Petter Krus

Fluid and Mechatronic Systems, Linköping University

Concept Generation and Modelling with Large Language Models

INTRODUCTION

- Large Language Models (LLMs) and AI Democratization: The advent of ChatGPT at the end of 2022 marked a significant milestone in making artificial intelligence accessible to a broader audience. This development has opened up new research opportunities, particularly in applying AI to various domains.
- Objective of the Paper: This presentation presents the use of ChatGPT 4o, a large language model, in the engineering design process. The primary focus is on automating the configuration of e.g. propulsion systems and actuation systems using ChatGPT.
- Case Study: The paper uses the configuration of aircraft hybrid propulsion systems as an example. Demonstrating how system configurations can be generated and represented as UML component diagrams.
- Challenges and Solutions: LLMs, including ChatGPT, are inherently non-deterministic, leading to variability in outputs. The paper explores methods to increase the probability of correct results and optimize prompts for better outcomes.
- Significance: The study shows promising results, indicating that LLMs have immense potential invarious aspects of engineering system design.

LARGE LANGUAGE MODELS AS BLACK BOXES

- The application of LLM:s has little to do with their internal workings. AI-scientists does not much of an edge in the applications of LLM over the domain expert.
- The approach to research of LLM application is maybe more like social science.

Generation of Architectures

- Prompt:"Generate a hybrid electric aircraft system with a fuel cell. Show the result as code for a UML-component diagram in PlantUML"
- The diagram is very basic and not entirely correct..

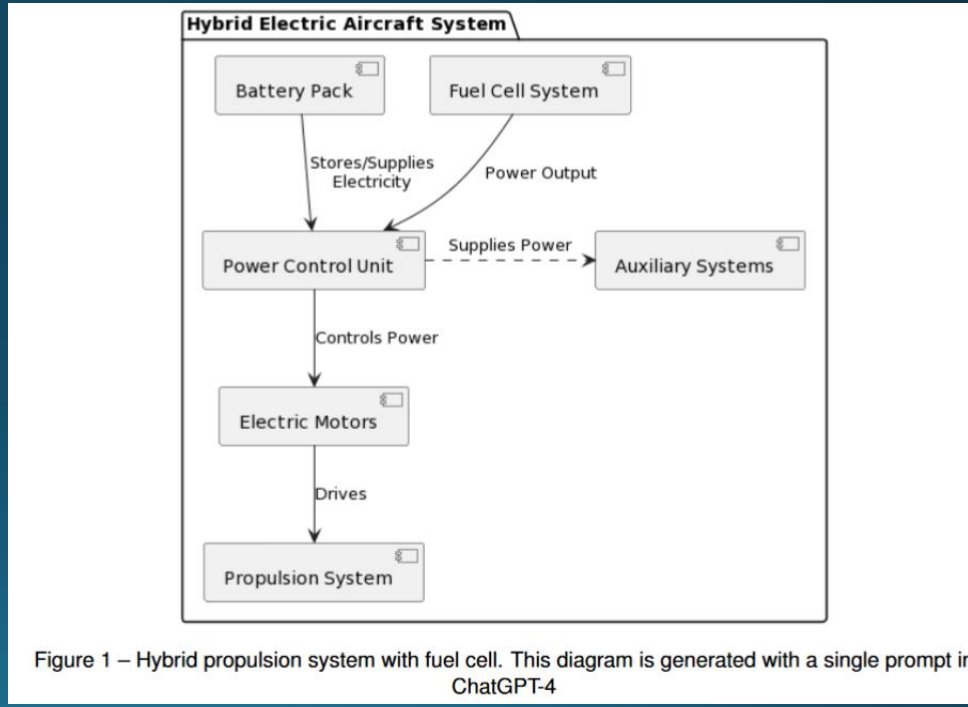
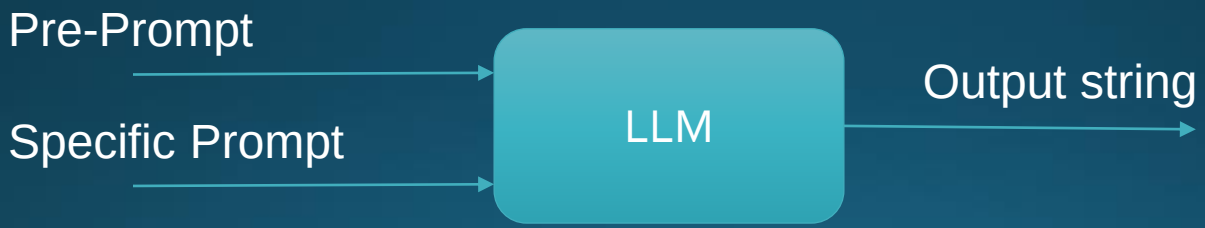


Figure 1 – Hybrid propulsion system with fuel cell. This diagram is generated with a single prompt in ChatGPT-4

LARGE LANGUAGE MODELS AS BLACK BOXES

- The prompt can be structured to have a general pre-defined prompt the can be reused, and one specific part.
- The pre-prompt can also contain documents and this can also be used to create a Custom GPT that can be saved for later use.

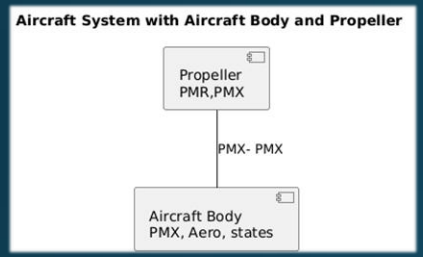


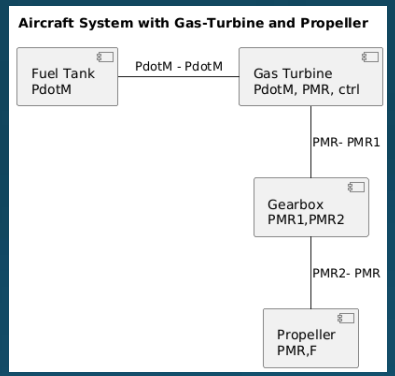
PRE DEFINED PROMPT WITH MICRO-TEMPLATES

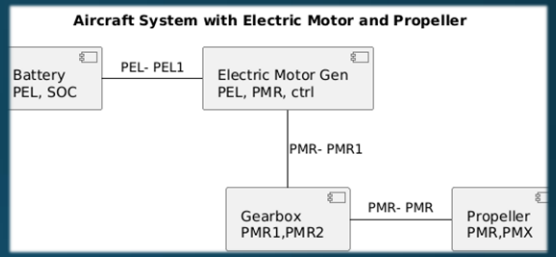
```
# Aircraft Propulsion System Configuration

This section details the standard configuration for connecting components
in an aircraft propulsion system, focusing on hybrid systems that
incorporate gas turbines, fuel cells, batteries, and electric machines.

## Component Descriptions

- **Gas Turbine**: Converts chemical power (\( \dot{P}_M \)) from a fuel
  source (Fuel Tank or Hydrogen Storage) into rotational power (MR). The
  MR port can only be connected to one component.

- **Fuel Cell**: Converts chemical power (\( \dot{P}_M \)) from Hydrogen
  Storage into electric power (EL).

- **Propeller**: Converts rotational power into linear power, i.e., thrust.

- **Gearbox**: Connects one or more rotational power sources (MR) to a
  single output rotational power (MR).

- **Electric Motor/Generator**: Converts electric power (EL) from a Battery
  into mechanical rotational power (MR) or vice versa. The MR port can only
  be connected to one component.

- **Battery Pack**: A storage unit that provides electric power (EL). It
  can have multiple connections.

## Component Connection Principles

- **Port Matching**: Ensure that ports on each component are connected
  to ports of a similar type on other components. For example, electric
  power ports (EL) should connect to other EL ports, while mass flow
  ports (\( \dot{P}_M \)) should connect to other \( \dot{P}_M \) ports.
  Only include components that are used.

- **Dashed Connections**: When using dashed lines ("-[dashed]-"), do not
```

