This is a repository copy of *Using GPT-4 to Generate Failure Logic*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/215140/

Conference or Workshop Item:
Clegg, Kester Dean orcid.org/0000-0002-4484-3291, McDermid, John Alexander orcid.org/0000-0003-4745-4272 and Habli, Ibrahim orcid.org/0000-0003-2736-8238 (2024) Using GPT-4 to Generate Failure Logic. In: UNSPECIFIED.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Using GPT-4 to Generate Failure Logic

Kester Clegg, Ibrahim Habli, and John McDermid

Centre for Assuring Autonomy, University of York, U.K.
{`kester.clegg, ibrahim.habli, john.mcdermid`}`@york.ac.uk`

Abstract. Large Language Models' (LLMs) ability to explain complex texts has raised the question of whether their encoded knowledge is sufficient to reason about system failures. The current weaknesses in LLMs, like misalignment and hallucinations suggest they make unsuitable safety analysts, but could "fast but flawed" analysis still be useful? LLMs can rapidly parse system descriptions for design mitigation strategies like redundancy, they can trace failure propagation from common mode faults (such as loss of power or hydraulics) to higher level events and even incorporate non-functional risks from outside the functional specification into an analysis. But despite their knowledge of hardware component failure modes, we found LLMs remain weak at failure logic reasoning. We used OpenAI's Generative Pre-trained Transformer (GPT) Builder to develop a specific role for analysing failure logic and generating the corresponding fault tree visualisation. Although there are no objective measures that qualitatively assess failure logic analysis (i.e. logical errors have variable significance) or whether the choice of higher-level failure modes is a "good model" of system failure, we report on the iterative process of developing the GPT, our inability to override the underlying model behaviour to counter its weaknesses, and conclude by reflecting on the productivity gains of using LLMs despite their flawed reasoning.

Keywords: Large Language Models · Fault Tree Analysis · Failure Logic

1 Introduction

LLMs have become an almost universal tool for text generation and knowledge retrieval. In safety analysis this presents an opportunity to improve productivity in areas that have traditionally been difficult to optimise. To date, LLMs have been tried out in areas that make use of their creative potential [9] where factual inaccuracy is less of a problem, rather than areas like failure logic which requires accurate reasoning about how physical systems can fail. We accepted from the outset the known weaknesses of LLMs, like hallucination and poor training data, but given the 'chat' style interfaces for most LLMs we wanted to target a process that involves interactive discussion and refinement typical of safety analysts working with system and software engineers. Our selected task was to perform failure logic analysis from a top down perspective, as would be typically done for a (preliminary) System Safety Analysis (SSA). One advantage of choosing this was that the level of specialist technical detail could be kept low, and it could be further extended by visualising the corresponding fault trees, a task which is time-consuming to perform manually and often requires proprietary graphical software or bespoke profiles within system modelling languages like SysML [4].

Analysing a system for failure logic requires an understanding of how hardware failure modes can affect system or software functionality and how those faults propagate through the system [7]. LLMs have been criticised as more suited to creative work than analytical output based on critical reasoning [5], although techniques have been published that appear to help with such tasks, such as 'tree of thought' techniques and critical prompts [11]. Our own experience backs up the literature reporting greater success with these techniques, but the weaknesses we have experienced also form part of the changing picture of how we will work with these 'acceptably imperfect' tools in the future. Experiments in code generation and other content creation tasks have demonstrated productivity gains despite the tendency of LLMs to output incomplete or incorrect answers [8][10].

The contributions of this paper are to identify specific weaknesses in GPT-4 with respect to failure logic analysis and the inability of our GPT's system instructions to override the default behaviour of GPT-4. We describe the GPT Builder interface provided by OpenAI and our development of FLAGPT (§2). We give the task scenarios (§3), present the results (§4.1 and §4.2) and finally discuss the implications and future potential for this technology in safety analysis (§5). We discuss the iterative process of coaxing better quality answers from LLMs in §2, §4.4 and §5, noting that human oversight remains critical.

2 FLAGPT: developing a Failure Logic Analysis GPT

OpenAI offers various tools and APIs for developers to customize and integrate GPTs into their applications, enabling the creation of bespoke GPT models tailored to specific needs. They offer two ways to do this. One requires no programming skills and can be used through 'system instructions' (aka 'prompt engineering') that can be generated automatically by GPT-4 through conversations with the Chat GPT Builder interface, where the user describes the type of GPT they want and how it should respond or format its output. The second is via their API that allows for fine-tuning, vector embeddings and other customisations. OpenAI estimates that fine-tuning requires a minimum of 50-100 'question + answer' examples to train on. This amount of system descriptions with corresponding failure logic analyses wasn't available in the research time we had, and neither was it obvious how we would evaluate the system responses (see 3 for a more detailed discussion) and so we chose to use the GPT Builder interface as it provides a 'low barrier' entry point based on prompt engineering.

A bespoke GPT is created using the GPT Builder by initially describing the type of tasks you want the GPT to perform. GPT builder will then create a logo and auto-generate some system instructions. You can upload a document that GPT builder will use to create vector embeddings for fine-tuning, although we had mixed results with this approach. It should be noted that each time you converse with the Create feature in the builder interface, your system instructions may be updated in unexpected ways or even overwritten. The Create feature remains useful to get testing quickly, but we got better results by handwriting the system instructions ourselves.

FLAGPT's system instructions are available in our github (see Acknowledgements) in two parts. The first part instructs FLAGPT to analyse all parts of a system for its failure logic and list all events and logic gates. The second part uses a template of a text-based drawing environment for LaTeX called TikZ[1] that can represent fault trees. Initially we told FLAGPT to create a LaTeX document using the TikZ commands. However, the increased output length caused a lot of errors and misalignment, resulting in generic LaTeX + TikZ output. We got better results by reducing what was asked of FLAGPT. Instead of an entire LaTeX file with TikZ commands, we restricted it to an example of node declarations and definitions for the fault tree, followed by the list of logic gates and how they connect. This was used as an input file to a LaTeX document we predefined. It is important to note that we were using the fault trees as a visual aid to identify errors in the fault logic, not to conduct fault tree analysis. However, there is no reason this couldn't be done by changing the output format using a different template, for example a spreadsheet that could be imported into the Fault Tree+ module of Isograph's Reliability Workbench™.

