75th International Astronautical Congress (IAC), Milan, Italy, 14-18 October 2024.
Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

IAC-24-D1,6,8,x86846

A computation engine for numerical system requirements generation in LLM-based spacecraft design assistants

Ramón María García Alarcia[a]*, Alessandro Golkar[a]

[a] *Department of Aerospace and Geodesy, Technical University of Munich, Lise-Meitner-Strasse 9, 85521 Ottobrunn, Germany*, ramon.garcia-alarcia@tum.de, golkar@tum.de
* Corresponding Author

Abstract

The design of spacecraft, and more broadly of space missions, can greatly benefit from LLM-based design assistants supporting engineers in the extraction of relevant information from past missions for its reuse in the new one, as well as in the automatic generation of system parameters or requirements in text at different levels. However, due to the mathematical nature of technical systems, many of the spacecraft parameters or requirements are numerical and involve calculations or the application of first principles. While Large Language Models excel at text generation, predicting the next words in a sequence with the highest probability, they are not suited, being statistical engines, to perform deterministic computations.

In this paper, we present the architecture of a computation engine to which numerical calculations are outsourced during the process of generating parameters or requirements by an LLM-based design assistant. We present the integration with the LLM, measure its performance, and benchmark and discuss the improvement in parameters generation with a space mission as a use case.

In particular, the computation engine hereby developed follows an object-oriented programming paradigm, mimicking through the classes the subsystems of a spacecraft, through the class properties their parameters, and through the class functions the equations and first principles used to calculate the parameters' values, which can be then expressed by the LLM as numerical system parameters or requirements with natural language. A high-level programming language and a modular approach are followed in seek of high readability and modifiability, allowing the engine to be extended and its granularity increased in future iterations. Different alternatives for the integration of the computation engine with the LLM and in the design flow are evaluated and compared, and parameters such as the computation speed and error are measured and assessed along different computation platforms.

By presenting this computation engine for spacecraft design, we aim at patching the shortcomings of LLM-based design assistants in numerical calculations, paving the way for their adoption and thus helping accelerate and simplify spacecraft design tasks for a broader range of institutions and individuals.

Keywords: spacecraft design assistant, computation engine, numerical system parameters, Large Language Model, Generative AI

Acronyms/Abbreviations

Application Programming Interface (API)
Artificial Intelligence (AI)
Attitude Determination and Control System (ADCS)
Commercial-Off-The-Shelf (COTS)
Communication (COM)
Computation Engine (CE)
Computer-Aided Design (CAD)
Electrical Power System (EPS)
Generative Adversarial Network (GAN)
Generative Pre-trained Transformer (GPT)
Large Language Model (LLM)
On-Board Computer (OBC)
Open Source (OS)
Payload (PLD)
Propulsion (PRO)
Structure (STR)
Thermal Control System (TCS)

1. Introduction

We are living in times of constant technological and industrial transformation. Space exploration and utilization, historically performed by large, risk-averse, budget-hungry organizations and companies, is not extraneous to progress. The space industry has been disrupted in the last decade by the lowering cost and increased opportunities for access to space, the introduction of the CubeSat standard, and the increased utilization of Commercial-Off-The-Shelf (COTS) components. This has allowed a greater number of actors to realize a bigger number of missions at lower costs. As a general, enabling technology, Artificial Intelligence has also gone a long way in the last decade, and in the last years, Generative AI and Large Language Models (LLM) have taken over the public arena and are more and more used in support and automatization of a

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

myriad of tasks. The space industry has progressively incorporated AI technology, fundamentally into downstream tasks related to data processing, with Deep Learning models. Studies and demonstrations on how to apply AI to upstream tasks, for instance in mission operations, increasing spacecraft autonomy, are underway. Still, the application of Generative AI and LLMs is quite a recent topic of discussion for space activities, and one of the tasks to which it can naturally be applied is at the beginning of a space project: the design of a space mission and its spacecraft.

Indeed, the design of spacecraft is a process or a set of tasks that is still performed in a very classical and manual manner, producing varied documentation and relying on heritage information to make decisions and make appropriate design choices. LLMs are very good both at producing documentation in different formats and tones, as well as ingesting heritage information and data and using it appropriately to generate responses to varied questions. Introducing LLMs can effectively support spacecraft designers, modernizing and automatizing the spacecraft design process, at least in its initial stages. However, the introduction of LLMs in the design process of any technical system faces a fundamental challenge: designing technical systems requires numbers to be produced, equations to be applied, first principles to be considered. LLMs, on the other hand, are fundamentally statistical engines made to work with text, predicting which is the word that comes next in a sentence with the highest probability – not deterministic engines that apply an equation or a first principle and faithfully generate a result transforming numerical inputs to numerical outputs. This prevents LLMs and other Generative AI technologies from being directly applicable, with trustworthiness, to problems related to the design of technical systems.

