Using GPT-4 Turbo To Automatically Identify Defeaters In Assurance Cases

Kimya Khakzad Shahandashti
York University
Toronto, Ontario, Canada
kimya@yorku.ca

Alvine Boaye Belle
York University
Toronto, Ontario, Canada
alvine.belle@lassonde.yorku.ca

Mohammad Mahdi Mohajer
York University
Toronto, Ontario, Canada
mmm98@yorku.ca

Oluwafemi Odu
York University
Toronto, Ontario, Canada
olufemi2@yorku.ca

Timothy C. Lethbridge
University of Ottawa
Ottawa, Ontario, Canada
timothy.lethbridge@uottawa.ca

Hadi Hemmati
York University
Toronto, Ontario, Canada
hemmati@yorku.ca

Song Wang
York University
Toronto, Ontario, Canada
wangsong@yorku.ca

***Abstract*—Assurance cases (ACs) are convincing arguments, supported by a body of evidence and aiming at demonstrating that a system will function as intended. Producers of systems can rely on assurance cases to demonstrate to regulatory authorities how they have complied with existing industrial standards (e.g., ISO 26262, DO-178C). Defeaters are arguments that challenge the effectiveness of assurance cases. Their presence in assurance cases could compromise the reliability of these assurance cases and make them inadequate for verifying a system's capabilities (e.g., safety, and security). This may lead to system failure, which could have severe outcomes, including loss of life. Therefore, identifying and mitigating defeaters is key to improving assurance cases robustness and reliability. In this paper, we focus on the identification of defeaters. Thus, we rely on GPT-4 Turbo, a Large Language Model developed by OpenAI, to automate the generation (identification) of defeaters in assurance cases. Our approach uses the Eliminative Argumentation (EA) notation to represent assurance cases. Besides, we leverage the Chain of Thought prompting technique to improve GPT-4 Turbo's reasoning capabilities. We conducted experiments on various reference assurance case fragments from the nuclear and aviation domains to evaluate the ability of GPT-4 Turbo to automatically generate defeaters. Although the quality of our experiments results is relatively moderate, the analysis of these results still provides valuable insights on the effectiveness of GPT-4 Turbo in generating defeaters.**

***Index Terms*—Large Language Models, Assurance Cases, Assurance Deficits, Defeaters, System Certification**

I. Introduction

The concept of assurance case refers to: *"a reasoned and compelling argument, supported by a body of evidence, that a system, service or organisation will operate as intended for a defined application in a defined environment."* [16]. The main purpose of an assurance case is to demonstrate that a particular system sufficiently supports its intended non-functional requirements [2], [13], [18]. Such requirements include security, reliability, and safety [2], [13], [18]. Assurance cases can be represented using various textual notations such as unstructured text [18], and semi-structured text [19]. Still, it is pretty common to visually represent assurance cases by relying on visual (diagrammatic) notations such as GSN (Goal Structuring Notation) [16] and EA (Eliminative Argumentation) [15].

Flawed reasoning, inaccuracies, or incomplete evidence can lead to the introduction of defects referred to as defeaters in safety arguments [31]. Such assurance deficits can result in an overestimation of system reliability and a tolerance for certain faults, which may ultimately contribute to failures in system safety [31]. Detecting and mitigating defeaters in assurance cases is crucial since they undermine the trust in the assurance of a system, potentially hindering the verification of mission-critical system capabilities [39].

Some approaches (e.g., [17], [33], [45], [50]) allow identifying defeaters in assurance cases [21]. However, only some of them are automatic. Thus, the other ones usually require significant manual intervention and expertise in identifying and mitigating defeaters in assurance cases. By integrating the strengths of automation with the nuanced understanding of human experts, a semi-automatic approach holds the promise of significantly improving the effectiveness of defeater identification, paving the way for a more robust certification process.

To bridge that gap, we use Large language models (LLMs) to automatically generate defeaters in assurance cases represented using Eliminative Argumentation. LLMs are advanced AI models that have progressively become prominent in natural language processing (NLP) [34]. Such models include GPT-4 Turbo. The latter exhibits an increased efficiency in producing responses and is more deterministic [36].

This work extends our previous research (i.e., [41]) which evaluated the effectiveness of using GPT-4 Turbo for creating defeaters, demonstrating excellent proficiency. Consequently, in this paper, we continue to rely on GPT-4 Turbo to generate defeaters.

The contributions of our paper are the following:

- **Contribution 1**: we refine the approach we introduced in [41] by crafting a set of predicate-based rules that allow leveraging GPT-4 Turbo to automate the generation of defeaters. This is critical for identifying and mitigating potential argument weaknesses.

- **Contribution 2**: We leverage the Chain of Thought (CoT) prompting technique for prompting our GPT-4 Turbo model. This allows for the explicit delineation of reasoning steps involved in problem-solving processes i.e. in the defeater identification.
- **Contribution 3**: We conduct four distinct experiments to assess how contextual information and examples in the prompts influence the GPT-4 turbo's effectiveness.

We further describe our work in the remainder of this paper.

II. Background

A. Assurance Cases (ACs)

Assurance cases are structured as a hierarchy of claims, with lower-level claims drawing on concrete evidence (e.g., formal reviews, simulations), and also serving as evidence to justify claims higher in the hierarchy [12]. The purpose of an assurance case is to demonstrate that a system or service meets specific non-functional requirements such as safety, security, and reliability [2], [13], [18]. This allows verifying that mission-critical systems' capabilities are correctly implemented and therefore prevents system failure. The latter may result in loss of life, severe injuries, large-scale environmental damage, property destruction, and major economic loss. The use of assurance cases also helps verify that systems comply with industrial standards (e.g., ISO 26262, DO-178C) [12], [26]. Depending on the non-functional requirements they target, assurance cases can be categorized into several types: safety cases [3], security cases [11], dependability cases [4], reliability cases [52], etc.

