NORDDESIGN CONFERENCE
REYKJAVIK, ICELAND, 12-14 AUGUST 2024

Evaluation of Different Large Language Model Agent Frameworks for Design Engineering Tasks

Pradas Gomez Alejandro, Panarotto Massimo, Isaksson Ola

Chalmers University of Technology, Sweden

Abstract: This paper evaluates Large Language Models (LLMs) ability to support engineering tasks. Reasoning frameworks such as agents and multi-agents are described and compared. The frameworks are implemented with the LangChain python package for an engineering task. The results show that a supportive reasoning framework can increase the quality of responses compared to a standalone LLM. Their applicability to other engineering tasks is discussed. Finally, a perspective of task ownership is presented between the designer, the traditional software, and the Generative AI.

Keywords: Artificial Intelligence (AI), Design Cognition, Large Language Models (LLM), Generative AI, Design Automation

1 Introduction

Public knowledge of Large Language Models (LLMs), which are part of the Generative AI discipline, has surged in 2023. This increase is mainly due to the release of ChatGPT, an OpenAI application that features LLMs at its core. Due to its natural language interface, the public has extensively tested its capabilities for generating text output. The simultaneous introduction of the product, LLM, and the technology, ChatGPT or Gemini, has led to an implicit association. The public, including design engineering and aerospace researchers, often treats these two aspects in a similar way. Additionally, Microsoft has released LLM-based features to facilitate and support their software users in performing tasks. These features underscore the users' commanding position and the supportive role of these applications, referred to as co-pilots, which help offload cognitive tasks.

Products are the result of the design process, and the designer's role is central to them. Until now, there has been no widespread technology that challenges human reasoning abilities applied across general and diverse fields. For example, Suh (2021) highlights the four domains and the critical role that the designer has in mapping across them and “zigzag” between them. Meanwhile, the reasoning capabilities of LLMs are currently under debate (Bommasani, R et al., 2022), This paper hypothesizes that LLMs possess sufficient reasoning abilities to perform design engineering tasks to a level useful for designers in delegating a subset of these tasks. This hypothesis is based on the recent frameworks developed and their application to computer science activities (Park et al., 2023; Suri et al., 2023).

The approach is to evaluate the capabilities of LLM models in situations where a clear task is requested with given inputs and outputs. The level of task delegation is explored to evaluate the performance of these models. The objective is to exemplify one task and demonstrate how a concatenation of such tasks can continuously support the designer through any stage of the design process.

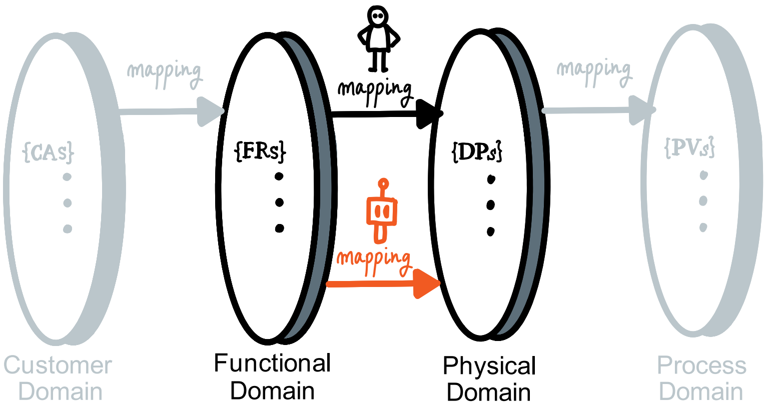

Figure 1: Positioning the paper in the design activity of mapping across design domains (Suh, 2021). Highlighted in orange is the activity of LLM based frameworks under evaluation in this paper.

Many research publications have recently appeared that evaluated the ability of ChatGPT or Gemini to perform various design engineering tasks, see, for example, Pierson and Ha (2024). Their focus is on the evaluation of a given single LLM framework (mainly chat bots). This paper, on the other hand, focuses on the state-of-the-art LLM frameworks powered by different LLM models available and on how well they perform a given task.

This paper does not focus on the ability of AI to *predict* a result given a dataset of similar input-output values. Instead, it advocates the ability for the LLMs to *imitate* the rationality abilities of the designer. Engineering companies typically lack datasets for the rationale of the different design processes: What was the problem and how it was solved in each instance. However, they have available "Method Descriptions" or "Design Practice" documents to explain how the process shall be performed. It is tentative from an engineering design support perspective to instruct these models with the input data and the process to follow, expecting that LLMs can diligently perform the steps. (Yang et al., 2024). Such a support system could be considered a *Generative Agent*, as it is based in a Generative AI. (Park et al., 2023)
The research questions guiding the research are:

- What LLM reasoning frameworks are available that are task-oriented?
- How effective are they in supporting design engineering tasks?