Fortunately, external tools can be interfaced with LLMs to bridge their shortcomings. For instance, LLMs can call external functions, giving them some input, and collecting back their output to integrate it in the flow of natural language generation. More and more models, be them open-source or proprietary, are including function calling capabilities, and developers are creating tools to enhance the generation capabilities of the LLMs. In this sense, computation engines are one sort of tools that can be developed that are able to compute numerical values for technical systems. In this work, we present what, to the best of our knowledge, is the first computation engine to calculate spacecraft design parameters in the framework of an LLM-based design assistant. The computation engine, built with modularity and modifiability in mind, coded in Rust, is unit tested at the level of functions. When the LLM identifies the need to produce a numerical value, it calls the appropriate function from the computation engine with the required inputs, retrieves the output value, and includes it in its text generation. The improvement of accuracy in numerical system parameters or requirements generation is assessed. Given the increase of accuracy, reassurance and trustworthiness in an LLM-based design assistant is increased. This opens the way for future adoption by companies and institutions developing space missions.

2. Related works

Engineers working in different industries have long made use of software to support engineering design tasks, for instance, Computer-Aided Design (CAD) tools. Such tools greatly reduce the time spent in designing, help perform checkups and simulations and improve the overall quality of designs. However, even if they have progressed over the years, the degree to which they automate processes is still low.

Little by little, CAD tools incorporate or give way to design assistants, able for instance, to provide recommendations to the user [1]. Fields such as software engineering have also introduced assistants along the design and development tasks [2], recently benefitting largely from developments in Large Language Models for coding tasks [3]. While LLMs are less used in engineering design for other fields, interest in them is also increasing, for instance in electronics design [4].

Moving to space, the creation of design assistants for space missions or spacecraft has been a matter of research for the past years. Particularly, there has been research and development creating a design assistant for Earth observation distributed missions [5], and more recently an exploration of the use of LLMs for tradespace exploration in space mission design [6], moving from rule-based to transformer-based systems. Others have explored the application of LLMs across different tasks of the space mission lifecycle, implementing the transformer architecture on requirements classification [7] or adding LLM for knowledge management and information reuse on space missions [8].

We have been exploring the usage of LLMs for the design of spacecraft. Already last year [9], we identified the necessity of implementing both systems engineering models that would structure inputs and outputs on the design assistant and provide ontologies for the models to learn, which we covered in a recent work [10], as well as the need to add a computation engine for numerical design parameters to be properly generated. The computation engine, for which an initial student

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

project was performed in our lab [11], is tackled in this work.

Regarding calling functions or tools from LLMs, a variety of works tackling different tools for different applications have been performed recently. For instance, recent research trained a transformer model to be able to decide to call external Application Programming Interfaces (API) to interface tools such as a calculator, a question-and-answer system, a search tool, a translator, or a calendar [12]. An in-depth thesis covers particularly the fine-tuning of LLMs to learn them to use tools [13]. An improvement of performance of up to 2.8 times has been measured in the resolution of complex tasks by LLMs such as GPT-4 interfacing tools, with respect to the same LLM alone [14].

This work, with respect to the current state-of-the-art and published literature on the topic, differs in the following aspects: (a) with respect to spacecraft design assistants, the one we present in our work is an LLM-based one, moving forward from more classical, rule-based tools (b) with respect to other tools callable from LLMs, we develop one for the design of space missions and spacecraft, to the best of our knowledge a novel development.

3. Methodology

This section explains on the one hand how the computation engine has been developed, validated, and integrated into a bigger LLM-based spacecraft design assistant; and also, the performance of the overall integrated system and how we assess the improvement in requirement generation by the design assistant when incorporating the computation engine into the system, with respect to an assistant that does not have this computation engine.

3.1 Development of the computation engine

As previously stated, the computation engine is a compendium of functions that calculate different numerical parameters of a space mission or spacecraft. The functions do not need to be in a particular configuration or order. However, we follow a more structured approach that is outlined in this subsection.

Before talking about the computation engine that has been developed, a series of considerations need to be explained. First, our computation engine is generic. This is because space missions and spacecraft can be very different. In many cases, there is little resemblance between a large telecommunications satellite in GEO, built by a large system integrator such as Airbus Defence and Space, and a small CubeSat in LEO built by a university or a research centre. Despite this, they all share some characteristics: there will be a physical Structure (STR), and you will have a bigger or smaller Attitude Determination and Control System (ADCS) for pointing, Communications (COM) system to talk to the spacecraft and receive data from it, Electrical Power System (EPS) for powering the spacecraft, On-Board Computer (OBC) and data handling to take decisions onboard. On top, you might have a Propulsion (PRO) system to change orbits, and a Thermal Control System (TCS) to maintain proper temperatures inside the spacecraft. And of course, each satellite will have one or more Payload (PLD) that capture the mission data or provide the mission service.

By creating a generic Computation Engine (CE), we focus on these common systems and their main parameters, but we do not tackle systems that are specific to certain missions or spacecraft classes. Additionally, the PRO and TCS systems are not implemented in this early version of the CE, since they are not present in all satellites (especially on nanosatellites and CubeSats) and are more complex systems.

