AI-Supported Eliminative Argumentation: Practical Experience Generating Defeaters to Increase Confidence in Assurance Cases

T. Viger *et al.*, "AI-Supported Eliminative Argumentation: Practical Experience Generating Defeaters to Increase Confidence in Assurance Cases,"
2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), Tsukuba, Japan, 2024, pp. 284-294,
doi: 10.1109/ISSRE62328.2024.00035.

Copyright:

AI-Supported Eliminative Argumentation: Practical Experience Generating Defeaters to Increase Confidence in Assurance Cases

***Abstract*—Assurance cases (AC) are structured arguments that justify why a system is acceptably safe. Though ACs can increase confidence that systems will operate safely and reliably, they are also susceptible to problems such as reasoning errors and confirmation bias. Recent work proposed AI-Supported Eliminative Argumentation (AI-EA), a framework leveraging Generative AI (GAI) models to support AC development by identifying potential reasons why the argument may be invalid (a.k.a. defeaters) so that they can be mitigated. However, this framework was not implemented and its effectiveness was not assessed empirically.**

In this practical experience paper, we implement AI-EA, explain and justify our design choices, and report on our practical experience in empirically evaluating its effectiveness in collaboration with an industrial partner from the safety domain. Our evaluation considers 171 AI-generated defeaters across two industrial case studies from the nuclear and automotive domains. Our findings show that GAI can generate informative defeaters with few significant hallucinations and that 25% of the generated defeaters were confirmed by developers of each AC to represent reasonable doubts or errors in the argument. Our implementation and results are made publicly available.

I. Introduction and Motivation

In safety-critical domains, engineers must demonstrate that their systems operate safely and correctly, as system failures and unexpected behavior can lead to catastrophic consequences. This is often done by constructing *Assurance Cases* (AC). An AC is a comprehensive argument explaining how a system meets its safety objectives (e.g., the mitigation of identified hazards), typically by appealing to a body of evidence artifacts. While ACs can be expressed in various formats, structured modeling languages such as Goal Structuring Notation (GSN) [12] have become popular due to their emphasis on rigorous argumentation. In a structured assurance case, a high-level claim about the system (e.g., "*The system is safe*") is iteratively decomposed through argumentation into simpler, lower-level sub-claims, until those sub-claims can be directly supported by evidence.

ACs play an essential role in ensuring that complex systems are dependable and trustworthy, as they organize a wide array of evidence artifacts into a comprehensive argument showing the role each piece of evidence plays in supporting system safety. However, they are also prone to errors such as incomplete arguments, logical fallacies, and inadequate evidence [11]. These flaws can give stakeholders unwarranted confidence, potentially resulting in the deployment of unsafe systems. *Eliminative Argumentation* (EA) [10] is an AC development methodology that aims to increase the rigor of ACs by explicitly identifying and mitigating potential *defeaters* to their arguments (i.e., reasons why the arguments might not be sound). EA has been demonstrated as a useful methodology for creating robust, rigorous ACs for industrial systems [5], [17]; however, identifying defeaters is a creative process depends on an engineer's skill and expertise in critically analyzing ACs. Developers may omit important defeaters due to confirmation bias or blind spots, reducing the effectiveness of EA. Involving multiple engineers with varied backgrounds and perspectives can help mitigate blind spots in defeater identification, though this requires additional resources and might not be feasible if only a handful of engineers have the technical knowledge about a system.

Recent progress in generative artificial intelligence (GAI) has revolutionized many areas of software engineering, and work is currently underway to evaluate the role GAI can play in supporting software implementation, modeling, testing and verification (see [9] for a recent review). Supporting AC development has also been recently identified as a potential area of application for GAI [20], [22]., though using GAI to support safety assurance activities presents a challenge: Most GAI models are black boxes, making it difficult for engineers to understand why specific outputs are generated and whether they are valid. For this reason, recent work [22] suggested that using GAI to create ACs from scratch has significant risks as GAI hallucinations during AC development can lead to false confidence that a system is safe. Instead, the authors proposed AI-Supported Eliminative Argumentation (AI-EA) an approach that uses GAI models to *provide safety experts with candidate defeaters for their arguments* – an idea also supported by others ([20]). These works outline high-level processes where a GAI model generates a set of potential defeaters for an AC, which engineers can then analyze to identify novel and useful candidates worth investigating. The authors argue that GAI is well-suited for this task for several reasons: First, the extensive data used to train large language models like GPT-4 along with their computational power enable them to compare ACs against large amounts of historical data, potentially allowing them to identify defeaters to an argument that would be in the blind spot of safety engineers. Second, the consequences of a GAI *hallucination* during defeater generation are significantly lower than its potential risks in AC creation. While a hallucinated defeater (i.e., a defeater that is actually not applicable to the AC) may cause false doubts in a system that require a safety engineer's time

to investigate, they do not cause false confidence that could result in deploying an unsafe system. Finally, GAI has been shown as an effective tool for supporting hazard analysis of cyber-physical systems, which has similarities to AC defeater identification [6]. However, these works [20], [22] do not implement their approaches and empirically evaluate their effectiveness on industrial case studies. This is a limitation for practitioners that need to understand the benefits and limitations of these technologies when applied in practice.

To address this problem, in this practical experience paper, we present an implementation of AI-EA. We explain and justify the design choices of our implementation and report on our practical experience empirically evaluating its effectiveness in collaboration with an industrial partner from the safety domain. Our evaluation includes an analysis of 171 AI-generated defeaters across two medium-sized industrial ACs from the nuclear and automotive domains, containing 506 and 317 nodes respectively. Our results show that AI-EA generated defeaters that were relevant and informative, that less than 6% of defeaters were significant hallucinations, and approximately 25% of them corresponded to real errors and risks. We made our implementation and results publicly available.[1]

This paper is organized as follows. Section II provides background information on ACs, EA, GAI, and AI-enabled EA (AI-EA). Section III presents our implementation of the AI-EA framework. Section IV discussed the results of our empirical evaluation. Section V presents threats to validity. Section VI summarizes related work. Section VII concludes with a summary and future directions.