2 Related work

Large language models at the core are token predictors based on the transformer architecture (Vaswani et al., 2017). On their own, they are incapable of reasoning. A foundation model such as gpt-4 or Gemini (Bubeck et al., 2023; Gemini Team, 2023) needs further training and alignment to be useful *assistants* or to provide reasonable and thoughtful responses. For some models, this involves Reinforcement Learning by Human Feedback (RLHF) (Ouyang et al., 2022) that rewards responses that are useful. Additionally, RLHF punishes inherited behaviors from training data, such as harmful responses or shift model biases. The models can be further aligned to perform specific tasks, for which curated datasets need to be available. The process is called *fine-tuning*. These datasets related to the specific problem to be solved are significantly smaller than the datasets used to originally train the model, and the computational effort required is modest in comparison. If the weights and architecture of the neural network model are made public, such as Llama 2 or Mixtral (Jiang et al., 2024; Touvron et al., 2023), they can be trained and run on a local machine with sufficient memory on its GPU. Otherwise, model providers, such as OpenAI (gpt-4) or Google (Gemini), offer fine-tuning via an API service. Both avenues of injecting knowledge to LLMs, require extensive datasets, significant computational power and AI/ML domain expertise to perform it, things that are not readily available on every department in industry.

In this paper, a third avenue is explored to inject context knowledge: in context learning. It consists of providing standard models with the appropriate information in their context window. In combination with prompt engineering and flow management, it provides both the relevant information and the alignment of reasoning behavior to increase the LLM response performance on engineering design tasks. The available frameworks and implementations are described next.

2.1 Reasoning frameworks

To maximize the reasoning abilities of the models, some researchers have suggested different techniques to force the model to answer in a particular format. Some authors compare it to "forcing the model to use its System 2 thinking" (Kahneman, 2012). These techniques involve using an input - a prompt – to the model that forces a particular output. The most straightforward is to request the model to think 'step by step' (Kojima et al., 2023). Chain-of-Thought (Wei et al., 2023) proposes giving a similar example in the prompt solved in detail, which encourages the LLM to replicate the detailed explanation, ultimately increasing the chances of getting the expected reasoning. The ReAct framework (Yao et al., 2022) request the model to answer in a particular format: Thought, Reason, Action, and Observation. The paper combines chain-of-thought techniques with internal planning abilities (Alderson-Day and Fernyhough, 2015). The technique has been very popular for the last 2 years. Other techniques involve allowing the LLM to criticize their own answers, giving the model an opportunity to correct itself (Madaan et al., 2023).

A recent trend in the state of the art is to incorporate multiple agents to complement each other's reasoning (Li et al., 2023). Their interactions can be collaborative (Talebirad and Nadiri, 2023) or adversarial (Chan et al., 2023; Du et al., 2023; Liang et al., 2023). In general, they report benefits in the quality and accuracy of the responses. What is not highlighted in these papers is the computational cost that each token in the conversation costs. The effectiveness of the response increases at the expense of generating more tokens. The implementations in this paper track the token (input and output) to evaluate their impact.

2.2 Execution framework types

Four types of main execution frameworks have been identified, namely, a chat, chain, agent, and multi-agent. They all have different usage scenarios as described below.

A **chat-based model** has a collection of input and output messages passed to an LLM that is aligned to continue conversations. Optionally, they include a ‘System message’ that sets up the scene and helps the LLM follow the conversation in a particular style or additional background. This architecture is the core of chat bot applications. Recent applications have aligned the LLMs to use tools when needed, allowing the chat bot application to check for sources online, or execute code as needed.

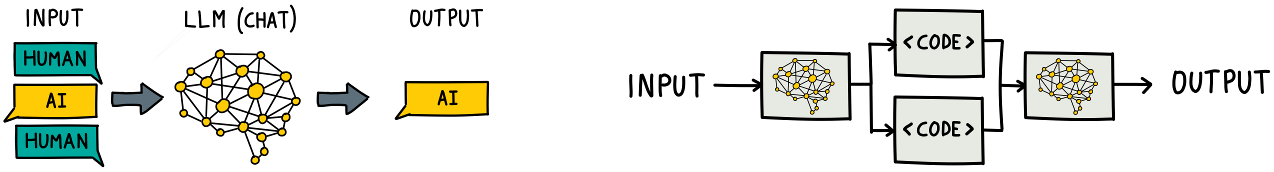

Figure 2: left: LLM Chat inputs and outputs. Right: A chain workflow combining LLMs and traditional code.

A **chain** is a predefined workflow by the user where the LLM model performs pre-agreed activities or a direct acyclic graph. The main advantages over a chat aligned LLM is that it can follow accurately a procedure. This is useful when it is a closed problem and intermediate reporting steps are expected. For example, in the verification of aerospace structures, there are methodologies or design practices to follow. For those cases, chains provide a good compromise to automate activities: The user defines the top-level workflow, and the LLM performs each of the steps independently. The workflow does not need to follow a single path; it can branch out triggered by user-defined decision criteria. This is a good automation strategy when a chat application has been used before and the user is confident that the model is capable of consistently performing. In chains, the user can combine LLM-only activities with hard-coded operations, such as functions or tools. Some of those tools can be performing calculations, update values on databases, or sending emails. Other steps can be the retrieval of context information (Retrieval Augmented Generation, RAG) for the LLM to provide a better answer at a later step in the chain. Examples of these chains are LlamaIndex. To enxend the flexibility of chains, a **graph** technique is used to define a workflow with possible iteration loops. Beside the iteration, the mayor difference between chain and graphs is that in graphs the steps are not hard-coded, and it is up to the LLM to decide what path to take depending on the input of the user, the outcome of the RAG or LLM output.