B. Eliminative Argumentation (EA)

Eliminative Argumentation (EA) is a graphical notation for representing assurance cases. It builds on the notion of *defeasible reasoning* [10], [15]. The latter supports the recursive challenging of claims to iteratively present and mitigate defeaters to increase the confidence in the assurance cases [10], [15]. Eliminative Argumentation notation employs a directed, non-cyclic graph to outline the structure of an argument [15]. A distinguishing feature of this notation is that it can capture and represent "doubts" or defeaters regarding the reliability of claims, evidence, or the derived logical conclusions [9].

According to Goodenough et al. [15], an eliminative argument consists of five primary components: **Claims (C)**, which are statements needing further argumentative support to establish credibility; **Evidence (E)**, encompassing observations, data, or artifacts; **Inference Rules (IR)**, which are guidelines for logically integrating multiple claims or defeaters to support a higher-level claim; **Defeaters**, which challenge the validity of claims, evidence, or inference rules; and **Strategies (S)**, which outline the method for organizing a group of claims or defeaters. Moreover, it is optional to use a **Context (CX)** element to elaborate on a primary element.

The most common categories of defeaters are [10], [15]:

- Those that present doubts about Claims: they are known as **Rebutting defeaters (R)**

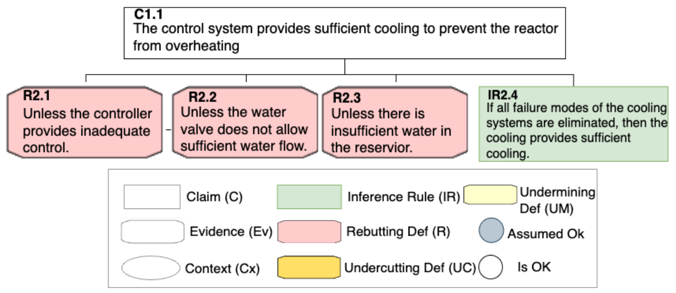
C1.1
The control system provides sufficient cooling to prevent the reactor from overheating
R2.1
Unless the controller provides inadequate control.
R2.2
Unless the water valve does not allow sufficient water flow.
R2.3
Unless there is insufficient water in the reservior.
IR2.4
If all failure modes of the cooling systems are eliminated, then the cooling provides sufficient cooling.
Claim (C)
Inference Rule (IR)
Undermining Def (UM)
Evidence (Ev)
Rebutting Def (R)
Assumed Ok
Context (Cx)
Undercutting Def (UC)
Is OK

Fig. 1. Fragment of EA assurance case adapted from [10]

- Those that present doubt about Evidence: they are called **Undermining defeaters (UM)**
- Those that present doubt about inference rules: they are referred to as **Undercutting defeaters (UC)**.

Eliminative Argumentation also proposes two argument terminators [15]: 1) the **Assumed OK** terminator, which indicates that no additional argument or evidence is needed for a defeater; and 2) the **Is OK** terminator that applies to an inference rule, signifying that it has no undercutting defeaters.

Figure 1 illustrates a fragment of an assurance case complying with the Eliminative Argumentation notation and created for a chemical reactor.

C. Large language models (LLMs)

Large language models (LLMs) are artificial intelligence systems that have become prominent in natural language processing (NLP) because they are trained on vast datasets, which allows them to answer queries with very high accuracy [25], [34]. In this paper, we have decided to leverage GPT-4 Turbo, an LLM developed by OpenAI. This choice is guided by the high performance of this model, as well as its ability to produce outputs that are more deterministic [35].

Prompting involves providing a language model with a carefully designed query or instruction to elicit a specific response, serving as a critical technique for leveraging the capabilities of LLMs in various tasks [46]. Chain of Thought (CoT) prompting enhances reasoning by structuring complex problems into smaller, manageable sub-tasks [46], [51]. Zero-shot prompting, where a model attempts to solve tasks without any prior examples, benefits from CoT by prompting models to articulate their reasoning steps explicitly [22]. K-shot prompting, which provides models with k examples to learn from, also sees benefits from incorporating CoT strategies [14]. Moreover, the roles within the OpenAI API, distinguish between "system", "user", and "assistant". The "system" role provides high-level instructions, the "user" role presents queries or prompts, and the "assistant" is the model's response [8].

III. Related Work

A. Approaches using LLMs to support the automation of Software Engineering tasks

Several approaches have focused on the use of Large Language Models to automatically support various software-related activities such as [48], [47], [41], [5], [30], and [7]. For instance, Weyssow et al. [48] addressed the difficulty of creating metamodels, which establish intricate connections among concepts within Model-Driven Engineering (MDE). The researchers suggest a methodology that employs Deep Learning, particularly leveraging pre-trained language models, to offer suggestions of pertinent domain concepts to metamodel creators. By educating a model with an extensive collection of metamodels to assimilate both structural and lexical characteristics, the study demonstrates the model's capability to deliver precise suggestions for concept renaming instances.

Kang et al. [20] examined the ability of GPT-3.5 in pinpointing the location of code faults, observing that the Large Language Model frequently succeeded in identifying the erroneous method at the initial attempt. Li et al. [24] implemented a combined approach using differential testing alongside ChatGPT to improve its capacity for generating test cases that reveal faults in programs with bugs. Chen et al. [6] leveraged GPT-4 to support requirements engineering activities. More specifically, they relied on GPT-4 capability to generate models aligned with the Goal-oriented Requirement Language (GRL). Their research emphasized the profound grasp GPT-4 possesses regarding goal modeling. Sivakumar et al. [43] gauged GPT-4 ability to understand GSN and to generate GSN elements. They also explored the use of GPT-4 to automatically generate safety cases. Our approach adapts the work of Chen et al. [6] as well as the one of Sivakumar et al. [43].