II. Background

A. Assurance Cases and Eliminative Argumentation

ACs are structured arguments, typically supported by evidence, that a system will adequately achieve its intended quality attributes (e.g., safety, security, reliability) [12]. ACs may be expressed in different formats, including narrative text, graphical tree-like notations (e.g., GSN [14], EA [10]), or some combination thereof. In this work, we used Eliminative Argumentation (EA) due to its focus on defeaters; however, other notations also incorporate similar elements (e.g., GSN "challenges" [12]). An AC expressed in EA recursively decomposes a top-level claim into sub-claims and supporting evidence. Defeater nodes enable engineers to express doubts about the validity of an argument, and can be used to challenge claims, evidence and reasoning steps of the argument. Defeaters may be mitigated by additional argumentation. If a defeater cannot be mitigated, it is a source of "residual doubt" in the AC. As sources of doubt (defeaters) are identified and eliminated, confidence in the argument increases. EA has been applied to several industrial-scale projects in the automotive, rail, industrial control system domains [5], [13], [17], and practitioner experience has shown EA to be an effective approach for increasing confidence in ACs [5]. For instance, a recent paper by Hobbs et al. describes the use of EA to gain confidence in the safety case for a "startup function" that initializes a real-time operating system for use in automotive control systems [13].

Figure 1 presents an AC fragment created using the EA modeling tool *Socrates*[2], from a case study in the automotive domain. The fragment considers the response of an Adaptive Cruise Control (ACC) system to failures of dependent subsystems, namely the braking system and the radar system. The main claim (Claim `c5550`) is that the ACC will disengage when the braking or radar systems fail. A strategy node (`S5552`) decomposes the main claim by considering sources of doubt that might undermine it. Two sources of doubt are shown: the possibility that the ACC's software controller does not disengage the system upon receiving an error report (`D5553`) and the possibility that the braking or radar monitoring systems do not report an error (`D5564`). To eliminate `D5553`, a software safety requirement for the ACC's software controller is expressed as a claim (`C5554`) which is then supported by verification results (`E5555`, `E5558`). However, the quality of the evidence is undermined by a further defeater that questions whether the tests cover the appropriate conditions (`D5556`). In this example, no counter-argument is made to resolve this defeater, and it remains a source of *residual doubt* (`Res`) in the argument.

B. AI-Supported Eliminative Argumentation

AI-Supported Eliminative Argumentation (AI-EA) [22] has been recently proposed to support engineers in AC design. Figure 2 shows the AI-EA framework, which consists of four stages: *Design*, *Generate*, *Filter*, and *Investigate*. The *Design* stage requires safety analysts to design the AC using conventional methods (e.g., EA). The *Generate* stage provides the AC and relevant background information to the GAI model, and prompts the model to generate potential defeaters. The *Filter* stage requires safety analysts to filter defeaters that are not useful (i.e., defeaters that are not legitimate doubts because they are irrelevant, inconsistent or contain incorrect information). The *Investigate* stage requires safety analysts to investigate the remaining defeaters and document their mitigation (or lack thereof) in the AC. For example, consider a safety engineer developing an AC for the ACC system. During the *Design* phase, the engineer may create the AC fragment depicted in Figure 1, but overlook the defeater `D5556`, which is annotated with a special dashed red outline. In the *Generate* phase, the engineer provides background information describing the ACC and a textual representation of the argument fragment in Figure 1 to the GAI model. She then prompts the model to identify potential defeaters in the fragment. AI-EA returns two defeaters about the unit tests:

1) *What if the ACC does not re-engage after the unit tests complete?*
2) *What if the unit tests did not exercise all conditions in the source code?*

[1] https://drive.google.com/file/d/1U3VrFinQ5NpQCtNmNQ6ZF_FTBw8LWoD7/view?usp=sharing

[2] https://criticalsystemslabs.com/socrates

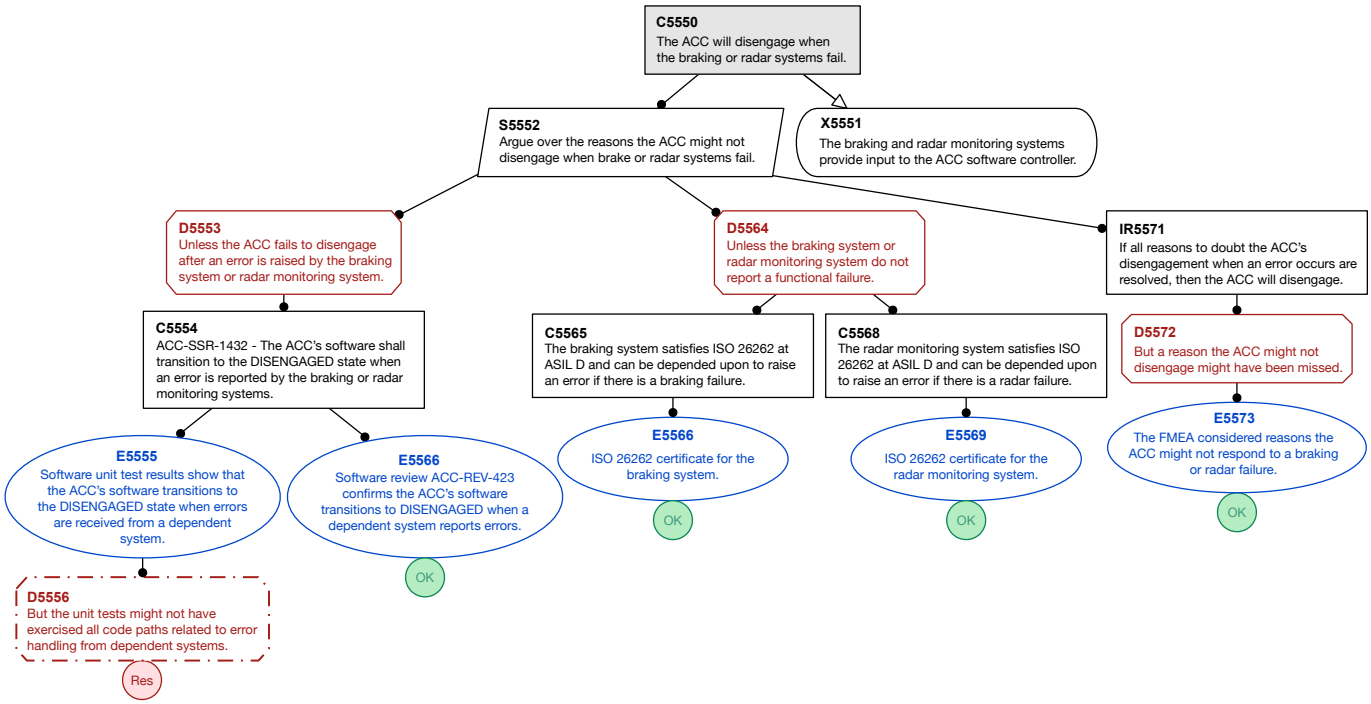

Fig. 1. AC fragment adapted from the ACC case study.

