Chair of Spacecraft Systems
TUM School of Engineering and Design
Technical University of Munich

Development and assessment of a computation engine module for an LLM-enhanced spacecraft design assistant

Roberto Aldea Velayos ✉

✉ roberto.aldea-velayosr@tum.de

September 20, 2024

Abstract — This paper presents a novel computation engine designed to enhance the capabilities of Large Language Models (LLMs) to support the initial phases of spacecraft design. With the integration of a Python-based computation engine, we address some of the inherent limitations of LLMs, such as the lack of computational ability and difficulty in tracing decision-making processes back to specific data inputs. The computation engine is supported by OpenAI's API models and the tool call feature to enable accurate, and traceable decision-making within spacecraft system configurations, focusing on missions involving Earth observation using CubeSats in Sun-Synchronous orbits. Through experimental validation across various simulated mission scenarios, the engine demonstrates significant improvements in reducing output uniformity and enhancing stability in design parameter predictions. The system's design allows for future expansions and adaptability to include a broader range of spacecraft types and mission objectives, marking a significant step forward in the application of AI in aerospace engineering.

1 Introduction

In recent years, Artificial Intelligence (AI) has significantly expanded its applications, particularly with Large Language Models (LLMs) that are increasingly being utilized across various tasks. These models are not only popular in conversational AI but have also demonstrated substantial capabilities as engineering tools capable of addressing complex challenges. Their potential as AI assistants in the engineering design process is undeniable.

Despite their advantages, LLMs inherently possess limitations that restrain their utility in prolonged engineering projects. One critical issue is the generation of 'hallucinations' or the production of erroneous or fictitious information. Additionally, these models struggle with computational or mathematical tasks and lack transparent reasoning processes, making it difficult to trace their outputs back to specific training data or internal reasoning. This lack of traceability is problematic in engineering projects where documentation and the ability to backtrack through the decision-making process are crucial.

To address these shortcomings, this paper proposes the integration of a "computation engine." This tool handles the necessary computations, thus enhancing the reliability and traceability of the answers provided by LLMs. By coupling the LLM with a computation engine, the model not only gains the ability to generate reliable numerical data but also enhances the decision-making process by selecting appropriate computational functions at each step.

The focus of this paper is on designing the architecture and an initial implementation of a computation engine, developed only for the initial phases of spacecraft design. More precisely, the CE focuses on the spacecraft's payload, orbit, and ground-segment parameters, which are critical to the design of the rest of the spacecraft platform. These key areas have been the primary focus of the design to ensure that the foundational aspects of the spacecraft are robustly developed. Even with this reduced design scope, the task remains as a complex endeavor that requires handling numerous variables and configurations, which, for this paper, will be simplified under certain assumptions to make the problem simpler. This work, therefore, serves as a foundation that could be expanded to include broader mission types and scenarios.

For this paper, the mission scope is reduced with the following assumptions:

- The mission type is restricted to Earth observation.
- The orbit is assumed to be always Sun-Synchronous and circular.
- The spacecraft is assumed to always be a CubeSat.

1.1 State of the Art

Recent advancements in artificial intelligence have paved the way for sophisticated applications in spacecraft design, particularly utilizing Large Language

Models (LLMs). Existing literature demonstrates significant interest in improving LLMs with computational tools to enhance their utility in engineering tasks. For example, the works described in [10], [7], and [8] introduce frameworks where LLMs are enhanced with capability-augmenting tools, significantly improving their performance in computational tasks. This integration facilitates more complex calculations and aids in mathematical reasoning (as in [1]), essential for technical domains like aerospace engineering.

In the specific realm of engineering and design, [5], [2] and [3] have pioneered in the use of LLMs to assist in early design phases. More precisely, [2] focuses on domain-specific data-sets to train LLMs for tailored tasks in spacecraft design, while [3] explores the broader application of AI, though not specifically LLMs, in the preliminary stages of spacecraft system development. These studies illustrate the potential of LLMs to transform traditional engineering workflows into more efficient and data-driven processes.

This paper extends this innovation by integrating a computational engine with an LLM, specifically addressing the challenges and limitations observed in previous models. Unlike earlier approaches that primarily utilize LLMs for data retrieval or basic computational tasks, this framework uses a Python-based computational engine to perform necessary calculations, thereby enhancing the LLM's capability to handle the complex requirements of spacecraft design. This integration not only improves the accuracy and efficiency of design tasks but also ensures traceability and reliability—key factors in engineering applications where precision is crucial.

1.2 Research gap and questions

Having briefly explored the existing literature on the subject, this paper aims to answer the following questions.

- What are the challenges and limitations of integrating LLMs with computational engines for spacecraft design, and how can these be addressed?
- To what extent can a Python-based framework, utilizing OpenAI's API for tool (or function) calls, improve the efficiency and accuracy of spacecraft design tasks?
- How can the developed computation engine module be assessed in terms of performance, usability, and impact on the design process?

2 Methodology

The Computation Engine (CE) has been developed in Python following an object-oriented structure in accordance to the spacecraft model built and described in [4].

