brakes.

The LLMs were asked “Will the bicycle be able to stop if the front brakes fail?” All LLMs correctly identified the bicycle would be able to stop using the rear brakes.

The LLMs were asked “Will the bicycle be able to stop if both brakes fail?” All LLMs identified this would be a failure of the braking system and gave alternate options for stopping the bicycle. All LLMs also identified that the braking system was a critical system for the bike and recommended regular inspection and maintenance.

Results, Discussion, and Limitations

In each case, the LLMs were able to identify all parts from the prompt describing the bicycle system. However, when it came to creating a SysML v2 textual model from the prompt, the models performed differently (Table 1).Using the few-shot learning prompts from the ChatGPT-4 model as a guide and adding SysML v2 knowledge, the SSE-SM was able to correctly model the bicycle on the first attempt.

The LLMs were all able to add pedals to the model although they utilized different methods to do so. ChatGPT-3.5 added the requirement as a constraint while the other two LLMs correctly added the requirement as a requirement.

Table 1. LLM Comparison

Criteria	Correct Response?		
	ChatGPT-3.5	ChatGPT-4	SSE-SM
Model Creation			
The LLM identified all parts	Yes	Yes	Yes
The LLM defined all parts	No	Yes	Yes
The LLM identified all connections	No	Yes	Yes
The LLM modeled all connections	No	Yes – required few-shot prompts	Yes
The LLM developed model was able to be graphically displayed	No	Yes – required few-shot prompts	Yes
Model Modification			
The LLM modified the model by adding pedals	Yes	Yes	Yes
The LLM modified the model by adding the weight requirement	No	Yes	Yes
Model Query			
The LLM identified the number of braking systems	Yes	Yes	Yes
The LLM reasoned the bicycle would be able to stop if only the front brakes failed	Yes	Yes	Yes
The LLM reasoned the bicycle would be able to stop if only the rear brakes failed	Yes	Yes	Yes
The LLM reasoned the bicycle would not be able to stop using the brakes if both brakes failed	Yes	Yes	Yes

In addition to the widely acknowledged limitations of LLMs (e.g. incorrect answers based on "hallucinations," replicability of results), the case study in this paper exhibited several limitations:

- A simple model not representative of a complex system was used for demonstration of capability. It is not known if the LLMs tested will perform as well or better when behavioral and/or more complex structural models are required. To overcome this limitation, future research could focus on building models that are more inclusive of the SysML v2 textual notation.
- SysML v2 is not widely adopted. This led to a limited amount of information available for RAG. Future research could focus on developing more SysML v2 example models for LLM training, finetuning, and RAG.

Future Work

There are many opportunities to build upon the research and case study presented in this paper:

- **Systems Modeling Benchmark:** Develop a benchmark to quantify LLM's capabilities to perform system modeling functions. SysEngBench (Bell et al., in press) is a recently introduced systems engineering benchmark for LLMs and is a good candidate for further contributions in the areas of system modeling.
- **SysML v2 Model Library:** A corpus of SysML v2 model examples is likely required to increase the capability of LLMs to perform systems modeling functions via RAG or finetuning. These new models should be more inclusive of the SysML v2 language and provide different, but correct, methods to model the same system.
- **Systems Modeling Domain-Specific LM:** Where LLMs have a broad understanding of language, a domain-specific model narrows its scope in the pursuit of deep expertise in a certain area. Domain-specific LMs could be LLMs, but Small Language Models (SLMs) may also be a feasible option. SLMs require much less memory and compute to infer responses and may be run on a variety of devices. Locally ran LMs are also desirable for work with sensitive data.
- **Emergent Property Discovery:** Explore the ability of LLMs to discover emergent properties in Systems of Systems via a System of Systems model.
- **Model Conversion:** Explore the ability of LLMs to convert SysML v1.x models to SysML v2. While mass conversion of models will likely require specialized tools, LLMs may be able to assist in building these tools.
- **AI Assistance Cost Factors:** As AI disrupts software and systems development, cost models used by the DoD will need to be updated. Madachy et al. (in press) introduced six new cost factors that may apply to a variety of cost estimation techniques. Capturing time savings from using general and domain-specific LLMs in systems modeling can help inform new AI related-cost factors.

