Abstract

Large Language Models (LLMs) are a subset of Generative Artificial Intelligence (AI) that has dramatically increased its capabilities over the past two years. Many LLMs can now consume multimodal media, demonstrating further potential for advancements in digital transformation. Systems Engineering is undergoing this digital transformation as many in the field are exploring ways to simplify and automate tedious drudgery. This paper investigates the effectiveness of both fine-tuned and unmodified Generative Pre-trained Transformer (GPT) models in translating Systems Modeling Language Version 2 (SysMLv2) illustrations into textual system descriptions. This research aims to bridge the knowledge gap between junior and senior systems engineers by providing a SysMLv2 Translator that expertly describes SysMLv2 model images for senior systems engineers to verify. Currently, a model translator does not exist for SysMLv2 models, creating a research objective to ascertain if Large Language Models (LLMs) can determine if a system modeled by SysMLv2 meets system requirements based on expert knowledge. To explore the LLM’s ability, three GPT models will be utilized: a control GPT (Chat GPT Version 4-o), the SysMLv2 Model Interpreter (a GPT fine-tuned using .txt and .sysml files), and the SysMLv2 Translator (a GPT fine-tuned using .json files). The models' outputs are assessed using the BERTScore and MAUVE Framework metrics to automatically evaluate their performance with a two-sample t-test analysis to compare performance. GPTs that used both fine-tuning and prompt received results indicating that fine-tuned GPTs, particularly the SysMLv2 Translator, performed significantly better than the unmodified GPT, indicating that fine-tuning and prompt engineering improves output quality. Our discoveries should highlight the immense potential of utilizing large language models to expedite and reimagine the digital transformation of knowledge and processes for the next generation of engineers.

Introduction

This research aims to ensure that newly created SysMLv2 models are aligned with system requirements according to a senior engineer’s knowledge. The verification process is crucial for maintaining system integrity, allowing expert engineers to retire and new engineers to take their place. Essentially, by creating a verification model with human assistance, systems can be rapidly digitalized to reduce the manpower and effort required to maintain the systems, simultaneously transferring the knowledge of vital system processes and requirements to the junior engineers. Furthermore, digitalizing systems into models yields many benefits, including enhanced visualization and understanding of systems, team collaboration, and a basis for future innovation. In addition to quality assurance, a verification model promotes knowledge transfer and increases communication between colleagues. With a textual interpretation of the system, senior engineers are enabled to verify the accuracy of the modeled system and understand the model errors without understanding SysMLv2. The verification model also sparks a dialogue that will increase communication about the modeled systems. As a direct byproduct, the improved communication will open the door for efficient collaboration, garnering a higher understanding of the systems from a simple image upload of the system’s model representation.

Furthermore, creating a verification model advances the digital transformation of Systems Engineering. From the analog methods of Document-Based Systems Engineering (DBSE) to creating advancements for the age of Model-Based Systems Engineering (MBSE), systems engineers are now embracing modern approaches to handle complex, contemporary issues. Despite the tangibility and systematic approach DBSE offers, many engineers are looking to digitalize the field due to the knowledge gap, fiscal challenges, and tedious nature of DBSE. The birth of MBSE serves to digitalize knowledge, artifacts, and documentation into data, code, and models. MBSE promotes flexibility, adaptability, and reusability through its capability to be updated rapidly, altered as needed, and copied as a basis for future systems to be modeled after. MBSE also eliminates the need for document, blueprint, and artifact management, removing the need to store physical system documentation and allowing data, code, and models to be stored intangibly and copied to make backups.

Despite the complexity and expertise required to create digital models, a verification model for SysMLv2 visualizations can significantly reduce the manpower needed to maintain and develop them. Fiscally, businesses and government agencies utilizing the verification model can save valuable time and resources by reducing the labor required to accurately model a system. Reducing the rigor of the workload will create a healthier, more enjoyable work environment whose efficiency would reflect the improvements made to the verification process of modeling systems. Furthermore, the verification model allows the team to focus elsewhere, particularly on exploring new opportunities and projects, minimizing the need for manual verification. By reducing waste, labor, and time, systems engineers will see enhanced innovation, increased productivity, improved quality, and expedited project completion.

Background

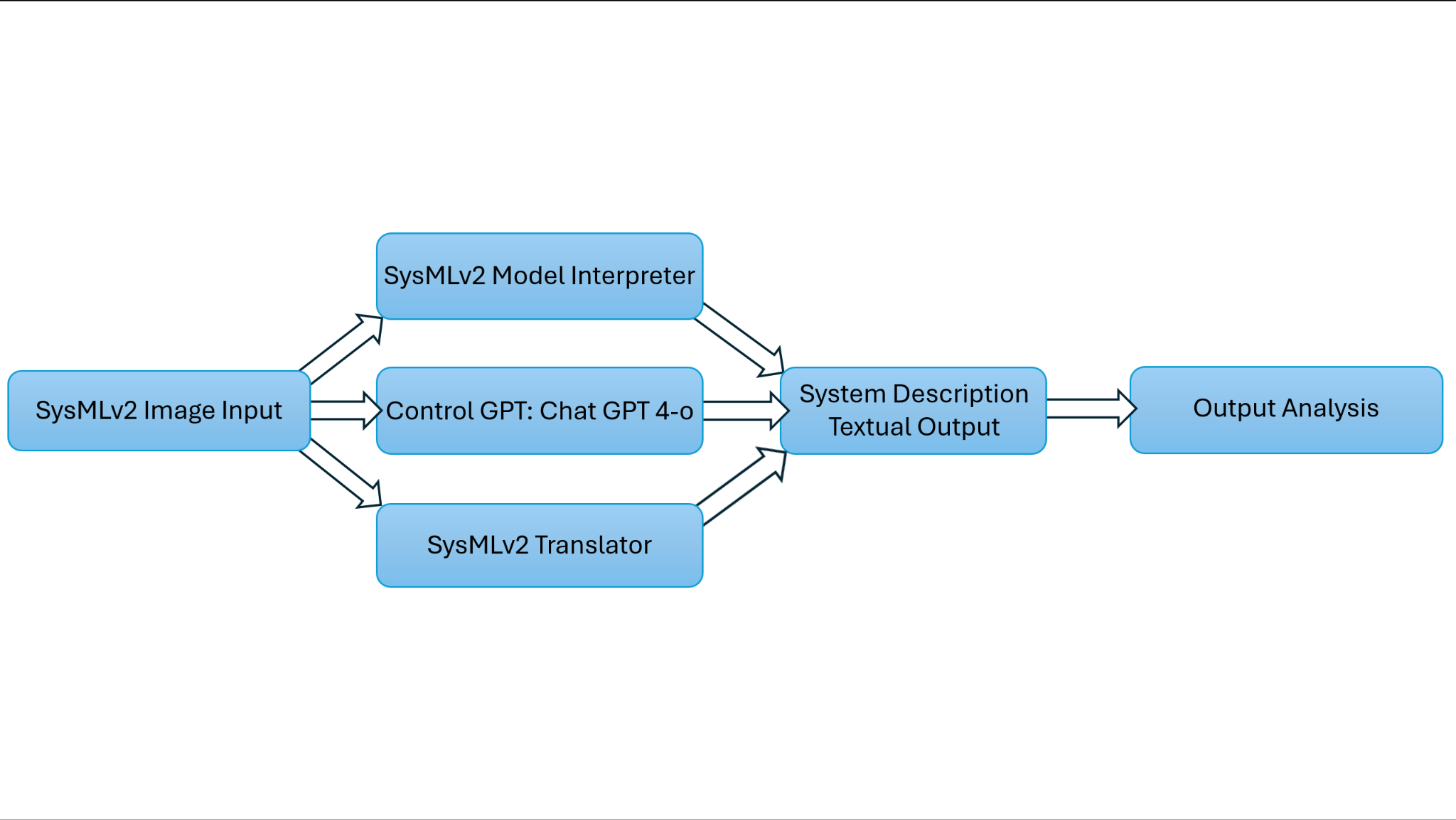
The digital transformation of systems engineering (SE) has introduced the integration of generative AI into SE. To this end, we hope to continue the foundational work laid out by Husain, Wach, and Topcu in their paper, “Can Large Language Models Accelerate Digital Transformation by Generating Expert-Like Systems Engineering Artifacts? Insights from an Empirical Exploration”. The basis of their research provides a vital proof of concept in the integration of generative AI in SE. They demonstrated that LLMs were capable of producing expert-quality SE artifacts. Significant improvements were noticed using prompt-engineering techniques. These improvements were noticeable across various LLMs (GPT 4, GPT-3.5 Turbo, and Claude) and showcased the potential of LLMs to expedite SE tasks.