The CE uses a combination of chain-steps, functions and databases of off-the-shelf spacecraft components to populate a generic spacecraft model and its parameters. The order followed by the CE for the design of the spacecraft is calculated in [4], which is payload-orbit-ground segment.

The CE has been developed thanks to OpenAI's API [6] and their models, in this case "gpt-4-turbo-2024-04-09" and "gpt-3.5-turbo-1106" have been used. More precisely, the CE uses an OpenAI's API function called tool-call, where the model can be fed with a list of tools (or computation functions) which then can be accessed to and chosen by the model if seen necessary.

2.1 Framework and Structure of the Computation Engine

The Computation Engine's basic structure flow can be seen in Figure 1. The CE takes a prompt in the form of a Mission Statement, which, along with some internal instructions, is inputted to the model (in this case gpt-4) and 6 parameters are produced as a result. The ground sampling distance (GSD), the maximum revisit time, the region's parameters (that include the radius, the latitude and longitude) and finally the number of Ground Stations. These parameters will then be used in posterior design steps.

The GSD is then used as an input to an algorithm that selects a payload, in this case a camera, from the database of different cameras. This algorithm compares the GSD given by the LLM and the GSD stated in the camera's parameters, and selects the most suitable camera for the altitude range of 400-900km, setting the GSD and the maximum altitude required. The parameters from the payload are then, along the parameters from the region of interest and the maximum region revisit time, which were inferred by the LLM, used in the next step, to calculate the orbital parameters of the mission.

Finally, the calculated orbital parameters, along with the number of ground stations, inferred by the LLM previously, are used to choose the ground stations that will be used in the mission.

Algorithm 1 shows part of the CE's pseudo-code.

Algorithm 1 Process for Extracting Satellite Orbit Parameters and Handling Specific Function Calls

Input: User provides a mission statement.
Prepare the internal prompt using the mission statement.
Internal Prompt:
"Given the following mission statement for Earth observation: {mission_statement}\n
Infer the following parameters necessary for satellite orbit calculation.
If in the mission statement any of the following parameters is stated, then use that value/s:
1. Maximum ground sampling distance (GSD) required in m. (If the generated value is lower than 4m, just say 4)
2. Region of Interest (Longitude, Latitude in degrees and Radius in km, the radius must be chosen so that the area given by the spherical cup of this radius covers the entire region.)
3. Maximum Revisit Time in days
4. Number of ground stations to be used during the mission, heavily depends on if the mission requires data downlink urgency. If not required just say 1
Provide estimates or typical values for Earth observation satellites focusing on this mission.
Respond only with the asked values in the following format without the units: GSD in m, longitude in degrees, latitude in degrees, radius in km, revisit time in days.
Don't write anything that are not the numbers asked for in the asked format. Example response: 4, -75.0, 0.0, 500, 5, 3"
Process Prompt with GPT-4
Store the parameters output in a list.
List functions from computation engine relevant for further modeling.
Save these functions in a list of dictionaries to be used later.
Prompt the model to start with the design in the followed order.
Check if the model's response involves a function call.
if response.choices[0].finish_reason == 'function_call' **then**
    (Example)
    function_call = response.choices[0].message.function_call
    function_name = function_call.name
    if function_name == 'orbit_params_revisit_time' **then**
        Call the corresponding function
        Populate accordingly the space_mission class based on the function results
    end if
end if
Repeat until the model is completely populated. =0

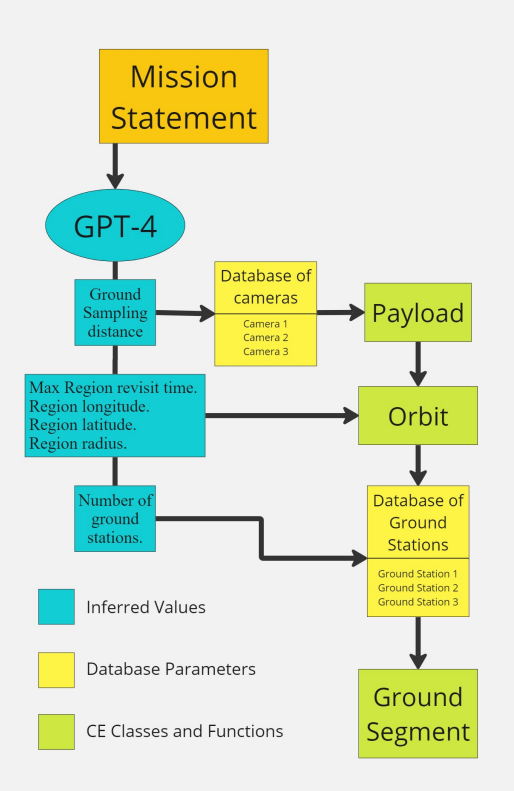

Figure 1 Computation Engine Diagram.

2.2 Analyzed Parameters

To measure the improvement of performance of the LLM assistant with the computation engine, 8 generic Earth observation missions will be prompted in 10 different instances. To compare with the parameters calculated with the CE, the same parameters will be asked to be inferred by 'gpt-4' without any external tool (The prompt used to inferr this parameters can be found in the Appendix).