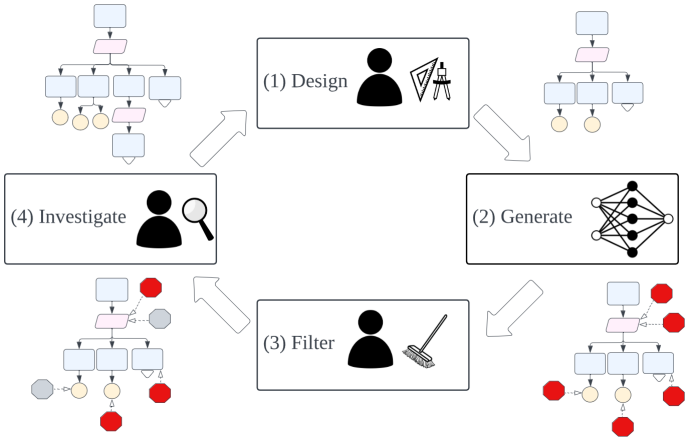

Fig. 2. Overview of the AI-EA framework, as presented in [22].

During the *Filter* phase, the engineer recognizes defeater (a) as a hallucination and filters it out. However, defeater (b) expresses doubt about the adequacy of test evidence, which is consistent with the argument fragment and does not appear to be a hallucination. She thus creates a new defeater (Figure 1) to capture this concern, *Investigates* whether this defeater represents a reasonable doubt to the system and, if so, whether it can be mitigated. In this example, a potential mitigation could come from test coverage analyses indicating that full path coverage has been achieved. If mitigation of the defeater cannot be demonstrated, it should then be documented in the AC as a source of residual uncertainty, as in Figure 1.

III. Implementing AI-EA

Figure 3 presents our implementation of the AI-EA framework. Our implementation focuses on the *Generate* step (i.e., leveraging GAI models to identify defeaters for an existing AC) and the *Filter* step (i.e., processing generated defeaters to provide a set of reasonable candidate defeaters to investigate). We do not restrict how the ACs should be developed during the *Design* step; any AC represented in GSN or EA notation can be used as input. The output of our implementation is a set of candidate defeaters to be investigated by domain experts in the *Investigate* step.

A. Defeater Generation

The *Generate* phase uses GAI to generate a set of potential defeaters to a given AC. Our implementation uses the GPT-4 large language model [18]. To interact with the GAI model, our implementation uses three separate prompts. Prompt 1 takes input from the AC developer to provide background information about the system under analysis and the AC argument in question to the GAI model. Prompt 2 generically asks the model to identify reasons to doubt the provided argument fragment without providing contextual information about what type of defeaters should be generated or what they should be used for, and Prompt 3 asks the model to identify defeaters in the argument fragment after providing contextual information about EA, the role that defeaters play in supporting AC development, and describing different categories of defeaters.

Prompt 1: Background information. The input used to instantiate Prompt 1 templates consists of two parts: 1) a textual description of the system under analysis, and 2) a fragment of

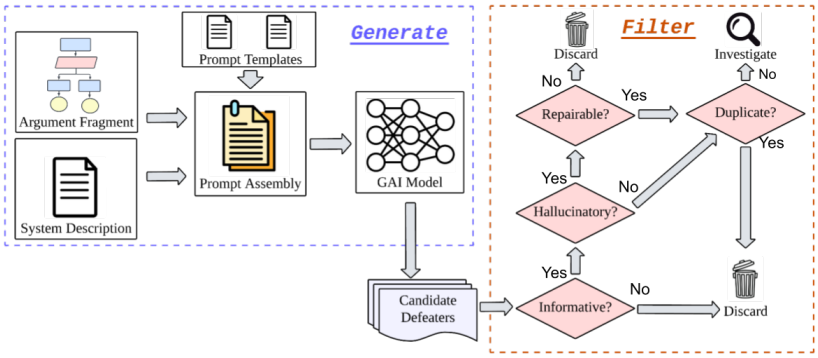

Fig. 3. Overview of our implementation of AI-supported Eliminative Argumentation.

the AC argument that captures a single reasoning step (i.e., the decomposition of a parent claim into sub-claims). The system description includes the purpose of the system, its operating environment, its components, and their relations. The argument fragment consists of a parent claim, a decomposition strategy, its sub-claims, and any context/assumption/evidence nodes directly connected to those subclaims.

Prompt 2: Identify Generic Doubts. This prompt generically asks the model if there are potential reasons to doubt the AC fragment that was provided as input in Prompt 1. Prompt 2 does not provide any background information on EA or its role in safety assurance. The prompt is given as follows:

> **P2:** *"Are there any potential reasons to doubt this assurance case? List any identified doubts in the form ⟨⟨ Doubt 1: ...⟩⟩, beginning with the symbols "⟨⟨" and ending with the symbols "⟩⟩", with one doubt per line. Where possible, highlight the following three components in each doubt: The *what*, meaning the identified potential flaw in the system or argument; the *where*, meaning the portion of the argument or component of the system which is the subject of the doubt; and a *why*, which is an explanation of why this argument could be defeated by the identified doubt."*

The word "doubt" was used instead of "defeater" for this prompt, as preliminary experiments revealed that GPT-4 would frequently fail to find doubts in the arguments when the word "defeater" was used without providing context. This is likely because the word *"defeater"* can take on slightly different meanings in different contexts, and one generic definition of a defeater is *evidence that a belief is false*, whereas in the context of Eliminative Argumentation, defeaters include *potential* weaknesses that could defeat the argument if not appropriately mitigated.

Prompt 3: Identify Defeaters Given Context about EA. In contrast to Prompt 2, Prompt 3 adds a preamble which defines EA, the role that defeaters play in supporting AC development, and different defeater types. The description of EA was provided to remove potential ambiguities around the words "doubt" and "defeater", and to give further context about how defeaters are used in safety analysis. Prompt 3 repeats Prompt 2 while replacing the word "doubt" with "defeater", and adding the following preamble:

> **Preamble (P3):** "*Eliminative argumentation is a methodology that aims to increase confidence in assurance cases by identifying doubts in an argument (called "defeaters") and showing how they have been mitigated. There are several different types of defeaters: A rebutting defeater identifies a potential reason why a claim could be false, an undermining defeater identifies a potential reason why evidence could be invalid, and an undercutting defeaters identify potential reasons why the truth of an argument's child claims is insufficient to determine whether its parent claim is true or not.*"

Prompt 3 gives contextual information that may help the model understand the types of defeaters it should produce; however, providing this context may also cause it to omit certain defeaters which represent valid concerns but do not correspond to the model's understanding of the task. For example, if the model erroneously interpreted the preamble in Prompt 3 to mean that all identified defeaters should increase confidence in the AC, it may then avoid generating defeaters that identify a blatant error in the argument. Prompt 2 is used in conjunction with Prompt 3 which leads to increasing the breadth of possible defeaters the model can generate and to mitigate potential omissions if a misinterpretation like this occurs. The choice to aggregate results generated by Prompt 2 and Prompt 3 was informed by preliminary experiments which indicated that each prompt yields useful but different results.