Husain, Wach, and Topcu focused on generating SE artifacts using LLMs. After generating SE artifacts from various prompts, the LLM's performance was assessed using the MAUVE framework, an automatic evaluation tool for neural and human text. Their research demonstrated the significance of using prompt engineering regarding the prompts' structure, context, and specificity to obtain higher-quality outputs to generate SE artifacts. The emphasis of these findings led to our motivation to extend their work by fine-tuning GPT models for Model-Based Systems Engineering (MBSE) purposes.

Our research focused on implementing LLMs to assist with MBSE tasks. Specifically, we investigated the effectiveness of both fine-tuned and unmodified GPT models in translating SysMLv2 model images into accurate textual descriptions. This approach aims to bridge the knowledge gap between junior and senior systems engineers and ensure that SysMLv2 models meet system requirements based on expert knowledge.

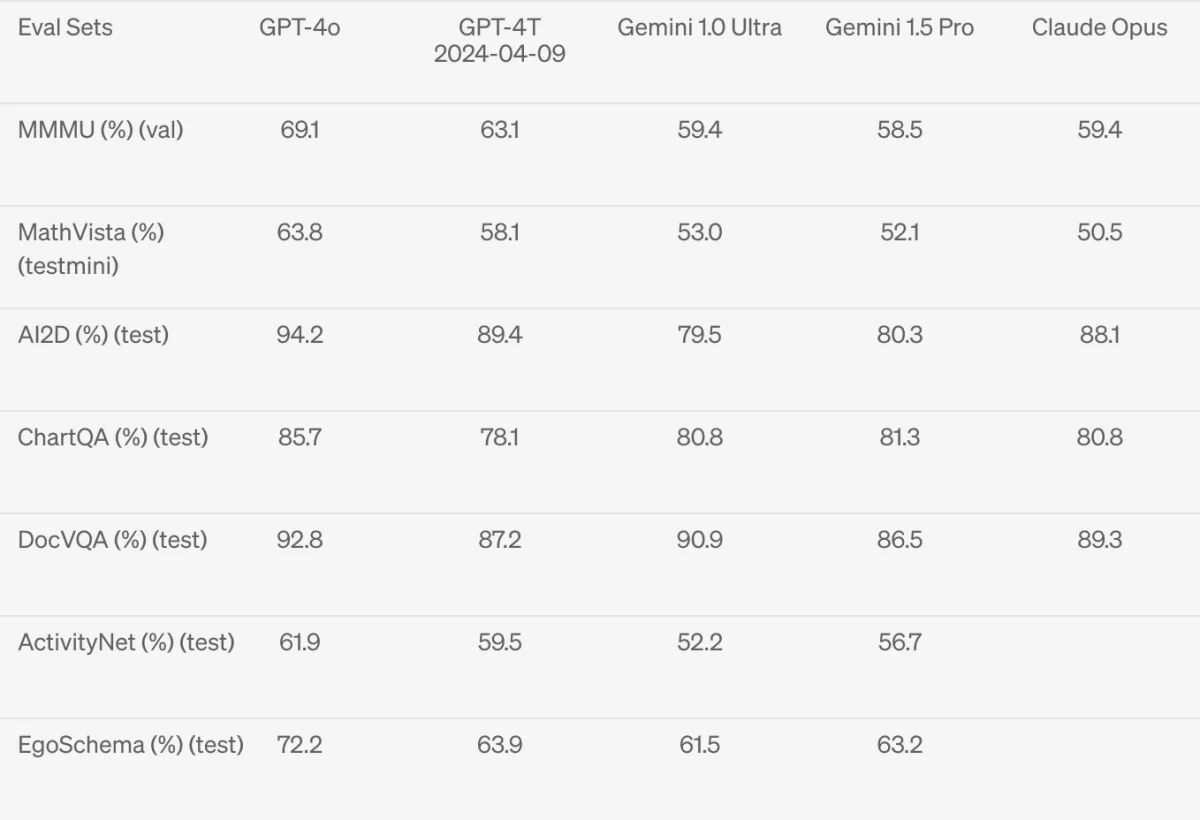
Our research findings align with those of Husain, Wach, and Topcu, particularly in highlighting the importance of prompt engineering and demonstrating the potential of LLMs assisting in SE and MBSE tasks. Our work continues on the path of previous research by exploring the application of LLMs in a specific domain of MBSE, SysMLv2 model verification. This application is necessary to bridge the knowledge gap between senior and junior engineers separated by digital barriers. Junior engineers are more inclined to understand and use SysMLv2 to model systems, while senior engineers have gained expert knowledge primarily through analog experience. To accomplish this objective, an investigation into the groundwork of implementing a full-scale SysMLv2 verification model must be conducted. Because SysMLv2 is not widely used, especially outside of SE, there exists a lack of relevant data. While various methods and approaches were considered, Chat GPT’s front-end fine-tuning functionality was chosen due to its simplicity and the sample size of available data. Back-end fine-tuning was determined to be unnecessary and was reserved to be completed following a proof of concept.