3 Task and system descriptions

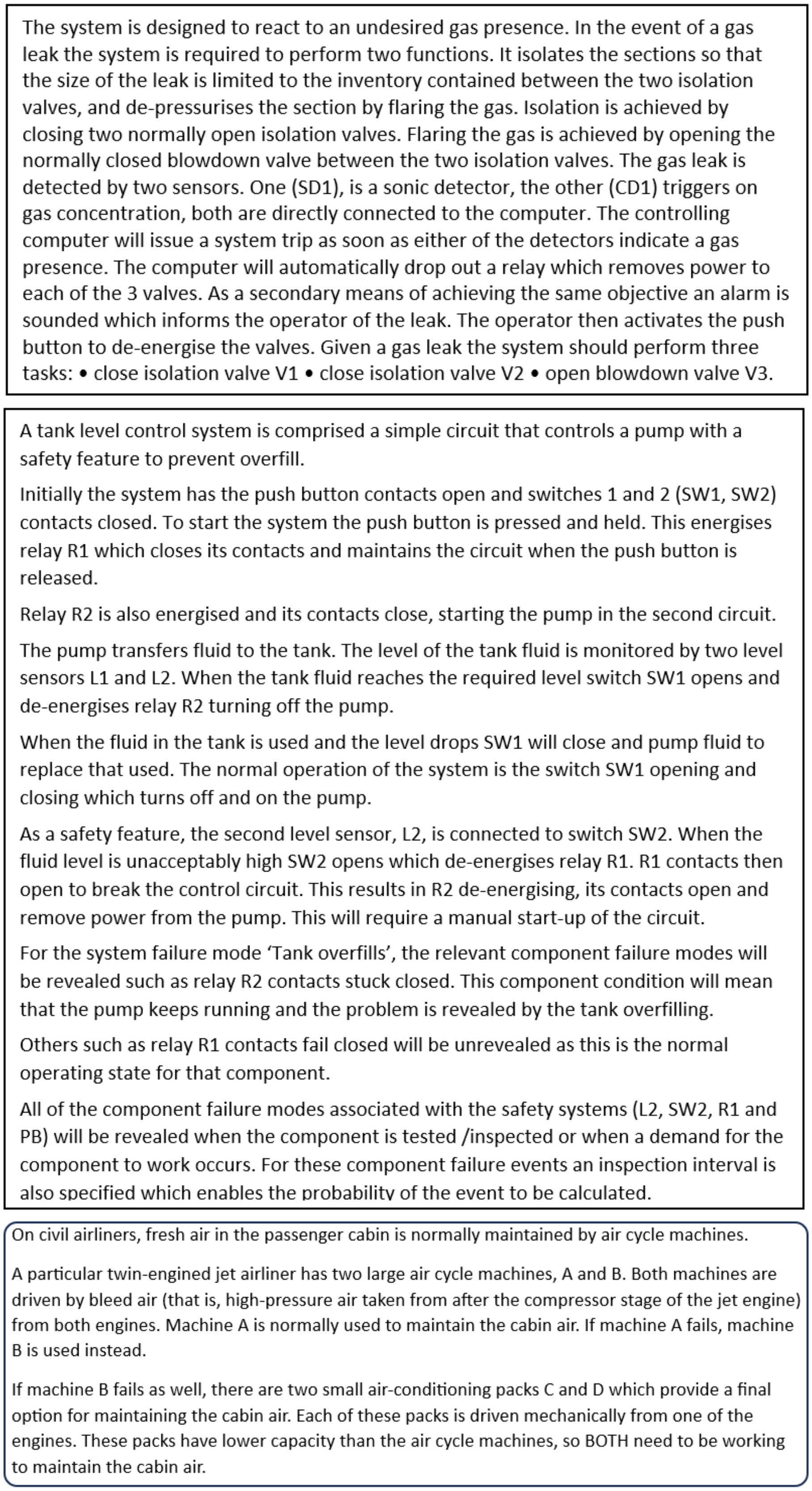
The task to analyse a system for its failure logic is started by either uploading a text file containing the system description and then instructing FLAGPT to "Analyse this system for the top level failure '[your event name]' ".[2] Explicitly naming the top level event ensures that it will get used as the final event of the fault tree, although left unprompted FLAGPT was able to derive appropriate names itself. FLAGPT should briefly describe how it will proceed and list the top level failure events in the fault tree hierarchy and the failure logic connecting those events.

There is limited system descriptions and domain experts available that could generate a quantitative assessment of failure logic analysis quality, and our intention here is not to exhaustively test LLM performance. Fault logic modelling is heavily reliant on domain expertise, and trying to compare different system specifications and their subsequent analysis ends up being a subjective exercise even with a panel of adjudicators. Increasing input token lengths to LLMs can result in greater misalignment [6] and we limited our system description lengths accordingly. But it is also true that not all errors are of equal severity in fault logic, making meaningful quantitative analysis of LLM performance difficult. Similar arguments also apply to qualitative assessments on the output,

---

[1] Further details are available at `https://en.wikipedia.org/wiki/PGF/TikZ`

[2] It is also possible to upload a diagram of the system and make use of GPT-4's ability to generate image-to-text. However, combining images with system descriptions does not improve the quality of FLAGPT's analysis.



The system is designed to react to an undesired gas presence. In the event of a gas leak the system is required to perform two functions. It isolates the sections so that the size of the leak is limited to the inventory contained between the two isolation valves, and de-pressurises the section by flaring the gas. Isolation is achieved by closing two normally open isolation valves. Flaring the gas is achieved by opening the normally closed blowdown valve between the two isolation valves. The gas leak is detected by two sensors. One (SD1), is a sonic detector, the other (CD1) triggers on gas concentration, both are directly connected to the computer. The controlling computer will issue a system trip as soon as either of the detectors indicate a gas presence. The computer will automatically drop out a relay which removes power to each of the 3 valves. As a secondary means of achieving the same objective an alarm is sounded which informs the operator of the leak. The operator then activates the push button to de-energise the valves. Given a gas leak the system should perform three tasks: • close isolation valve V1 • close isolation valve V2 • open blowdown valve V3.