B. Approaches to Identify Defeaters in Assurance cases

Shahandashti et al. [21] notably surveyed approaches to identify defeaters. For instance, Groza et al. [17] introduce an approach based on description logic (DL) reasoning for pinpointing defeaters in Goal Structuring Notation. Their method has two steps: initially, they utilize hybrid logic to verify if the evidence nodes in the GSN diagram are confirmed against the Kripke model. Following this, through DL reasoning, they identify which objectives in the GSN framework lack supporting verified evidence. Furthermore, Murugesan et al. [33] advocate for semantic analysis to detect defeaters. Muram and Javed [32] present ATTEST, a framework based on natural language processing (NLP) for assurance. This framework initially processes textual information through several NLP techniques and then develops rules that capture both the syntactic knowledge derived from NLP activities and the semantic insights from model frameworks and their interactions. These rules are then applied to assess argument validity, confirm their soundness, evaluate their sufficiency, spot potential defeaters, and choose counter-evidence. Yuan et al. [50] investigate the automatic detection of defeaters through a predicate-based framework. They create an ontology comprising constant symbols, function symbols, and predicate symbols, establishing the lexicon for GSN node expressions. Some of these approaches are manual, which may make the identification of defeaters time-consuming, error-prone and tedious.

Viger et al. [45] introduced a novel approach that aims at leveraging GPT-4 to automatically identify defeaters in assurance cases to enhance their reliability. However, their work is still preliminary. Besides, Viger et al. have not carried out yet experiments to validate their work. In [41], we extended the work of Chen et al. [6] and Sivakumar et al. [43] to assess GPT-4 Turbo efficiency. Our results revealed its excellent proficiency in understanding and applying Eliminative Argumentation notation. Consequently, in this paper, we aim to use GPT-4 Turbo for identifying defeaters within assurance cases.

Noteworthy, in our previous work i.e. [41], our goal was to enhance the verification and reliability of assurance cases by devising a novel method aiming at automating defeater identification and mitigation. We therefore proposed a 3-phase approach consisting in utilizing GPT-4 Turbo for identifying and mitigating defeaters in assurance cases represented using Eliminative Argumentation notation. Figure 2 provides a high-level overview of that approach. We have completed **Phase I** of that approach in [41]. That Phase consisted in extracting structural and semantic rules from Eliminative Argumentation, and deriving structural and semantic-based questions from these rules. Table I reports these rules. Phase I also consisted in relying on these structural and semantic-based questions together with generation-based ones, to evaluate GPT-4 Turbo's proficiency in Eliminative Argumentation. Our evaluation showed that GPT-4 turbo demonstrates an excellent proficiency in Eliminative Argumentation.

As we explained in [41], **Phase II** of the proposed approach centers around applying GPT-4 Turbo to identify defeaters. That phase consists of guiding GPT-4 Turbo through the Chain of Thought prompting techniques to clarify the reasoning behind defeater identification. This is possible by: 1) integrating the reasoning steps into the examples at hand; or 2) asking the LLM to provide a detailed explanation of the thought process [22], [46]. This phase also consists in using predicate-based structural rules since LLMs can effectively encode rule-based knowledge [49]. This phase also aims at involving experts (e.g., two experts) to review and/or refine the defeaters GPT-4 Turbo generated. This allows ensuring the validity as well as the applicability of the defeaters GPT-4 turbo generated.

As we further explained in [41], **Phase III** aims at leveraging GPT-4 Turbo to support the mitigation of the various defeaters generated in **Phase II**. Noteworthy, in [41], **Phases II** and **III** were still at the proposal stage i.e. we did not detailed nor implemented both phases in [41]. In this paper, we extend the work we proposed in [41] by further detailing Phase II and carrying out experiments to validate that phase. We will implement **Phase III** in future work.

IV. Formalization of predicate-based structural rules

To carry out Phase II of our approach (i.e. defeater identification), we proposed in [41] to rely on predicate-based rules for illustrating EA elements and their connections. The integration of predicate-based rules into the prompting process

TABLE I
THE SET OF STRUCTURAL AND SEMANTIC RULES WE PROPOSED IN [41]

Category	**Name**	**Structural rules**	**Semantic rules**
EA Element	Claim	Connected to: Context, Rebutting Defeater	A claim is stated as a predicate, a true or false statement.
EA Element	Evidence	Connected to: Rebutting defeater, Undermining defeater, Undercutting defeater, Inference rule, Evidence	Evidence is in the form "[Noun phrase] showing P" with P asserting an interpretation of data relevant to the argument.
EA Element	Context	Connected to: Claim	It gives additional information about the content of a fundamental element and is optional.
EA Element	Inference Rule	Connected to Rebutting defeater, Undermining defeater, Undercutting defeater, Claim, Evidence	They are predicates (P $\rightarrow$ Q), where either P or Q (but not both) is an eliminated defeater.
EA Element	Undercutting Defeater	Connected to: Inference Rule	Is a doubt about the validity of an inference rule (P $\rightarrow$ Q), preceded by "Unless"
EA Element	Undermining Defeater	Connected to Evidence	Is a predicate associated with evidence, preceded by "But". It challenges the validity of the data comprising the evidence.
EA Element	Rebutting Defeater	Connected to: Claim	Is a predicate associated with a claim, preceded by "Unless"
Argument Terminator	Assumed OK	Attached to Rebutting defeater, Undermining defeater, Undercutting defeater, Claim, Evidence	It asserts that some defeater is (assumed to be) false.
Argument Terminator	Is OK	Attached to Inference Rule, Claim, Evidence	It applies to inference rules, indicating no undercutting defeaters due to the rule being a tautology.