Methods

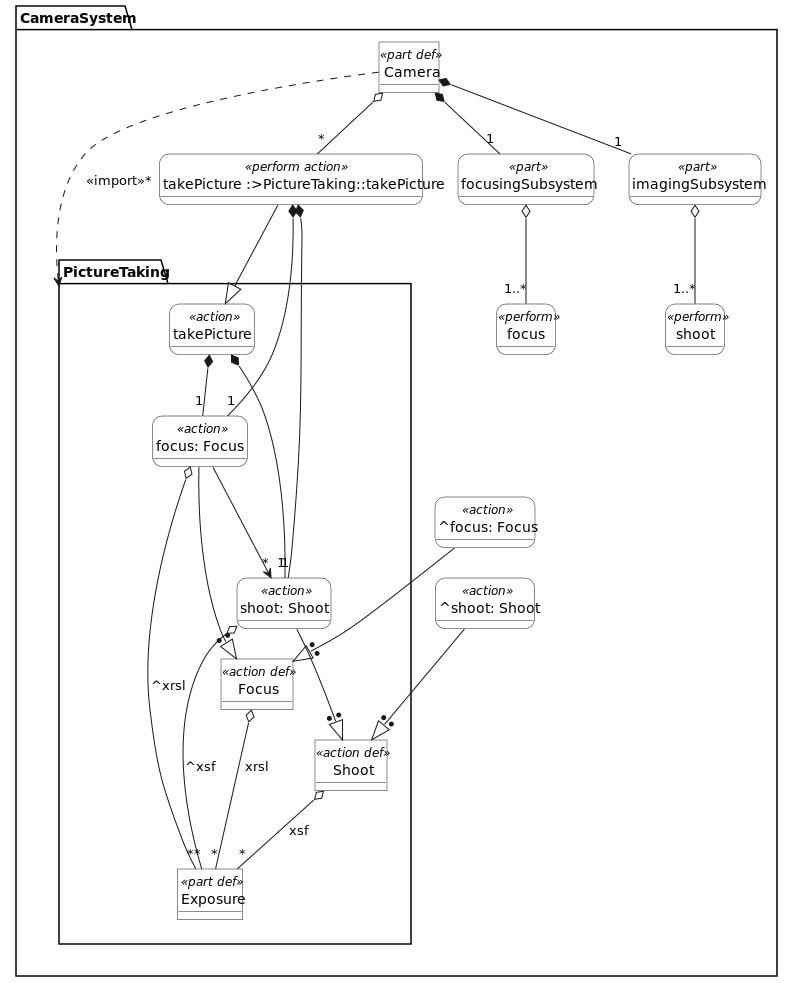
  
**Figure 1: Architecture and Process Flow of Final Product**

To begin determining whether LLMs can be used for model verification, it is necessary to employ a structured and systematic approach. First, an LLM of choice must be identified. Three LLMs were considered: Chat GPT, Claude, and Google Gemini. While each LLM has its limitations and advantages, Chat GPT was chosen for its simplicity, collaboration capability, and quality of response. Further research into utilizing LLMs as system verification models should explore the advancements of other LLMs like Claude, Google Gemini, and Mistral AI. Next, the models were gathered. Twenty-five SysMLv2 model visualizations of varying complexity were collected primarily from the SysML-v2-Release GitHub repository, while others were found in various SysMLv2 textbooks [3], [4], [5]. Model descriptions were developed through a blend of research and supervised input from a language model to establish the baseline descriptions for model verification. The language model was provided the full context of the problem space to create descriptions and was primarily tasked with refining the information. From these descriptions, models can be fine-tuned to produce outputs better suited for system verification.

Before fine-tuning, it is necessary to determine whether it elicits a significant advantage over using a regular multimodal GPT. While both GPT-4 and GPT 4-o contain the functionality for analyzing and interpreting images, GPT 4-o offers increased image interpretation performance.

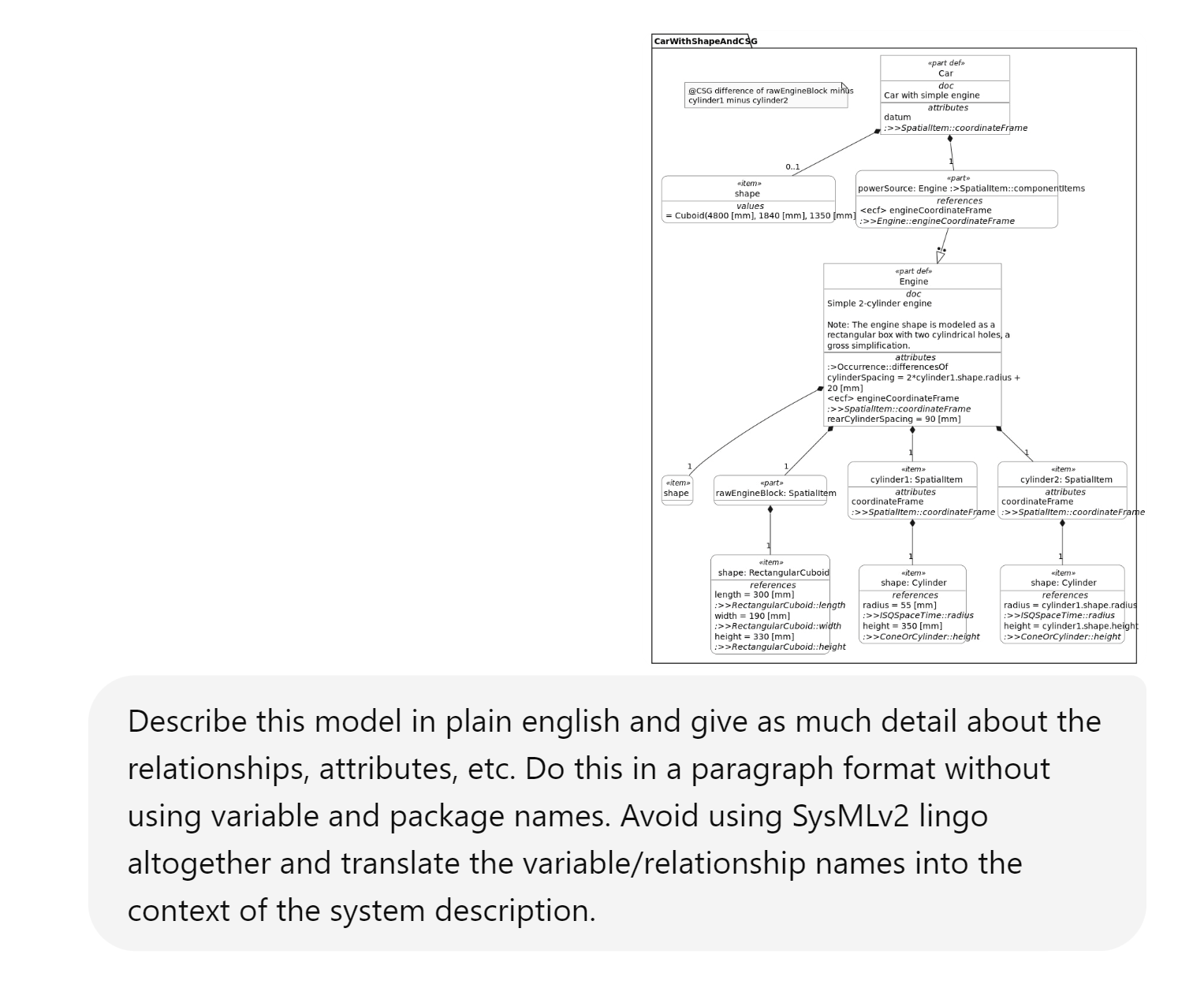
**Figure 2: Vision Understanding Comparison of Various LLM Models**