Beyond being generic the computation engine hereby presented is also not exhaustive. This means that it does not compute all the numerical properties of all the subsystems present in a generic spacecraft, but only those that are of major importance and defining the global budgets of the system (e.g., mass budget, power budget, radiofrequency link budget, data budget, etc.). This is done, on the one hand for the sake of simplicity and saving time, as the focus of this work is having a functional prototype allowing to prove the suitability and benefits of such system in the framework of a spacecraft design assistant.

As an example, the computation engine can cover some of the following details of interest for a space mission and spacecraft:

- **Space Mission**: The required delta-V from an insertion orbit to a final orbit [m/s], the required delta-V for deorbiting the spacecraft [m/s], the lifetime [years].
- **Spacecraft:** The general parameters of the spacecraft, including total mass [kg], volume [U], power consumption [W], which are the main components of the mass budget and power budget.
- **AOCS:** The pointing accuracy [deg], of the spacecraft, the required momentum [Nm], being the main components of the pointing budget.
- **COM:** The transmitted power [W], the frequency [Hz], the bandwidth [Hz], and the maximum range [m] for the downlink, the

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

received power [W], the Signal-to-Noise Ratio [dB], and the Bit Error Rate for the uplink.
- **EPS:** The power generated by a solar array [W], the required battery cells.
- **OBC:** The required processor speed [MHz], the required data storage [MB].

For the reason of its limited scope and the need for extensions, the computation engine is built with modularity and modifiability in mind so that it can be improved in the future. Indeed, the base of the code* is released publicly in a Gitlab repository for the space community to use it, modify it, and expand it according to the needs of each user.

The CE is programmed with a high-level programming language, the Rust language, creating modules for the spacecraft, and for each of its subsystems. Modules implement structs, containing the parameters of each subsystem as variables, and functions, that calculate the values of such parameters when called. We selected Rust because it is a compiled -and thus fast to execute-, cross-platform, and modern language with an increasing user community, being adopted more and more as the language where AI model inference is performed. The CE is composed, as depicted by Figure 1, a *main* file which imports all necessary modules, creates the system message, and handles user input/output. An llm.rs file takes care of the API calls to OpenAI, creating the necessary JSON structures for exchange of messages. *functions_handler.rs* creates the JSON structure with the callable functions and its parameters. It also performs the calling of functions when requested by the LLM. *ce.rs* is an instrumental file to import all the CE classes. A *spacemission* class instantiates an *orbit* class and a *spacecraft* class. The *spacecraft* class instantiates multiple satellite subsystems.

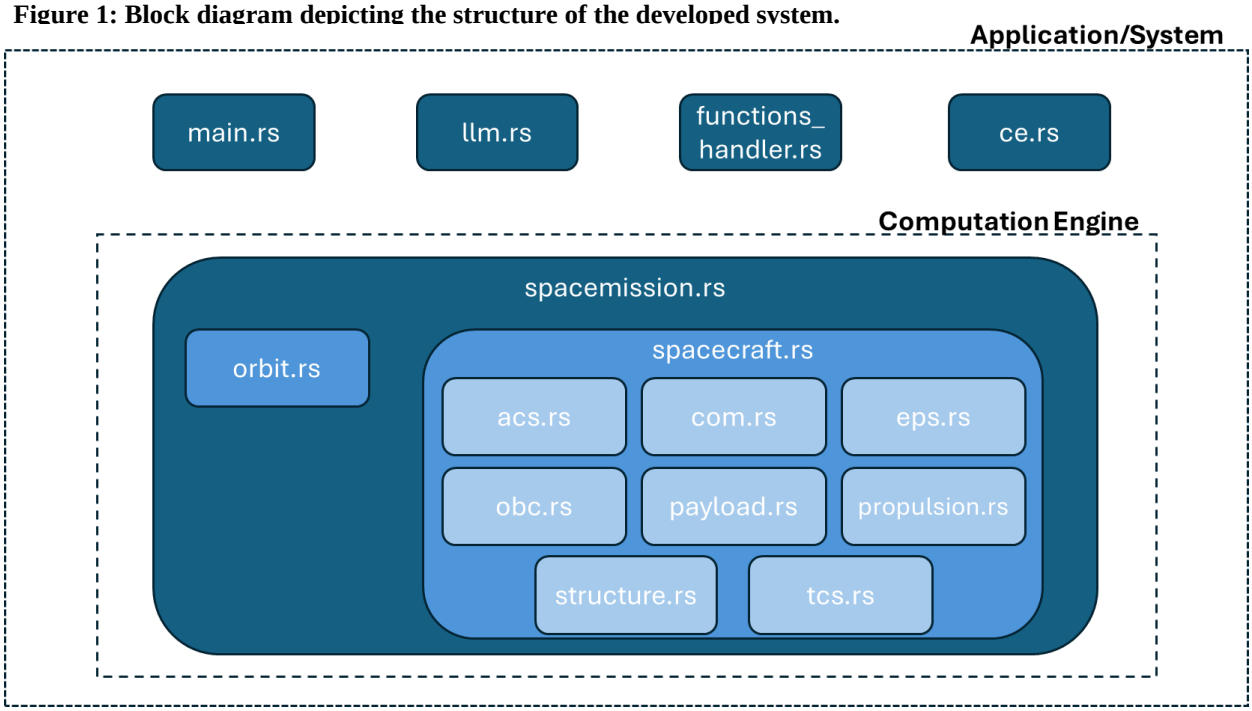
3.2 Integration of the computation engine with the LLM