An **agent** is the combination of LLM models and the infrastructure to access tools, retrieve context information, and store memory of its interactions. They are task-oriented models. They are provided with a configuration that defines their behavior, the tools they have available, and their expected activities. The main difference from chains is that they are given freedom to achieve the overall goal. It may be suggested to follow a procedure. However, it is not guaranteed that they will. Agents are useful when the task does not require following a specific process. They can tackle more generic tasks than chains. The agent’s ability to provide a successful outcome depends on the reasoning abilities of the LLM. For this, different reasoning frameworks can be built on top of LLMs and agents. Agent framework examples besides LangChain include Baby-AGI, OpenAI GPT (formerly known as “plugins”), AutoChain, or Google’s Gemini agents.

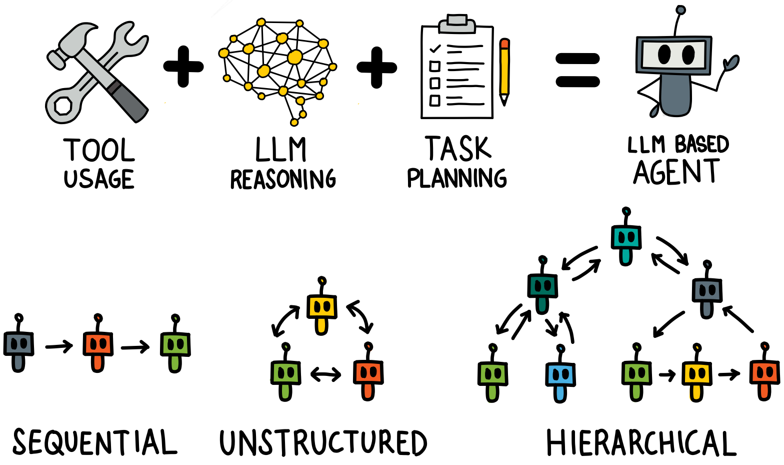

Figure 4: Top: Key components of an LLM agent. Bottom: Different multi-agent architectures

A **multi-agent** architecture is a collection of agents that interact with each other towards a common goal. According to LangChain, multi-agents are multiple actors powered by language models connected in a specific way. The connection can be defined using graphs as defined above. Connections can be considered as conversations between agents or workflow processes. They can be hierarchical, where the top agent decides who talks or performs an action, or sequential (predefined by the user). There may be a top agent controlling the different agents, or the agents may collectively decide (Li et al., 2023). The level of delegation between agents is manifold. The workflow can be strictly defined by the user, decided by the agent, or collectively decided. Workflows can also be nested with a combination of hierarchical and sequential. There is a lot of flexibility in its configuration.

The available tools to configure the multi-agent architectures are also evolving as rapidly as the publications. Given the goal of autonomous design support, we ignore GUI applications such as ChatGPT or Gemini and we focus on the frameworks that allow interactions via APIs. HuggingFace ("Hugging Face," 2024) offers automations, models and datasets to customize models, and offers hosting services for LLM applications. There is an active community built around it. For building agent architectures, LangChain is predominantly the largest ecosystem to create chat automations, chains, agents, and multi-agent architectures. There are other multi-agent architectures, such as CrewAI that builds upon LangChain, or AutoGen.

Therefore, a chat, a chain, an agent, and a multi-agent architecture will be explored in this paper.

3 Methodology

To answer Research Question 1 (*What LLM frameworks are available that are task-oriented?*), a literature review has been conducted on both design engineering and generative AI fields. The answer to Research Question 2 (*How effective are they in supporting design engineering tasks?*) was performed with the implementation of the different LLM reasoning frameworks and their respective evaluations.

3.1 Literature Methodology

For the application of LLM on engineering and design tasks, the Scopus database was queried on the following topics that appear in the title or abstract:

OR:		**OR:**		**OR:**
Application		Large Language Models		Engineering
Evaluation	**AND**	LLM	**AND**	Design
Usage		ChatGPT, Bard		Aerospace
Comparison		GPT-4, PalM, Gemini, Llama2, Mixtral		Automotive

The first two columns were used to search on article title, abstract, and keywords, returning 2,867 documents. Notably, the majority corresponds to the 2021-2024 period. The third column had to be added to keep the evaluations within the domain of Engineering Design, and limit it to the Engineering subject area returning 616 documents. The documents were filtered only on title, resulting in 33 documents. Most of the documents referred to the acronym "LLM" having a different meaning for other disciplines.