In Prompts 2 and 3, the model is told to structure its responses into three parts: a "*What*" component, a "*Where*" component, and a "*Why*" component. The "*What*" component contains either an event, scenario, behavior, or aspect of the system under study or a flaw in the AC argument. The "*Where*" component references one or more aspects of the system under study, or one or more nodes in the AC argument. The "*Why*" component explains why the entity identified in the "*What*" component could pose a credible threat to the AC argument. For example, defeater `D5556` from Figure 1 could be expressed by the model as below.

What: Insufficiency of unit tests to demonstrate correct error handling.
Where: In CHILD EVIDENCE `E5555`.
Why: Without evidence that sufficient coverage has been achieved, it is possible that the test results may not have exercised code paths related to error handling, thereby weakening the evidence supporting the correctness of error handling.

We selected this format for Prompts 2 and 3 after approximately 10 trial-and-error experiments that qualitatively assessed the results from different prompts. We found that explicitly separating the "What", 'Where," and "Why" components helped mitigate the generation of wordy and rambling defeaters that would sometimes occur otherwise. Additionally, this format gave us a more structured approach to identifying hallucinations. Hallucinations can be detected in this format by (i) ensuring that the information provided in each component is accurate, and (ii) ensuring that components are consistent and relevant to each other.

B. Filtering

The filtering step concerns the manual inspection of the generated defeaters to determine which (if any) are worthy of further investigation. Our implementation of AI-EA uses a three-step filtering process.

Step 1: Dismiss Uninformative Defeaters. To be worth investigating, a defeater must be sufficiently informative and concrete to indicate to the AC developer the potential issue and where they should focus their attention to assess it. Generic residual uncertainties, such as the defeater "There may be an unconsidered failure that could defeat the argument" do not provide concrete information about what types of failures could arise and where/why they might defeat the argument and can be filtered out. In contrast, consider the defeater "Claim `C1040` does not consider the potential for lapses of control when the ACC transitions from a Disengaged to an Engaged state. These transitions could reduce vehicle safety due to driver confusion". This defeater indicates a specific doubt, where it impacts the argument, and why it may defeat the argument if not mitigated, giving engineers a concrete concern to investigate.

In general, the usefulness of defeaters depends on context, and it relies on a reviewer's subjective assessment to determine whether the defeater is sufficiently informative. In our implementation, reviewers filter uninformative defeaters by scoring each generated defeater based on the extent to which they provide a concrete *What*, *Where* and *Why* component.

Step 2: Identify (and Repair) Hallucinations. Generative AI models can produce hallucinations: outputs that appear convincing but contain errors or fabricated information. When a hallucination is identified, then either (i) the defeater contains a minor error or discrepancy, but the reviewer can easily identify what the defeater *should* be saying; or (ii) the defeater contains a significant error or inconsistency, and it is either meaningless or difficult to interpret. For these cases, the engineer can respectively fix or discard the defeater.

Step 3: Identify Duplicates. As GAI models are non-deterministic, repeating the same prompts can cause the model to identify new defeaters; however, this comes with the trade-off that duplicate defeaters can also be produced. Once boiler-plate and severely hallucinatory defeaters have been dismissed, duplicates must also be filtered out. There are two kinds of duplicate defeaters: duplicates within the set of generated defeaters itself (i.e., two generated defeaters identifying the same doubt), and defeaters that are already covered in a portion of the real AC (which may or may not have been provided as input to the model). Duplicates of both kinds should be filtered out to avoid redundant investigation, especially if AC development is being performed collaboratively (as is often the case). Our evaluation of AI-EA has reviewers manually evaluate defeaters to identify duplicates, though this stage could be partially automated using syntactic heuristics to identify potential duplicates. The GAI model itself may also be able to provide support for identifying and filtering duplicates, though we have not yet incorporated this into our implementation.

After performing these steps, the remaining defeaters should be novel, informative, and hallucination-free, and thus provide a reasonable starting point for further investigation. It then falls to the engineer to assess which, if any, of these defeaters pose credible risks to the argument and require mitigation.

IV. Evaluation

For AI-EA to be effective in supporting EA, the GAI model must be able to generate useful defeaters that survive the *Filtering* stage. Evaluating the effectiveness of AI-EA requires a method for assessing the usefulness of AI-generated defeaters. This is challenging as there are no established frameworks for evaluating the usefulness of defeaters or comparing defeaters against one another. Assessing whether a defeater is sufficiently useful to include in an AC is a largely subjective process based on an engineer's experience; however, as noted in Section III, there are some criteria that can strongly suggest that a defeater is *not* useful, such as if it is uninformative or not consistent with the AC and its associated system. We first evaluate our implementation of AI-EA by considering the following three research questions, which measure the informativeness, coherence and usefulness of defeaters, respectively:

RQ1 (Informativeness). How frequently do GAI-generated defeaters *concretely* describe "*What*" the identified doubt is, "*Where*" it impacts the system/argument and "*Why*" it could defeat the argument? (Section IV-A)

RQ2 (Coherence). How frequently do generated defeaters contain hallucinations, and how severe are they? (Section IV-B)

RQ3 (Usefulness). To what extent can GAI models generate defeaters that give useful insights to safety engineers? (Section IV-C)

In addition to evaluating the quality of individual defeaters, we also measure whether the *semantics* of the defeaters – that is, the concepts and entities they refer to – vary based on the

system description and target argument provided to the GAI model. One of the main potential benefits of AI-EA is that safety engineers can have blind spots and biases related to the systems they develop, and consequently, they may overlook or omit important defeaters. If GAI models continually produce the same type of defeaters regardless of the input they are given (e.g., only questioning the validity of evidence), this would significantly limit the usefulness of AI-EA and indicate that the model is not actually considering its input as part of its decision-making. We assess whether the concepts referred to in the defeaters are appropriate in the context of the provided inputs with the following question:

RQ4 (Semantics). Does GAI identify the same types of defeaters across every system/argument, or can they generate different types of defeaters depending on the system description and strategy provided as input? Do these variations reflect the contents of its inputs? (Section IV-D)

Models and Methods. Our study subjects are the *Adaptive Cruise Control (ACC)*, and the *CERN Large Hadron Collider (LHC) Machine Protection System (MPS)*[3] [17].