Therefore, by setting the baseline performance with GPT 4-o, it is possible to determine whether or not a fine-tuned GPT can outperform an adept multimodal model. To establish a baseline, GPT 4-o was prompted with .sysml model visualizations akin to the one used in Figure 2.



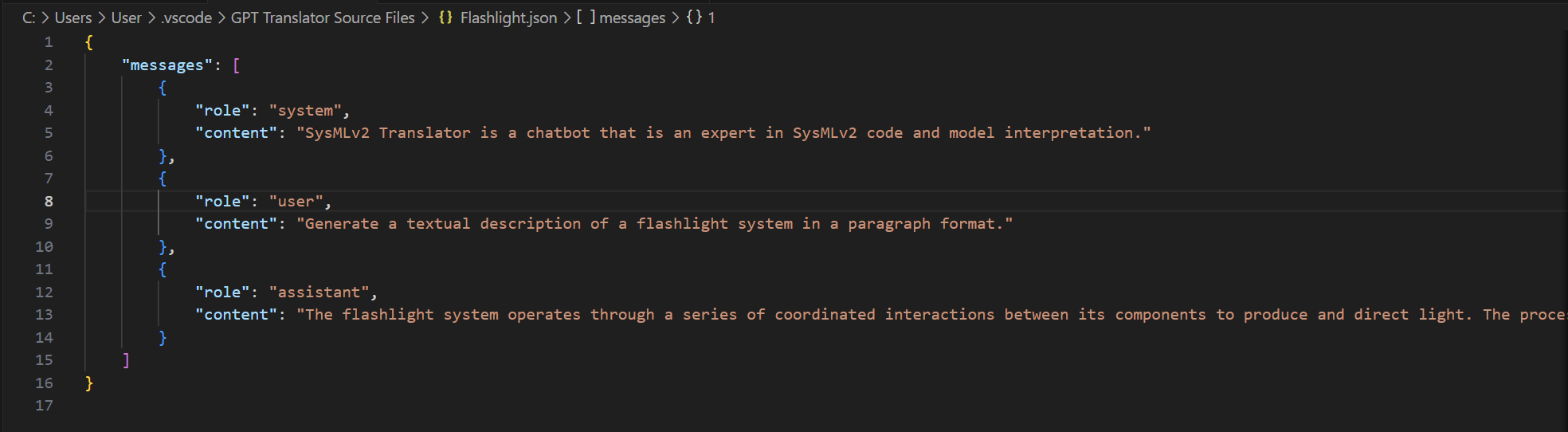
**Figure 3: SysMLv2 diagram of a camera system**

Working with GPT-4o, some habits needed to be ironed out. Responses were often not in a paragraph format but rather bulleted or numbered. Furthermore, GPT-4o commonly used variable and package names like “PictureTaking” or “focusingSubsystem” from Figure 2, which often muddied the responses. To correct this, all prompts to the Control GPT, GPT-4o, were given additional instructions using prompt-engineering techniques to produce an adequate response (see Figure 3). Finally, after prompting the twenty-five selected models, the first GPTs’ work was completed, and all outputs were saved into text files.

**Figure 4: Prompting Chat GPT 4-o to meet specifications.**

The recent models released by Open AI contain advanced functionality for creating a fine-tuned GPT with capabilities that include web browsing, image generation, code interpretation, and data analysis. The fine-tuning functionality also provides an additional “Instructions” input that the GPT follows for every prompt to avoid redundant format and specification requests.

Additionally, front-end functionality allows up to twenty unique input files that can contribute to diversifying and enhancing the LLM's already advanced knowledge. Utilizing these features, the first fine-tuned GPT for system verification was created, called the “SysMLv2 Model Interpreter.” Twenty files containing information on ten different models were added to the SysMLv2 Model Interpreter’s knowledge base to fine-tune the model. Each model included a .txt file detailing the system requirements and structure and a .sysml file to textually and graphically represent various mechanisms like flashlights, cameras, or cars. The SysMLv2 Model Interpreter was given instructions to use at every model iteration, and at this point, the GPT was complete and ready to prompt. Similarly to the Control GPT, GPT-4o, twenty-five models were used to determine the limitations of LLMs serving the role of a verification model.

After finishing the SysMLv2 Model Interpreter, it was deemed necessary to investigate the benefits of using a more efficient file type. Using a JSON file, the new GPT could have twenty models in its knowledge instead of ten. Rather than providing mere references to code and descriptions, the JSON file assigned the GPT roles and instructions for responding to specific user requests. The new GPT was named the “SysMLv2 Translator.” Figure 4 depicts the role, prompt, and desired response when a SysMLv2 model image is detected.  


**Figure 5: Example of a JSON file used to fine-tune the SysMLv2 Translator**

The SysMLv2 Translator’s instructions differ to clarify the intentions of the JSON files, but on the front end, they are quite similar. Similarly to the previous two GPTs, the SysMLv2 Translator was prompted to interpret the same twenty-five models, and the responses were stored in text files for scoring.

After recording the responses for twenty-five different models, an evaluation metric was needed to determine the LLM’s success in interpreting the models. Two automatic evaluation metrics were employed: the BERTScore and the MAUVE Framework. The BERTScore computes a similarity score using a tokenized approach. By comparing each token in the GPT response to every token in the actual model description, the BERTScore calculates the precision, recall, and harmonic mean of precision and recall scores to calculate the F1 measure. The F1 measure is a standardized result (0-1.0) used to determine the accuracy of two texts to one another [2]. The F1 score will be a significant metric referenced throughout this paper as the BERTScore. The BERTScore was implemented in a Jupyter Notebook Python environment through a simple package installation. Comparatively, the BERTScore score involves more steps than the MAUVE Framework but has the potential to yield better performance. The setup in Figure 5 details the needed libraries, modules, configuration, and logging requirements to prepare the BERTScore for full functionality.