To investigate the state-of-the-art frameworks, an additional database was used: ArXiv.org. The authors are aware that the articles published in this database are not reviewed by independent sources, potentially compromising their quality. However, they are relevant because they are the primary publication database for the field of generative AI. All major players, such as OpenAI, Meta, Google, or Microsoft choose arXiv.org to publish their findings. Academic players have also followed this approach, optionally publishing journals or conferences as a posterior activity. For that reason, papers found there are also searched for by author and date on Scopus in case a peer-reviewed source exists. To compensate for that lack of rigor, number of citations (tracked by Semantic Schoolar) was used to determine the popularity and related to that, the relevance of each paper. Keywords used are LLM Agent, Architecture, Reasoning, Cognition, Framework.

3.2 LLM performance evaluation methodology

The task of evaluating the performance of the LLM was selected based on the following rationale.

- It is a task where a clear assessment of success can be defined: Correct or incorrect
- The task is a process that is currently done manually, without a simple AI or automatic alternative solution.
- The complexity of the task makes it realistic for an LLM, while keeping the input size within margins for the available LLMs.
- It was a task that the authors were familiar with (Pradas Gómez et al., 2023).

Details of the design task are provided in Section 4. Other than the reasons above, the design task could be any and it is encouraged for the reader to think on a design task where this could be applied, as the intention of the paper is to inspire the reader to consider other applications.

Research Question 2: How effective are they in supporting design engineering tasks? – uses the term *effective*. The working definition of efficiency in this study is the percentage of correct answers that the model completes, given exactly the same input over a number of n=10 trials.

The LLMs used for the evaluation have a significant effect on the effectiveness result. They are affected by factors such as model size (parameters), training data size (tokens), training data type and quality, epochs during training, or human reinforcement alignment. Therefore, to take into account the inter-model capabilities, different model sizes and generations are examined. The models used are gpt-3.5-turbo-0125 (175b parameters) and gpt-4-0125-preview (state-of-the-art, parameter count not published), both from OpenAI.

The evaluation of Chat LLMs can be done manually, as most of the papers on engineering design have done up to date. However, for more complex frameworks, the interactions and steps are automated. We have selected the python package LangChain as it has the basic components to build the LLM frameworks. In addition, it can track token consumption and intermediate calls to LLMs using the complementary LangSmith service. The tracking of the LLM calls makes it useful for debugging the application. In addition, it allows one to easily inspect the internal model reasoning in LLM Based Agents, or the conversations between Agents in a Multi-Agent Framework. Finally, this framework allows to call the same architecture repetitively to extract minimum, average, and maximum effectiveness values for each configuration.

4 Engineering Task Problem Description

During the embodiment of a conceptual design, the functional model must be realized in a physical form. Aerospace structural components have a highly integrated product architecture (Ulrich, 1995). The activity of connecting the functional to physical domains in the design solution is not a one-to-one mapping exercise. If the designer is using a function-means (Hubka and Eder, 1988) or Enhanced Function-Means (Schachinger and Johannesson, 2000) the tree architecture has the purpose of exploring the different means to fulfill a functional requirement, for which the tree is structured around the cascading decisions that the designer makes. However, physical modeling of the product can be performed in a CAD assembly or a Knowledge-Based Engineering (KBE) application, to name a few. The architecture of the physical modeling is based on practicalities such as physical similarity, external interfaces, or manufacturing considerations. Therefore, there is a conceptual association that the designer needs to perform between two very different trees.

The goal of LLM frameworks is to map the objects in the functional and physical domains. The design features are modeled as design solutions (DS) in an enhanced function means, while the same features are modelled on the physical domain as primitive classes in the source code of a KBE library. In addition, these design feature objects contain variables that need to be modified and are declared in the functional domain. The actual product to be studied is a Turbine Rear Structure (TRS), described in detail by Pradas et al., (2023). Without the support of an LLM, the design task would be as follows: a designer would find a variable in EF-M DS "*Outer Case*" named "*thickness*" with the value range *(2,3)*. The designer would look at the KBE code structure and use her judgment to associate the design solution with the appropriate primitive. Then, she would find that the path of the variable should be `trs.outer_case.thickness`; and finally, she would enrich the variable metadata by adding this path to the `kbe_path` property.