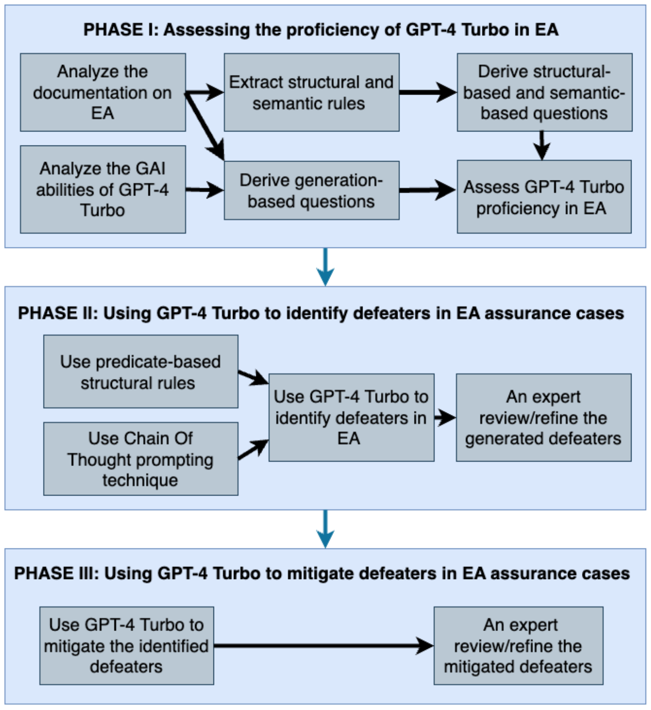
PHASE I: Assessing the proficiency of GPT-4 Turbo in EA
Analyze the documentation on EA
Extract structural and semantic rules
Derive structural-based and semantic-based questions
Analyze the GAI abilities of GPT-4 Turbo
Derive generation-based questions
Assess GPT-4 Turbo proficiency in EA
PHASE II: Using GPT-4 Turbo to identify defeaters in EA assurance cases
Use predicate-based structural rules
Use Chain Of Thought prompting technique
Use GPT-4 Turbo to identify defeaters in EA
An expert review/refine the generated defeaters
PHASE III: Using GPT-4 Turbo to mitigate defeaters in EA assurance cases
Use GPT-4 Turbo to mitigate the identified defeaters
An expert review/refine the mitigated defeaters

Fig. 2. Overview of the approach we introduced in [41]

of AI language models has been a focus of recent research. As Yang et al. demonstrated in [49], LLMs can effectively encode rule-based knowledge into their parameters through a method known as rule distillation. Their work not only enhances the efficiency of the models compared to example-based learning but also improves their generalization capabilities. Similarly, Zhu et al. [53] introduced the Hypotheses-to-Theories (HtT) framework, which underscores the significance of learning a rule library for reasoning with LLMs. Their method involves an induction stage for generating and verifying rules and a deduction stage for applying these rules, leading to substantial improvements in the model's accuracy and reasoning abilities.

The aforementioned studies underscore the potential and

TABLE II
TABLE OF EA DEFEATERS AND THEIR PREDICATE-BASED RULES

Defeater Type	**Predicate-Based Rule**
Rebutting Defeater (R)	R#({{TEXT}}, C(s)#)
Undercutting Defeater (UC)	UC#({{TEXT}}, IR(s)#)
Undermining Defeater (UM)	UM#({{TEXT}}, E(s)#)

effectiveness of applying predicate-based rules in guiding the prompting process, thereby enhancing the reasoning capabilities and accuracy of AI language models. This is crucial because prompting methods relying on an LLM's implicit knowledge are prone to "*hallucinations*" [28]. Such hallucinations usually occur when the model generates answers that are incorrect or even repetitive [28]. To tackle that issue in [41], we proposed to rely on predicates to create various predicate-based rules from the structural rules EA embodies. Our aim was to leverage these predicate-based rules to prompt GPT-4 Turbo in a way that enables it to generate defeaters more effectively. In accordance with that proposal, we now introduce in this paper the following general predicate-based rule for EA elements:

$$\boxed{\text{ELEMENT\#}(\{\{\text{TEXT}\}\},\text{CONNECTEDELEMENT(S)\#})}$$

An `ELEMENT` can be any of the following EA elements: Claims (C), Evidence (E), Inference Rules (IR), Rebutting Defeater (R), Undercutting Defeater (UC), or Undermining Defeater (UM). The `CONNECTEDELEMENT(S)` represents a list of possible EA elements that are connected to `ELEMENT`. The # symbol indicates the unique number for each element.

Utilizing the general predicate-based rule and the semantic and structural rules outlined in Table I, we derived specific predicate-based rules for generating defeaters. These rules, as detailed in Table II, guide the prompting process. According to these predicate rules, a Rebutting Defeater (RD) is exclusively connected to Claim(s) (C), an Undercutting Defeater (UC) is linked only to Inference Rule(s) (IR), and an Undermining Defeater (UM) is associated solely with Evidence(s) (E).

V. Experimental Setup

A. Research Question

The objective of this preliminary study is to answer the following research question (RQ):

RQ: Is GPT-4 Turbo able to generate defeaters in EA notation? To investigate that research question, we conducted 4 distinct experiments designed to evaluate the effectiveness of GPT-4 Turbo in accurately generating defeaters using the defined predicate-based rules and Chain of Thought prompting technique. We designed these experiments to investigate the effectiveness of including examples and contextual information about the system. In our work, the **context** refers to any information about the system that was mentioned in the paper describing that system and its assurance case.

- **Experiment 1: Zero-shot without Context** - This experiment evaluates GPT-4 Turbo's ability to generate defeaters using a zero-shot approach, where the model is not provided with any specific examples. In this experiment, we do not provide any contextual information about the system.
- **Experiment 2: One-shot without Context** - This experiment incorporates a one-shot learning approach, where the model is given a single example to guide its response, still without any contextual information about the system.
- **Experiment 3: Zero-shot with Context** - This experiment also applies a zero-shot methodology. Unlike Experiment 1, it provides the model with context about the system.
- **Experiment 4: One-shot with Context** - Building upon the previous experiments, this test combines one-shot learning with some contextual information about the system.

B. Dataset

Our dataset consists of 9 partial assurance cases (assurance case fragments) that we selected from two assurance cases complying with EA and documented in the literature. Our selection strategy aimed to focus on assurance cases from diverse sectors, including nuclear and aviation, to validate the versatility and domain independence of our approach. The remainder of this section describes our dataset.