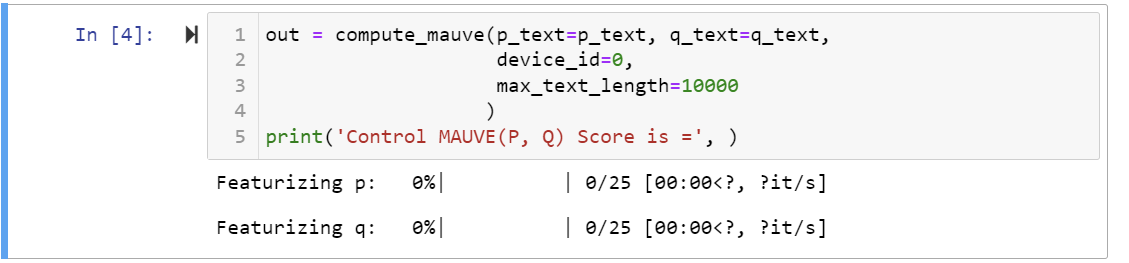
**Figure 6:** **BERTScore Control GPT Responses vs Actual Descriptions Initial Setup**

The setup for the three GPTs is the same, aside from reconstructing the file paths and adjusting the variable names. All twenty-five GPT responses and model definitions were recorded into two separate files, the first for the GPT responses and the second for the actual model definitions. Both files are separated by two newline characters between each response. The code block used to construct the file paths also contains a method for reading the files one model at a time and assigning them to an index in a list. After creating two lists of length 25, the precision, recall, and F1 scores can finally be calculated. The scores for each model are then recorded into a single tensor with twenty-five scores from 0-1.0.

Conversely, the MAUVE score is much simpler to set up. After installing the package, the first requirement is to sort the responses into two arrays/lists. Instead of doing a file import for the responses and descriptions, the lists were hard-coded to analyze subgroups of choice. The subgroups analyzed were how unmodified and fine-tuned GPTs performed with simpler models versus more complex models. The first array consists of the twenty-five system/model descriptions, while the second array consists of the GPTs’ responses to each model. The MAUVE demonstration calls the first array refs (references) and the second array cands (candidates). The “refs” array contains the model descriptions for this study, and the “cands” array contains all GPT-produced textual descriptions. Three comparisons will be necessary since there are three GPTs, resulting in three “cands” arrays to one “refs” array. The MAUVE score is calculated by the function MAUVE(P, Q), where P represents the “refs” array, and Q represents the “cands” array. “The score is standardized like the BERTScore such that MAUVE(P, Q) lies in the range (0, 1.0]. Furthermore, MAUVE(P, Q) = 1 if and only if Q = P” [1]. In the context of this research, a perfect score is only attainable if the texts are identical. After the models have been added to the appropriate array, all left to do is a simple command using the MAUVE library demonstrated in Figures 6 and 7. To import the MAUVE automatic evaluation metric, use the command “from mauve import compute-mauve.” Note variable names are solely a representation of P and Q and are not new values.

|  |  |
| --- | --- |

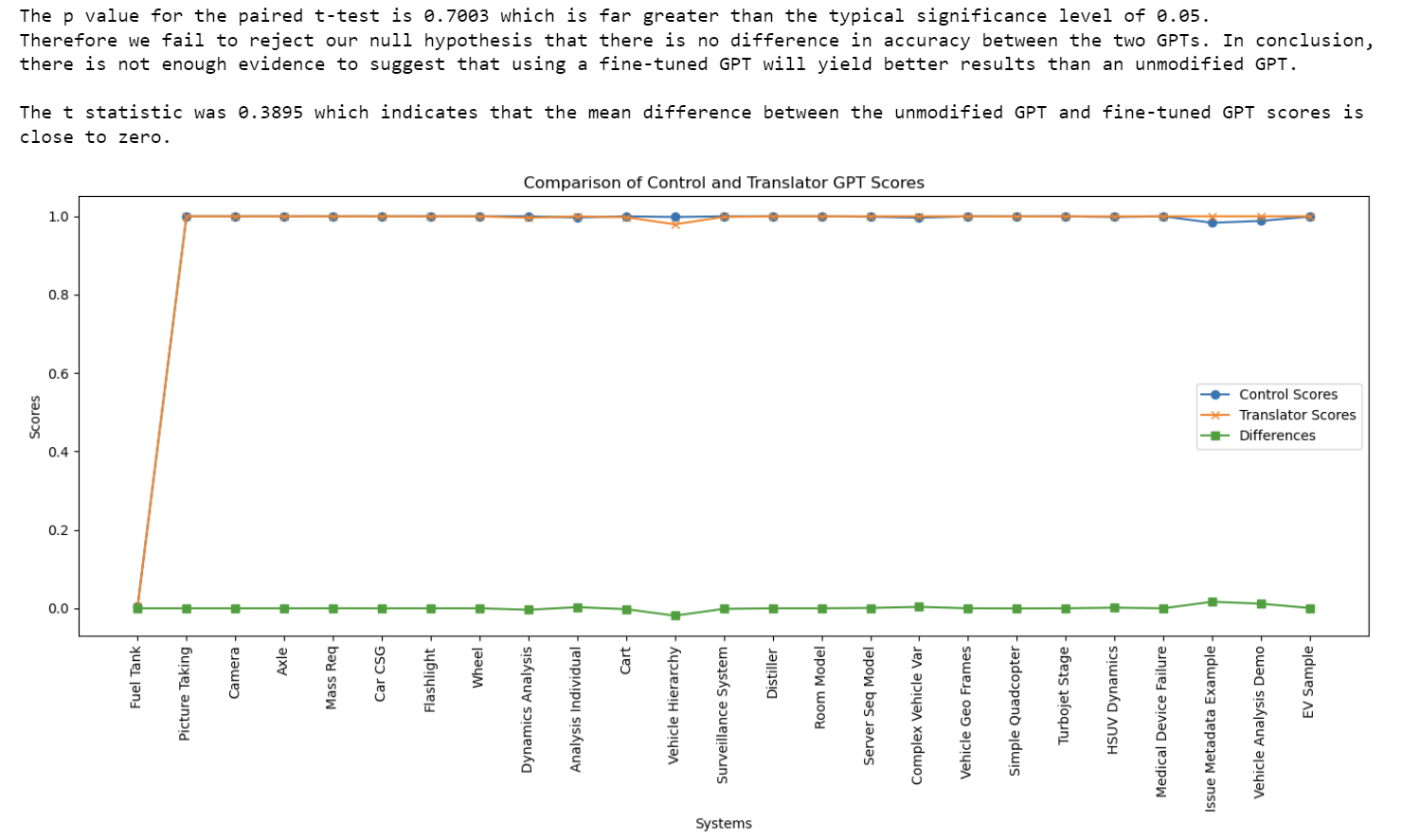
**Figure 7: Model Description Array and Control GPT Description Array**