A tank level control system is comprised a simple circuit that controls a pump with a safety feature to prevent overfill.

Initially the system has the push button contacts open and switches 1 and 2 (SW1, SW2) contacts closed. To start the system the push button is pressed and held. This energises relay R1 which closes its contacts and maintains the circuit when the push button is released.

Relay R2 is also energised and its contacts close, starting the pump in the second circuit.

The pump transfers fluid to the tank. The level of the tank fluid is monitored by two level sensors L1 and L2. When the tank fluid reaches the required level switch SW1 opens and de-energises relay R2 turning off the pump.

When the fluid in the tank is used and the level drops SW1 will close and pump fluid to replace that used. The normal operation of the system is the switch SW1 opening and closing which turns off and on the pump.

As a safety feature, the second level sensor, L2, is connected to switch SW2. When the fluid level is unacceptably high SW2 opens which de-energises relay R1. R1 contacts then open to break the control circuit. This results in R2 de-energising, its contacts open and remove power from the pump. This will require a manual start-up of the circuit.

For the system failure mode 'Tank overfills', the relevant component failure modes will be revealed such as relay R2 contacts stuck closed. This component condition will mean that the pump keeps running and the problem is revealed by the tank overfilling.

Others such as relay R1 contacts fail closed will be unrevealed as this is the normal operating state for that component.

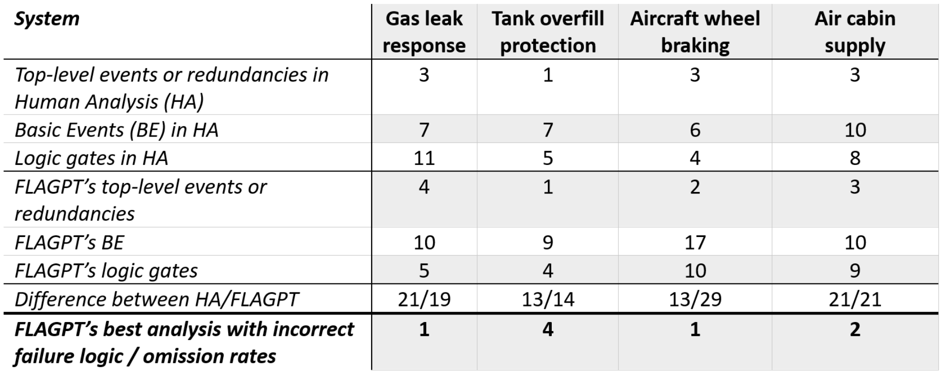
All of the component failure modes associated with the safety systems (L2, SW2, R1 and PB) will be revealed when the component is tested /inspected or when a demand for the component to work occurs. For these component failure events an inspection interval is also specified which enables the probability of the event to be calculated.

On civil airliners, fresh air in the passenger cabin is normally maintained by air cycle machines.

A particular twin-engined jet airliner has two large air cycle machines, A and B. Both machines are driven by bleed air (that is, high-pressure air taken from after the compressor stage of the jet engine) from both engines. Machine A is normally used to maintain the cabin air. If machine A fails, machine B is used instead.

If machine B fails as well, there are two small air-conditioning packs C and D which provide a final option for maintaining the cabin air. Each of these packs is driven mechanically from one of the engines. These packs have lower capacity than the air cycle machines, so BOTH need to be working to maintain the cabin air.

Fig. 1. System descriptions for gas leak response, tank overfill and cabin air supply.



System	**Gas leak response**	**Tank overfill protection**	**Aircraft wheel braking**	**Air cabin supply**
Top-level events or redundancies in Human Analysis (HA)	3	1	3	3
Basic Events (BE) in HA	7	7	6	10
Logic gates in HA	11	5	4	8
FLAGPT's top-level events or redundancies	4	1	2	3
FLAGPT's BE	10	9	17	10
FLAGPT's logic gates	5	4	10	9
Difference between HA/FLAGPT	21/19	13/14	13/29	21/21
FLAGPT's best analysis with incorrect failure logic / omission rates	**1**	**4**	**1**	**2**

Table 1. Failure logic complexity for each system. The human analyses (by respective authors) are given in the top half of the table and FLAGPT's in the bottom.

particularly when viewed from the explanatory value of fault trees to display system failure modes. Instead we focus on our experience of co-working with an LLM and the issues we encountered. We kept our experiments to four short text descriptions that varied in terms of their redundancy, failure modes and failure logic. These were: a gas leak response system (from [2]), a tank overfill protection system (from [2]), an aircraft wheel braking system from ARP4761 [1] and an air cabin supply system (from University of York coursework). With the exception of the example from ARP4761 [1] these are shown in Fig. 1.

While there is no objective measure we can apply to gauge difficulty for failure analysis, and given that higher level failure modes can be subjective placeholders to aid human comprehension, we nonetheless have attempted to show differences between the system descriptions by counting the number of basic events, number of logic gates and levels of redundancy or top level failure events produced in each source by the authors. Table 1 shows these numbers alongside FLAGPT's for comparison. We acknowledge that neither counting the number of gates or basic events, nor summing them to produce a comparative total, gives any kind of meaningful measure of difficulty in comprehending the failure logic for the systems. Context is critical, perhaps particularly for LLMs [3]. One might argue that assigning incorrect failure logic to a conjunction of events (e.g. AND instead of OR) is a more serious error with regard to quality than omitting a basic event, however omission of a failure event could also be regarded as critical. We added these two faults as a final row on the table, again acknowledging that where in the fault tree the incorrect logic or omission occurs could make a big difference that we have not attempted to quantify.

In the case of omission, we would also like to point out that on occasions GPT-4 seemed to default to a minimal ('lazy') analysis and would either fail to develop top level events any further or would not include basic events in either the analysis or visualisation. As we couldn't control GPT-4's underlying behaviour (i.e. FLAGPT's system instructions were the same each time it was run) we would have to re-prompt FLAGPT to follow its system instructions. In terms of correctly parsing system redundancy and failure logic, FLAGPT's difficulties seemed to centre around the exact wording or type of subsystems. For example, systems that run in parallel to provide full capacity (like the air conditioning packs in the air cabin supply system) were always misinterpreted as AND failure logic, possibly due to use of the word 'BOTH' in the description. Likewise the triple redundancy in the aircraft wheel braking system was often not spotted by FLAGPT, as it appeared to consider the alternative braking system as the emergency back up, and not recognise that the emergency system was a manual override if all else fails. Both could be corrected by additional prompting from a safety analyst who is familiar with the system design, but a novel, unfamiliar specification would be more difficult to check.