In what follows, we will explain the typical way in which a tool is integrated with an LLM, which is the one followed in this work as well. When interacting with an LLM, a JSON structure is normally exchanged. These structures include some prompts as well as parameters. Among the prompts, the most important is the response one, which is displayed to the user typically in a chat interface. Additionally, a typical prompt is the system one, which tells the LLM how it needs to behave -e.g., what tone to use in the responses-, and typical parameters are the temperature -which regulates the stochasticity and thus the creativity of the LLM-. When integrating external tools, the JSON structure contains a field for tools, making explicit to the LLM the existence of functions and sharing their name and their input parameters, as depicted by Code 1. The LLM, by having been exposed to the existence of these functions, decides to use them at discretion.

When an LLM has decided to use one function, this is specified in the returned JSON structure, with the inputs that the LLM has decided to provide, as shown in Code 2. The user's system needs to read this, locally run the selected function with the provided response on the user side and return a JSON structure with the values

* Available at: https://gitlab.lrz.de/rgalarcia/ce_base

Figure 1: Block diagram depicting the structure of the developed system.



Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

outputted by the function. A JSON structure containing the output values is sent to the LLM. The LLM integrates that into its text and provides a message with the numerical value incorporated.

Code 1: Portion of the JSON shared with the LLM, indicating the functions available for calling, with one function.

```
{
    "type": "function",
    "function":
    {
        "name":
"calculate_orbital_period",
        "description": "Calculate the
orbital period of a satellite in
Earth's orbit based on its altitude.",
        "parameters":
        {
            "type": "object",
            "properties":
            {
                "altitude_km":
                {
                    "type": "number",
                    "description": "The
altitude of the satellite in
kilometers."
                }
            },
            "required": ["altitude_km"]
        }
    }
}
```

Code 2: Portion of the JSON in which the LLM requests the calling of a function.

```
{
"tool_calls": [
          {
            "id": "call_ID123",
            "type": "function",
            "function": {
              "name":
"calculate_orbital_period",
              "arguments":
"{\"altitude_km\":1000}"
            }
          }
        ]
}
```

The behaviour explained before is typically the same for both open-source (OS), locally running LLMs, and proprietary, API-based LLMs.

In our case, we use a closed-source, API-based LLM through OpenAI. The selected model when using the Computation Engine is the *gpt-3.5-turbo-0125*, due to the lower costs in comparison to more recent models such as *gpt4* or *gpt4o*, and yet solid behaviour in understanding the available functions and calling whenever necessary. However, due to the need of additional general knowledge and reasoning capabilities, *gpt-4o-2024-08-06* is used in this work's evaluation when not using a CE. Both models are called setting their internal temperature parameter to zero, with the goal of obtaining more reproducible results by reducing the randomness of the answers.

Due to the scarcity of internally available computational resources, it has not been possible in the frame of this initial work to use an open source, locally running LLM. The LLMs that can run on small consumer devices are those of small size, with a typical number of parameters ranging from 2 billion to 7 billion. Unfortunately, such small models are unable to properly understand the functions that are available to be called, and calling them with the correct name and arguments, and format. Only medium models, typically of 70 billion parameters, can do this consistently. Unfortunately, such medium models are already unable to run on consumer devices and require a more powerful computational architecture.

3.3 Usage of the system

From a user's perspective, it is rather simple to use the system. The user only needs to ask design questions, whether they involve the calculation of numerical parameters or requirements, or not. The system takes care of instantiating the appropriate Computation Engine classes with their parameters and functions, it creates the system prompt that is shared with the LLM informing of its intended behaviour and the availability of functions, and it also rotes queries from the user to the LLM and vice versa for the model responses, while keeping the conversation context and including it in each user-LLM exchange. Additionally, the system records performance metrics, such as the number of tokens that were sent to the LLM or retrieved from it. Naturally, the system is able to handle the LLM instructions of calling a specific function with a series of parameters, returning the result to the LLM and fetching the final text back from it. This operation happens in a completely transparent fashion to the user, who only sees the final result. The flowchart depicted by Figure 2 helps understand the actions and interactions happening when a user utilizes the system asking a design question, when it is not involving numerical calculations and when it requires so.