The parameters chosen to analyze the performance of the CE are:

- Average region contact time in a single orbit in the span of one Week.
- Maximum eclipse duration in a single orbit in the span of one Week.
- Maximum region revisit time in a single orbit in the span of one week.
- Average camera data production in a single orbit in the span of one Week.
- Average ground station contact time in a single orbit in the span of one Week.

These parameters are crucial in spacecraft design as they significantly influence the engineering decisions

regarding key spacecraft components. Each parameter's accurate computation is essential for designing efficient and effective spacecraft systems, thereby explaining their selection for this comparative analysis.

Before detailing how each parameter is calculated, it is crucial to define the initial conditions and assumptions that govern the satellite's behavior in our simulations:

- **Altitude:** Determined during the payload configuration step and is instrumental in defining the satellite's orbital path.
- **Right Ascension of the Ascending Node (RAAN):** Calculated to ensure that in the first orbit, the satellite passes directly above the region of interest.
- **Inclination:** Set to synchronize with the Sun (Sun-Synchronous Orbit) to maintain consistent lighting conditions over the region of interest.
- **Argument of Perigee (AOP) and True Anomaly (TA):** Both parameters are set to zero, simplifying the calculation by focusing the orbit at its perigee at the starting position.
- **Initial Time for Simulation:** Set to January 1, 2025, at 17:15 UTC, marking the start of the observation period.
- **Simulation Period:** Extends over one week to provide a comprehensive view of the satellite's operations across multiple orbits.

Now, we proceed to describe the specific calculations for each operational parameter using the defined initial conditions.

2.2.1 Average Region Contact Time in a Single Orbit

The average region contact time per orbit is determined using the `find_events` function from the Skyfield library[9], which calculates visibility events based on the satellite's configured orbit and the observer's location. By assuming the region is akin to an observer with a certain minimum elevation, this function can be used to calculate this parameter.

2.2.2 Maximum Eclipse Duration in a Single Orbit

The maximum eclipse duration per orbit is calculated by assessing the intervals during which the satellite is in Earth's shadow, using the `is_sunlit` function from Skyfield[9].

2.2.3 Maximum Region Revisit Time in a Single Orbit

The maximum time between successive visits to a specific region is computed using the `find_events` function from Skyfield[9]. Similarly to subsection 1, this function can be used to calculate this parameter.

2.2.4 Average Camera Data Production in a Single Orbit

This calculation is based on the formula that combines coverage time with the satellite camera's operational characteristics:

$$Avg\ Data\ Production = \frac{1}{8} \times DR \times Avg\ Coverage\ Time \tag{1}$$

where the *Data Rate (DR)* is determined by the satellite's velocity and camera specs:

$$DR = \frac{Pixel\ Depth \times Resolution \times Ground\ Track\ Velocity}{GSD} \tag{2}$$

2.2.5 Average Ground Station Contact Time in a Single Orbit

Ground station contact times are calculated using `find_events` from Skyfield[9], simulating the satellite's visibility from a fixed ground location.

2.3 Uniformity and Stability

Two factors will be analyzed in the results. First, the appearance of uniformity in the parameters inferred by the LLM. Uniformity in the LLM means that it answers with the same responses to different inputs, which means that instead of trying to predict or calculate the parameter for each case, it just responds with the same value every time, presumably a value chosen from the training data. In this case it could then be said that if there is too much uniformity in the responses, the model's accuracy for this type of tasks is not adequate.

Second, the stability. In this case, a good stability would mean that if the assistant would give different parameters for two instances of the same prompt, that at least these parameters would be close to each other. Perfect stability would mean that the assistant always gives the same parameter responses for the same missions, no matter how many times instanced.

2.4 Experimentation

Large Language Models (LLMs) can be adjusted to produce responses with varying degrees of randomness, controlled by a parameter known as 'temperature' (denoted as 't'). The temperature setting influences the predictability of the model's outputs: a lower temperature leads to more deterministic and consistent responses, while a higher temperature allows for greater variability and creativity in the generated outputs. To thoroughly evaluate the influence of this parameter on model performance, we examine three different temperature settings: t=0, t=0.5, and t=1, in the cases where only the LLM is used without the computational engine (CE). These settings represent a spectrum from no randomness (t=0), moderate randomness (t=0.5), to high randomness (t=1). When the CE is used, we consistently use a temperature setting of t=0 to ensure maximum reliability and precision in the outputs, aligning with the strict accurateness requirements of spacecraft design.

The mission statements used for the performance benchmark are the following:

- Mission A: Observe and study the deforestation process in the Amazon forest.
- Mission B: Track and alert of wildfires in North America.
- Mission C: Track and study the plastic pollution in the Pacific Ocean.
- Mission D: Observe and study the urbanization process in Europe.
- Mission E: Provide real-time monitoring of volcanic activity in Southeast Asia for early warning systems.
- Mission F: Survey and study coral reef degradation in the Caribbean to support conservation efforts.
- Mission G: Measure and map air pollution in major urban centers around the world to inform health and environmental policies.
- Mission H: Track seasonal agricultural crop health across Africa to aid in food security planning.