*1) **CERN LHC Machine Protection System:*** Constructed by CERN (European Organization for Nuclear Research), the Large Hadron Collider (LHC) stands as the most significant particle accelerator globally [29]. Its Machine Protection System (MPS) monitors operations to prevent damage from unstable particles. The MPS integrates four key sub-systems: the Beam Loss Monitoring System (BLMS), Beam Interlock System (BIS), Beam Dumping System (BDS), and Safe Machine Parameter Controller (SMPC). The assurance case of the MPS complies with EA. That assurance case comprises 509 nodes and is accessible publicly [1]. That assurance case focuses on the **nuclear** domain, and comprises 105 potential defeaters. That assurance case is very large and is therefore represented by several fragments (i.e. partial assurance cases) in [1]. We randomly collected eight of these assurance case fragments. We refer to them as **Cases 1, 2 3, 4, 5, 6, 7, and 8**, respectively. For illustration purposes, we describe Case 1.

Case 1: Figure 3 illustrates this fragment. The latter breaks down claim C0030. That claim asserts that ”The BIS will effectively communicate a beam permit loss to the BDS within a timeframe of under 100 microseconds”. The decomposition of that claim occurs through Strategy S0654, which examines potential failure scenarios that might obstruct, postpone, or hinder the BIS's ability to send a beam dump request to the BDS. The defeaters D0031, D0036, D0438, and D0512 symbolize these failure scenarios. These defeaters present doubt about the claim, so one can consider them rebutting.

*2) **Air Traffic Control System:*** This systems falls in the domain of **aviation**. Referenced from [9], the associated assurance case complies with EA notation and originates from Raytheon Canada's experience in developing a safety case for Nav Canada's air traffic management system. This safety case focuses on scenarios where the same discrete SSR (Special Service Request) code, a unique aircraft identifier from air traffic control, is assigned to multiple aircrafts in the same airspace. Such duplications, serving as a ”primary key” in air traffic systems, risk hazardous situations like incorrect radar data association, and endangering aircraft separation. Given the nature of our problem, i.e. defeater identification, we were only able to collect the following fragment from that assurance case:

Case 9: Figure 4 illustrates a fragment of the assurance case in [9]. That fragment contains a claim called C0001. The latter argues that discrete SSR codes assigned to aircraft in the controlled airspace are unique over a tolerable interval of X seconds. Two defeaters (i.e. D0003 and D004) defeat that claim.

Table III reports each case's defeaters (with original names).

C. Data pre-processing

After collecting the nine assurance case fragments (i.e. cases 1-9) we first pre-processed them as follows:

- In EA, inference rules are a more structured version of GSN strategies [15]. For the sake of simplicity and to foster data reduction, we, therefore, removed strategies from the analyzed fragments. This also allows obtaining assurance case fragments that align with the structural and semantic rules we extracted from EA.
- Defeaters' names all start with “D”. Hence, they seem generic. Thus, we renamed these defeaters based on their categories to better label them. Thus, the names of rebutting, undermining, and undercutting defeaters respectively start with “R”, “UM”, and “UC”.
- Originally, the nine assurance case fragments were represented in EA -a visual notation. We therefore converted these fragments into the textual form (compliant with the predicate rule) i.e. structured prose complying with EA. That textual format is suitable for GPT-4 turbo processing.

.

TABLE III
SUMMARY OF THE DATASET

Case	ID	Name	Category	Content of the defeater
Case 1	1	D0031	Rebutting	Unless the beam loop is damaged in a way that interferes with the transmission of the loss of the beam permit.
	2	D0036	Rebutting	Unless transmission of the withdrawal of the beam permit is too slow (i.e. not less than 100 microseconds)
	3	D0438	Rebutting	Unless the BIC power fails and thus the beam permit withdrawal signal is not sent or received by the BIC loop.
	4	D0512	Rebutting	Unless the Beam Loss Signal is not received by a Beam Interlock Controller within 70 microseconds of the BLMS detecting high beam loss.
Case 2	1	D0121	Rebutting	Unless the Surface Electronics fails to withdraw the user permit to enable a 'dump signal' to the BIC.
	2	D0252	Rebutting	Unless the Combiner Card fails to combine signals from Beam Energy Tracker, Beam Permit and Surface Card Dump Signal, Thus preventing a notification of a beam dump being required to the BIC.
Case 3	1	D0380	Undermining	Unless a malfunction, either in the MKB magnets themselves or their connection to the power supply, causes them to fail to activate.
Case 4	1	D0305	Undermining	Unless a malfunction, either in the MKB itself or its connection to the power supply, causes the MKD to fail to activate
Case 5	1	D0078	Rebutting	Unless the MKD loses connection to its power supply.
	2	D0107	Rebutting	Unless the generated electromagnetic field is not within its specified tolerance.
	3	D0551	Rebutting	Unless the generated electromagnetic field is not within its specified tolerance.
Case 6	1	D0685	Undercutting	Unless communications between the subsystems fails.
	2	D0686	Undercutting	Unless undiagnosed events led to a spurious beam dump.
Case 7	1	D0431	Rebutting	Unless the critical components have a misleading connection status and appear to be online and operational.
	2	D0432	Rebutting	Unless the BIC's signal transmission hardware is damaged and communication to the BDS compromised.
Case 8	1	D0521	Rebutting	Damage or Compromised signal transmission pathways between the Beam Loss Monitors and their respective Beam Interlock Controllers.
	2	D0522	Rebutting	Incorrect integration of signals at AND gates, which lead to a Beam Dump Permit not being produced when required.
	3	D0523	Rebutting	A Beam Permit is produced by the BIS, but damage or compromised signal transmission pathways from the BIS to the BDS impedes the transmission of the Beam Dump Permit.
Case 9	1	D0003	Rebutting	Unless an aircraft with an in-use SSR code enters the airspace and its code is not re-assigned within X seconds.
	2	D0004	Rebutting	Unless a communications error occurs when the SSR code is transmitted that is not corrected within X seconds.