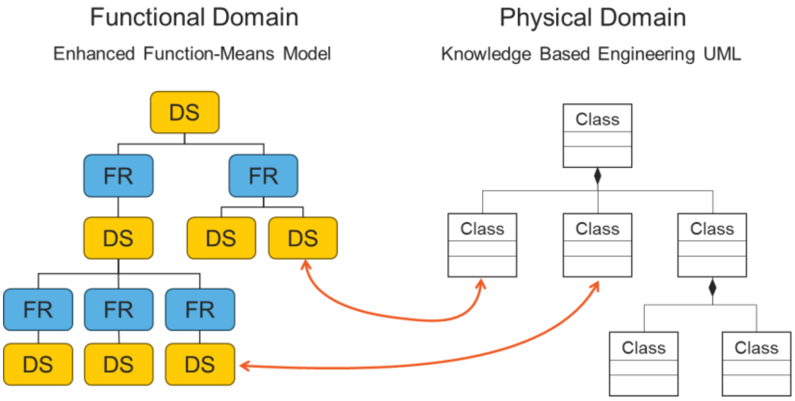

Figure 5: Function Modeling to the left and Physical model to the right. The arrow between models shows the mapping that the designer needs to perform to connect both domains. This mapping is the goal of the LLM in this paper.

4.1 Task description: inputs and expected outputs

Instructions are passed to the model to: (1) identify the design solutions (DS) in the EF-M tree that had variables (all variables need to be mapped to the KBE model), and then (2) to provide the path of those variables in the KBE application.

In addition, the results are requested to be reported in a tabular format. The EF-M model contains in total five variables that the model needs to find and then map to a KBE map. The correct result, performed manually, is presented in Table 2.

Table 2: Expected solution for the task. This information is not known to the LLM based solution. It is used to assess the correctness of the result and therefore the success.

Extracted information contained in functional model			**Expected (correct) solution**
Design Solution	**Variable Name**	**Variable Value**	**Variable path in KBE application**
Outer Case	thickness	(2,3)	trs.outer_case.thickness
Vane	vane_count	[6,7,8,9,10,11,12]	trs.number_of_vanes
Vane	thickness	(1,3)	trs.vane.vane_thickness
Single Lug	lug_type	“single”	trs.outer_case.lug_type
Double Lug	lug_type	“double”	trs.outer_case.lug_type

The EF-M model is provided in a standardized JSON format. In addition, the physical model is provided as the source code of a developed KBE application, including the source code of the library of KBE primitives. For the chat interface, both inputs are wrapped in a text prompt, together with the task description and two examples. For the agent architectures, the agent was given a tool to access a library of KBE applications, and the EF-M was passed as part of the prompt.

5 Results

5.1 Literature Results

The literature review shows that the 33 identified papers performed an evaluation of the capabilities of large language models using exclusively a chat application (predominantly ChatGPT) . A random selection of 10 more papers in the 616 unfiltered papers also confirms this trend. Therefore, there is a research gap in the application of the other frameworks – especially agent and multi-agent architectures – in design engineering tasks.

5.2 Results of the framework evaluation

5.2.1 Chat models

A system message to configure the model was given to the model instructing on the overall task and describing the inputs and expected output and format, including a one-shot example. All required information was passed within the prompt.

Table 3: Success rate results for the chat architecture

Model	**User suggests intermediate steps**	**Success rate**
gpt-3.5-turbo-0125	No	0%
gpt-3.5-turbo-0125	Yes	20%
gpt-4-0125-preview	No	10%
gpt-4-0125-preview	Yes	30%

The results indicated a preference for complicated tasks to “suggest the model” intermediate steps. Sometimes they were followed, sometimes not. This led to the next approach, to strictly specify intermediate steps and outcomes with chains.

5.2.2 Chains

A sequential list of activities were specified by the designer, and then some of those were passed to a chat LLM. LangChain modules controlled the execution and the transfer of information between each of the steps. The models used the OpenAI “tool calling” functionality, which enforced a certain output format, which was parsed and sent to the next step.

Table 4: Success rate results for the chain architecture

step	**Description**	**gpt-3.5**	**gpt-4**
1	Extract variables from E-FM tree	100%	100%
2	Create the class hierarchy from KBE application code	90%	100%
3	Expand class hierarchy with library code	100%	90%
4	Connect EF-M variables with class inputs	67%	100%
	Total Success rate	**60%**	**90%**

This approach requires a higher level of programming experience. In return, it forces the model to comply with a response format. Combining it with a user-driven task decomposition, this approach has proven to be very effective for both models, is simple tasks, like steps 1-3. Step 4 highlighted the challenges of gpt-3.5 sized models in complex reasoning tasks.

5.2.3 Agents

The Agent model was implemented using the LangChain OpenAI Tool scheme. In the previous implementations, the EF-M tree definition, KBE application, and library are passed directly as raw data in the prompt. To showcase the capabilities of agents, the file names were mentioned in the prompt, and the ability to look in a directory was given as a tool to the model. In the intermediate steps, the models correctly retrieved the content of the files. However, there was a major difference between the effectiveness of the models, showing how larger models outperform medium-sized models.

Table 5: Model performance comparison for the agent architecture

	gpt-3.5-turbo-0125	gpt-4-0125-preview
Success rate	0%	90%

The verbose and conditioning of the agents required around 9k tokens, making this method unfeasible for models with context windows below this value, such as Llama 2.