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

3.4 Assessment of the trustworthiness and performance of the system

To assess the performance of the system, we measure the following two parameters:

- Accuracy of the system without a Computation Engine, measured through the relative error between the output result and the ground truth for each of a series of questions.
- Accuracy of the system with an integrated CE, measured as per above.
- Number of input tokens (sent by the developed application to the LLM) without a CE
- Number of output tokens (sent from the LLM to the developed application) without a CE
- Number of input tokens with a CE
- Number of output tokens with a CE

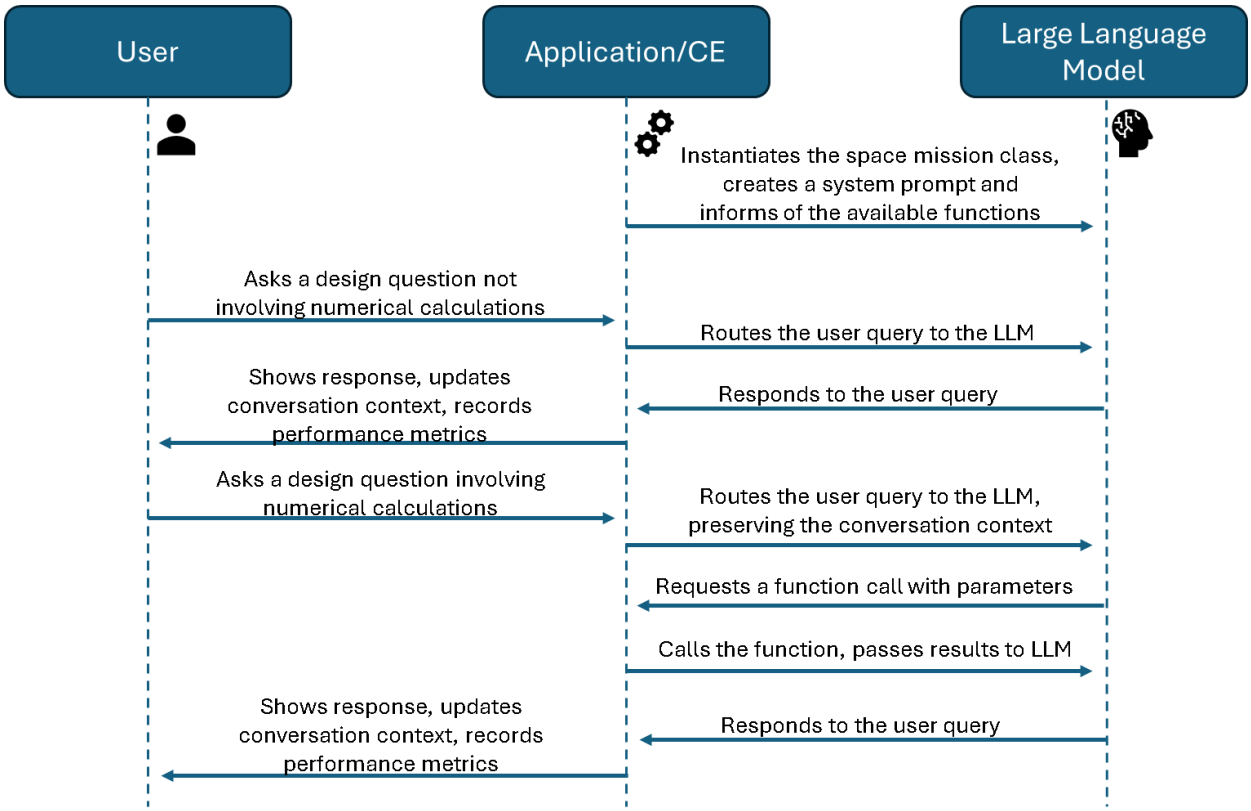
On the one hand, we expect the accuracy of the system to greatly improve when incorporating the CE, since the numerical calculations, which as explained are not performed properly by an LLM, will be outsourced to functions. On the other hand, a slight increase in the number of tokens, both at input and at output, would be expected to be recorded when using a CE.

Input tokens are expected to increase because, along with the conversation, the different functions available to the LLM to call must be included, with a lengthy text structure in JSON depicted before in Section 3.3. Additionally, the output tokens are expected to increase because when needing to call a function, the LLM will produce an additional response, an intermediary one, asking for the function to be called (before producing the final response, shown to the user, with the response text).

Thus, a trade-off appears. One of the valuable conclusions of this work is to understand whether the benefits of including the CE outgrow the burden in terms of token usage (which incur a cost, either paid to the LLM provider with the LLM calls or paid internally by the usage of computational equipment for a longer time). We expect the benefits to be bigger than the drawbacks and the CE to be a useful and cost-effective tool in advancing LLMs and LLM-based design assistants in the context of space systems engineering.

It is also important to mention that, when having locally running LLMs, which is not the case of this work, a broader set of metrics could be measured to better characterise the performance of the system. For instance, measuring the token generation speed

Figure 2: Flowchart depicting the usage of the developed system.



Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

(token/s), the CPU and GPU utilization (%), the RAM consumption (GB), or the power consumption (W), and how they change with and without the integration of the CE, would give a full picture that we are unable to give at this initial stage of our work presented here.

For all the parameters mentioned before, measures are first taken during a conversation with a pure LLM, and then in a conversation with an LLM with the integrated computation engine, making use of it. We have devised a series of questions that request the activation of the CE to compute some numerical values, relating to different spacecraft subsystem designs. The questions are adapted from academic books and exams and created on our own. All questions are previously solved to have the correct numerical value, the ground truth. They aim to cover the different areas that the CE covers.