4 Results and Working Experience

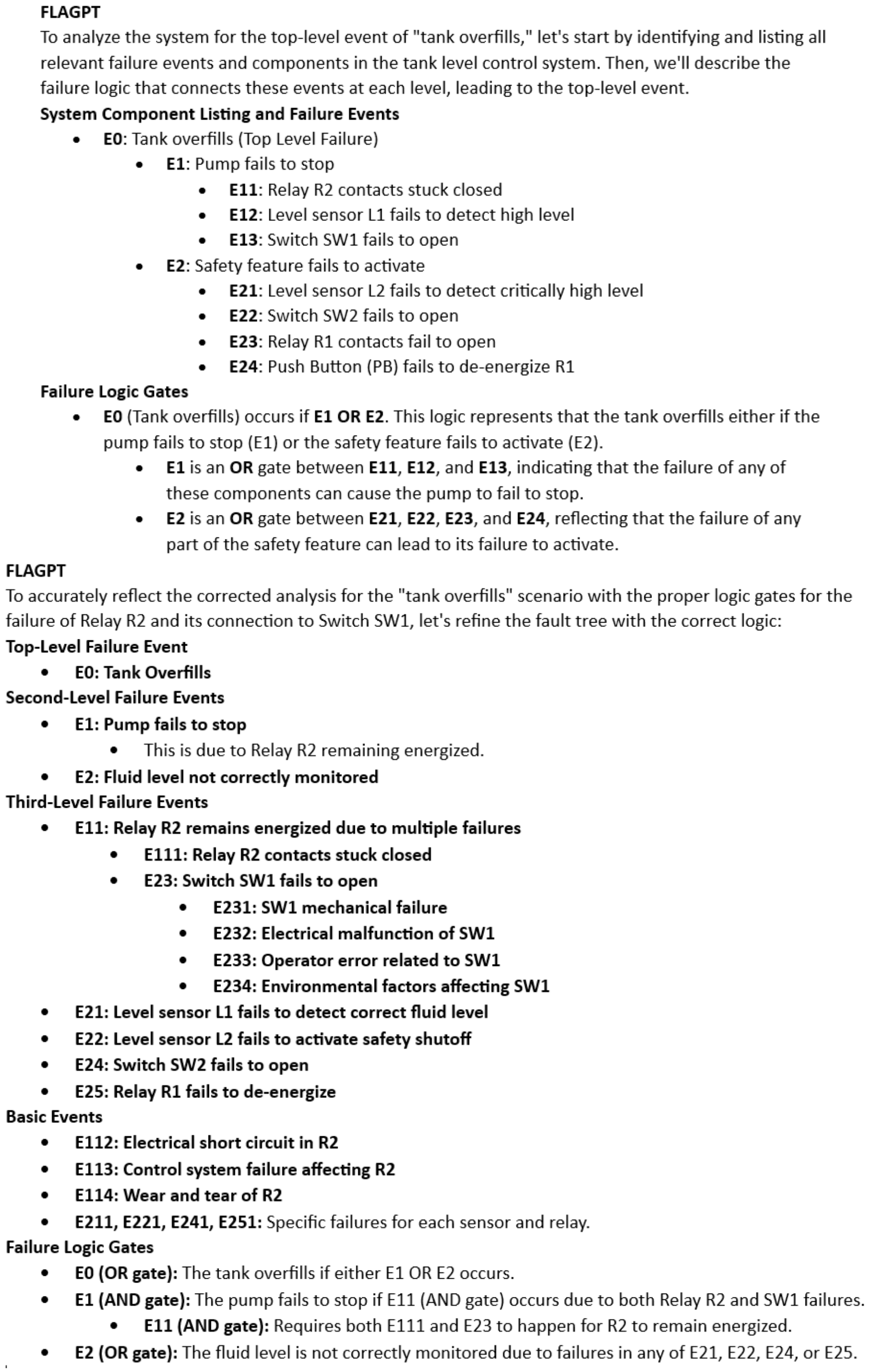
4.1 Tank Overfill failure event

First analysis of the tank overfill scenario will typically produce the output in the top half of Fig. 2. Visualisation of the fault tree (Fig. 3) reveals that the 'Relay R2 remains energised' failure event was undeveloped for its dependencies. Our first attempt at improvement was to prompt "You need to improve your analysis. Expand why Relay R2 remains energised". This did not change the analysis. Next we asked "What about SW1?". Again FLAGPT refused to adjust its analysis. Finally after several more attempts and further expansion, we corrected FLAGPT by explicitly prompting "Shouldn't E111 and E23 be connected by an AND gate for R2 to fail?". This generated the analysis in the lower half of Fig. 2 and the fault tree shown in the lower half of Fig. 3. Note there are several errors and omissions in this analysis, for example the push button is not mentioned and 'R1 remaining energised' is not developed. Overall this is a poor analysis that should start from the failure of the pump to stop and develop through to R2 and then R1 remaining energised.

This corrective interaction between FLAGPT and the user is typical.[3] In most cases we would expect between 5–10 attempts to correct the analysis. The analysis may still be incorrect at this point, but 'good enough' so that the user can take it and finish it manually. The iterative process is perhaps unfamiliar to tool users, where typically we expect exactly the same output each time a program is run. LLMs offer a different user experience, where results vary and close supervision by the user is needed before using the output. Despite generating only partially correct outputs, the speed at which the analysis and visualisation can be performed results in greater productivity. Within our research group we found most users would take the time saved plus some manual corrections over having to do the entire analysis and visualisation manually.

---

[3] We analysed each scenario 15–20 times, ensuring that option to allow GPT-4 to learn from our dialogue was turned off.