5.2.4 Multi-Agents:

In CrewAi and LangChain, a hierarchical agent architecture was configured in CrewAi and LangChain with 3 roles: task manager, engineer, and checker. The role of the task manager is to use the engineer and checker as needed to solve the mapping of the variables in the EF-M to the KBE application. The engineer proposes solutions, and the checker ensures that the solutions are correct. Up to 4 loops of feedback were allowed between the checker and the engineer.

Table 6 Model performance comparison for the multi-agent architecture

	gpt-3.5-turbo-0125	gpt-4-0125-preview
Success rate	90%	100%

The table below summarizes and compares the performance of each architecture.

Table 7: Comparison of the overall framework

LLM Approach	Success rate (n=10)		Average tokens used	Instruction level from user	Adherence to plan
	gpt-3.5	gpt-4			
Chat	20%	30%	9,000	Low / High	Low
Chain	60%	90%	9,000	High	High
Agent	0%	90%	9,000	Low	Low
Multi-Agent	90%	100%	15,000 per loop.	Low or High	High

6 Discussion

In this chapter, the results of the LLM implementation are discussed. In addition, its significance to the design process is explored in the second part.

6.1 Result Discussions

Chat frameworks on their own are highly unreliable (0% to 30% success rate). Without tools to verify facts or generate calculations, all they can do is relate to the information used in their training, which may not be applicable to the task at hand. They are good at understanding basic user intent, and a low development effort is required.

Chains require programming to be set up. They require to be developed for a specific application and also require the developer to break down the problem sequentially, instead of letting the LLM reason about it. They do not have the possibility to iterate independently, but they significantly improve the effectiveness to 60% (gpt-3.5) or 90% (gpt-4). However, it is below the desired 100% effectiveness for most design engineering applications. The authors believe that the percentage of success could be further improved with better implementation and developer experience.

The agent architecture performs similarly on gpt-4 compared to chains, but drops on gpt-3. The advantage over chains is that the steps are not predefined by a human, so the flexibility is greater. In addition, its capability to call external tools is a step forward in avoiding hallucinations and to interface with other software systems. On the other hand, agents do not always comply with instructions. There are distinctive use cases for both. Finally, the difference in performance between gpt-3.5 and 4 (0% vs. 90%) indicates that the reasoning capability of the LLM has a large influence on the agent performance.

Multi-Agents is a workflow abstraction that can be very powerful. They increase the effectiveness of the models (90% for gpt-3 and 100% for gpt-4) A detailed comparison of all the possible sub-architectures and task delegation levels has not been explored. This architecture may be useful when it is desired to have a more robust response. Robustness can be measured in many ways: Is the response complete? Has it considered certain angles? Does it satisfy all checklists? There can be an agent in the framework that advocates for each perspective. However, this comes at a cost, mainly due to the amount of tokens needed to be generated. Each token has a computational cost that affects response time, computer energy consumption, and fills the context window. At this stage, the authors only recommend this architecture for enhancing the reasoning capabilities when standalone agents are not able to successfully perform the task. Their framework is constructed on many levels, each containing many parameters that are very sensitive to an effective response. E.g. the LLM settings (temperature, repetition), The agent tools description, reasoning framework, multi-agent architecture, model type, fine-tuning of the model, access to relevant databases, quality of the referenced data, etc. The context window limitation of 32k token may be a limiting factor for some models. In the most powerful models, context windows tend to be increased (e.g., Claude 3 claims now a context window of hundreds of thousands of tokens without decreasing attention), but in locally run models, these context windows are today limiting factors for multi-agent architectures.

This multi-agent architecture successfully manages to capture the specialization of team members. Engineering design is normally conducted by a multidisciplinary design team of humas, each specialized in their domain. While LLMs are useful, a general-purpose LLM capable of replacing or supporting such multidisciplinary teams could be very challenging to achieve on its own. Therefore, a multi-agent architecture may be appropriate when modeling the team behavior, as we expand in the next section.

As discussed previously the selection of this engineering design task was based on the previous research of the authors. The performance of the four LLM architectures is expected to be equivalent for similar difficulty tasks. For easier tasks, a simple chat may be sufficient, and for more complex tasks where the models are unable to identify the logic, a multi-agent debate may be needed to enhance the performance.

6.2 Wider implications

LLM Agent framework architectures are an active research topic in the computer science community. No dominant approach has yet emerged. A combination of Chains, Agents and Multi-Agents may be able to concatenate to perform a significant percentage of the design activities, but further research is required to quantify the effectiveness of the design support.

The multi-level architecture shows that medium models as GPT-3.5 can have a significant improvement on reasoning when they are checked externally, and they can try again. It opens up to have similar results with open-source, smaller models such as Mixtral, which perform similarly on other models. This is significant as the models can be run within the premises of the companies, and no customer or confidential data have to be sent to the LLM providers. Furthermore, the license is sufficiently permissive to be used by companies.