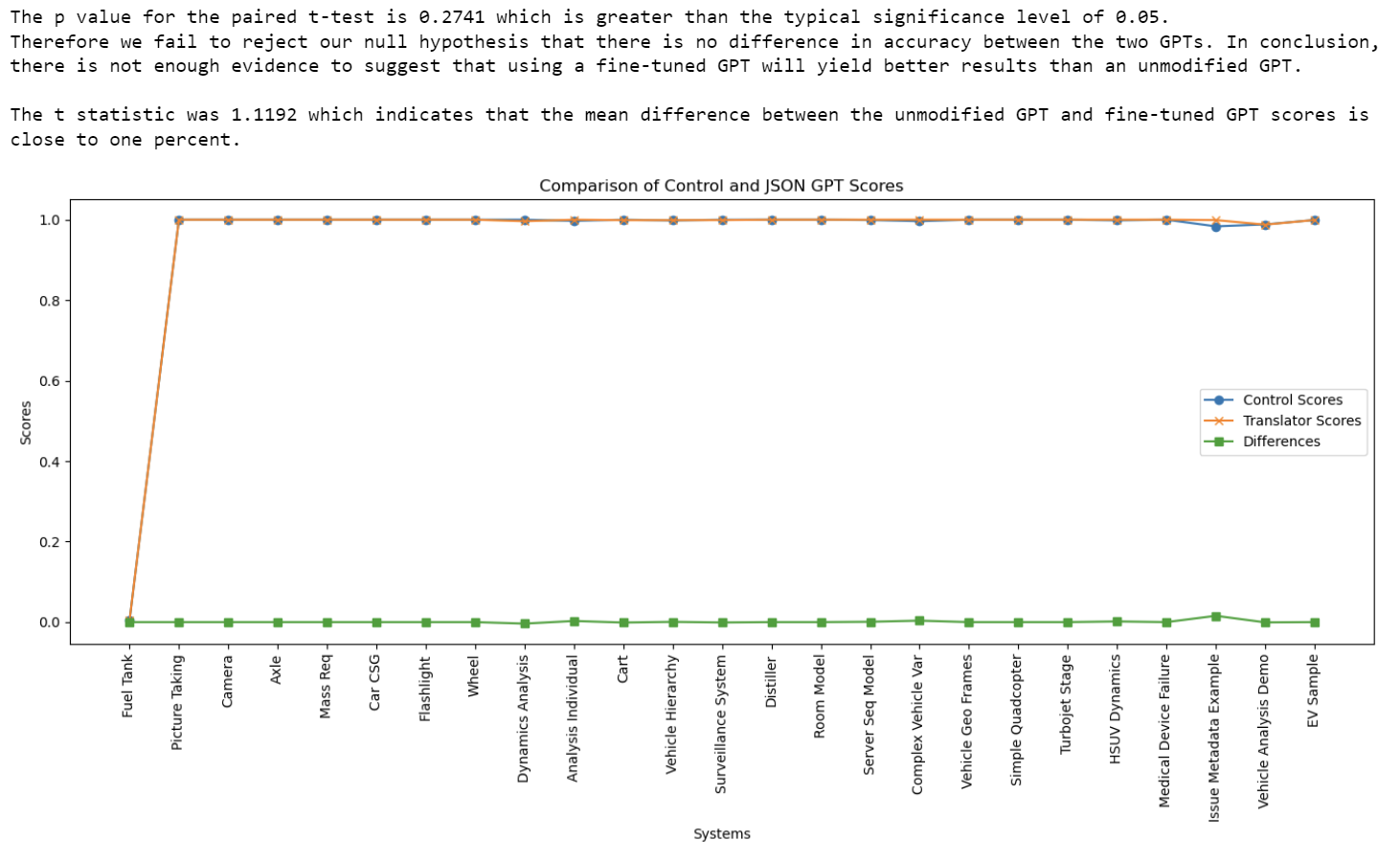


**Figure 8: Calculating a Standardized Score for the Control GPT’s Performance**

Lastly, an additional score was calculated to assess each GPT's ability to translate simple models compared to more complex visualizations. The complex scores were identified based on the image’s pixel size and relationship complexity. The BERTScore was also assessed in the same manner using the same qualifications.

Results

Results for the MAUVE evaluation show that all three models scored above the 99th percentile. The SysMLv2 Model Interpreter received the highest score of 1.0, while the Control GPT and SysMLv2 Translator tied, earning identical scores of 0.9993110441356944 across all 25 models. Each iteration in Figures 8 and 9 represents the MAUVE Score at n model iterations. For example, the Axle model is the fourth model evaluated, and its score represents the MAUVE evaluation for the first four models since the Axle model is at n=4. The graph was created to show the change in scores at every model iteration. The final MAUVE score for the GPT models occurs at n=25, the MAUVE assessment score for the EV Sample.

**Figure 9: MAUVE Score for SysMLv2 Model Interpreter vs GPT-4o at n Models**

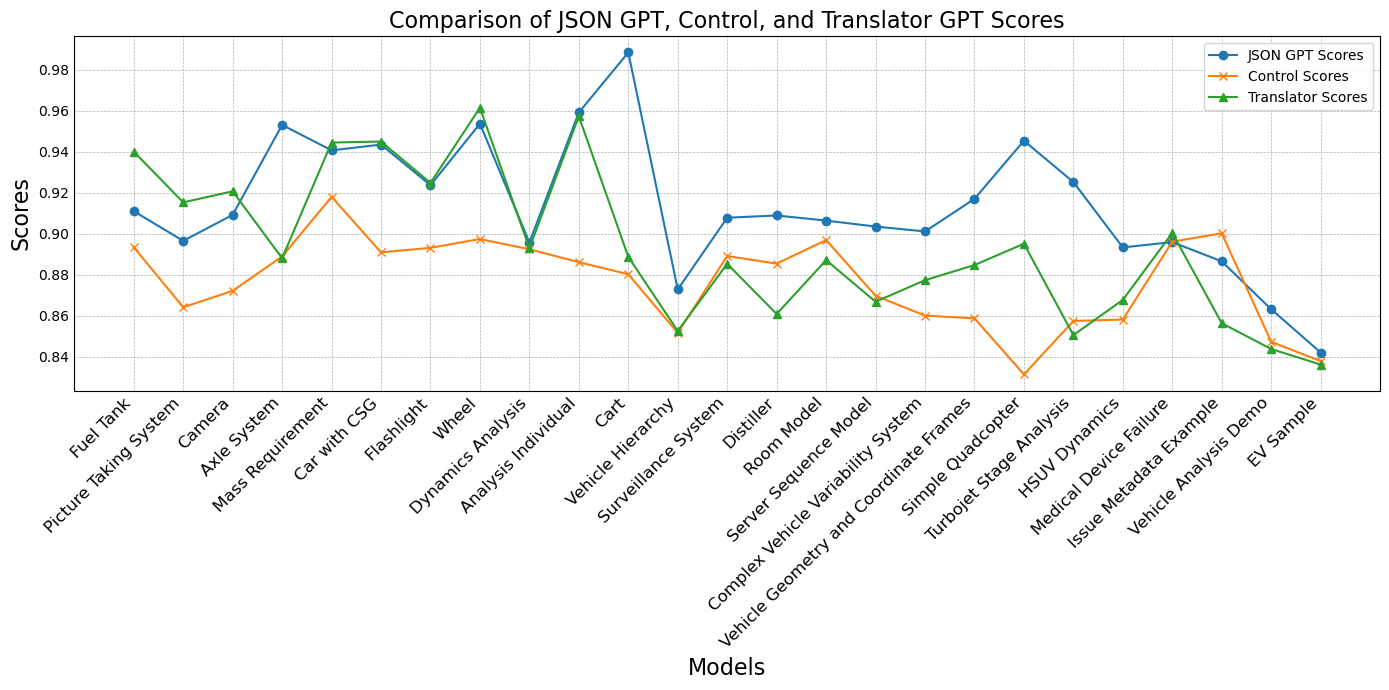
**Figure 10: MAUVE Score for the SysMLv2 Translator vs. GPT-4o at n Models**