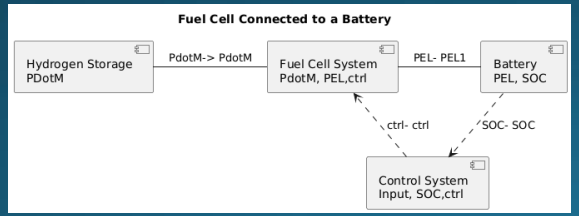
```
## Micro Templates

Here are some micro templates that show the style of the UML diagram.

### Propeller Connected to an Aircraft

\```plantuml
@startuml
' Define the components with ports
component "Aircraft Body\nPMX, Aero, states" as AircraftBody
component "Propeller\nPMR,PMX" as Propeller
' Connections
Propeller -down- AircraftBody: PMX- PMX
' Diagram Title
title Aircraft System with Aircraft Body and Propeller
@enduml
\```

### Connection Between a Battery and a Motor

\```plantuml
@startuml
' Define the components with ports
component "Battery\nPEL, SOC" as Battery
component "Electric Motor Gen\nPEL, PMR, ctrl" as ElMotorGen
component "Propeller\nPMR,PMX" as Propeller
component "Gearbox\nPMR1,PMR2" as Gearbox
' Connections
Battery -right- ElMotorGen: PEL- PEL1
ElMotorGen -down- Gearbox: PMR- PMR1
Gearbox -right- Propeller: PMR- PMR
' Diagram Title
title Aircraft System with Electric Motor and Propeller
@enduml
\```
```

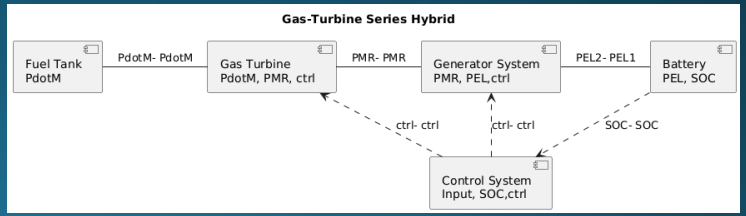
Micro Templates in the Pre-Prompt



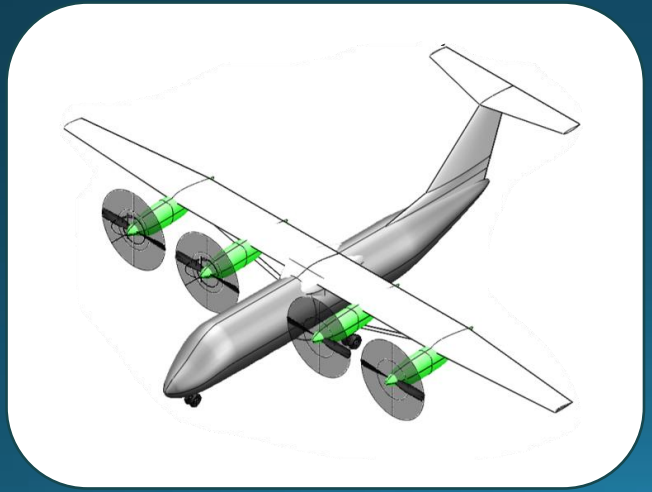


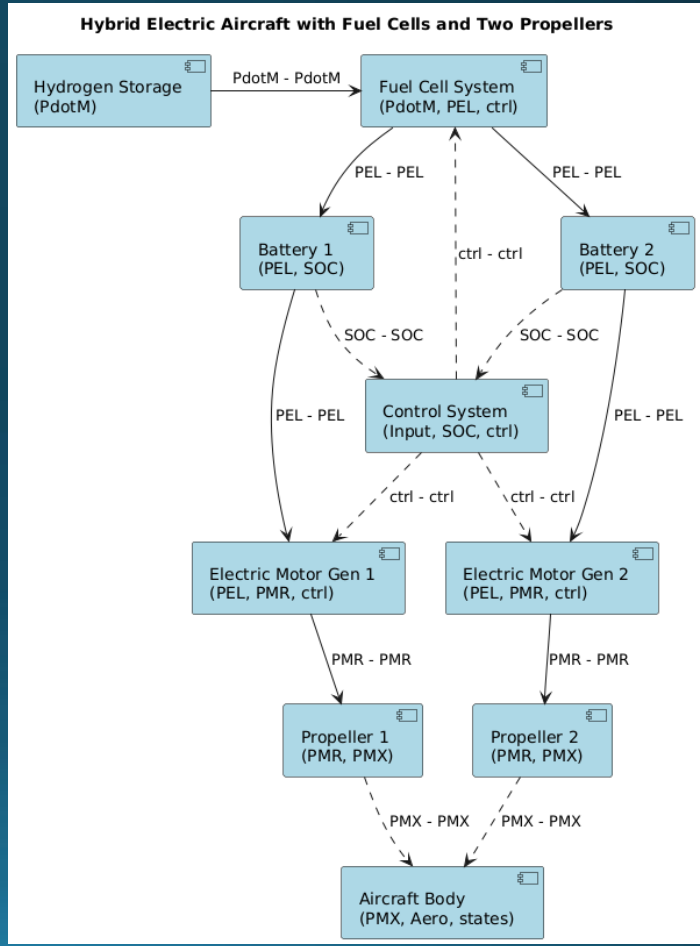






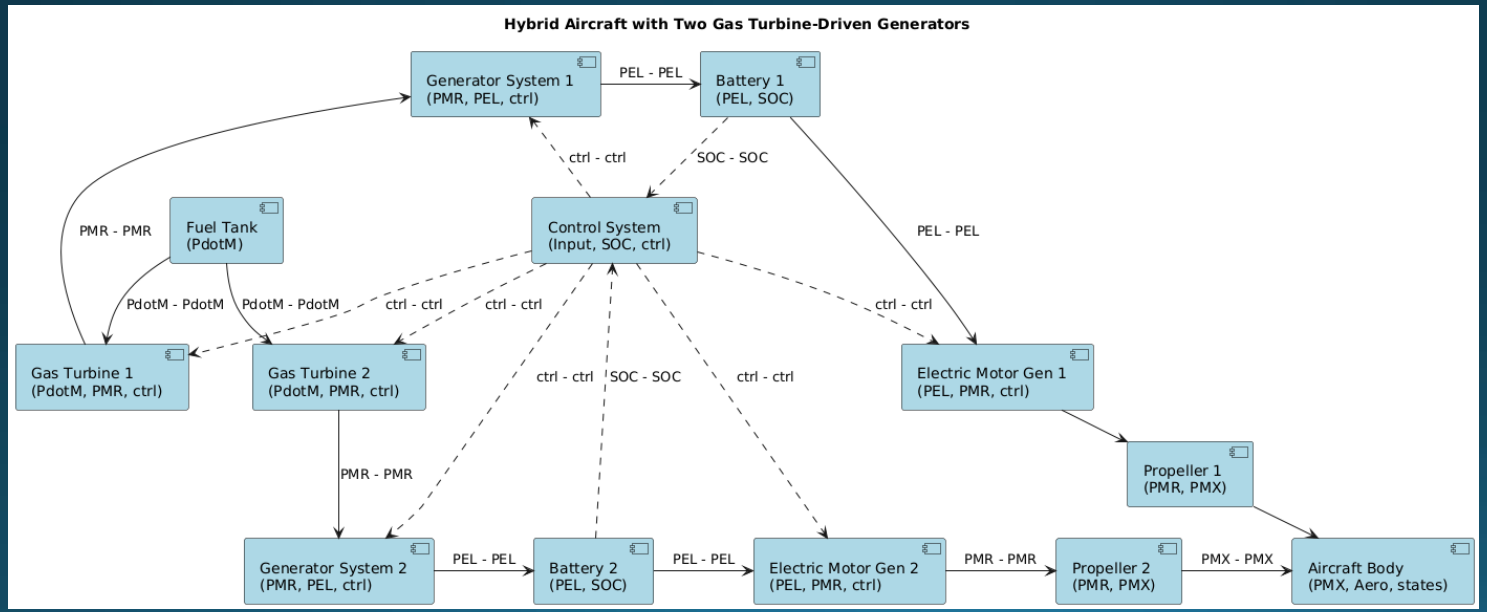
- Prompt: "Generate a Concept for a Hybrid Electric Aircraft with Two Propellers and Fuel Cells."