It is tempting to support models by “stepping in” when they fail in simple tasks. Patching a step where they fail with explicit instructions or code, like in the chain example. There are scenarios where it is useful to force the model to follow certain logic and generate evidence. For example, in the detailed analysis of complex aerospace products. Agents and multi-agent frameworks could be useful in the near term. However, despite the improvement in sort-term effectiveness, it is likely that bigger models trained with more data will outperform these frameworks to improve reasoning capabilities (Sutton, 2019).

How can these architectures be useful in the long term? First, they leave a record of their decision-making rationale, something that is expensive to obtain in the engineering design community. We could use these synthetic data to evaluate and curate a list of good design rationale. This dataset could later be feed to the next-generation models to generate better decisions, without the need for frameworks on top of it. The key difference between this proposed dataset and traditional machine learning datasets is that it contains the rationale for following a certain design process, as opposed to a dataset of product properties. The former approach helps the designer think how to design; the second approach tells the designer how the product could look like, given a set of input parameters.

The LLM multi-agent architecture opens a new way of modeling the behaviors of design engineering teams and activities. The dynamics of a multi-agent hierarchical architecture reminds us of an engineering team for an aerospace design office: A team lead (Engineer in Charge) coordinates with different discipline leads (design definition, solid mechanics, aerodynamics, etc.) who in turn have different engineers to support the tasks. Each engineer in the organization has a purpose and a specialization, with access to different tools to give a piece of the puzzle to create a compliant design. The way the agents explicitly express their goals and thoughts, and the way the interactions occur based on the output of each task can be very realistic. With such models of an engineering organization, design researchers have now an additional means to test novel methodologies before a final validation in a real company.

The development of LLM based applications has also unique characteristics at this early maturity stage. It is not clear what part of the process shall be left to the engineer to define and what part of the process shall be left to the traditional code (traditional or built as a chain) or up to the LLM as an Agent. There will be a level of shared responsibility between the three: Engineer, hard coded software and LLM reasoning.

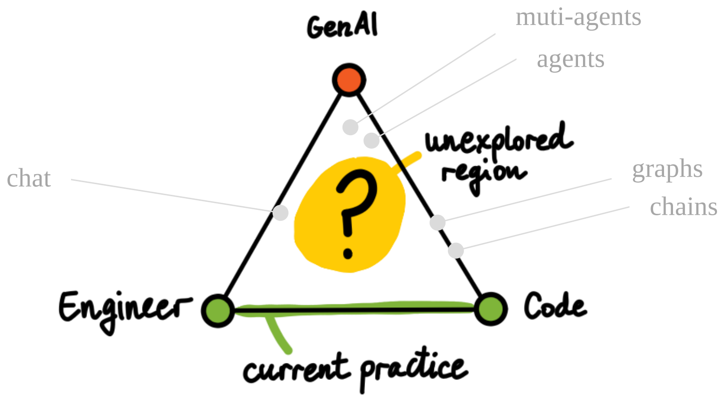

Figure 6: The triangle of shared responsibility that has been created when GenAI and LLMs are making design decisions. The frameworks used in this paper (in gray) are placed in this space taking into account how much design ownership they are given.

Finally, we found that the performance of the LLM application is highly dependent on:

1. How the task has been defined by the application developer
2. The exact prompt used to elicit a response, including examples and system messages.
3. The minor version updates of the large language models
4. The hyperparameters used in the large language models

These observations are in line with the findings of other researchers (Reitenbach et al., 2024; Singhvi et al., 2024).

7 Conclusions

The literature review on reasoning agents in the engineering design research community has shown a gap in the application of LLM reasoning frameworks. On the other hand, the large language model community is quickly and actively researching different means to get LLMs smarter, including building reasoning frameworks. The implementation of different frameworks on a proposed design task revealed that each one have advantages and disadvantages. In terms of effectiveness, using a multi-agent framework significantly increases the response accuracy. Frameworks are sensitive to many parameters, and more development is needed in the architectures to ensure reliability on a wide range of tasks. However, they may have specific applications within certain automation activities, and they open up the door to get better design rationale data through the recording of their interactions. Finally, the triangle of shared ownership: Designer, Generative AI (LLM) and traditional hard coding needs to be further researched to understand how to best benefit from this technology.

References

Alderson-Day, B., Fernyhough, C., 2015. Inner speech: development, cognitive functions, phenomenology, and neurobiology. Psychological bulletin 141, 931.

Bommasani, R et al., 2022. On the Opportunities and Risks of Foundation Models. https://doi.org/10.48550/arXiv.2108.07258

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M.T., Zhang, Y., 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. https://doi.org/10.48550/arXiv.2303.12712

Chan, C.-M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., Liu, Z., 2023. ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. https://doi.org/10.48550/arXiv.2308.07201

Du, Y., Li, S., Torralba, A., Tenenbaum, J.B., Mordatch, I., 2023. Improving Factuality and Reasoning in Language Models through Multiagent Debate. https://doi.org/10.48550/arXiv.2305.14325

Gemini Team, 2023. Gemini: A Family of Highly Capable Multimodal Models.