Finally, for all the different stated cases, they will be run again but with the regions size (the radius parameter) set explicitly in the mission statements. This has been done because the model varies too much in the inferred values of the region's radius, even with the temperature set to 0, and this parameter affects greatly the other design parameters, thus making the analysis of the results very difficult.

3 Results

Figure 2 to Figure 6 illustrate the impact of fixing the region's size on the parameters computed by the computational engine (CE). The comparison reveals that when the region's size is not predetermined, there is a significant variation in the calculated values across different instances for some missions. Conversely, fixing the region's size leads to noticeably more consistent parameter values across multiple instances, which can be clearly notices in Figure 4 and Figure 6. This consistency underscores the sensitivity of the design process to the size of the region. Therefore, to ensure reliability and to minimize variations in our analysis, we will focus exclusively on results where the region size is pre-fixed.

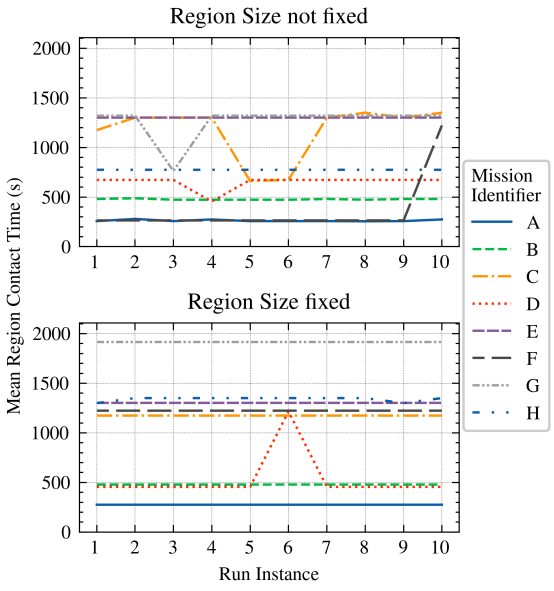

Figure 2 Comparison of Average Region Contact Times in a Single Orbit for Each Mission in One Week, with CE, With and Without Fixed Region Size.

Figure 7 to Figure 11 illustrate the parameters predicted by GPT-4 without the computational engine (CE) across various settings of the temperature parameter (t=0, t=0.5, and t=1). Notably, even under the condition of highest randomness (t=1), certain parameters consistently appear identical across different mission samples. This phenomenon can be clearly

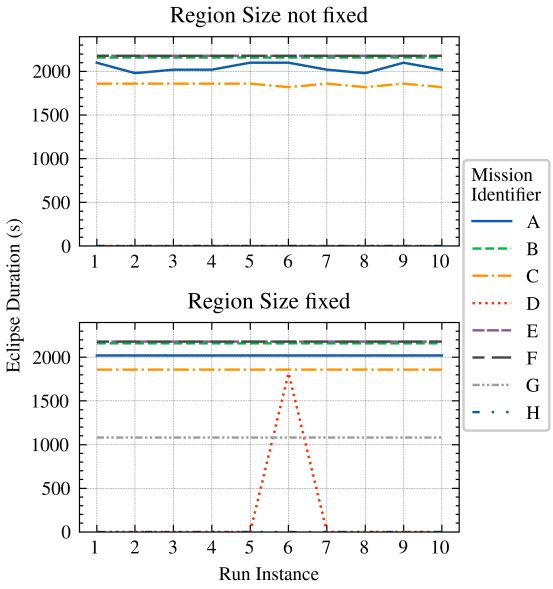

Figure 3 Maximum Eclipse Durations in a Single Orbit for Each Mission in One Week, with CE, With and Without Fixed Region Size.

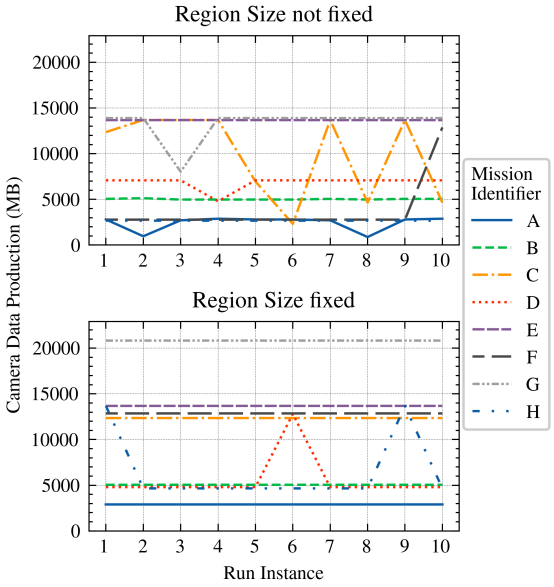

Figure 5 Average Camera Data Productions in a Single Orbit for Each Mission in One Week, with CE, With and Without Fixed Region Size.