During the evaluation, on the prompt side, there is an initial prompt, with a *system* role, which says: *"You are a helpful spacecraft design assistant. If available, use supplied tools to assist the user"*. Then, the evaluation starts by asking the LLM -with the *user* role- to open a design session and create a space mission with a spacecraft of certain characteristics. In particular, the following series of five prompts are used. First: *"Let's open a design session, create a new space mission with name EventSat"*. Second: *"The goal of the mission is to detect, classify, and identify objects in space"*. Third: *"The mission consists of one 6U CubeSat spacecraft"*. Fourth: *"The mission orbit is a 500 km altitude Sun-Synchronous Orbit (SSO). Set the orbit parameters"*. Fifth: *"The CubeSat has a 2U payload consisting of an event camera coupled with a small telescopic optic for space-to-space imaging"*. Even if there is no ground truth for these answers to be checked, we expect the LLM to call the functions that instantiate the space mission class object and its sub-objects (e.g. orbit, spacecraft, and spacecraft subsystem classes). What follows in Table 1 are the questions, following in the design session, that do have a ground truth and that are used for the trustworthiness and performance evaluation.

One of the things that can be noted by inspecting the Table, one of the benefits of using LLMs (with the preservation of the conversation context) is their understanding of the context when asked new questions. For instance, a user asking a question "And in GEO?" to a rule-based system would probably not produce any meaningful result. But when asked to an LLM that has the previous pieces of the conversation, knowing that it was asked to calculate the orbital period before, it is clear for the LLM that it now needs to call a function to compute the orbital period with a parameter altitude of 35 786 km. A regular system would not understand that it must call a function, would not understand which specific function it needs to call, would not know that it needs to give an altitude in km as an input, and that satellites in GEO are in a specific and well-known altitude.

Table 1: Evaluation questions and expected answers

Question	Ground truth
MISSION - ORBIT	
What is its orbital period of EventSat?	94.47 min
What is the Delta_V required to go down to a circular orbit of 300 km for its disposal?	113.32 m/s
The satellite has a mass of 12 kg. What is its lifetime at its planned orbit?	3.02 years
And at 300 km?	0.041 years
PAYLOAD	
Set the EventSat payload power consumption to 11.5 W and the EventSat payload mass to 2 kg. In application of the diffraction limit and assuming a visible light observation, focal length of 500 mm and an aperture of 90 mm, what is the maximum distance at which the payload will detect satellites of 2 m size?	120.78 km
ATTITUDE DETERMINATION AND CONTROL	
Set the ADCS power consumption to 3.9 W and the ADCS mass to 0.62 kg. What is the required momentum storage in the reaction wheels to counteract a maximum torque of 3.7e-5 Nm, during 1/4 of the orbit?	0.037 Nms
COMMUNICATIONS	
The satellite is equipped with a UHF band transmitter. Set the communications system power to 2 W and the communications system mass to 0.30 kg, transmitting at a frequency of 436.5 MHz. No antenna gains and system losses are considered. What is the power received by a ground station on Earth?	-106.2 dBm
The satellite is equipped with a UHF band receiver. It receives a signal of bandwidth 40 kHz and power -90 dBm. The temperature of the receiving system is 700 K. What is the Signal to Noise Ratio?	34.13 dB
Assuming a digital modulation	0.256

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

QPSK and a symbol rate of 1 MBaud are used, what is the expected Bit Error Rate?	
ELECTRICAL POWER	
The satellite has solar arrays, their combined mass is 0.68 kg, add this to the EPS mass. The solar array area is of 0.125 square meters, with an efficiency of 25%. What is the amount of power that can be generated by the satellite?	42.53 W
The satellite has a battery of mass 2.18 kg, add this to the EPS mass. The battery cells have a capacity of 10.5 Wh. What is the number of cells required to accumulate the power generated for one orbit, assuming one third of the time the satellite is eclipsed?	5 cells
ON-BOARD COMPUTING AND DATA	
Set the OBC power consumption to 2.15 W and the OBC mass to 0.13 kg. The satellite is equipped with a payload of maximum speed 1.6 Gbps. What is the required clock speed for a CPU of 32 bits?	50 MHz
The payload works for 20% of an orbital period. What is the required amount of data storage capacity for one orbit?	226.73 GB
SPACECRAFT	
What is the total mass of EventSat?	5.91 kg (partial knowledge)
What is the peak power consumption of EventSat?	19.55 W

4. Results

This section presents the results of the evaluation of the trustworthiness and the performance of the developed system, by following the strategy previously outlined in Section 3.4.

The number of input tokens for the different questions, with the CE and without the CE, is presented in Figure 3. The same is done for the output tokens in Figure 4.