The ACC case study is an AC developed a company in the safety domain with experience assessing safety and security risks associated with complex hardware/software-intensive systems. A portion of the ACC argument was presented in Section II. The full AC is publicly available and was designed to be representative of confidential ACs that have been previously developed for real systems in the automotive domain. [4]. The argument for the ACC was expressed using EA and is of small-to-medium size, consisting of 317 nodes. It contains 10 strategies, four of which were analyzed in our evaluation. It took the creators between two and three months to create the ACC model.

The CERN LHC case study is an AC from the nuclear domain related to the CERN particle accelerator [15]. The AC concerns the Machine Protection System (MPS) responsible for protecting the LHC from damage during operation. The argument contains 506 total nodes and 32 strategies, five of which were analyzed in our evaluation. The MPS AC was developed over a three month period as a collaboration between academic researchers and a company from the safety domain, and validated by experts from the European Organization for Nuclear Research (CERN)[5].

We used the state-of-the-art large language model GPT-4 [18] to generate 171 defeaters across nine different argument fragments from our two study subjects. These fragments were selected from various levels of each AC, ranging from top-level to low-level arguments with claims directly connected to evidence. For each argument, we performed two rounds of defeater generation following the process outlined in Section III-A, which were then aggregated. Our evaluation was conducted by three authors, with two authors evaluating each case study. Each of these authors has more than three years of experience researching ACs, and one author was directly involved in creating each AC.

[3]https://cds.cern.ch/record/2854725?ln=en

[4]https://safetycasepro.com/examples/ACC/

[5]https://home.cern/

A. RQ1. Informativeness

Methodology. To evaluate the informativeness of GAI-generated defeaters, we assessed the extent to which each defeater was able to provide a concrete *What*, *Where* and *Why* component. Two authors assigned a score between zero and two to the "what", "where", and "why" components of each defeater (as described in Section III-B) based on whether the component was absent, present but generic, or present and concrete. As an example of how defeaters were scored, consider the following defeater:

> "`ACC D66:`
> *What: The failure modes of the ACC when disengaged could have been overlooked.*
> *Where: In PARENT CLAIM* `6000`*, where it is claimed that the ACC does not reduce the safety of the vehicle while disengaged.*
> *Why: The assurance case does not study all potential failure modes of the ACC when disengaged. Some failure modes could affect safety, either directly or indirectly, even when the ACC is disengaged.*"

Reviewers assigned a score of one to the "*What*" component of this defeater as, while it indicates that failure modes could have been overlooked in a particular setting (i.e., while the system is disengaged), it does not indicate what these overlooked failure modes might be. The "*Where*" component was given a score of two, as the defeater precisely indicates a specific claim that was the subject of concern. The "*Why*" component was given a score of one, as though the defeater indicates that these overlooked failure modes are relevant to safety and could defeat the argument, it does not concretely state what aspects of the system's safety could be impacted by these failures. Summing these scores yields a value of four out of six, which gives an approximate measure of the defeater's informativeness.

Results. The average scores for each component of each defeater are given in Table I. The scores were calculated as the average given by the two reviewers. Across all 171 defeaters, only 19 received a summed score of less than 4/6. In contrast, 88 received a score of 5.5/6 or greater; the following defeater is one such example:

> "`CERN D43:`
> *What: Redundancy of the BLMS components.*
> *Where: CHILD INFERENCE* `110`*.*
> *Why: While the assurance case mentions redundancy within the Machine Protection System (MPS), it fails to address the specific redundancies within the BLMS component. Without knowing the redundancy measures in place for the detectors, Tunnel Electronics and Surface Electronics, we cannot be assured of the system's ability to handle any malfunctioning components of the BLMS properly.*"

We also observed that the average scores across each component were greater than 1.5/2. This shows that (i) each

TABLE I
THE AVERAGE CLASSIFICATION SCORE OF EACH COMPONENT OF EACH DEFEATER.

System	Avg. What	Avg. Where	Avg. Why
CERN	1.68	1.75	1.68
ACC	1.66	1.54	1.70

component was usually present and concrete in each defeater, (ii) nearly half of the generated defeaters included concrete *What*, *Where* and *Why* components, and (iii) only a very small portion of the generated defeaters contained no concrete information. We observed that the few generated defeaters with very low informativeness scores ($\leq$2.5) mostly corresponded to (i) hallucinations, and (ii) extremely generic concerns about the argument itself, e.g., asserting that a claim in the argument could be incorrect, but not indicating which claim or what issues it may have.

The two AC reviewers gave the same score to 67.4% of defeater components across both ACs. Reviewer scores had a difference of 1 for 30.7% of defeater components, and only $\tilde{1}$.7% of components were assigned a score of 2 by one reviewer and 0 by the other. This result suggests that while measuring the informativeness of GAI-generated defeaters is subjective, reviewer assessment is consistent.

B. RQ2. Coherence

Methodology. Two authors analyzed the dataset to assess whether each defeater contained a hallucination and, upon identifying a hallucination, recorded (i) the component of the defeater that the hallucination pertained to (i.e., the "*What*", "*Where*", or "*Why*" aspect); (ii) the nature of the hallucination, i.e., whether the hallucination was inaccurate (i.e., producing false output) or irrelevant (i.e., producing output not related to a doubt in the AC); and (iii) whether the defeater was still comprehensible and applicable to the AC despite the hallucination (i.e., a "*minor*" hallucination), or if the hallucination caused the defeater to be inapplicable to the AC (i.e., a "*major*" hallucination).

Results. For each defeater component, Table II shows the percentage of defeaters from our dataset that contained a hallucination in that component. It also highlights the percentage of defeaters that contained minor and major hallucinations and the defeaters that contained inaccurate and irrelevant hallucinations.

The results show that for the ACC and the CERN ACs respectively, 13 out of 74 and 20 out of 97 defeaters contained a hallucination in at least one component of the AC. Of these 33 hallucinations, 24 were minor issues that did not significantly impact the reviewers' ability to understand the defeater or its relevance to the AC. For example, the ACC defeater

> "*The verification of the ACC's initial state being Disengaged doesn't appear to consider the possibility of a system bug or malfunction causing it to change state uncommanded. Child evidence* `1012` *only demonstrates the initial startup state, not what might happen afterwards. This could cast doubt on Child Claim* `1011`."

identified a reasonable doubt in the argument (i.e., that an uncommanded state changes could occur), but was classified as a *"Minor/Inaccurate/Where"* hallucination as it attributes this doubt to the wrong claim. `C1011` ("The initial state of the ACC shall be Disengaged.") only makes an assertion about the system's initial state.