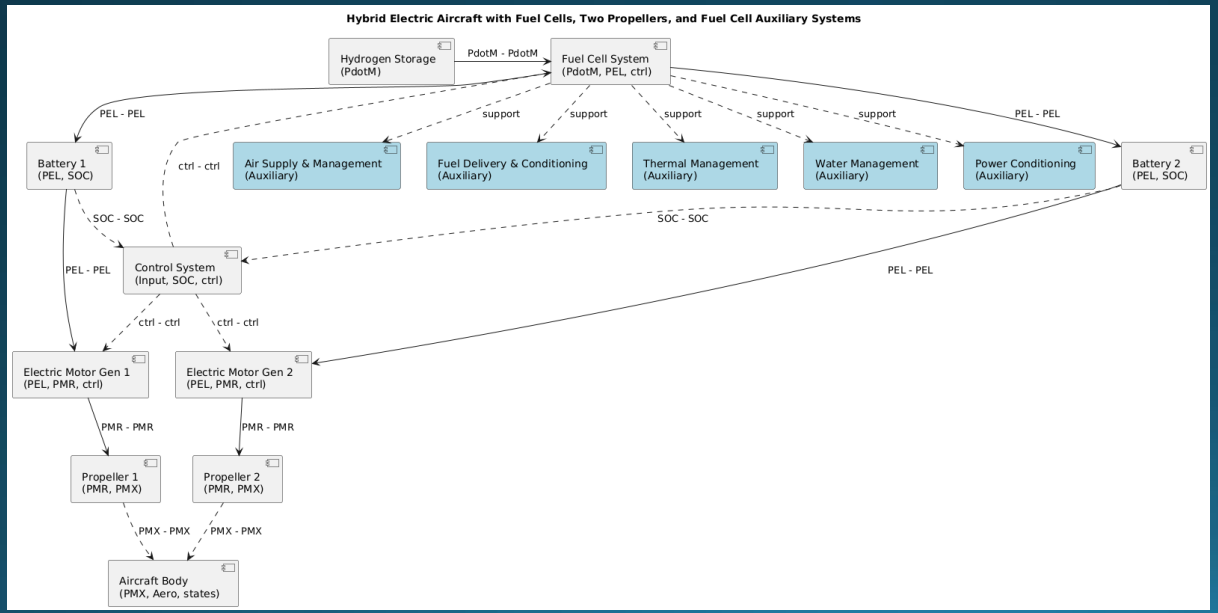
Variations

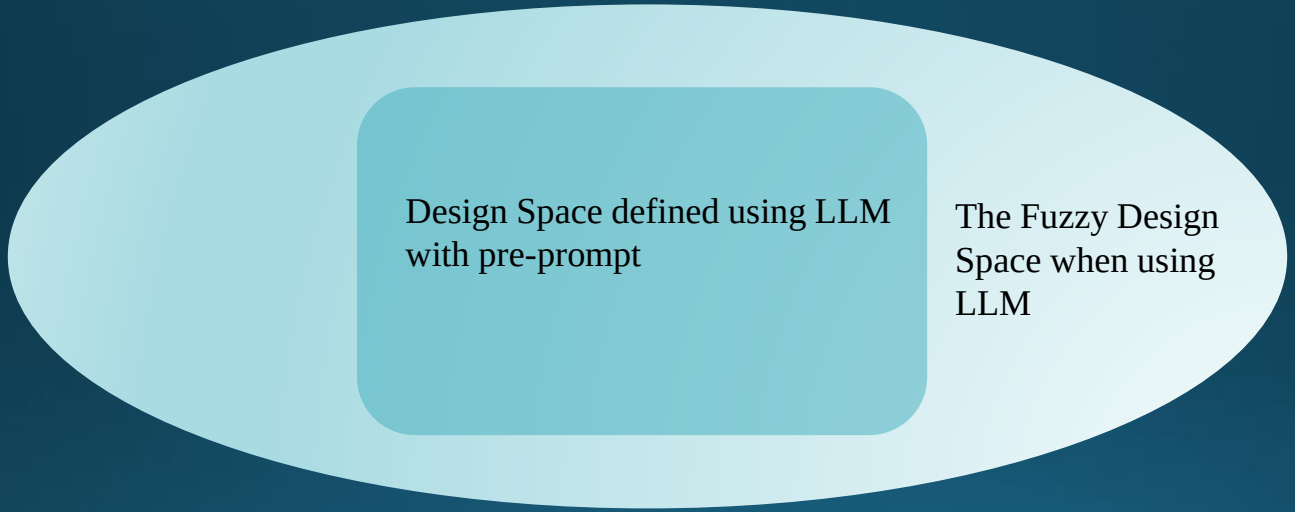
- Prompt: "Generate a system with two gas turbine driven generators"



EXPANDING THE SYSTEM BEYOND THE PRE-PROMPT

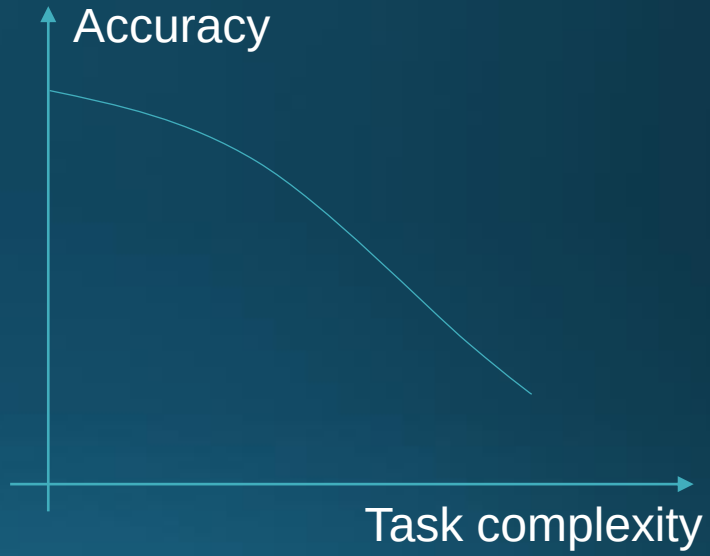
- Prompting to ad auxiliary systems for the fuel-cell (This information was not in the pre prompt)




Design Space defined using LLM with pre-prompt
The Fuzzy Design Space when using LLM

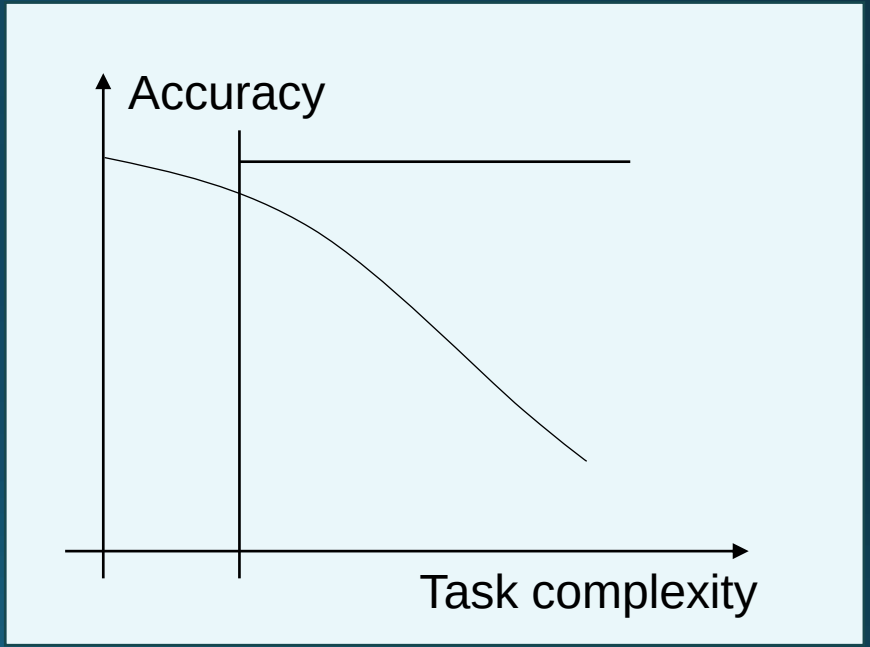
Scaling Property of LLM

- The scaling property of LLMs have to be considered
- Just because a simple system can be generated there is no guarantee a more complex can be generated