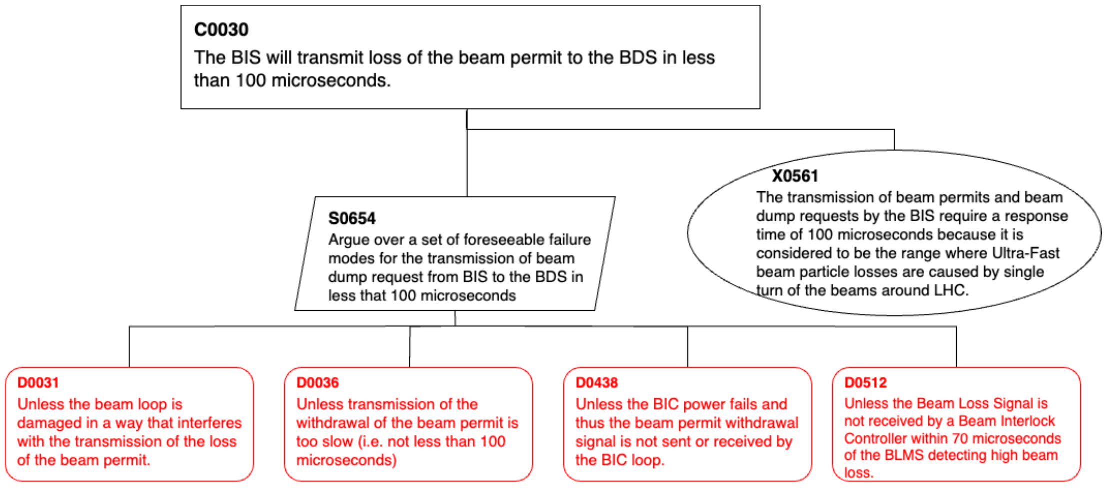
C0030
The BIS will transmit loss of the beam permit to the BDS in less than 100 microseconds.
S0654
Argue over a set of foreseeable failure modes for the transmission of beam dump request from BIS to the BDS in less that 100 microseconds
X0561
The transmission of beam permits and beam dump requests by the BIS require a response time of 100 microseconds because it is considered to be the range where Ultra-Fast beam particle losses are caused by single turn of the beams around LHC.
D0031
Unless the beam loop is damaged in a way that interferes with the transmission of the loss of the beam permit.
D0036
Unless transmission of the withdrawal of the beam permit is too slow (i.e. not less than 100 microseconds)
D0438
Unless the BIC power fails and thus the beam permit withdrawal signal is not sent or received by the BIC loop.
D0512
Unless the Beam Loss Signal is not received by a Beam Interlock Controller within 70 microseconds of the BLMS detecting high beam loss.

Fig. 3. Case 1 (adapted from [29])

D. GPT-4 Turbo Setting

To interact with the GPT-4 Turbo model when performing our experiments, we relied on the OpenAI API[1]. Setting the `seed` parameter is critical to support the generation of more reproducible outputs from GPT-4 Turbo when feeding it with identical inputs [36]. To ensure outputs are predominantly consistent across API calls, it is possible to set a specific integer for the seed parameter and consistently use this same value for any requests where you're aiming for deterministic results [37]. Additionally, maintaining identical settings for all other parameters, like the prompt or temperature, is crucial for achieving this consistency. However, it's important to bear in mind that slight variations in determinism can still occur due to occasional necessary updates OpenAI makes to the model configurations. To aid in tracking these updates, the *system_fingerprint* field is made available [38].

E. Prompting Process: on the use of the Chain of Thought prompting technique

OpenAI's guide[2] has proposed several best practices to support prompt engineering. We followed these practices when interacting with GPT-4 Turbo. For this purpose, we carefully

[1]https://openai.com/api/. To make sure we get more deterministic responses from the model, we set the value of the `seed` parameter in the API

[2]https://platform.openai.com/docs/guides/prompt-engineering

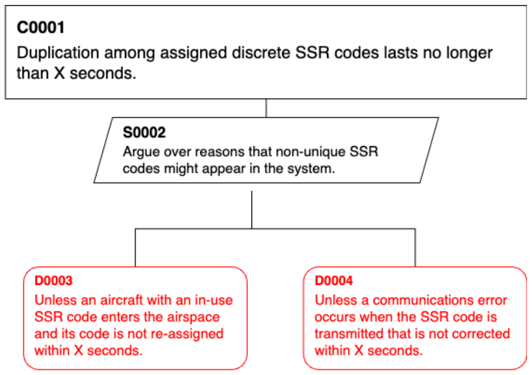
C0001
Duplication among assigned discrete SSR codes lasts no longer than X seconds.
S0002
Argue over reasons that non-unique SSR codes might appear in the system.
D0003
Unless an aircraft with an in-use SSR code enters the airspace and its code is not re-assigned within X seconds.
D0004
Unless a communications error occurs when the SSR code is transmitted that is not corrected within X seconds.

Fig. 4. Case 9 (adapted from [9])

crafted both *'system'* and *'user'* prompts to effectively guide the model. In this regard, the 'user' prompts consist in the direct questions asked to the model, and designed to retrieve either specific information or analysis. The main purpose of the 'system' prompts was to orient GPT-4 Turbo appropriately, to make sure it was aware of its role as an assistant when processing our inquiries.

We also leveraged the CoT (Chain of Thought) prompting technique. The latter is a technique used in natural language processing to enhance the problem-solving capabilities of language models [46]. It involves prompting the model to generate intermediate steps or reasoning paths when tackling complex questions or tasks [46]. CoT usually improves the performance of language models on tasks that require reasoning, such as arithmetic calculations, common sense reasoning, and text-based problem-solving [44].

The different prompts we used in our various experiments are available on GitHub[3]. In the following, we discuss the user and system prompt templates that we used for experiment 4 since it is a more complex experiment. We annotated each part of the prompt and discussed each part separately.