Though most hallucinations were minor, 9 of the 33 identified hallucinations produced output that was either incomprehensible or misinterpreted key details about the system/argument that made them inapplicable to the AC. For example, the ACC assurance case defeater

> "`ACC 41`*: The assurance case does not appear to address the transition periods between Engaged and Disengaged, or vice versa. These transitions might be critical in terms of safety. If the system latency during transitions leads to safety concerns, it could defeat claims like Child Claims* `8000` *and* `7000`"

was classified as a *"Major/Inaccurate/What"* hallucination, as the purported issue, i.e., that the AC does not address transitions between Engaged and Disengaged states, is false. The fragment of the AC given to the model contained claims addressing these exact transitions.

While GAI models clearly have the potential to misinterpret the input they are given and generate false/incomprehensible defeaters, our results show that major hallucinations are few and far between. Most of the identified hallucinations pertained to minor details that could be easily identified and repaired by AC reviewers, e.g., defeaters that identified a valid concern but attributed it to the wrong claim in the argument.

C. RQ3: Usefulness

Methodology. Two authors assessed each defeater to identify those that represented reasonable doubts that could strengthen the argument if mitigated. These assessments were done subjectively based on the authors' domain expertise and their knowledge as developers of the ACs. A discussion was then held between the reviewers to produce a single list of useful defeaters.

Results. For the ACC assurance case, we identified that out of the 44 unique defeaters examined (excluding 30 duplicate defeaters), 15 raised realistic doubts not directly covered in the model's input. For the CERN AC, 18 out of 78 unique generated defeaters were useful. For example, a useful defeater for the CERN AC correctly identified that one of the argument's strategies did not consider the speed of communication between the subsystems it argues over, and the parent claim only holds under the implicit assumption that no communication delays occur. A useful defeater for the ACC accurately identified limitations in the argument's evidence, which only tested the system's integration with typical hardware components.

TABLE II
THE PERCENTAGE OF DEFEATERS CONTAINING HALLUCINATIONS OF THE SPECIFIED TYPES.

System	Defeater Component			Severity		Types	
	What	Where	Why	Minor	Major	Inacc.	Irrel.
MPS	4.2%	10.3%	6.2%	16.5%	4.2 %	11.3%	9.3%
ACC	9.4%	2.7%	5.4%	10.8%	6.7%	12.1%	5.4%

The GAI model also produced several defeaters that identified real errors in the ACs. For example, consider the following defeater:

> "`CERN 35`**:**
> What*: The actual time required for activating the MKD magnets.*
> Where*: CHILD EVIDENCE `304`: Design specifications of MKD transmission lines show that magnet activation time is $>$ 2.8 microseconds (Sec. 17.3.1).*
> Why*: The assurance fragment mentions that the MKD magnet activation time is "greater than" 2.8 microseconds, but does not specify by how much. Thus there is a possibility of it potentially taking longer than the 92 microseconds stated in the parent claim*".

The defeater correctly identifies that `Evidence 304` gives no upper bound on the activation time for one of the system's magnets. In reality, the node `Evidence 304` contained a typo: the AC developer intended to write "$<$ 2.8"rather than "$>$ 2.8". This exemplifies the model's ability to identify logical flaws that human AC developers may miss.

In summary, approximately one-quarter of the GAI-generated defeaters (excluding duplicates) identified realistic doubts that would strengthen the argument if included and mitigated. These defeaters ranged from unconsidered hazard scenarios to unstated implicit assumptions and concrete errors in the AC's nodes.

D. RQ4. Semantics

Methodology. We created a taxonomy of semantic topics for the generated defeaters in an incremental fashion. We took an initial, random sample of defeaters and identified a set of recurring themes (e.g., "synchronization issues", "inadequate evidence"). We then attempted to map each defeater from the dataset to one of these topics. If no matching topic was present in the taxonomy, a new class was added, and previous classifications were revised to maintain consistency. This process was conducted separately by two authors, who then compared their results to collectively arrive at a consistent classification scheme for the generated defeaters. Once our classification scheme was finalized, we counted the distribution of each defeater class for each input argument of both ACs.

Results. Table III presents the relative frequency of the topics of the generated defeaters. The arguments studied for the MPS and ACC are numbered #1-#5 and #6-#9, respectively. We observed variability in the semantics of generated defeaters at both the *system* level and the *argument* level. At the system level, we see a significant degree of variance between the defeaters generated for the MPS versus the ACC, e.g., some topics appear frequently in defeaters of one AC and do not appear in the defeaters of the other at all. The system descriptions of the MPS and its AC emphasize its component-based architecture, so it is natural to propose defeaters related to component interactions (e.g., communication delays or synchronization errors). Conversely, the model frequently proposes defeaters related to the black-box nature of the ACC's sensor system, whereas there are no black-box components in the MPS. At the argument level, we can also observe semantic variability, albeit to a lesser degree. We believe that being the MPS an industrial, complex system leads to a higher variability. We find that these variations between each system's arguments are sensible. For example, argument #1 is an argument over the primary components of the MPS; therefore a higher number of defeaters refer to communication and synchronization issues between the components. Argument #3 pertains only to the timing of a single component's actions, hence there is an overt emphasis on delays rather than the integrity of communication between components.

Our results assuage the concerns that either (a) the content of the input has no effect on the semantics of the generated defeaters, or (b) the semantic relationships between the input and output are random or incoherent.

V. LESSONS LEARNED AND THREATS TO VALIDITY

Lessons Learned. Our results show that the GAI model generated a high portion of informative, non-hallucinated defeaters that should not be dismissed in *Filter* stage of AI-EA (**RQ1** and **RQ2**). Approximately one quarter of the non-duplicated defeaters identified realistic doubts that would strengthen the argument if mitigated, including errors in each AC that were previously unnoticed (**RQ3**). None of the defeaters identified as *useful* for the ACC assurance case received a combined informativeness score of less than 4.5, and 13 of them received a score of higher than 5.5. This suggests that, while the presence of concrete "*What*"/"*Where*"/"*Why*" components is not sufficient for a defeater to be useful, the absence of one or more components is strongly correlated lack of usefulness of a defeater. Filtering out defeaters with low informativeness scores is thus unlikely to cause engineers to overlook useful defeaters. Finally, the semantics of the generated defeaters are meaningfully connected to the content of the argument fragment passed as input to the model (**RQ4**). Our results demonstrate that AI-EA is capable of supporting

TABLE III
RELATIVE FREQUENCY OF TOPICS MENTIONED IN GENERATED DEFEATERS (EXCLUDING SEVERE HALLUCINATIONS AND DUPLICATES). SOME DEFEATERS CORRESPOND TO MORE THAN ONE TOPIC, SO COLUMNS DO NOT ALWAYS ADD TO 100%.