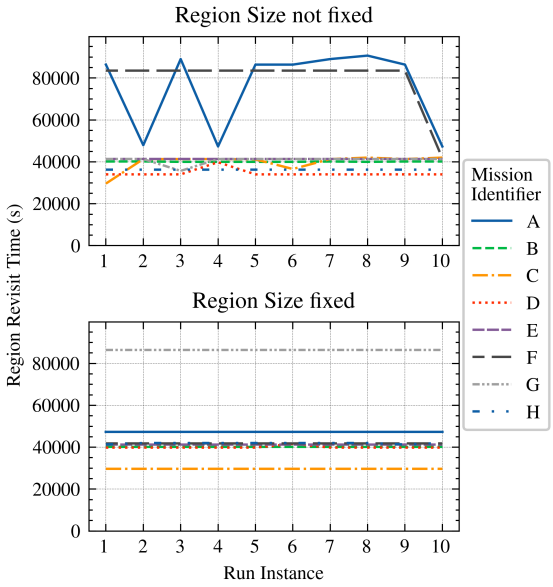

Figure 4 Maximum Region Revisit Times in a Single Orbit for Each Mission in One Week, with CE, With and Without Fixed Region Size.

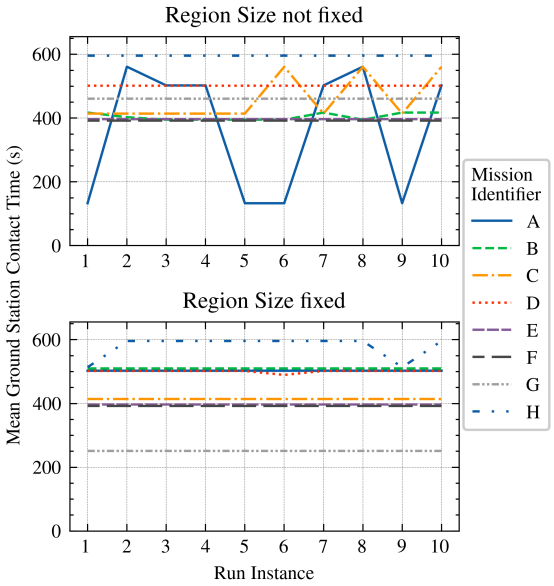

Figure 6 Average Ground Station Contact Times in a Single Orbit for Each Mission in One Week, with CE, With and Without Fixed Region Size.

seen in Figure 7 with the values ca. 600s and 1800s, in Figure 8 with ca. 2400s and 5400s, in Figure 9 with ca. 80000s and 180000s, in Figure 10 with ca. 300MB and 500MB and finally in Figure 11 with ca. 300s and 600s. This uniformity in responses, despite the anticipated variability with higher randomness, suggests a potential issue akin to overfitting. In such cases, the model might be relying on repetitive patterns learned during training rather than adapting to the specifics of each new mission, which could compromise the accuracy and reliability needed of the outputs for engineering applications. In contrast, when integrated with the CE, the outputs vary logically by mission, indicating that each set of mission parameters is being uniquely calculated.

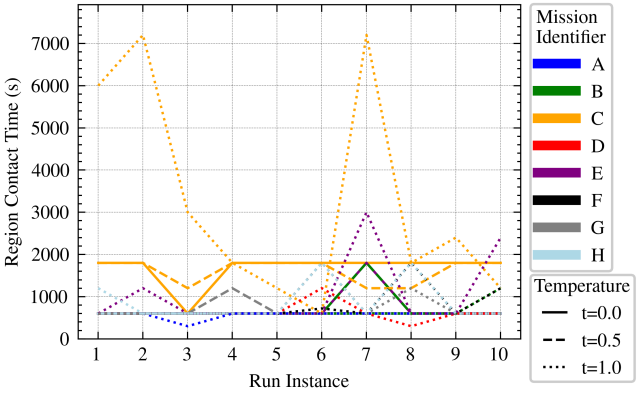

Figure 7 Average Region Contact Times in a Single Orbit for Each Mission in One Week, Only GPT, Fixed Region Size, for Each Temperature.

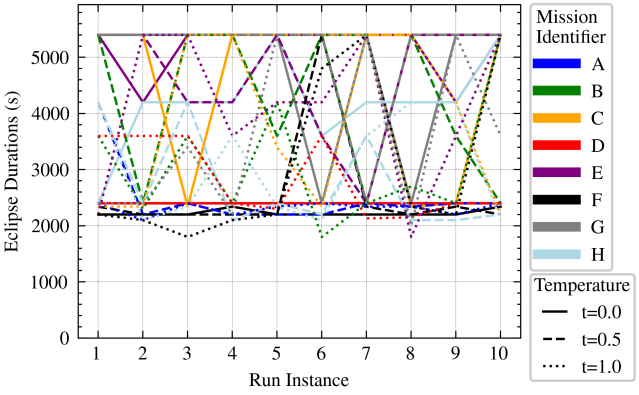

Figure 8 Maximum Eclipse Durations in a Single Orbit for Each Mission in One Week, only GPT, Fixed Region Size, for Each Temperature.