Hubka, V., Eder, W.E., 1988. Theory of Technical Systems: A Total Concept Theory for Engineering Design. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-52121-8

Hugging Face [WWW Document], 2024. URL https://huggingface.co/ (accessed 2.14.24).

Jiang, A.Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D.S., Casas, D. de las, Hanna, E.B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L.R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T.L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., Sayed, W.E., 2024. Mixtral of Experts. https://doi.org/10.48550/arXiv.2401.04088

Kahneman, D., 2012. Thinking, fast and slow, Penguin psychology. Penguin Books, London.

Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y., 2023. Large Language Models are Zero-Shot Reasoners. https://doi.org/10.48550/arXiv.2205.11916

Li, G., Hammoud, H.A.A.K., Itani, H., Khizbullin, D., Ghanem, B., 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. https://doi.org/10.48550/arXiv.2303.17760

Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Tu, Z., Shi, S., 2023. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate. https://doi.org/10.48550/arXiv.2305.19118

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P., 2023. Self-Refine: Iterative Refinement with Self-Feedback. https://doi.org/10.48550/arXiv.2303.17651

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., Lowe, R., 2022. Training language models to follow instructions with human feedback. https://doi.org/10.48550/arXiv.2203.02155

Park, J.S., O'Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., Bernstein, M.S., 2023. Generative Agents: Interactive Simulacra of Human Behavior. https://doi.org/10.48550/arXiv.2304.03442

Pierson, K.C., Ha, M.J., 2024. Usage of ChatGPT for Engineering Design and Analysis Tool Development, in: AIAA SCITECH 2024 Forum. Presented at the AIAA SCITECH 2024 Forum, American Institute of Aeronautics and Astronautics, Orlando, FL. https://doi.org/10.2514/6.2024-0914

Pradas Gómez, A., Panarotto, M., Isaksson, O., 2023. Design automation strategies for aerospace components during conceptual design phases, in: Aerospace Europe Conference 2023–10TH EUCASS–9TH CEAS, Lausanne. https://doi.org/10.13009/EUCASS2023-578

Reitenbach, S., Siggel, M., Bolemant, M., 2024. Enhanced Workflow Management using an Artificial Intelligence ChatBot, in: AIAA SCITECH 2024 Forum. Presented at the AIAA SCITECH 2024 Forum, American Institute of Aeronautics and Astronautics, Orlando, FL. https://doi.org/10.2514/6.2024-0917

Schachinger, P., Johannesson, H.L., 2000. Computer modelling of design specifications. Journal of Engineering Design 11, 317–329. https://doi.org/10.1080/0954482001000935

Singhvi, A., Shetty, M., Tan, S., Potts, C., Sen, K., Zaharia, M., Khattab, O., 2024. DSPy Assertions: Computational Constraints for Self-Refining Language Model Pipelines. https://doi.org/10.48550/arXiv.2312.13382

Suh, N.P., 2021. Introduction to Design, in: Suh, N.P., Cavique, M., Foley, J.T. (Eds.), Design Engineering and Science. Springer International Publishing, Cham, pp. 1–33. https://doi.org/10.1007/978-3-030-49232-8_1

Suri, S., Das, S.N., Singi, K., Dey, K., Sharma, V.S., Kaulgud, V., 2023. Software Engineering Using Autonomous Agents: Are We There Yet?, in: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). Presented at the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, Luxembourg, Luxembourg, pp. 1855–1857. https://doi.org/10.1109/ASE56229.2023.00174

Sutton, R., 2019. The bitter lesson. URL https://www.cs.utexas.edu/~eunsol/courses/data/bitter_lesson.pdf (accessed 2.15.24).

Talebirad, Y., Nadiri, A., 2023. Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents. https://doi.org/10.48550/arXiv.2306.03314

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C.C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P.S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E.M., Subramanian, R., Tan, X.E., Tang, B., Taylor, R., Williams, A., Kuan, J.X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., Scialom, T., 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. https://doi.org/10.48550/arXiv.2307.09288

Ulrich, K., 1995. The role of product architecture in the manufacturing firm. Research Policy 24, 419–440. https://doi.org/10.1016/0048-7333(94)00775-3

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is All you Need, in: Advances in Neural Information Processing Systems. Curran Associates, Inc.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D., 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. https://doi.org/10.48550/arXiv.2201.11903

Yang, K., Liu, J., Wu, J., Yang, C., Fung, Y.R., Li, S., Huang, Z., Cao, X., Wang, X., Wang, Y., Ji, H., Zhai, C., 2024. If LLM Is the Wizard, Then Code Is the Wand: A Survey on How Code Empowers Large Language Models to Serve as Intelligent Agents. https://doi.org/10.48550/arXiv.2401.00812

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y., 2022. ReAct: Synergizing Reasoning and Acting in Language Models. https://doi.org/10.48550/arXiv.2210.0362

Contact: Pradas Gomez, Alejandro, Chalmers University of Technology, alejandro.pradas@chalmers.se