- **System Prompt:**

You are an assistant who helps me to identify defeaters in Eliminative Argumentation notation. Eliminative Argumentation consists of 7 elements (i.e., Claim (C), Context (CX), Evidence (E), Inference Rule (IR), Undermining Defeater (UM), Undercutting Defeater (UC), Rebutting Defeater (R)) and 2 terminators (i.e., Assumed Ok (ASSUMED_OK), Is Ok (OK))① As input, you are given an assurance case in Eliminative Argumentation delimited by `@@@`. Moreover, we provided some context about the case in the input after CONTEXT.② For modeling an assurance case in Eliminative Argumentation, we use the following structure which shows the connections between Eliminative Argumentation notation:

– **ELEMENT** can be any of the 7 elements of Eliminative Argumentation.
– **CONNECTEDELEMENTS** is a list of possible eliminative argumentation elements that is connected to the ELEMENT.
– # indicates the unique number for each element.
– {{`TEXT`}} indicates descriptive text for each element that can be in the form of a noun-phrase, verb-phrase, or predicate.

The general structure is: ELEMENT#({{TEXT}}, CONNECTEDELEMENTS#)③

There are 3 types of defeaters for which we have defined semantic and structural rules:

1) Element: Undercutting Defeater (UC),
Semantic Rule: An undercutting defeater (UC) is a doubt about the validity of an inference rule ($P \rightarrow Q$), preceded by "Unless".
Structural Rule notation: UC#({{TEXT}},IR#)
2) Element: Undermining Defeater (UM),
Semantic Rule: An undermining defeater (UM) is a predicate associated with evidence, preceded by "But". It challenges the validity of the data comprising the evidence.
Structural Rule notation: UM#({{TEXT}},E#)
3) Element: Rebutting Defeater (R),
Semantic Rule: A rebutting defeater (R) is a predicate associated with a claim, preceded by "Unless".
Structural Rule notation: R#({{TEXT}},C#)④

Using these rules, try to identify defeaters for each element in the input⑤ Now, I will provide you with an example so you can understand the process better: For example, C1(Light turns on) means a Claim element with a unique number of 1 and the TEXT of Light turns on which does not have any connected element. R1(Unless the bulb is defective, C1) means a Rebutting Defeater element that is a defeater for the C1 Claim. Now, I want you to think through each step of this process. Begin by examining each element. Then, for each element, determine if it can have any defeater and categorize each one as either rebutting (it can be only for a Claim (C)), undermining (it can be only for an Evidence (E)), or undercutting (it can be only for an inference rule (IR)). Provide the output in the same structure as the input with the correct numbering in the modeling rules explained above.⑥

1) In this section of the prompt, the model is explicitly defined as an assistant, setting clear expectations for its role in facilitating the analysis of arguments within EA. By providing definitions for the seven elements and two terminators, the prompt establishes a comprehensive context, ensuring that the model understands and adheres to the specific

[3]https://github.com/kimixz/GPT4-Turbo-AC/tree/main

terminologies and structure required for accurately identifying the defeaters.

2) By specifying that the input is an assurance case in EA, enclosed within the delimiters @@@, this section of the prompt demarcates the boundaries of the user-provided data, ensuring precise extraction and processing by the model. Moreover, context signifies any relevant information about the system we are discussing. We typically present this information right after the section labeled "CONTEXT", in the input when it pertains to experiments 3 and 4.
3) In this section of the prompt, we describe the general predicate-based rule for EA elements as described in section IV.
4) In this section, we describe different types of defeaters and their structural and semantic rules as defined in Table II.
5) In this section, we specify the task of the model which is identifying the defeaters in the assurance case.
6) In this section, we are prompting the model using the CoT technique to think step by step and also specify how it should structure the output. If we use the zero-shot technique with CoT such as Experiment 1 and 3, we do not provide any example. However, we provide an example for experiments 2 and 4 as shown in the current prompt.

- **User Prompt:** The user prompt in the 4 experiments are the partial assurance cases we described in Section V-B in the format of the general predicate-based rule. The following is an example of the user prompt for case 1. Context signifies any relevant information about the system we are discussing. We typically present this information right after the section labeled "CONTEXT", in the input when it pertains to experiments 3 and 4. The context that we used in the prompts is available on Github[4].

```
@@@
C1(The BIS will transmit loss of the beam permit to the BDS in less than 100 microseconds)
IR1(Argue over a set of foreseeable failure modes for the transmission of beam dump request from the BIS to the BDS in less than 100 microseconds, C1)
@@@
CONTEXT:
```

F. Assessment Measures

To assess our experiment results, we rely on two criteria.

1) Ground-truth Similarity: it evaluates the lexical and semantic similarity between defeaters generated by GPT-4 Turbo and the defeaters in the reference assurance cases (ground-truths). The scripts for assessing ground-truth similarity are available on GitHub[5]. We rely on three measures to assess it:

[4]https://github.com/kimixz/GPT4-Turbo-AC/tree/main
[5]https://github.com/kimixz/GPT4-Turbo-AC/tree/main

- **Exact match or string matching measure (using fuzzy logic)**. That measure quantifies the similarity between two texts, considering partial matches and minor differences [42]. The score typically ranges from 0 to 1, where 0 indicates no similarity and 1 indicates a perfect or near-perfect match, including partial similarities. We rely on a Python library called fuzzywuzzy [40] to assess it.
- **Bleu score**: The Bleu score quantifies the similarity between the machine-generated text and the reference text. That score varies between 0 and 1, where 0 indicates no overlap (completely different or irrelevant text) and 1 indicates a perfect match (the generated text is identical to the reference text). We rely on sacrebleu (Python library) to compute BLEU scores.
- **Semantic similarity**: as in [43], we use the cosine similarity to assess the semantic alignment between the reference defeaters within the assurance case and those produced by GPT-4 Turbo. The cosine similarity value ranges from -1 to 1, where -1 denotes complete dissimilarity and 1 indicates exact similarity. To compute the cosine similarity, we rely on scikit-learn, a Python library.