Scaling Property of LLM

- Use the lower end of the complexity to make a program that can generate a more "complex" system along some degrees of freedom



From Configuration Rules to Python Code: A Process Outline

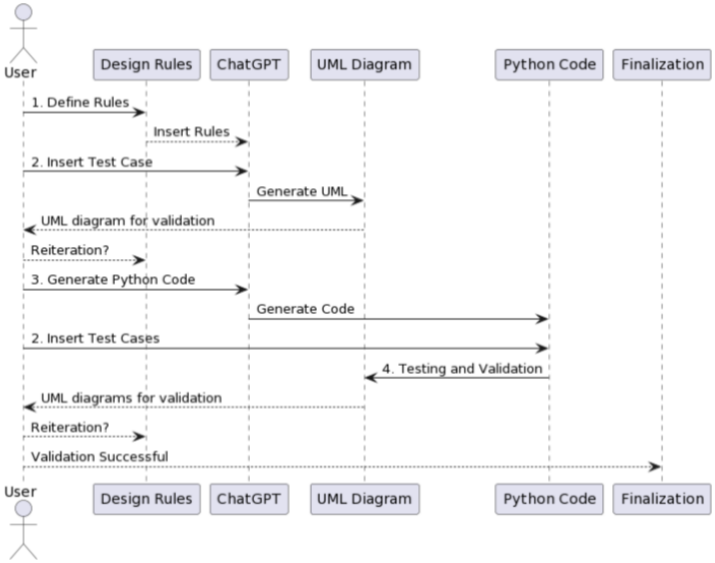
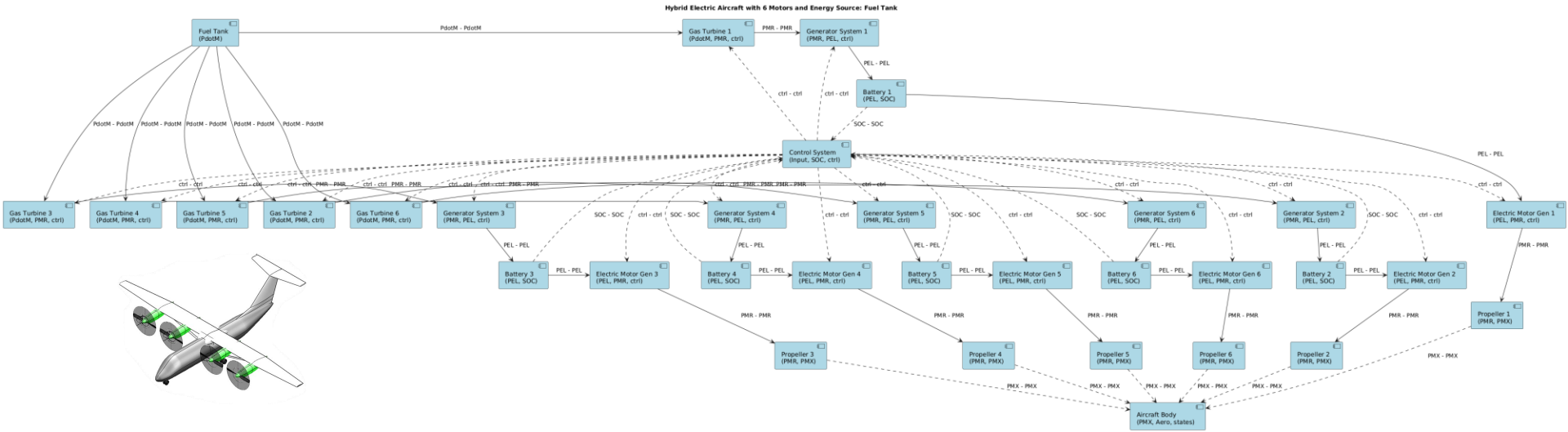


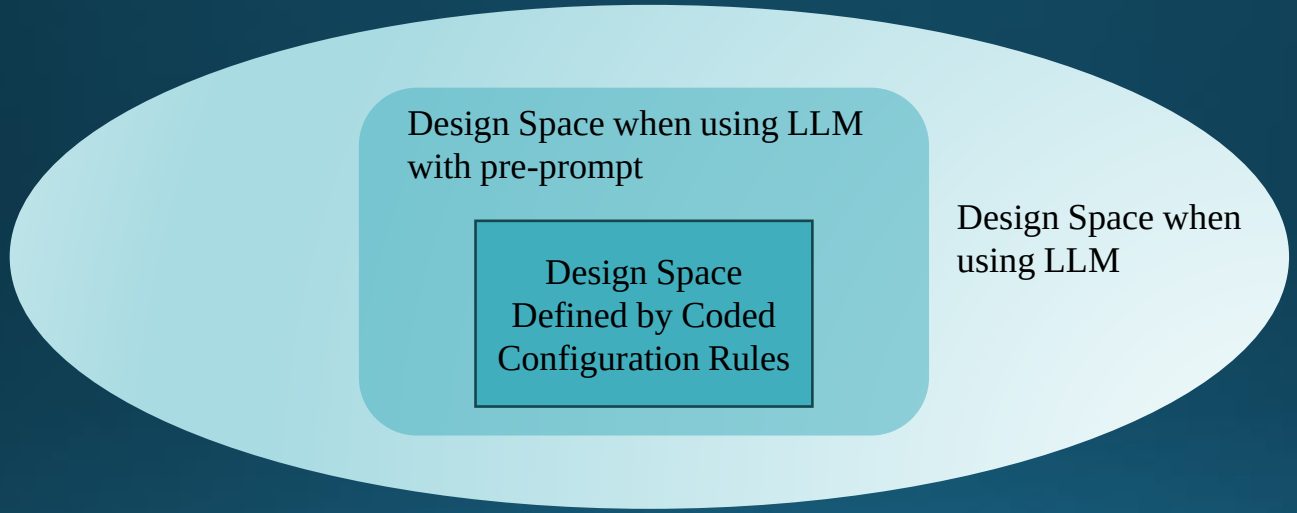
Figure 6 – UML Sequence Diagram illustrating the interaction process from configuration rules to Python configuration code.

From Python Code to Component Diagram

- When some satisfactory base line system has been generated the LLM is asked to generate Python code with some degrees of freedoms.

- Output from generated Python program
- plantuml_code = generate_plantuml_code(energy_source="fuel_tank",num_motors=4)




Design Space when using LLM with pre-prompt
Design Space Defined by Coded Configuration Rules
Design Space when using LLM

Incrementally Increase the Complexity of the Python Code

- Adding capability for different subsystems, by using the existing code as a starting point.

Prompt: "Also ad the possibility to have additional actuators (like flaps and landing gear) that are using a subset of the circuits".

```
# Example usage:
functions = ["Aileron Left", "Aileron Right", "Elevator Left", "Elevator Right", "
additional_actuators = {
    "Flaps Left": [1],
    "Flaps Right": [1],
    "Landing Gear Nose": [2],
    "Landing Gear Left": [2],
    "Landing Gear Right": [2]
}
circuits = ["Circuit 1", "Circuit 2"]
```