Topic	MPS Arguments					ACC Arguments			
	#1	#2	#3	#4	#5	#6	#7	#8	#9
Single point of failure	17%	10%	–	–	–	–	–	–	–
Lossy communication	8%	10%	–	–	13%	–	–	–	–
Delayed communication	42%	10%	14%	–	13%	–	–	–	–
Synchronization/delays	25%	10%	57%	10%	–	–	–	6%	–
Redundancy	8%	10%	–	10%	–	–	–	–	
Operating conditions	–	10%	14%	20%	13%	8%	9%	31%	13%
Black box uncertainty	–	–	–	–	–	25%	27%	13%	13%
Maintenance/monitoring	–	10%	–	10%	–	8%	–	–	–
Term/concept undefined	–	10%	–	10%	–	17%	18%	6%	–
Robustness unspecified	–	–	–	–	25%	–	–	–	–
Data accuracy	8%	–	–	–	13%	–	–	–	–
Human error	8%	–	–	–	–	–	–	19%	–
Unintended interactions	–	10%	–	10%	–	–	9%	–	–
Insufficient evidence	17%	10%	–	20%	–	17%	36%	13%	25%
Infeasible claim	–	–	–	–	–	17%	9%	–	–
Unspecified failure	16%	–	14%	10%	25%	8%	9%	13%	50%

practitioners in AC development by identifying a diverse range of relevant and nontrivial defeaters.

We also observed that, while defeaters with major hallucinations were not directly useful, the process of assessing their severity sometimes led to interesting discoveries in the argument. For example, assessing a hallucinated defeater for the ACC system led to a discussion between authors which ultimately resulted in a contradiction being identified in the argument. It is possible that the presence of certain types of errors in an AC will cause GAI models to produce specific types of hallucinations. Further study into these correlations (e.g., by analyzing AI-generated defeaters for ACs that contain intentional errors) may enable GAI hallucinations to play a direct role in identifying issues within ACs.

There are several techniques that can be applied to optimize the performance of GAI models for specific tasks. Models can be fine-tuned by ingesting additional documents and benchmark examples; however, this is not currently feasible in the context of AI-EA as there are very few publicly available ACs [16], and there are no established "gold-standard" benchmark EA ACs that are known to contain no errors. Ingesting additional documents may become more feasible as the library of publicly available ACs – especially those developed using EA – increases. We do not claim that the GAI model and prompts used in our implementation of AI-EA are optimal; instead, our work demonstrates the effectiveness of using GAI models out-of-the box with a reasonable set of prompts to generate defeaters in practice, and provides a set of evaluation criteria for AI-generated defeaters that can be used to facilitate further optimization.

One limitation of AI-EA is the manual effort required to identify and filter duplicates when a large number of defeaters is generated. In our experiments, we found that it was useful to repeat the same set of prompts to the model multiple times due to variations in the model's output. More repetitions are likely to generate a wider breadth of defeaters, but also increase the amount of manual effort required to identify and filter duplicates. In our implementation, we prompted the model twice for each argument fragment as we noticed that further prompts yielded diminishing returns, though it is possible that additional useful defeaters could have been generated with more repetitions. Techniques to partially automate the filtering of duplicates can be used to support more prompt repetitions, such as using syntactic heuristics to identify potential duplicates or by leveraging the GAI model itself to filter the generated dataset for duplicates.

Another limitation of AI-EA is that practitioners may have concerns related to data privacy when interacting with GAI models, as industrial ACs often contain proprietary information. This concern can be mitigated by sanitizing input to the model by removing confidential details or presenting these details at a higher level of abstraction, or by using a local GAI model. There were no data privacy concerns for our evaluation as all case studies we used were in the public domain.

Threats to Validity. Evaluating defeaters is a subjective task that depends on the judgment and experience of reviewers, and it is possible that another group of evaluators would have arrived at different results than ours. We mitigated this concern in three ways: two authors independently evaluated each defeater; we used a structured method to evaluate specific concrete aspects of each defeater (e.g., "*What*", "*Where*", "*Why*", whether the defeater was a hallucination, etc.); and we computed measures of agreement among independent reviewers. We conducted two rounds of defeater generation, with each round using two prompt templates; more or fewer rounds may have generated different results. Our study is based on one AC from the nuclear domain and one AC from the automotive domain developed by a safety company. ACs prepared in another context or by different authors may use different terminology or rely on different argument structures. This study was performed using GPT-4, with its default API configuration. Another model could have produced

different results. The rapid advancements in GAI models mean another model might become available shortly with better performance. Our main contribution is providing a baseline of empirical results that establish the feasibility of the AI-EA framework based on our practical experience generating and evaluating defeaters for real ACs, as well as providing a scheme for evaluating AI-generated defeaters that can be used to optimize AI-EA by measuring the effectiveness of different implementations.

VI. Related Work

The use of GAI models as tools for defeater identification was investigated by Viger et al. [22]. As mentioned in Section I, their proposal relies on GAI to generate a set of potential defeaters, which are then manually filtered by human experts. The present work significantly extends this proposal, providing structured prompts for defeater generation, evaluation criteria for manual filtering, and empirical results based on industrial case studies.

Shahandashti et al. [20] proposed using GAI for AC development through eliminative argumentation (EA) by focusing on three objectives: (i) to determine whether GPT-4 is "aware" of EA syntax and semantics (which the paper focused on); (ii) to use GAI models for defeater identification, and (iii) to use of GAI models for defeater mitigation. Shahandashti et al. provide empirical results for objective (i), which suggest that GPT-4 is familiar with the core concepts of EA. With respect to objective (ii), Shahandashti et al. mention possible approaches for LLM-based defeater identification, e.g., via Chain-of-Thought prompting techniques [23]. They also foresee the involvement of human experts in evaluating defeaters proposed by GAI models. However, no concrete examples of such prompts, evaluation criteria, or experimental results were given. Objective (ii) coincides directly with the proposal of Viger et al. [22] and with the focus of the present work which addresses the above limitation. Specifically, our work provides the first concrete implementation and empirical evaluation of AI-supported defeater generation. Our experiments use a traditional prompting scheme, so our results can provide a baseline for future experiments with more sophisticated prompting techniques.