Conclusion

LLMs have shown an ability to generate SysML v2 models with increasing capability as an LLM becomes aware of domain-specific knowledge. As SysML v2 becomes widely adopted, systems engineering domain-specific LLMs are a promising method to reduce the knowledge gap and training required to build, modify, and query systems models using plain language prompts. However, more research is required to ensure the accuracy and reliability of LLMs applied to systems modeling is acceptable. As we continue to explore the use of LLMs in systems modeling, a variety of applications will undoubtedly shape the future of AI and systems engineering.

References

Amazon Web Services, Inc. (2024). *What are large language models? - LLM AI explained - AWS*. https://aws.amazon.com/what-is/large-language-model/

Arya, N. (2024, January 3). *Run an LLM locally with LM Studio*. KDnuggets. https://www.kdnuggets.com/run-an-llm-locally-with-lm-studio

Bell, R., Madachy, R., & Longshore, R. (in press). *Introducing SysEngBench: A novel benchmark for assessing large language models in systems engineering*.

Carroll, E., & Malins, R. (2016). *Systematic literature review: How is model-based systems engineering justified?* (SAND--2016-2607). https://doi.org/10.2172/1561164

Friedenthal, S. (2023). *Introduction to the SysML v2 Language Graphical Notation*. https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/Intro%20to%20the%20SysML%20v2%20Language-Graphical%20Notation.pdf

Friedenthal, S., Griego, R., & Sampson, M. (2009). *INCOSE model based systems engineering (MBSE) initiative*.

Friedenthal, S., & Seidewitz, E. (2020). A preview of the next generation systems modeling language (SysML v2). *Systems Engineering NewsJournal*.

Huertas, J. F. (2023, March 7). Size isn't everything—How LLaMA democratizes access to large-language-models. *Shaped Blog*. https://www.shaped.ai/blog/size-isnt-everything-how-llama-democratizes-access-to-large-language-models

INCOSE. (2007). *INCOSE systems engineering vision 2020*. https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf

Madachy, R., Bell, R., & Longshore, R. (in press). *Systems acquisition cost modeling initiative for AI assistance*.

Nucci, A. (2024, January 20). RAG vs fine-tuning LLM: Comparing the Gen AI approaches. *Aisera: Best Generative AI Platform For Enterprise*. https://aisera.com/blog/llm-fine-tuning-vs-rag/

Oleszak, M. (2024, March 22). *Zero-shot and few-shot learning with LLMs*. Neptune.Ai. https://neptune.ai/blog/zero-shot-and-few-shot-learning-with-llms

OMG. (n.d.). *What is SysML? | OMG SysML*. Retrieved March 29, 2024, from https://www.omgsysml.org/what-is-sysml.htm

OMG. (2017). *OMG Systems Modeling Language version 1.5*. http://www.omg.org/spec/SysML/1.5

OMG. (2023). *OMG Systems Modeling Language (SysML) Version 2.0 Beta 1*. https://www.omg.org/spec/SysML/2.0/Language/

OpenAI. (2023, November 6). *Introducing GPTs*. https://openai.com/blog/introducing-gpts

Stachecki, F. (2024, January 9). *What is new in SysML 2.0?* eduMax. https://www.edumax.pro/blog/what-is-new-in-sysml-20

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Weizenbaum, J. (1966). ELIZA—A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. https://doi.org/10.1145/365153.365168

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., & Zhang, Y. (2024). A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 100211. https://doi.org/10.1016/j.hcc.2024.100211

Acquisition Research Program
Department of Defense Management
Naval Postgraduate School
555 Dyer Road, Ingersoll Hall
Monterey, CA 93943

www.acquisitionresearch.net