Incrementally Increase the Complexity of the Python Code: Example

- Flaps and Landing Gears are added to single (named) systems

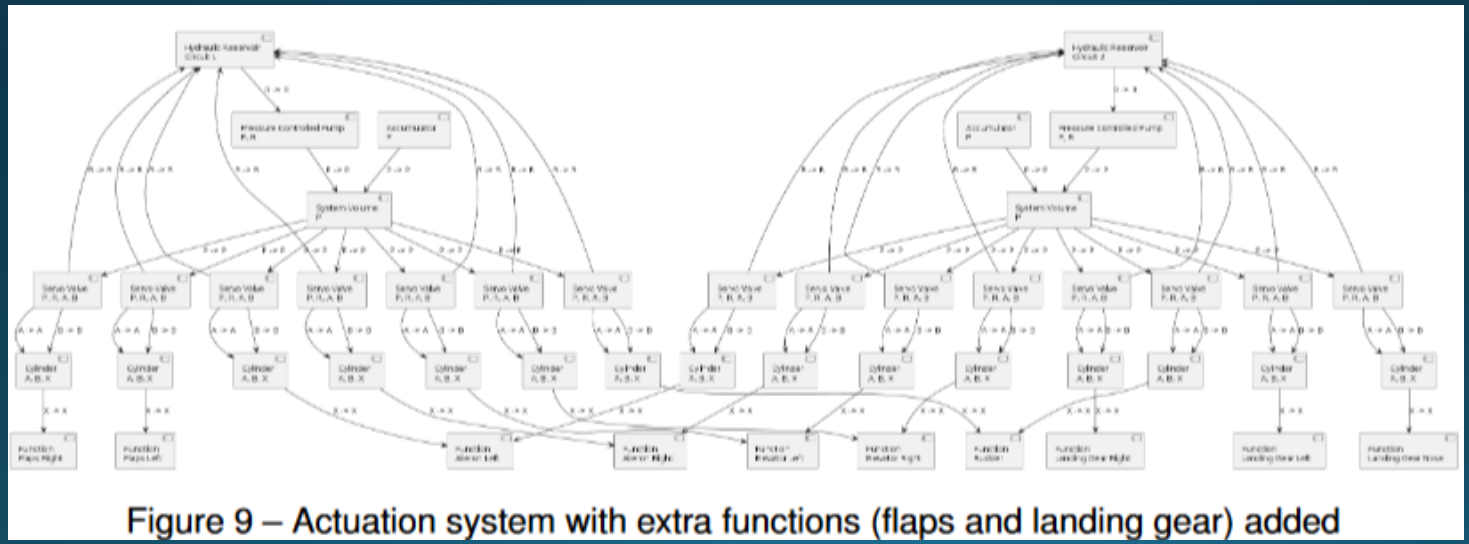
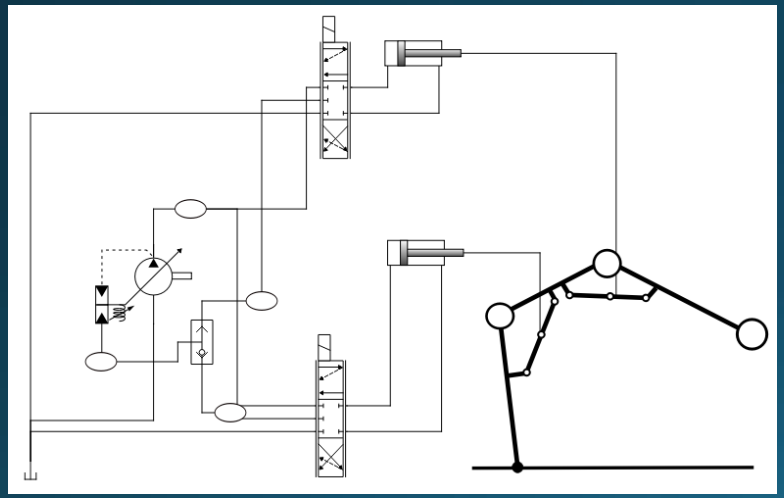


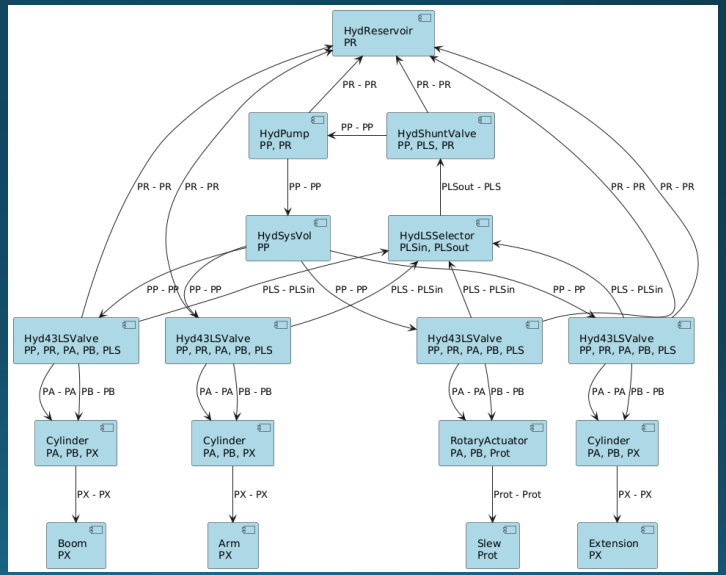
Figure 9 – Actuation system with extra functions (flaps and landing gear) added

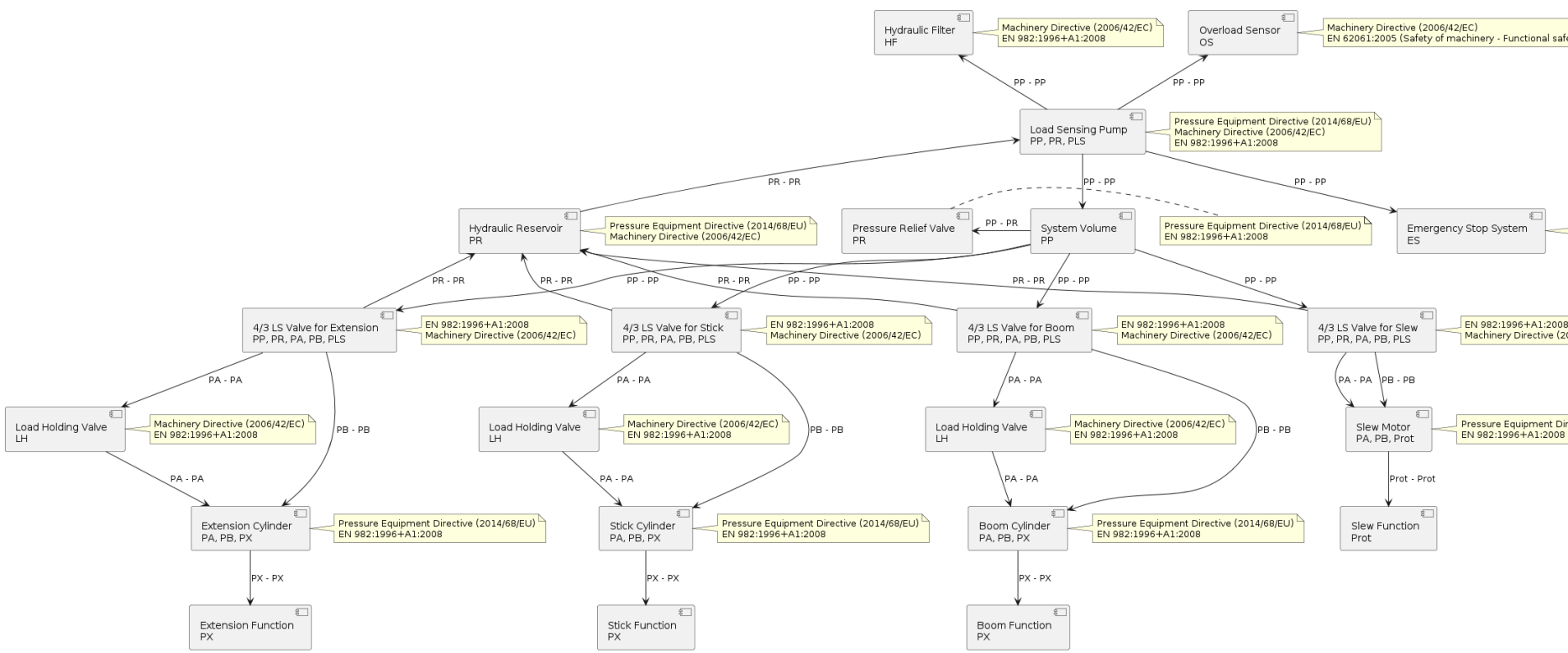
Load Sensing Hydraulic System for Truck Mounted Crane