Separating the models by their complexity revealed that the SysMLv2 Translator performed the best for complex models, while the SysMLv2 Model Interpreter performed the best for simple models. A model was determined to be complex if the image's pixel size was greater than 1920x1080 and the number of relationships (associations, dependencies, usages, etc.) was greater than 9. The full results of performance by complexity are shown in Table 1.

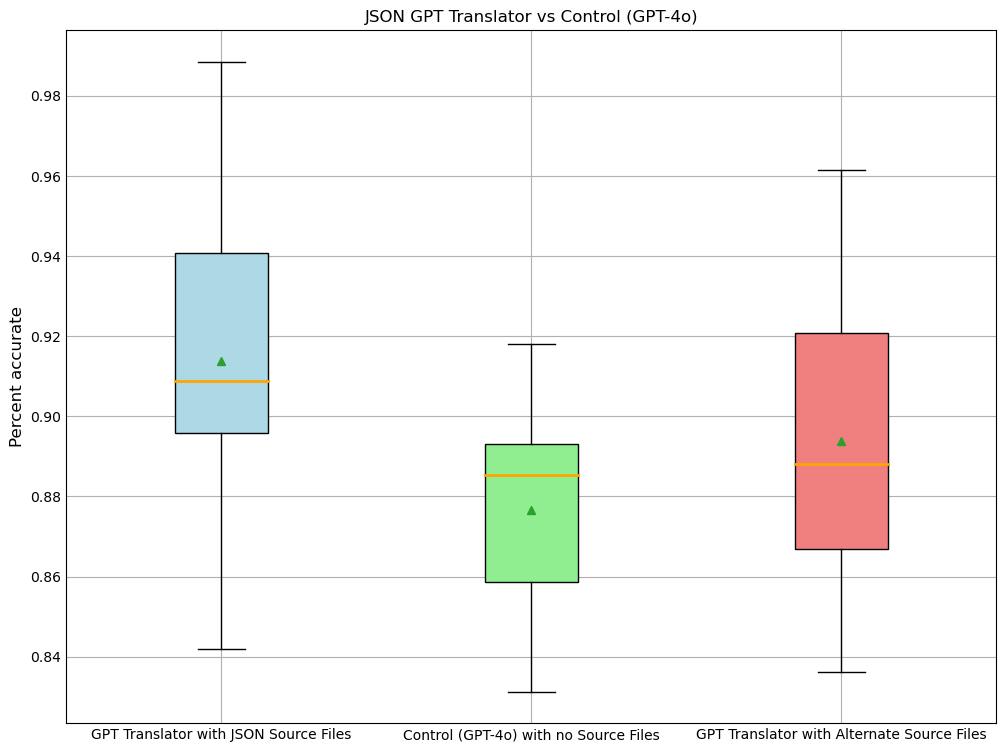
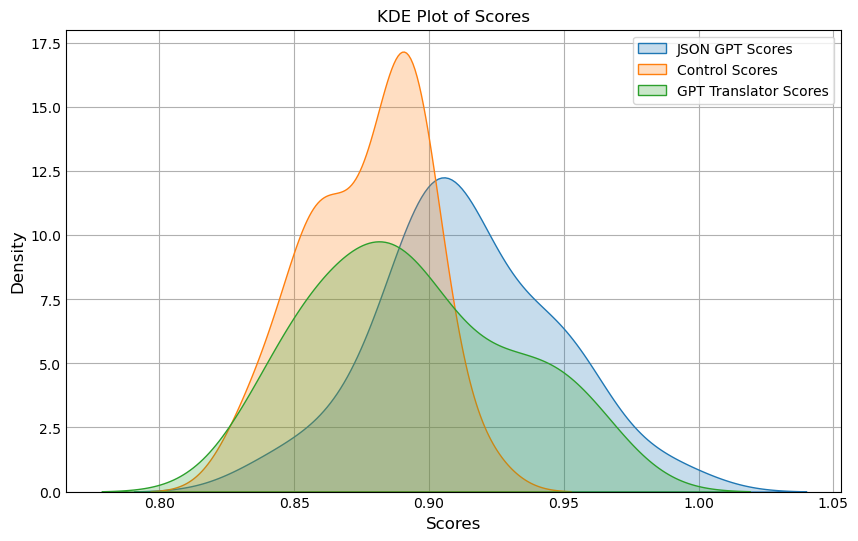
|  | Control GPT (GPT-4o) | SysMLv2 Model Interpreter | SysMLv2 Translator |
| --- | --- | --- | --- |
| Simple Model Score | 1.0 | 1.0000000000000007 | 1.0 |
| Complex Model Score | 0.9851028655309841 | 0.9999999999999994 | 0.9999999999999999 |
| Cumulative Model Score | 0.9993110441356944 | 1.0000000000000004 | 0.9999999999999999 |

**Table 1: MAUVE Scores for Simple vs. Complex Models by GPT**

The BERTScore results revealed lower scores than the MAUVE assessment, with the highest average GPT score of 0.9137. The highest-performing GPT was found to be the SysMLv2 Translator. The SysMLv2 Model Interpreter followed behind second with a 0.8937 average score, and the Control GPT performed the worst with a 0.8766 average score. Figure 8 represents the scores of every GPT at each model iteration.

  
**Figure 11: Line Graph of GPT’s F1 (BERT) Scores at each Model Iteration**

Furthermore, Figure 9 demonstrates a box plot where the orange line within the box represents the median score, and the dark green triangle represents the mean score for each GPT. Additionally, the boxplot helps demonstrate the interquartile range and identify the maximum and minimum scores from each GPT.