FLAGPT

To analyze the system for the top-level event of "tank overfills," let's start by identifying and listing all relevant failure events and components in the tank level control system. Then, we'll describe the failure logic that connects these events at each level, leading to the top-level event.

System Component Listing and Failure Events

- **E0**: Tank overfills (Top Level Failure)
  - **E1**: Pump fails to stop
    - **E11**: Relay R2 contacts stuck closed
    - **E12**: Level sensor L1 fails to detect high level
    - **E13**: Switch SW1 fails to open
  - **E2**: Safety feature fails to activate
    - **E21**: Level sensor L2 fails to detect critically high level
    - **E22**: Switch SW2 fails to open
    - **E23**: Relay R1 contacts fail to open
    - **E24**: Push Button (PB) fails to de-energize R1

Failure Logic Gates

- **E0** (Tank overfills) occurs if **E1 OR E2**. This logic represents that the tank overfills either if the pump fails to stop (E1) or the safety feature fails to activate (E2).
  - **E1** is an **OR** gate between **E11**, **E12**, and **E13**, indicating that the failure of any of these components can cause the pump to fail to stop.
  - **E2** is an **OR** gate between **E21**, **E22**, **E23**, and **E24**, reflecting that the failure of any part of the safety feature can lead to its failure to activate.

FLAGPT

To accurately reflect the corrected analysis for the "tank overfills" scenario with the proper logic gates for the failure of Relay R2 and its connection to Switch SW1, let's refine the fault tree with the correct logic:

Top-Level Failure Event

- **E0: Tank Overfills**

Second-Level Failure Events

- **E1: Pump fails to stop**
  - This is due to Relay R2 remaining energized.
- **E2: Fluid level not correctly monitored**

Third-Level Failure Events

- **E11: Relay R2 remains energized due to multiple failures**
  - **E111: Relay R2 contacts stuck closed**
  - **E23: Switch SW1 fails to open**
    - **E231: SW1 mechanical failure**
    - **E232: Electrical malfunction of SW1**
    - **E233: Operator error related to SW1**
    - **E234: Environmental factors affecting SW1**
- **E21: Level sensor L1 fails to detect correct fluid level**
- **E22: Level sensor L2 fails to activate safety shutoff**
- **E24: Switch SW2 fails to open**
- **E25: Relay R1 fails to de-energize**

Basic Events

- **E112: Electrical short circuit in R2**
- **E113: Control system failure affecting R2**
- **E114: Wear and tear of R2**
- **E211, E221, E241, E251:** Specific failures for each sensor and relay.

Failure Logic Gates

- **E0 (OR gate):** The tank overfills if either E1 OR E2 occurs.
- **E1 (AND gate):** The pump fails to stop if E11 (AND gate) occurs due to both Relay R2 and SW1 failures.
  - **E11 (AND gate):** Requires both E111 and E23 to happen for R2 to remain energized.
- **E2 (OR gate):** The fluid level is not correctly monitored due to failures in any of E21, E22, E24, or E25.

Fig. 2. FLAGPT's initial output (top) and 'corrected' output (bottom) after several prompts to re-engage with the analysis and correct specific faults. The fault tree is shown in Fig. 3

.