Design Space defined using LLM with pre-prompt

The Fuzzy Design Space when using LLM





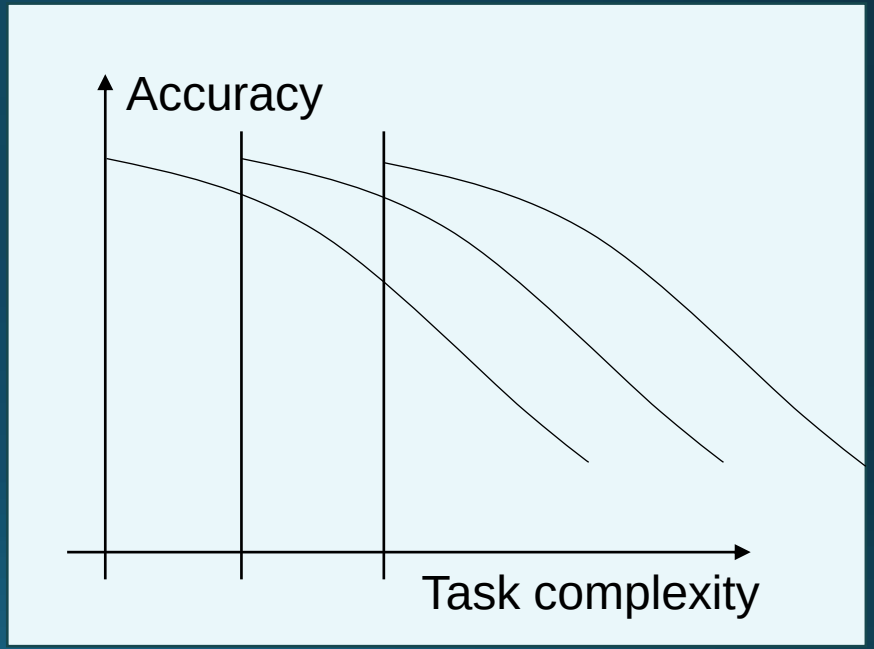


Insert Requirements from European Regulations.

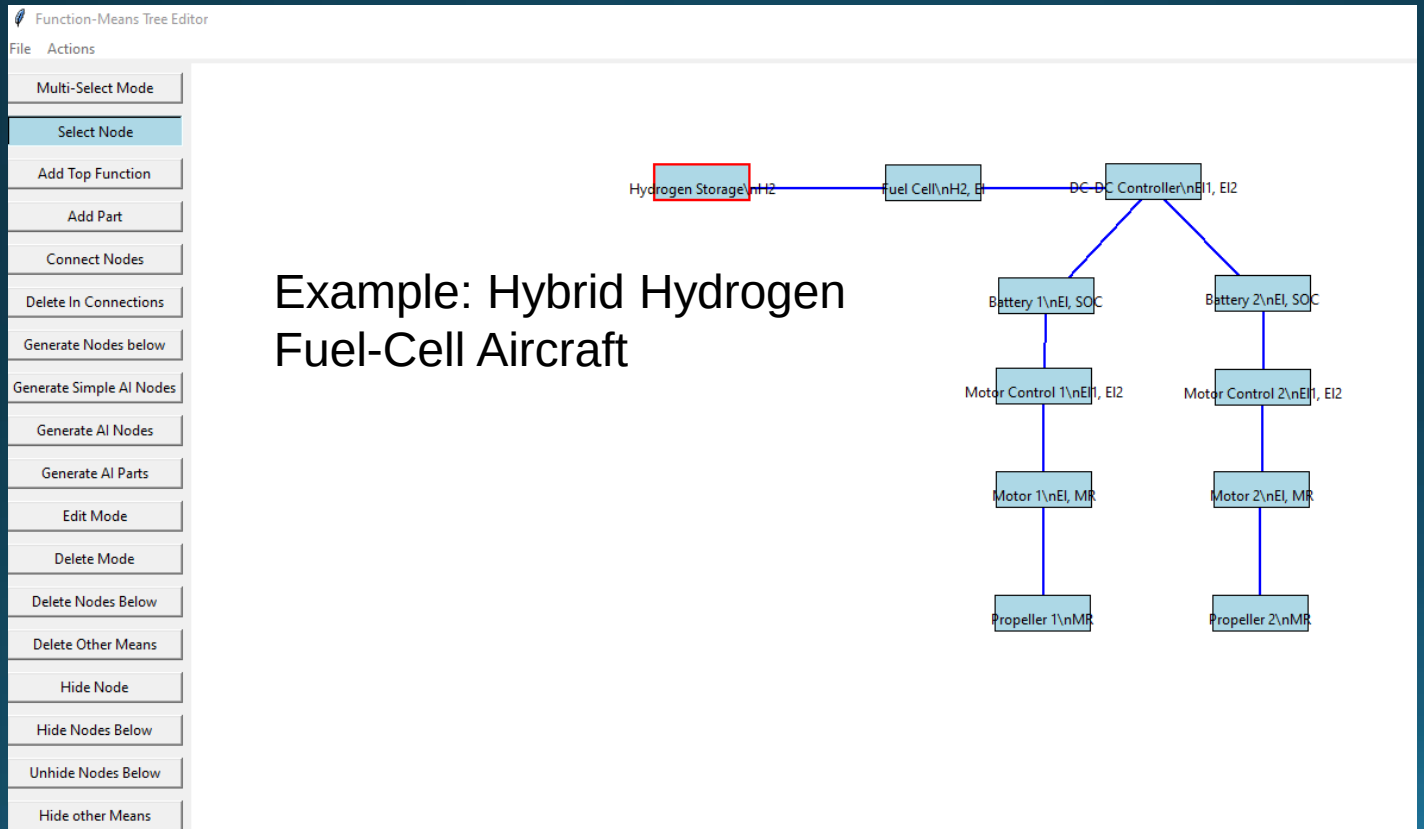
- Required valves to fulfill the regulations are added automatically.

Use API for Tools to Enable Interactive use of LLMs

- Allow for interactive use of LLM by using LLM-API inside a tool
- Example 1: System decomposition