Many researchers have investigated the strengths and limitations of GAI models in software and safety engineering processes. These include the writing of formal specifications [4], hazard analysis [6], goal modeling [3], software modeling [2], and testing [19]. The use of GAI in these tasks presents new opportunities for research in AC development techniques beyond only EA. For example, recent developments in AI-supported formalization [4] and proof engineering [7] could provide significant improvements to existing AC development techniques which leverage on formal methods [1], [8], [21]. One impediment to such research is the dearth of publicly available ACs to be used as training and testing data. The creation and interpretation of ACs and AC development logs as data [16] is one attempt to address this limitation, but more work is needed.

VII. Conclusion

In this paper, we presented an implementation of the AI-EA framework for leveraging GAI to identify defeaters in ACs. We detailed our implementation, presented and justified a set of criteria for evaluating AI-generated defeaters, and empirically evaluated the effectiveness of our implementation by assessing 171 AI-generated defeaters for two industrial ACs in collaboration with an industrial partner. Our results show that AI-EA was able to generate contextually relevant, informative and realistic defeaters without requiring reviewers to filter out large numbers of irrelevant or incoherent defeaters. Developers of each AC also confirmed that approximately 25% of the unique generated defeaters represented reasonable concerns or errors in the argument that could increase confidence if mitigated. Our dataset, evaluation and implementation are publicly available[6]. In future work, we intend to optimize our implementation by evaluating the impact of the different parameters on the quality of the results. Such parameters include the GAI model used and its settings, the type of background information given to the model, ingestion of additional training documents, and altering the mode of interaction with the GAI model (e.g., prompt refinement). We also plan to explore other areas of AC development that GAI can support, such as change-impact assessment and argument formalization.

References

[1] Bourbouh, H., Farrell, M., Mavridou, A., Sljivo, I., Brat, G., Dennis, L.A., Fisher, M.: Integrating Formal Verification and Assurance: An Inspection Rover Case Study. In: Proc. of NASA Formal Methods Symposium. pp. 53–71. Springer (2021)

[2] Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the Assessment of Generative AI in Modeling Tasks: An Experience Report with ChatGPT and UML. J. of Software and Systems Modeling (SoSyM) pp. 1–13 (2023)

[3] Chen, B., Chen, K., Hassani, S., Yang, Y., Amyot, D., Lessard, L., Mussbacher, G., Sabetzadeh, M., Varró, D.: On the Use of GPT-4 for Creating Goal Models: An Exploratory Study. In: Proc. of Workshops of International Conference on Requirements Engineering (RE'23). pp. 262–271. IEEE (2023)

[4] Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: Nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In: Proc. of International Conferene on Computer-Aided Verification (CAV'23). pp. 383–396 (2023)

[5] Diemert, S., Joyce, J.: Eliminative Argumentation for Arguing System Safety - A Practitioner's Experience. In: Proc. of IEEE International Systems Conference (SysCon'20). pp. 1–7 (Aug 2020)

[6] Diemert, S., Weber, J.H.: Can Large Language Models Assist in Hazard Analysis? In: Guiochet, J., Tonetta, S., Schoitsch, E., Roy, M., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops. pp. 410–422. Springer Nature Switzerland, Cham (2023)

[7] First, E., Rabe, M., Ringer, T., Brun, Y.: Baldur: Whole-Proof Generation and Repair with Large Language Models. In: Proc. of SIGSOFT International Conferene on Foundations of Software Engineering (ESEC/FSE'23). pp. 1229–1241 (2023)

[8] Foster, S., Nemouchi, Y., O'Halloran, C., Stephenson, K., Tudor, N.: Formal Model-Based Assurance Cases in Isabelle/SACM: An Autonomous Underwater Vehicle Case Study. In: Proc. of Formal Methods in Software Engineering (FormaliSE'20). pp. 11–21 (2020)

[6]https://drive.google.com/file/d/1U3VrFinQ5NpQCtNmNQ6ZF_FTBw8LWoD7/view?usp=sharing

[9] Fraiwan, M., Khasawneh, N.: A Review of ChatGPT Applications in Education, Marketing, Software Engineering, and Healthcare: Benefits, Drawbacks, and Research Directions. arXiv preprint arXiv:2305.00237 (2023)
[10] Goodenough, J.B., Weinstock, C.B., Klein, A.Z.: Eliminative Argumentation: A Basis for Arguing Confidence in System Properties. Tech. rep., SEI, CMU (2015)
[11] Greenwell, W.S., Knight, J.C., Holloway, C.M., Pease, J.J.: A Taxonomy of Fallacies in System Safety Arguments. In: Proc. Int. System Safety Conf. (2006)
[12] Group, A.C.W.: Goal Structuring Notation Community Standard (Version 3). Tech. rep., Safety Critical Systems Club (2021)
[13] Hobbs, C., Diemert, S., Joyce, J.: Driving the Development Process from the Safety Case. In: Safe AI Systems. Safety-Critical Systems Club, Bristol, UK (2024)
[14] Kelly, T.P.: Arguing safety - A Systematic Approach to Safety Case Management. Ph.D. thesis, University of York (1998)
[15] The Large Hadron Collider. https://home.cern/science/accelerators/large-hadron-collider (04 2022 [Online])
[16] Menghi, C., Viger, T., Di Sandro, A., Chechik, M.: Assurance Case Development as Data: A Manifesto. International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) (2023)
[17] Millet, L., Diemert, S., Rees, C., Viger, T., Chechik, M., Menghi, C., Joyce, J.: Assurance Case Arguments in the Large: The CERN LHC Machine Protection System. In: Proc. of Int. Conference on Computer Safety, Reliability and Security (SafeComp'23). pp. 3–10 (2023)
[18] OpenAI: GPT-4 Technical Report. arXiv:2303.08774 (2023)
[19] Schäfer, M., Nadi, S., Eghbali, A., Tip, F.: An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. IEEE Transactions on Software Engineering **50**(1), 85–105 (2024)
[20] Shahandashti, K.K., Sivakumar, M., Mohajer, M.M., Belle, A.B., Wang, S., Lethbridge, T.C.: Evaluating the Effectiveness of GPT-4 Turbo in Creating Defeaters for Assurance Cases. arXiv (2024)
[21] Viger, T., Murphy, L., Di Sandro, A., Menghi, C., Shahin, R., Chechik, M.: The ForeMoSt Approach to Building Valid Model-Based Safety Arguments. J. of Software and System Modeling (SoSyM) pp. 1–22 (2022)
[22] Viger, T., Murphy, L., Diemert, S., Menghi, C., Di Sandro, A., Chechik, M.: Supporting Assurance Case Development Using Generative AI. In: Proc. of SafeComp'23, Position paper (2023)
[23] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems **35**, 24824–24837 (2022)