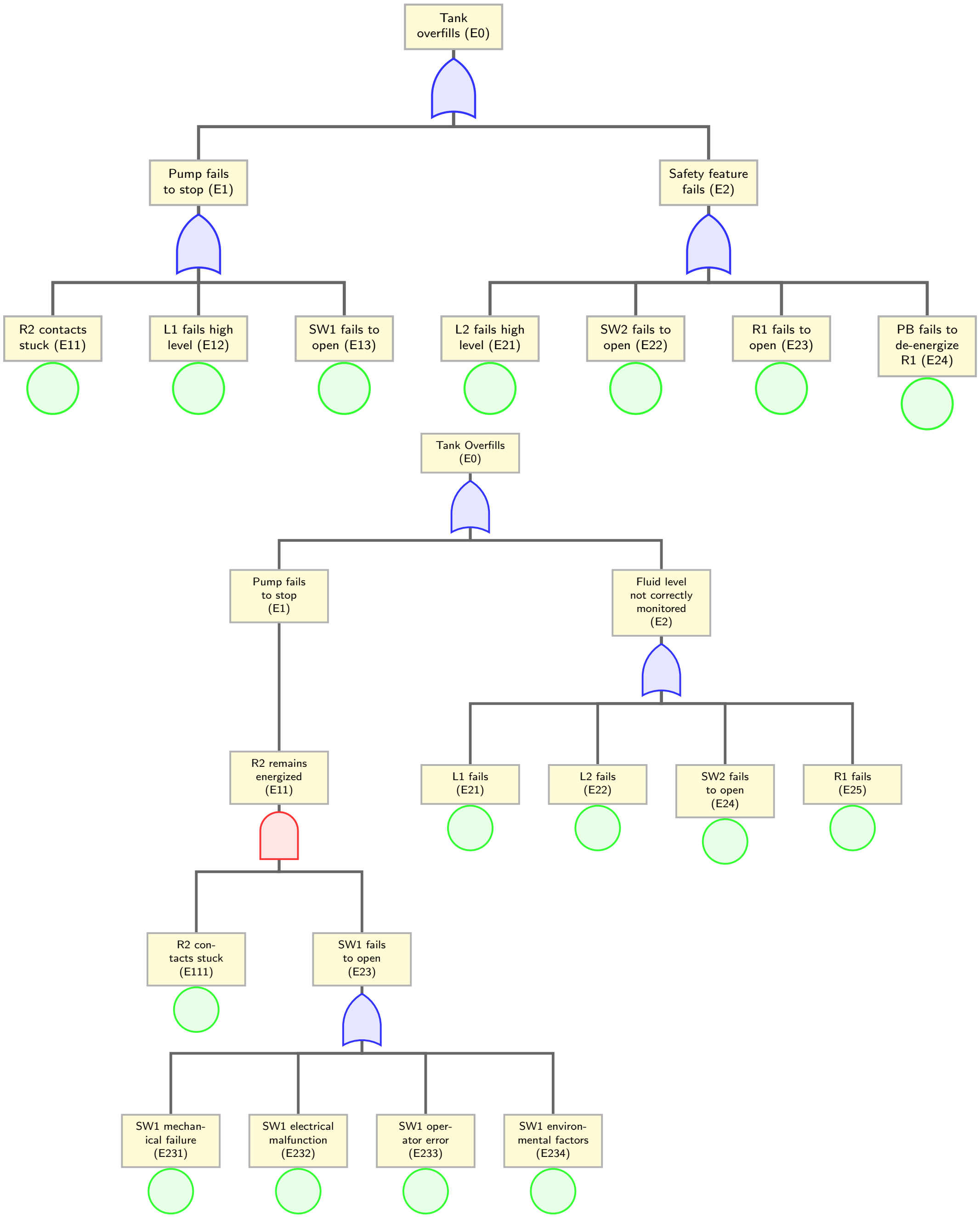

Tank overfills (E0)
Pump fails to stop (E1)
Safety feature fails (E2)
R2 contacts stuck (E11)
L1 fails high level (E12)
SW1 fails to open (E13)
L2 fails high level (E21)
SW2 fails to open (E22)
R1 fails to open (E23)
PB fails to de-energize R1 (E24)
Tank Overfills (E0)
Pump fails to stop (E1)
Fluid level not correctly monitored (E2)
R2 remains energized (E11)
L1 fails (E21)
L2 fails (E22)
SW2 fails to open (E24)
R1 fails (E25)
R2 contacts stuck (E111)
SW1 fails to open (E23)
SW1 mechanical failure (E231)
SW1 electrical malfunction (E232)
SW1 operator error (E233)
SW1 environmental factors (E234)

Fig. 3. FLAGPT's fault trees for tank overfill system (see Fig. 1). The top tree is a typical response before additional prompting to ask for further expansion on some events and correct some failure logic. Note E2 shows a completely different analysis of contributing events to the final tree's logic, with L1 no longer contributing to SW1.

4.2 Air bleed cabin supply system

The air bleed cabin supply example (see Fig. 1) which is given as an exercise to our MSc students (most of whom are experienced safety specialists), gave FLAGPT a great deal of difficulty. It frequently failed to develop the analysis sufficiently, even after being instructed to include mechanical failure of the air bleed supply in its analysis. FLAGPT would typically assign incorrect failure logic connecting E0 to the air cycle machines and back up air conditioning packs. When asked to check its answer it was able to correct this (i.e. initially assigns an AND gate but subsequently assigns OR), but this did not work for E3 (the failure of the air conditioning packs). It seems to be the wording (i.e. "...BOTH need to be working to maintain the cabin air") that causes FLAGPT to misinterpret E3 as meaning both air conditioning packs need to fail, therefore it must have an AND failure logic conjunction, even though its analysis described correctly how both packs were needed for the system to work. With explicit direction it could correct the logic and produce a reasonable fault tree (see Fig 4), but there seems to be a systemic error in GPT-4 for this type of failure event wording. We generated several synthetic examples based on this system description using GPT-4 and none were analysed correctly.

4.3 Aircraft wheel brake and gas leak systems

Both of these systems were at the lower end of failure logic complexity, with most of the failure logic represented by OR gates. FLAGPT did a reasonable job on both but additional prompting was still required to get fully developed fault trees and correct failure logic, e.g. the failure of both gas leak sensors was always incorrectly assigned an OR gate. The aircraft wheel braking system, while correctly separating the green and blue hydraulic systems and adding basic events like servo motor failure, never included the emergency brake system as an additional redundancy (see Fig. 5).

4.4 Alignment and consistency

Alignment (doing what the user wants) and consistency remain hurdles for analysts to adopt FLAGPT. Alignment is a particular issue, as the user is required to correct the analysis several times before an acceptable output is produced and depending on the length of this interaction, FLAGPT could start to drift from its system instructions, particularly on the second part of its task for visualisation. Despite forbidding it to add unwanted code, we could not override GPT-4's default behaviour (to output a standalone LaTeX document environment). In a commercial setting this would have meant paying for output tokens that were of no use. The second frustration was the lack of consistency. Once software is finished development you expect it to run repeatedly as programmed. However, FLAGPT's errors that had to be corrected could change on a repeat analysis. We found that the interactive process of getting FLAGPT to correct itself worked well provided you were confident in your own assessment of the system. If you