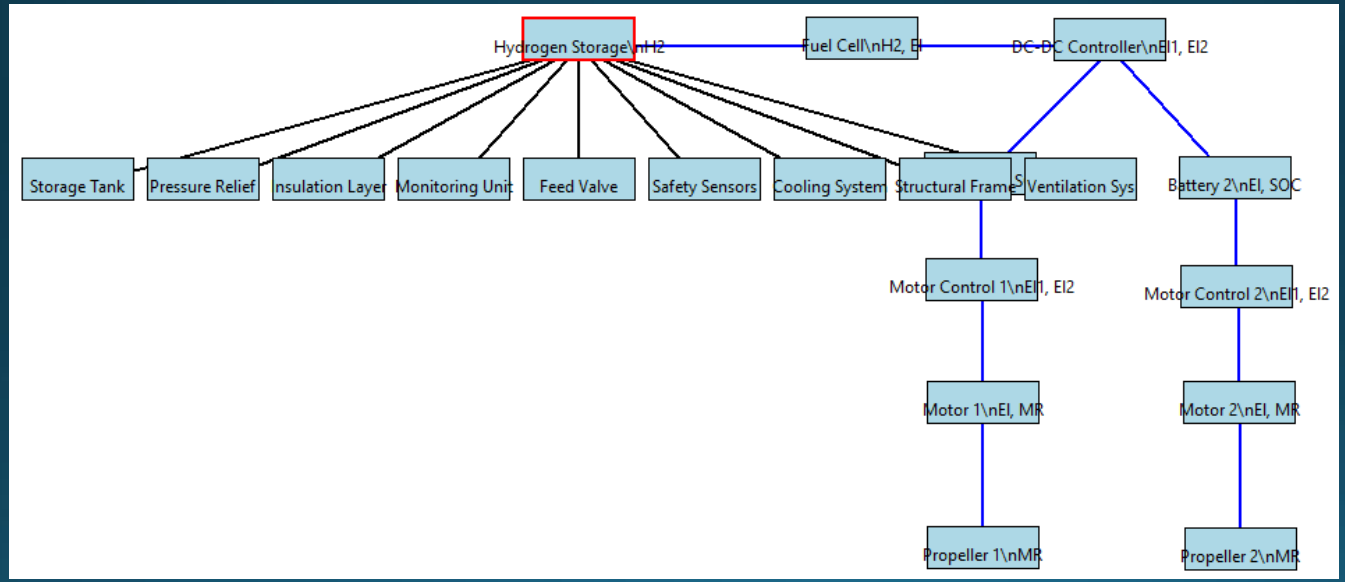


Importing (AI-generated) System Layout to Generate Subsystems/Components

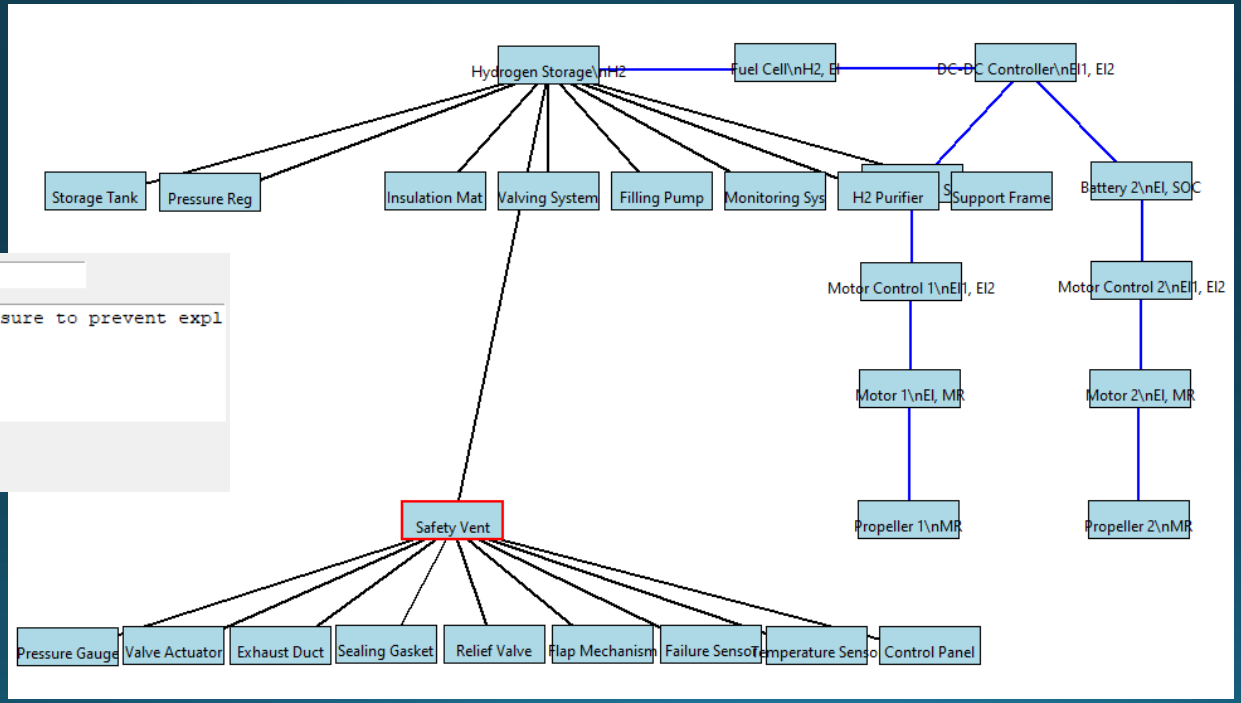


Example: Hybrid Hydrogen Fuel-Cell Aircraft

AI Generate Subsystems/Components



AI Generate Subsystems/Components



Name: Safety Vent

Comment: Releases excess pressure to prevent explosion.

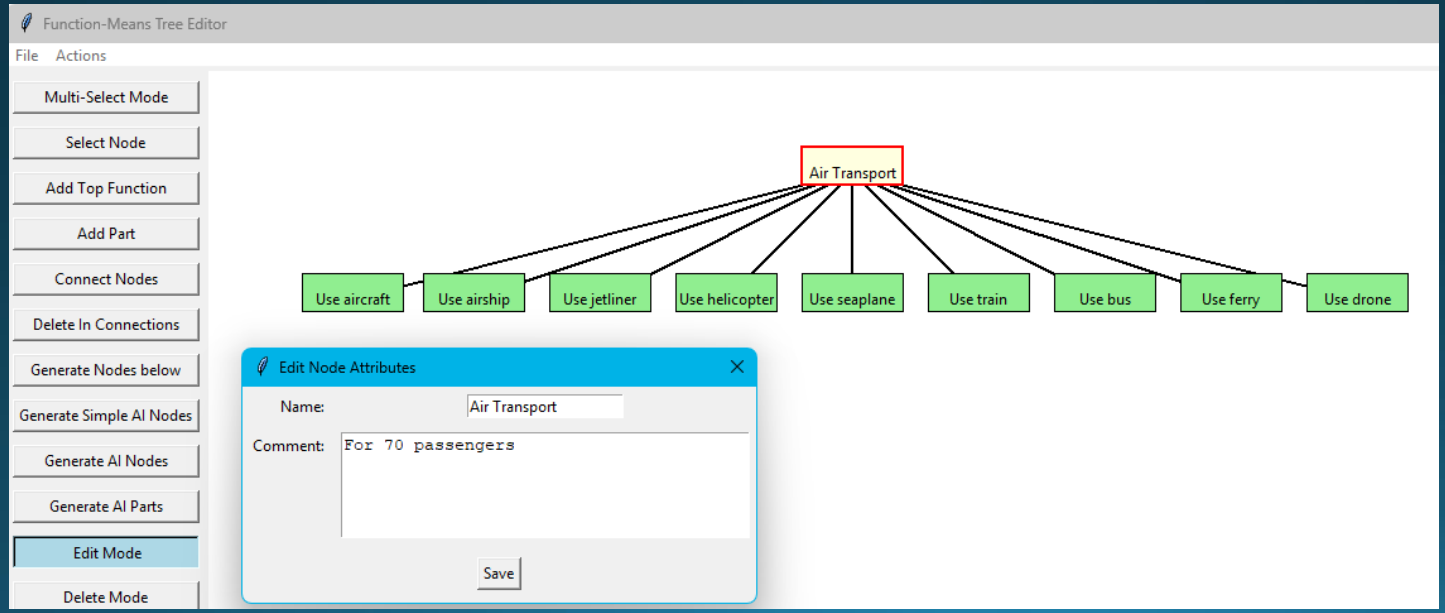
Save

```python
# Gather parent context for AI prompt
def gather_parent_context(node_id):
    for edge in connection_manager.edges:
        if edge["to"] == node_id:
            parent_node_id = edge["from"]
            parent_node = self.nodes[parent_node_id]
            return gather_parent_context(parent_node_id) + [parent_node["name"]]
    return []

parent_context = " > ".join(gather_parent_context(selected_node_id))
comment = selected_node.get("comment", "").strip()
node_info = f"'{selected_node['name']}'"
if comment:
    node_info += f" (Comment: {comment})"

# Embedded instructions for generating Part nodes
instructions = """
You are working on generating parts for a system design.
Use the following rules for generating Part nodes:
1. For a 'Part', generate up to 9 sub-parts that can integrate into the parent node.
2. Each sub-part should represent a distinct functional or structural component of the parent.
3. Each sub-part should have a concise name (≤15 characters) and a short description of its role.
4. Ensure sub-parts are diverse and cover various aspects such as structural, mechanical, electrical, or signal-related element

Format:
- Name: [Suggestion Name]
- Comment: [Suggestion Role/Description]
"""

prompt = f"""
{instructions}
Context: The current node is part of a system hierarchy '{parent_context}'
to design the part '{selected_node['name']}'.
The selected node is {node_info}.
Suggest exactly 9 Part sub-nodes for this '{selected_node['name']}'.
Each Part node should have a name (≤15 characters) and a description (≤50 characters).
"""

import openai
openai.api_key = self.api_key

print("Prompt:", prompt)

try:
    response = openai.ChatCompletion.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": instructions},
            {"role": "user", "content": prompt}
        ],
        max_tokens=250
    )
    raw_content = response.choices[0].message.content
```

```
Prompt:

        You are working on generating parts for a system design.
        Use the following rules for generating Part nodes:
        1. For a 'Part', generate up to 9 sub-parts that can integrate into the pare
        2. Each sub-part should represent a distinct functional or structural compon
        3. Each sub-part should have a concise name (≤15 characters) and a short des
        4. Ensure sub-parts are diverse and cover various aspects such as structural

        Format:
        - Name: [Suggestion Name]
        - Comment: [Suggestion Role/Description]

        Context: The current node is part of a system hierarchy 'Hydrogen Storage\nH
        to design the part 'Safety Vent'.
        The selected node is 'Safety Vent' (Comment: Releases excess pressure to pre
        Suggest exactly 9 Part sub-nodes for this 'Safety Vent'.
        Each Part node should have a name (≤15 characters) and a description (≤50 ch

Raw AI response: - Name: Pressure Relief
  - Comment: Opens to release excess gas pressure.

- Name: Check Valve
  - Comment: Prevents backflow of hydrogen gas.

- Name: Safety Seal
  - Comment: Ensures airtight closure to prevent leaks.

- Name: Flow Meter
  - Comment: Measures the rate of hydrogen flow.

- Name: Vent Cap
  - Comment: Covers the vent when not in use.

- Name: Temperature Sensor
  - Comment: Monitors vent temperature for safety.

- Name: Pressure Gauge
  - Comment: Displays current internal pressure levels.

- Name: Vibration Damp
  - Comment: Reduces vibrations during vent operation.

- Name: Warning Light
  - Comment: Illuminates for safety alerts during operation.
```