Despite the overall stability improvements with the computational engine (CE), variations in some parameters are still observed for the case of pre-fixed region

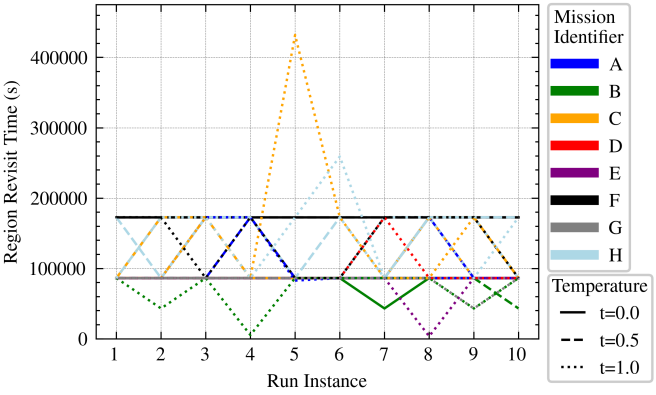

Figure 9 Maximum Region Revisit Times in a Single Orbit for Each Mission in One Week, Only GPT, Fixed Region Size, for Each Temperature.

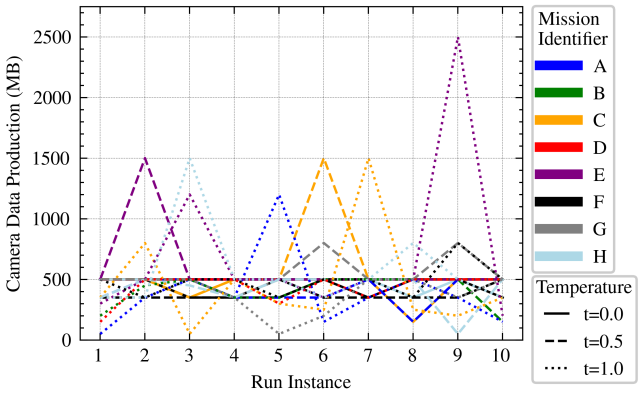

Figure 10 Average Camera Data Productions in a Single Orbit for Each Mission in One Week, Fixed Region Size, Only GPT, for Each Temperature.

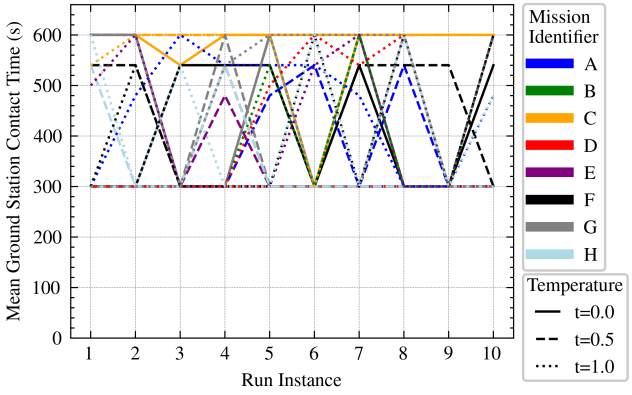

Figure 11 Average Ground Station Contact Times in a Single Orbit for Each Mission in One Week, Fixed Region Size, Only GPT, for Each Temperature.

size, notably for mission D and H as shown in Figure 2 to Figure 6. This variability can be easily traced to differences in the initial parameters, specifically to the GSD, which the model predicts differently across some instances. These differences in GSD are critical as they influence the subsequent calculation of orbital parameters, leading to varied outcomes. Unlike the parameters inferred directly by GPT-4, the variations with the CE demonstrate enhanced traceability, as the source of differences in calculated parameters can be easily identified and linked back to specific input variations, thereby improving the transparency and reliability of the design process.

Stability in the design process

Figure 12 to Figure 16 present histograms of the coefficient of variation (CV) for all evaluated parameters with pre-fixed region size. The coefficient of variation is a statistical measure of the relative dispersion of data points around the mean, defined as follows:

$$CV = \frac{\sigma}{\mu} \tag{3}$$

where σ is the standard deviation of P, calculated as:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - \mu)^2} \tag{4}$$

and μ is the mean of P, calculated as:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} P_i \tag{5}$$

The analysis reveals that, with the exception of missions D and H (as highlighted earlier), the parameters computed with the computational engine (CE) demonstrate perfect stability, showcasing no variance in outputs across multiple instances. This consistency is crucial, especially in an iterative design process where predictability and reliability are key. Consistent outputs ensure that any modifications to design inputs or mission parameters are the sole drivers of changes in the design, rather than variability in the tool's computational process. In contrast, the parameters predicted by the model alone without the CE exhibit significant fluctuations, even when the model's randomness setting (temperature) is at its lowest (t=0). This variability could lead to uncertainties in decision-making, as designers might not discern whether changes in output are due to altered inputs or inherent model instability.