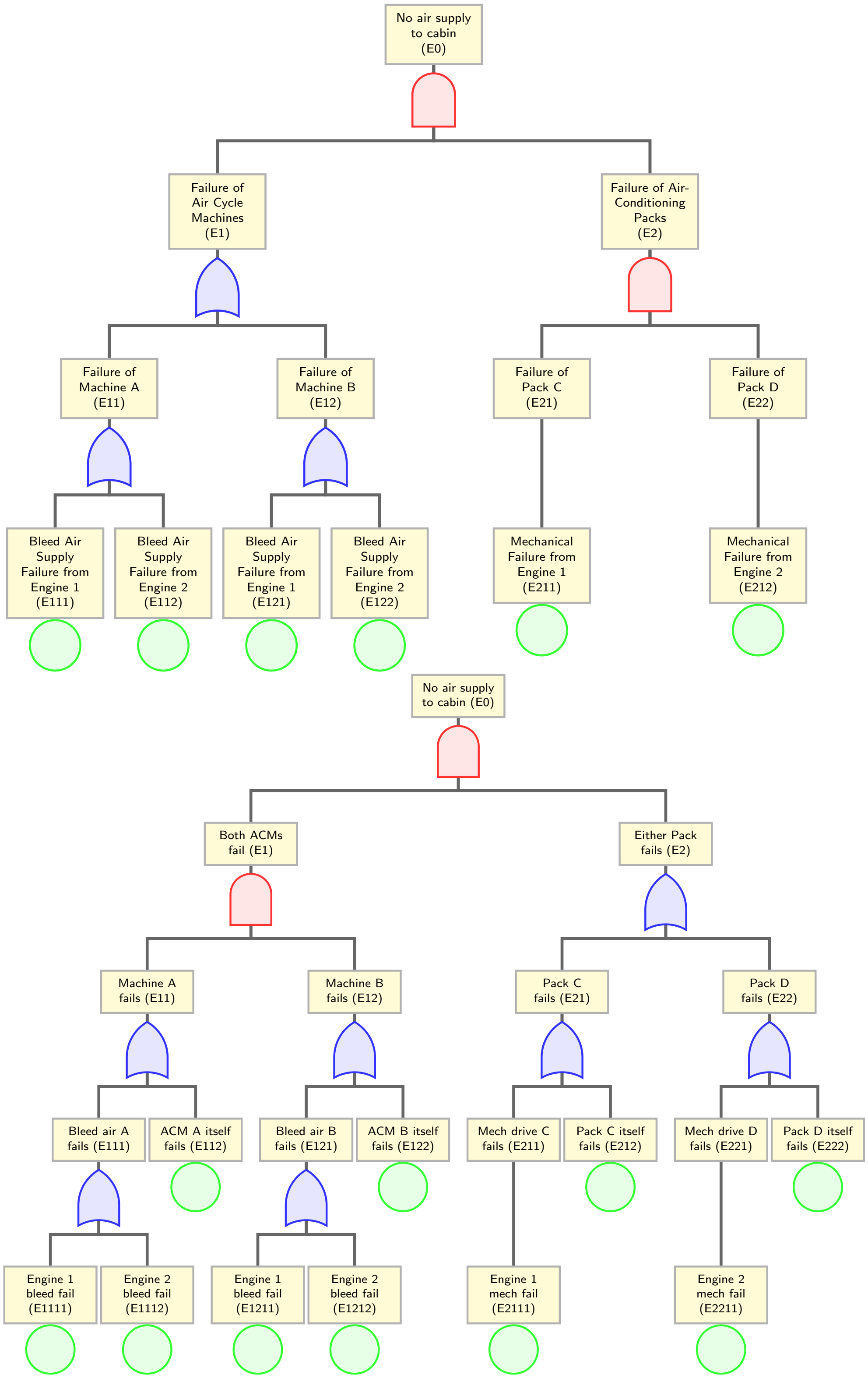

No air supply to cabin (E0)
Failure of Air Cycle Machines (E1)
Failure of Air-Conditioning Packs (E2)
Failure of Machine A (E11)
Failure of Machine B (E12)
Failure of Pack C (E21)
Failure of Pack D (E22)
Bleed Air Supply Failure from Engine 1 (E111)
Bleed Air Supply Failure from Engine 2 (E112)
Bleed Air Supply Failure from Engine 1 (E121)
Bleed Air Supply Failure from Engine 2 (E122)
Mechanical Failure from Engine 1 (E211)
Mechanical Failure from Engine 2 (E212)
No air supply to cabin (E0)
Both ACMs fail (E1)
Either Pack fails (E2)
Machine A fails (E11)
Machine B fails (E12)
Pack C fails (E21)
Pack D fails (E22)
Bleed air A fails (E111)
ACM A itself fails (E112)
Bleed air B fails (E121)
ACM B itself fails (E122)
Mech drive C fails (E211)
Pack C itself fails (E212)
Mech drive D fails (E221)
Pack D itself fails (E222)
Engine 1 bleed fail (E1111)
Engine 2 bleed fail (E1112)
Engine 1 bleed fail (E1211)
Engine 2 bleed fail (E1212)
Engine 1 mech fail (E2111)
Engine 2 mech fail (E2211)

Fig. 4. FLAGPT's fault trees for air bleed cabin supply system using description from Fig. 1. The top tree is before correction is applied through additional prompting.