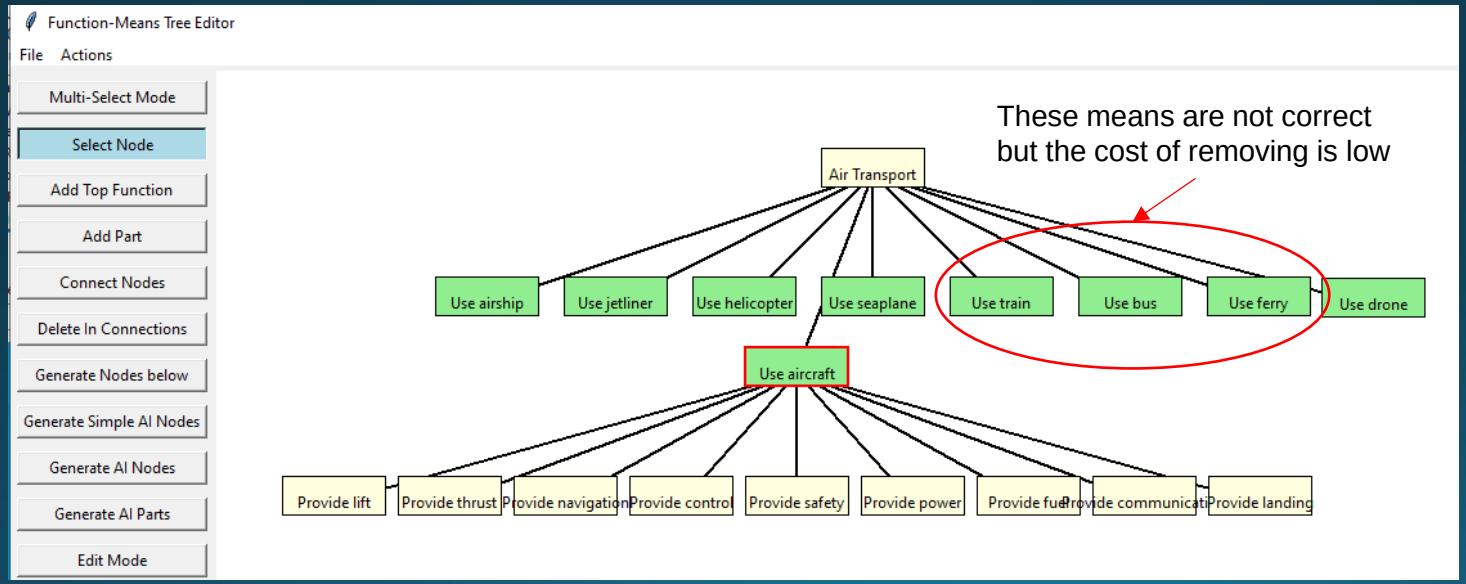
Use API for Tools to Enable Interactive use of LLMs

- Example 2: Function Means Tree
- LLM is used to generate suggestions for alternative solutions

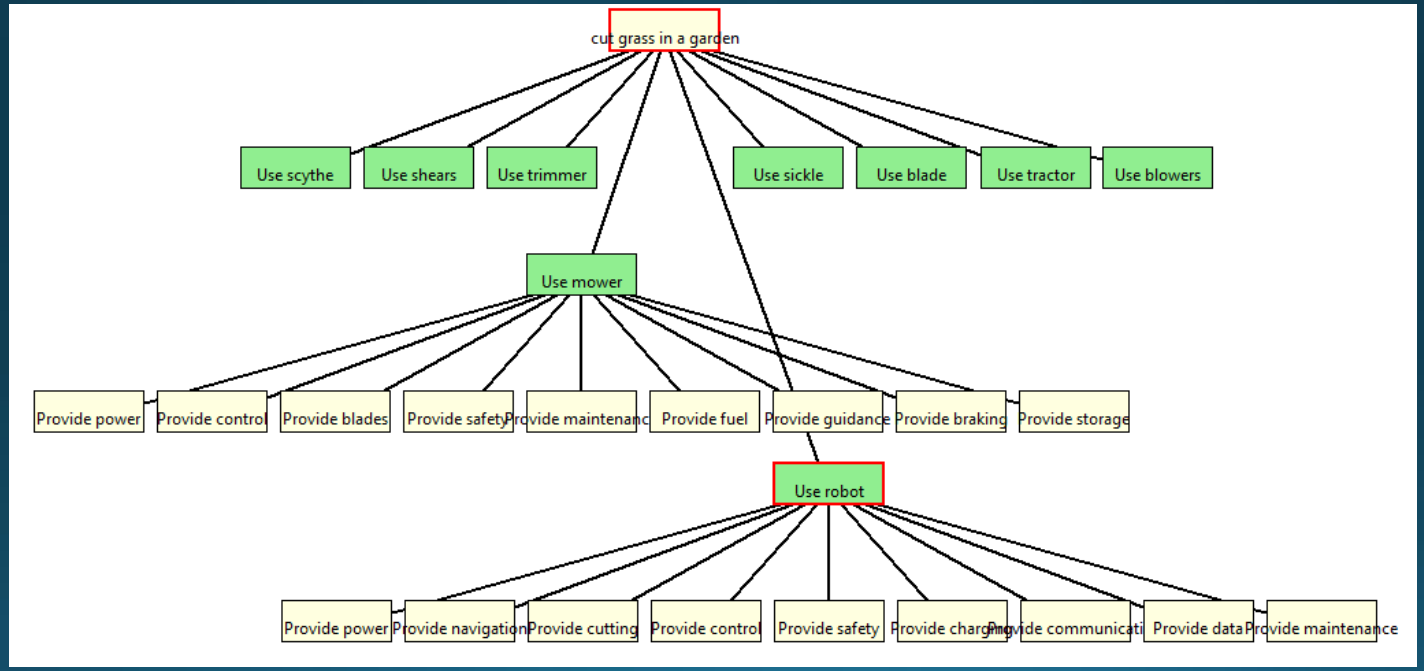


Use API for Tools to Enable Interactive use of LLMs

- Generating functions for the next level.



Lawn Mower Example



DISCUSSION

- **Potential Contribution of Large Language Models, LLM:**
  - ChatGPT and other large language models can significantly enhance the **conceptual design process** by assisting with complex system configurations.
  - **Limitations**: Difficulty in managing highly intricate and interdependent design rules due to the model's constraints, as system complexity grows.
  - **Solution**:
    - The use of micro templates to show examples of solutions greatly enhance the reliability.
    - Generating initial Python code to generate a confined design space,
      - followed by iterative improvement using LLM to expand and refine these rules step by step, enables efficient management of complexity.
    - Integrating LLM API in tools for using LLMs interactively.

DISCUSSION

- **Application in UML Component Diagrams:**
  - The model produced UML component diagrams in **PlantUML** due to its minimalist format and ease of use, facilitating a clear representation of system architecture.
  - However, other diagram formats can also be incorporated for more detailed or specific representations.
  - UML diagrams can be mapped to system libraries for **simulation purposes**, allowing system architectures to be manipulated, tested, and validated in simulations.

CONCLUSIONS

LLMs should be used when:

value of the correct answers > cost of the bad answers