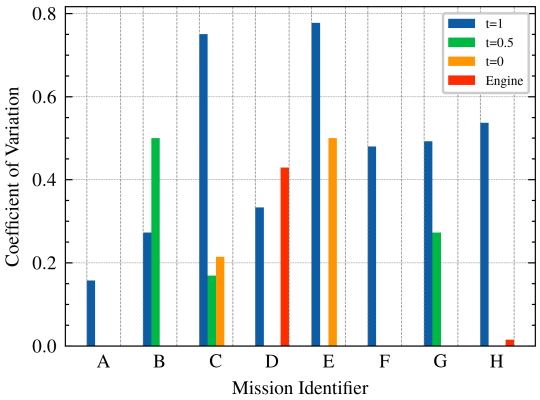

Figure 12 Coefficient of Variation of Average Region Contact Time for Each Mission with Fixed Region Size.

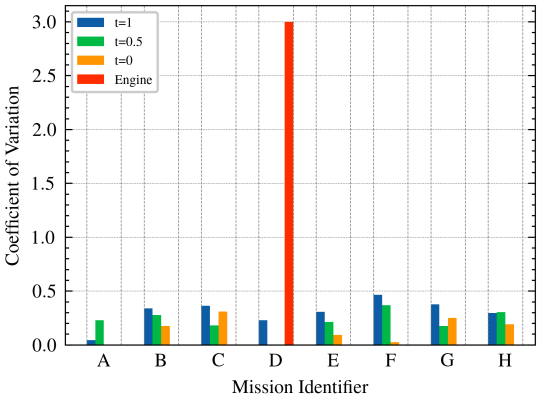

Figure 13 Coefficient of Variation of Maximum Eclipse Duration for Each Mission with Fixed Region Size.

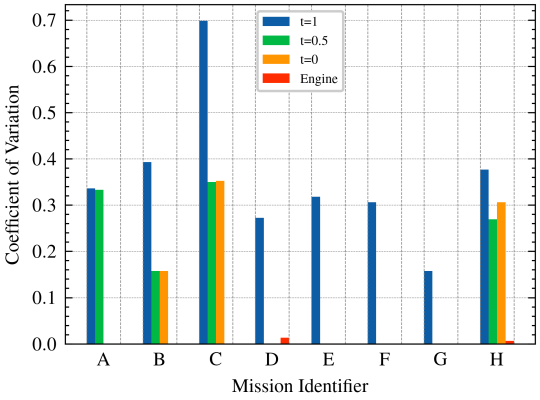

Figure 14 Coefficient of Variation of Maximum Region Revisit Time for Each Mission with Fixed Region Size.

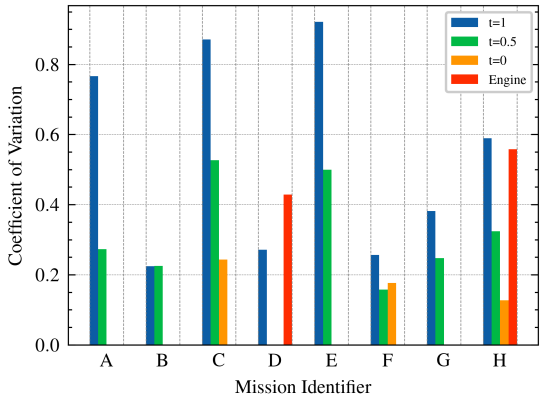

Figure 15 Coefficient of Variation of Average Camera Data Production for Each Mission with Fixed Region Size.

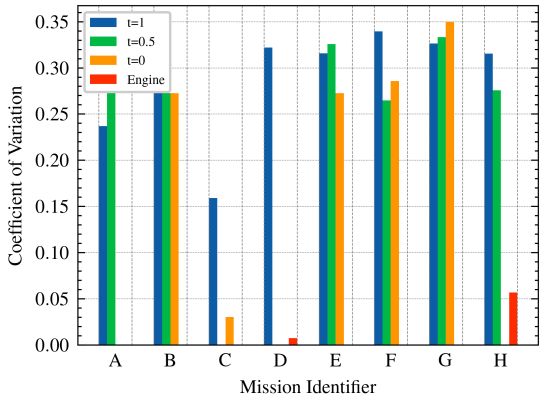

Figure 16 Coefficient of Variation of Average Ground Station Contact Time for Each Mission with Fixed Region Size.

4 Conclusions and Limitations

The development of a computational engine for use with LLMs represents an important step in the field of spacecraft design, particularly in the early design phases. This paper has demonstrated that the proposed framework not only achieves improved accuracy and efficiency in design tasks but also enhances the traceability of decision-making processes, which is critical in engineering.

The integration of the CE reduces the frequency at which the LLM needs to be invoked. By handling computations internally and only using the LLM when necessary for specific tasks, the CE minimizes the computational load that typically comes with continuously running large models. This selective use significantly enhances overall computational efficiency, as operating an LLM can be resource-intensive and costly, particularly when dealing with complex calculations and large data volumes.

The CE significantly reduces the likelihood of the uniformity phenomenon, ensuring more reliable outputs across varied inputs. It maintains high stability in its calculations, providing consistent results across numerous run instances.

The modular nature of the CE allows for the incorporation of additional spacecraft subsystems and support for a wider array of mission types. Future iterations could perhaps add user interaction, allowing for more control over each step in the design process.