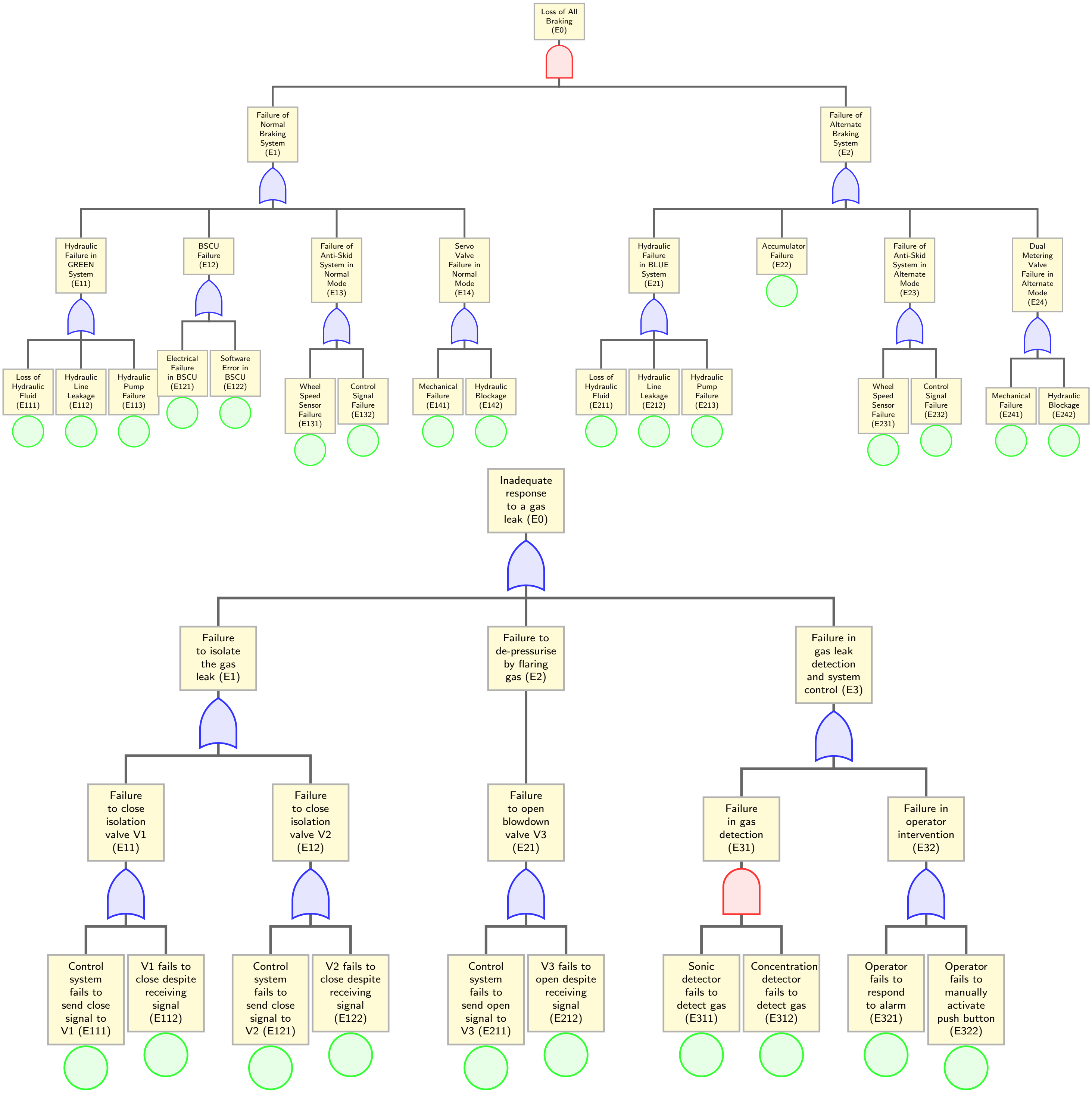

Loss of All Braking (E0)
Failure of Normal Braking System (E1)
Failure of Alternate Braking System (E2)
Hydraulic Failure in GREEN System (E11)
BSCU Failure (E12)
Failure of Anti-Skid System in Normal Mode (E13)
Servo Valve Failure in Normal Mode (E14)
Hydraulic Failure in BLUE System (E21)
Accumulator Failure (E22)
Failure of Anti-Skid System in Alternate Mode (E23)
Dual Metering Valve Failure in Alternate Mode (E24)
Loss of Hydraulic Fluid (E111)
Hydraulic Line Leakage (E112)
Hydraulic Pump Failure (E113)
Electrical Failure in BSCU (E121)
Software Error in BSCU (E122)
Wheel Speed Sensor Failure (E131)
Control Signal Failure (E132)
Mechanical Failure (E141)
Hydraulic Blockage (E142)
Loss of Hydraulic Fluid (E211)
Hydraulic Line Leakage (E212)
Hydraulic Pump Failure (E213)
Wheel Speed Sensor Failure (E231)
Control Signal Failure (E232)
Mechanical Failure (E241)
Hydraulic Blockage (E242)
Inadequate response to a gas leak (E0)
Failure to isolate the gas leak (E1)
Failure to de-pressurise by flaring gas (E2)
Failure in gas leak detection and system control (E3)
Failure to close isolation valve V1 (E11)
Failure to close isolation valve V2 (E12)
Failure to open blowdown valve V3 (E21)
Failure in gas detection (E31)
Failure in operator intervention (E32)
Control system fails to send close signal to V1 (E111)
V1 fails to close despite receiving signal (E112)
Control system fails to send close signal to V2 (E121)
V2 fails to close despite receiving signal (E122)
Control system fails to send open signal to V3 (E211)
V3 fails to open despite receiving signal (E212)
Sonic detector fails to detect gas (E311)
Concentration detector fails to detect gas (E312)
Operator fails to respond to alarm (E321)
Operator fails to manually activate push button (E322)

Fig. 5. FLAGPT's fault tree for 'loss of all braking' from wheel brake system description in ARP4761 (top). Note the missing Emergency Braking redundancy could be corrected by further prompting. The gas leak fault tree below shows separation of the valves and gas detection system (E3), and separate listing of events E111 and E121.

had no prior knowledge of the system design and its failure modes, it would be difficult to trust FLAGPT's analysis.

5 Conclusions

FLAGPT generates fast, flawed 'first drafts' of a failure logic analysis. Providing near instant visualisation helps to see where the analysis is wrong and allows the user to repeatedly query and correct it. Even outwith cases that were consistently misunderstood (e.g. cabin air supply), the quality of the analysis was highly variable, and reformulating FLAGPT's system instructions had little effect on solving misalignment issues due to the underlying model behaviour. There is evidence that LLM reasoning is affected by premise order [3], which infers how we write our system description matters in terms of how well the LLM will reason about failure logic junctions. For future work we intend to break the analysis task into sub tasks, each fulfilled by role-based agents that are overseen by an LLM, to see if this improves consistency and quality of the output.

Acknowledgements. Development supported by Assuring Autonomy International Program funded by Lloyds Register, UK. The system instructions for FLAGPT, system descriptions, LaTeX files and example outputs from FLAGPT are available on our github: `https://github.com/amlas-tool/FLAGPT`.

References

1. Guidelines And Methods For Conducting The Safety Assessment Process On Civil Airborne Systems And Equipment (1996). https://doi.org//10.4271/ARP4761
2. Andrews, J., Lunt, S.: An Introduction to Fault Tree Analysis. `https://www.nottingham.ac.uk/research/groups/resilience-engineering/documents/nxgen/outputs/an-introduction-to-fault-tree-analysis.pdf`
3. Chen, X., Chi, R.A., Wang, X., Zhou, D.: Premise order matters in reasoning with large language models (2024). https://doi.org/10.48550/arXiv.2402.08939
4. Clegg, K. et al: A SysML Profile for Fault Trees—Linking Safety Models to System Design. Springer International Publishing (2019)
5. Lappin, S.: Assessing the strengths and weaknesses of large language models (2023). https://doi.org/10.1007/s10849-023-09409-x
6. Levy, M. et al : The impact of input length on the reasoning performance of large language models (2024). https://doi.org/10.48550/arXiv.2402.14848
7. McDermid, J.: Software hazard and safety analysis. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. Springer (2002)
8. Noy, S., Zhang, W.: Experimental evidence on the productivity effects of generative artificial intelligence (March 2023). https://doi.org/10.2139/ssrn.4375283
9. Shahandashti, K. et al: Evaluating the effectiveness of gpt-4 turbo in creating defeaters for assurance cases (2024). https://doi.org/10.48550/arXiv.2401.17991
10. Shypula, A. et al: Learning performance—improving code edits (2023). https://doi.org/10.48550/arXiv.2302.07867
11. Wei, J. et al: Chain-of-thought prompting elicits reasoning in large language models (2023). https://doi.org/10.48550/arXiv.2201.11903