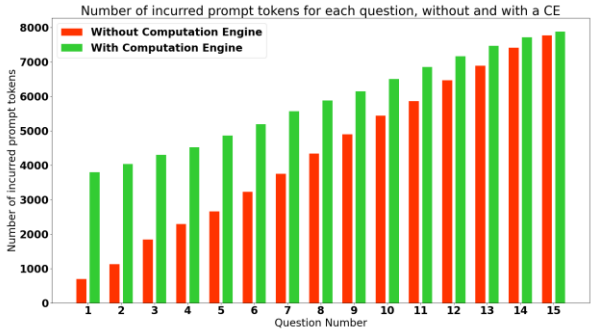


Figure 3: Number of prompt tokens for the different evaluation questions, with the CE incorporated and without it.

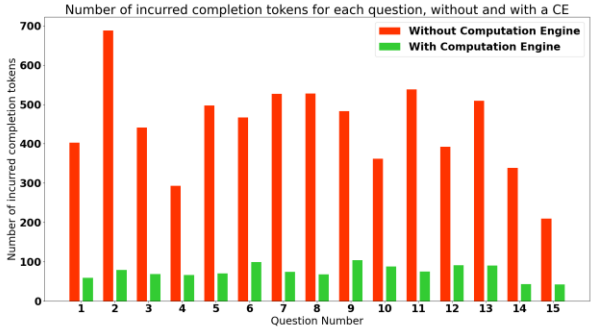


Figure 4: Number of completion tokens for the different evaluation questions, with the CE incorporated and without it.

Figure 5 shows the relative error (in %) with respect to the ground truth, without and with the CE, for the different evaluation questions.

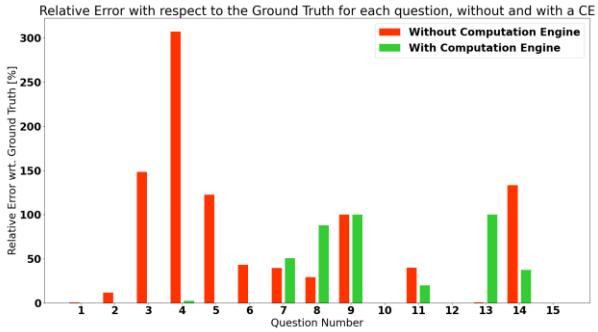


Figure 5: Relative Error with respect to the ground truth for the different evaluation questions.

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

5. Discussion

From the results, two clear observations, which can be directly traced back to our expectations or hypotheses, can be drawn.

On the one hand, our expectation that the system with the Computation Engine integrated would deliver more accurate results, and thus be more trustworthy, is confirmed. The relative error of the results with respect to the ground truth is better for almost all the questions (except some which require the LLM to blend both information that is stored in the CE objects as well as in previous LLM prompts – an issue that can be improved by further development of the CE and better prompting strategies). This better accuracy is undoubtedly related to the fact that the LLM knows how to call the functions and integrate the result from them into the generated text. Without the CE, the system struggles to answer the more complex questions, or those for which an answer was not available in the training dataset, thus the relative error is increased significantly.

On the other hand, our hypothesis of increased token consumption only partially holds true: for small conversations, the effect of the inclusion of a JSON data structure with the functions that can be called by the LLM is remarkable. However, since the answers with a CE integrated are significantly shorter (the LLM omits chain of thoughts and explanations), the effect over time, visible in large conversations, is that the inclusion of a CE results in more succinct and less token-costly conversations. Indeed, this behaviour can be seen in Figure 3 and Figure 4: in the first figure, we observe how the token advantage of not having an integrated CE is particularly significant (initial questions, 1-4), however the advantage is diluted over time (final questions, 12-15). This is because the answers with a CE are consistently less token-consuming, as seen in Figure 4.

When accounting for the incurred cost, since we are using *gpt-3.5* for when integrating the CE and *gpt-4o* without a CE, the advantage of using the CE is incontestable. The latter GPT model is around 4 times more expensive than the former in input tokens and about 8 times more expensive in output tokens. However, the former model is not capable of producing proper answers without a CE. Thus, when adding the cost variable in the equation, there is no doubt whatsoever in the benefit of integrating a CE, besides the trustworthiness and performance point of view that were already discussed.

It should not be left out from this discussion what remains an intrinsic problem with LLMs interfacing CEs. This evaluation is only a simplified usage case for the system, and in reality, we cannot always compute the desired numerical parameters or requirements because inputs are missing. Assumptions need to be accepted and made, either by the user or the LLM during the conversation, and the outputs can only be within a range of accuracy depending on the number of inputs accurately provided and the number of assumptions made. The practitioner needs to properly understand what is the maturity of the design decisions that are being made by properly assessing the maturity, and coverage, of the information and data that is being provided.

6. Future Work

There is room for improvement of the initial work presented here. Clearly, the Computation Engine can be extended. Firstly, by adding implementing additional subsystems that were left out in this work due to them being complex and not present in all satellites (namely, the propulsion system and the thermal control system). Secondly, by adding more level of detail to the subsystems, with additional parameters and functions that calculate those parameters. Additionally, two important topics can be addressed. On the one hand, a modification should be made to allow the CE to process multiple function calls in single LLM responses -a capability of OpenAI's LLMs that is currently not exploited-. On the other hand, and most importantly, the interfacing of newly created functions to the LLM is currently poorly scalable: after the coding of the function, it needs to be manually added to the JSON structure that is shared with the LLM, and a caller function needs to be created inside the function caller module. Automatic coding of such JSON structure and the piece of code in the function handler would greatly improve the usability of the CE.