  
**Figure 12: Box Plot for GPT Median, Mean, and Quartile Comparison**

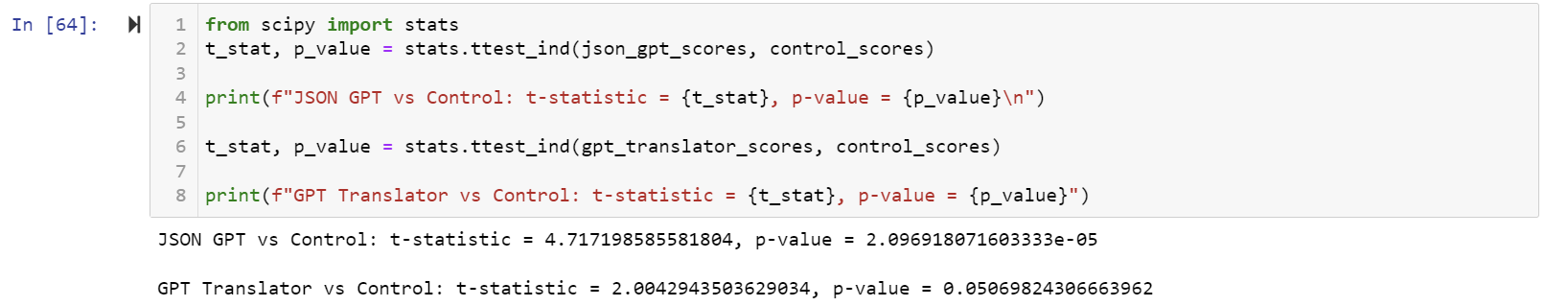
**Figure 13: Kernel Density Plot for GPTs’ F1 (BERT)Score Comparison**

Discussion

Upon initial examination, the automatic evaluation metrics were starkly contrasted. The MAUVE Framework seemed to produce results consistently above the 99th percentile of accuracy, while the BERTScore scored the range of [0.8313, 0.9885] across all the GPTs. A test was conducted to ensure that the scores were not impacted due to the number of instances. From previous research and the MAUVE’s guidelines for usage, there existed a recommendation to obtain at least fifty instances for textual comparison. After splitting the twenty-five GPT responses for each model in half, the sample size increased to fifty, meeting the necessary threshold. Despite this, scores remained more significant than the 99th percentile, suggesting that the MAUVE framework was unsuitable for evaluating LLM output based on semantic comparison. Instead, it focused on determining whether the cands/Q array was human-written. Thus, the LLM-produced output of the Q array was compared based on neural text versus human text rather than comparing semantic accuracy between the texts. This shortcoming led to the conclusion that the MAUVE Framework was not a valid automatic evaluation metric for determining whether an LLM can serve as a verification model.

The BERTScore outperformed the MAUVE Framework in terms of its method of comparison. Since the BERTScore tokenizes and compares the tokens to determine accuracy, it provided results demonstrating the capability for textual comparison. Furthermore, the BERTScore suggests that the SysMLv2 Translator is the most fit GPT of the group to serve as a verification model. However, the general intention of this research is to determine if fine-tuning reaps significantly superior results to an unmodified GPT. This distinction is important to explore because of the fiscal costs and labor that can be avoided if an unmodified GPT is just as capable as a fine-tuned GPT.

Two paired t-tests were conducted to perform a hypothesis test to determine whether the fine-tuned GPTs were significantly better than the unmodified control GPT (GPT-4). The first hypothesis test was between the Control GPT and the SysMLv2 Translator. The null hypothesis for this hypothesis test shows no significant difference between the SysMLv2 Translator and the Control GPT. The second hypothesis test is between the Control GPT and the SysMLv2 Model Interpreter, and the null hypothesis for this suggests no performance difference between the Control GPT and the SysMLv2 Model Interpreter. Figure 11 shows a paired t-test performed in Python for both hypothesis tests.

  
**Figure 14: Paired T-test Code and Outputs**

For the first hypothesis test between the Control GPT and the SysMLv2 Translator trained on JSON source files, the p-value of 0.00002097 was calculated. This p-value is less than the typical significance level of 0.05; therefore, we can reject the null hypothesis and conclude that there is a statistically significant difference in accurately translating SysMLv2 models to textual descriptions between a finetuned GPT using JSON source files and an unmodified GPT. For the second hypothesis test between the Control GPT and the SysMLv2 Model Interpreter, the p-value of 0.05069824 was found. This effectively matches the typical significance level of 0.05; therefore, we cannot conclude whether or not we can reject the null hypothesis, but we can make the statement that the evidence is trending toward being statistically significant.

Conclusion

The evidence presented suggests that a GPT fine-tuned on JSON files has the capacity to serve as a SysMLv2 verification model. Despite achieving statistical significance in performance compared to an unmodified GPT, there is more to be desired regarding accuracy. On average, the SysMLv2 Translator was 91.37% accurate. To be employed as a verification model, it would be favorable if the accuracy of the textual output achieved greater than 97 percent accuracy ±2 percent. Achieving this would require a more extensive knowledge base for the GPT and additional models of varying complexity. At the same time, the results open the possibility for front-end development with back-end fine-tuning. After demonstrating significant improvements, the fidelity of a GPT trained on more models should be explored.

Furthermore, considering alternative LLMs like Claude, Google Gemini, or Mistral presents potential for fiscal and technical improvement. This research has determined that there exists proof that the process of digitally transforming knowledge, information, and processes can be automated. The existence of a verification model to digitalize information presents a unique opportunity for knowledge transfer. Senior engineers can use technical, contemporary means to validate systems analogously. Artifacts, blueprints, and records can be digitalized and stored into data, models, or code for future use. Integrating AI into Model-Based Systems Engineering elevates its potential to the forefront of knowledge transfer for digital transformation. Ultimately, this research underscores the potential for multimodal LLMs to bridge the gap between traditional engineering practices and contemporary digital methodologies, fostering a seamless integration of legacy knowledge into future technological advancements.

References

1. Husain, M., P. Wach, and T.G. Topcu. Can Large Language Models Accelerate Digital Transformation by Generating Expert-Like Systems Engineering Artifacts? Insights from an Empirical Exploration. 2024. Cham: Springer Nature Switzerland.
2. Pillutla, K., et al., Mauve: Measuring the gap between neural text and human text using divergence frontiers. Advances in Neural Information Processing Systems, 2021. 34: p. 4816-4828.
3. Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, Yoav Artzi. "BERTScore: Evaluating Text Generation with BERT." *International Conference on Learning Representations*, 2020.
4. Sanford Friedenthal, E.S., Roger Burkhart, Eran Gery, Hisashi Miyashita, Hans Peter de Koning, Oystein Haugen, Tomas Juknevicius, Charles Krueger, Ivan Gomes, Miyako Wilson, Santiago Leon, William Piers, Tilo Schreiber, Zoltán Ujhelyi, OMG Systems Modeling Language™. June 2023. p. 638.
5. Sanford Friedenthal, A.M., Rick Steiner, A Practical Guide to SysML : The Systems Modeling Language. 2015, Elsevier/Morgan Kaufmann: Waltham, MA
6. Ed Seidewitz, M.B., SysML-v2-Release. 2024: Github