All the code and documentation can be found in https://gitlab.lrz.de/sps-public/spacemodelers

Limitations

Having explained the conclusions of this work, it is important to state and explain the different limitations that the taken approach to the problem implies:

- The CE's effectiveness is confined to the specific types and classes of missions for which it has been configured—primarily Earth observation missions using CubeSats in Sun-Synchronous orbits. This specialization restricts the adaptability of the CE to other mission types without significant reconfiguration or redevelopment.

- The current framework exhibits limited flexibility in accommodating specific design requirements that deviate from its predefined configurations. For instance, if a designer wishes to specify a particular type of camera or antenna with unique characteristics, the framework may not reliably

enforce these preferences throughout the design process. Even if these components were manually added to the database, the CE might overlook them in favor of other options, potentially leading to sub-optimal design choices.

- This computation engine relies heavily on the use of OpenAI's API and models, which could pose limitations when trying to use the CE with other open-source LLMs. This could also pose risks if the API's structure were to be heavily altered.

Addressing these limitations could significantly enhance the framework's utility and applicability. For example, integrating comprehensive modelling functionalities within the model's architecture could provide a more robust and versatile tool by making the model inherently understand an Spacecraft design process. This approach, as discussed in [4] involves incorporating systems engineering methodologies directly within LLMs to improve their accuracy and reliability in technical domains.

Furthermore, these limitations could be potentially mitigated by deeper integration of computational functions directly within the model's architecture, as suggested in [8]. The "ToolkenGPT" method, which augments LLMs with a vast array of computational tools via embeddings, demonstrates the potential benefits of such enhancements, suggesting that a similar strategy could be applied to extend the range of the CE's capabilities. This approach would make the model more flexible, because it would allow it to use any function at any given point of the design process if seen fit.

Future work for this paper would include:

- The addition of the rest of the spacecraft subsystems in the design process.
- The validation of the developed functions with unit tests.
- The addition of chain-of-thought methods to improve the performance of the LLM in the case where no CE is used.
- The addition of a tool to facilitate traceability in the design process of the CE.

Acknowledgment

Special thanks to Ramón García Alarcia for proposing the topic and supervising and helping me during the process. Thanks to the Chair of Spacecraft Systems of TUM for sponsoring the use of OpenAI's API models for the development and testing.

References

[1] D. Andor, L. He, K. Lee, and E. Pitler. Giving BERT a Calculator: Finding Operations and Arguments with Reading Comprehension. In *arXiv preprint arXiv:1909.00109*, 2019.

[2] A. Berquand, P. Darm, and A. Riccardi. Space-Transformers: Language Modeling for Space Systems. *IEEE Access*, 9:133111–133116, 2021.

[3] A. Berquand, F. Murdaca, A. Riccardi, T. Soares, S. Generé, N. Brauer, and K. Kumar. Artificial Intelligence for the Early Design Phases of Space Missions. In *IEEE*, 2019.

[4] R. García Alarcia, P. Russo, A. Renga, and A. Golkar. Bringing Systems Engineering Models to Large Language Models: An Integration of OPM with an LLM for Design Assistants. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MBSE-AI Integration, SciTePress*, pages 334–345, 2024.

[5] Z. Ji et al. ChipNeMo: Domain-Adapted LLMs for Chip Design. *ACM Comput. Surv.*, 55(12):1–32, 2023.

[6] OpenAI. OpenAI API. OpenAI Documentation, 2024. [Online]. Available: https://www.openai.com/api/.

[7] A. Parisi, Y. Zhao, and N. Fiedel. TALM: Tool Augmented Language Models. In *arXiv preprint arXiv:2205.12255*, 2022.

[8] R. Raileanu et al. ToolkenGPT: Augmenting Frozen Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[9] B. Rhodes. Skyfield: High Precision Research-Grade Astronomy for Python, 2024. [Online]. Available: https://rhodesmill.org/skyfield/.

[10] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *arXiv preprint arXiv:2302.04761*, 2023.

5 Appendix

The following is the prompt used to infer the parameters used for the analysis, for the case where the CE is not used:

Based on the following initial design parameters:
desired_gsd = initial_parameters['gsd_m']m
longitude of the area = initial_parameters['longitude_deg']
latitude of the area = initial_parameters['latitude_deg']
radius of the area = initial_parameters['radius_km']km
revisit time = initial_parameters['revisit_time_days'] days
Number of ground stations = initial_parameters['num_ground_stations']
That have been inferred from the following mission statement: mission_statement. Please infer the following parameters of the mission, knowing that the time span that you can assume to inferr or calculate them is one week, starting from the 01/01/2025 at 17:15:
1. Maximum region revisit time during the time span considered in seconds
2. Maximum eclipse duration within a single orbit during the time span considered in seconds
3. Mean contact time in one orbit with the region in seconds
4. Mean camera data production per coverage of the region in MB
5. Mean contact time with the ground station/s in one orbit in seconds
Respond only with the asked values separated by commas without the units. Don't write anything that are not the numbers asked for in the asked format.
Format of response example: 10000, 1000, 1000, 1000, 1000