On the evaluation side, additional testing will provide a more comprehensive understanding of the trustworthiness and performance of the system. This includes, among others, fully running the system locally, with open source LLMs, and measuring hardware-related parameters that are not available for measurement with an API-served LLM. This additional testing phase can include a broader set of LLMs to better characterize the impact that the choice of a particular model has on the overall trustworthiness and performance.

7. Conclusion

In this work, the initial version of a Computation Engine for LLM-based spacecraft design assistants has been presented. The architecture as well as the methodology followed for the development and the evaluation of an early version of the system have been introduced. The results of the evaluation show that, with

Copyright ©2024 by the International Astronautical Federation (IAF). All rights reserved.

the integration of the developed CE, a significant improvement is measured in the trustworthiness of LLM-based design assistants, with an impact on token usage that tends to dissipate with long conversations such as design conversations.

To conclude, this work is an important stepping stone towards the consecution of reliable generative spacecraft design assistants, overcoming the limitations of LLMs by adding a trustworthy numerical computation block.

Acknowledgments

I want to thank my student Roberto Aldea Velayos, who first developed a prototype in Python of the computation engine [11] and who, through his supervision and discussions, helped mature ideas in the area. Additionally, I want to express gratitude to my supervisor and lab colleagues for the many insightful discussions that we have had during the development of this work.

References

[1] A. Huet, R. Pinquié, F. Segonds, and P. Véron, "A cognitive design assistant for context-aware computer-aided design," *Procedia CIRP*, vol. 119, pp. 1029–1034, Jan. 2023, doi: 10.1016/j.procir.2023.03.146.

[2] A. F. Del Carpio and L. B. Angarita, "Assistant Solutions in Software Engineering: A Systematic Literature Review," in *2023 IEEE 14th International Conference on Software Engineering and Service Science (ICSESS)*, Oct. 2023, pp. 93–100. doi: 10.1109/ICSESS58500.2023.10293029.

[3] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A Survey on Large Language Models for Code Generation," Jun. 01, 2024, *arXiv*: arXiv:2406.00515. doi: 10.48550/arXiv.2406.00515.

[4] Z. He and B. Yu, "Large Language Models for EDA: Future or Mirage?," in *Proceedings of the 2024 International Symposium on Physical Design*, in ISPD '24. New York, NY, USA: Association for Computing Machinery, Mar. 2024, pp. 65–66. doi: 10.1145/3626184.3639700.

[5] A. V. i Martin and D. Selva, "Daphne: A Virtual Assistant for Designing Earth Observation Distributed Spacecraft Missions," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 30–48, 2020, doi: 10.1109/JSTARS.2019.2948921.

[6] G. Apaza and D. Selva, "Leveraging Large Language Models for Tradespace Exploration," *Journal of Spacecraft and Rockets*, vol. 0, no. 0, pp. 1–19, doi: 10.2514/1.A35834.

[7] A. Berquand, P. Darm, and A. Riccardi, "SpaceTransformers: Language Modeling for Space Systems," *IEEE Access*, vol. 9, pp. 133111–133122, 2021, doi: 10.1109/ACCESS.2021.3115659.

[8] M. Henriquez, A. Herd, P. Danènas, D. Dilijonas, and J. Ontiveros, "CLARK: Building Conversational Intelligence for Knowledge Management in the Space Domain," *ECKM*, vol. 24, no. 1, Art. no. 1, Sep. 2023, doi: 10.34190/eckm.24.1.1792.

[9] R. M. Garcia Alarcia and A. Golkar, "Architecture of a generative design tool for spacecraft and user front-end implementation through a chatbot smart design assistant," presented at the IAC 2023 Congress Proceedings, 74th International Astronautical Congress, 2023. Accessed: Aug. 28, 2024. [Online]. Available: https://mediatum.ub.tum.de/node?id=1724403&change_language=en

[10] R. M. Garcia 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*, Rome, Italy: INSTICC, 2024. doi: 10.5220/0012621900003645.

[11] R. Aldea Velayos, "Development and assessment of a computation engine module for an LLM-enhanced spacecraft design assistant," 2024, Accessed: Oct. 16, 2024. [Online]. Available: https://mediatum.ub.tum.de/node?id=1755552&change_language=en

[12] T. Schick *et al.*, "Toolformer: Language Models Can Teach Themselves to Use Tools," *Advances in Neural Information Processing Systems*, vol. 36, pp. 68539–68551, Dec. 2023.

[13] S. G. Patil, "Teaching Large Language Models to Use Tools at Scale".

[14] Y. Gu *et al.*, "Middleware for LLMs: Tools Are Instrumental for Language Agents in Complex Environments," Feb. 22, 2024, *arXiv*: arXiv:2402.14672. doi: 10.48550/arXiv.2402.14672.