2) Reasonability: This measure is used in [6] and [43]. Reasonable means the GPT-4 turbo generated EA "*element could reasonably be in the ground-truth but is not*" [6]. Two assessors with at least two years of experience in EA manually rate the reasonability of the defeaters GPT-4 turbo generated. For this purpose, the two assessors rely on a linear scale: ***1=Totally reasonable; 2=Mostly reasonable; 3=Moderately reasonable; 4=Slightly reasonable; 5=Unreasonable***. For testing the interrater reliability, we used Cohen's Kappa measure [27]. The value of that measure varies between - 1 to 1. When the value of the Cohen's Kappa measure is close to 1, this indicates a strong level of agreement between raters. When the value of the Cohen's Kappa measure is lower than 0, this indicates that there is no agreement between raters. To compute the value of that measure, we relied on the sklearn, a Python library.

VI. Results

A. Ground-truth similarity results

1) Exact match or string matching results: Table IV shows the averages of fuzzy similarity scores. Based on the mean similarity scores, we can conclude that Experiments 1 and 2 outperform Experiments 3 and 4. This suggests that for the specific task of generating defeaters with GPT-4 Turbo, providing contextual information about the system may not always lead to better outcomes. It is particularly interesting to note that the one-shot approach did not significantly outperform the zero-shot approach when context is not provided, but it did improve performance when context is included, even though it did not reach the effectiveness of the zero-shot scenarios without context.

2) BLEU score: Table V presents the averages of BLEU scores for the experiments. The analysis underscores the significant role of example-based learning (one-shot) in enhancing the model's ability to generate defeaters that closely

Experim. 1	Experim. 2	Experim. 3	Experim. 4
0.57	**0.56**	**0.39**	**0.49**

TABLE IV
AVERAGES OF FUZZY SIMILARITY SCORES

match the ground truth. Surprisingly, the addition of context in Experiments 3 and 4 did not lead to the anticipated improvements in performance, unless it was combined with example-based guidance, as evidenced by the superior results in Experiment 4. This observation illustrates the importance of providing a single, impactful example over supplying contextual information. The highest performance achieved in Experiment 4, where both context and one-shot learning were utilized, suggests that integrating contextual information with carefully chosen examples presents a more effective strategy for optimizing defeater generation tasks.

Experim. 1	Experim. 2	Experim. 3	Experim. 4
0.30	**0.46**	**0.34**	**0.48**

TABLE V
AVERAGES OF BLEU SCORES

3) Semantic similarity results: Table VI presents the cosine similarity scores. The analysis illustrates that Experiment 2, which includes one-shot learning without context, slightly outperforms Experiment 1, indicating a subtle yet positive effect of example-based guidance on model performance.

Experim. 1	Experim. 2	Experim. 3	Experim. 4
0.63	**0.64**	**0.57**	**0.60**

TABLE VI
AVERAGES OF COSINE SIMILARITY SCORES

The analysis of ground-truth similarity values reveals nuanced insights into GPT-4 effectiveness for each experimental setup. Experiment 2, which utilized a one-shot approach with CoT and predicate-based rules without context, consistently showed higher performance. This indicates that minimal guidance (a single example) can significantly impact the model's ability to generate defeaters closely aligned with the expected ones.

B. Reasonability results

As stated above, two assessors independently reviewed the defeaters GPT-4 generated and rated their reasonability. The corresponding value of the Cohen's Kappa measure is **0.54** which indicates a moderate agreement level between the two raters. Table VII reports the average of the reasonability results for the four experiments we performed on the 9 assurance case fragments. Experiment 1, with the highest mean score of 4.72, indicated that defeaters were mostly viewed as "Unreasonable," marking it as the least effective in generating reasonable defeaters. Conversely, Experiments 3 and 4 displayed a shift towards "Slightly reasonable" outputs, signaling improvements but still not reaching the ideal levels of reasonability. Noteworthy, Experiment 2 yields the best results, with a mean score of 2.94. Hence, it yields defeaters that are "Moderately reasonable".

Experim. 1	Experim. 2	Experim. 3	Experim. 4
4.72	**2.94**	**4**	**3.89**

TABLE VII
AVERAGE OF REASONABILITY SCORES

The analysis of the mean scores across all experiments suggest that, when provided with one example and no context, GPT-4 turbo can generate "Moderately reasonable" defeaters.

VII. THREATS TO VALIDITY

Our dataset consists of nine assurance case fragments i.e. cases 1-9 (see Section V-B). The work of Millet et al. [29] documents cases 1-8 and was published in 2023 by the SafeComp conference. Furthermore, the work of Diemert et al. [9] documents case 9 and was also published in 2023. Since the cut-off date of GPT-4 turbo is quite recent (i.e. 2023[6]), it may be possible that part of our dataset was already present in the training set of GPT-4 turbo. This may impede the generalization of our findings to new data. To mitigate that issue, we have pre-processed the data (i.e. cases at hand). This means that GPT-4 turbo has never seen most of the resulting pre-processed fragments. Still, to tackle that issue in future work, we will sample more recent data (i.e. assurance case fragments) that is not publicly available.

Our dataset consists in an unbalanced mix of assurance cases, with eight cases (assurance case fragments) from one domain (i.e. nuclear domain) and only one from another (i.e. aviation domain). Additionally, our dataset consists of isolated cases and does not leverage the potential interdependence of these cases, which is also a threat to the validity of our work. Both of these issues stem from the scarcity of publicly available data, which is largely due to the sensitive and critical nature of the information that assurance cases represent. This sensitivity often restricts the availability of such cases, limiting the diversity and quantity of data that one can comprehensively analyze.

Another threat to the validity of our work is the manual translation of assurance cases into a predicate-based textual format compliant with EA. This process necessitates manually inputting key information from an EA model into a predicate-based textual prompt. Such manual intervention carries the risk of introducing errors, particularly because predicate-based textual representations in EA lack syntax highlighting. We plan to tackle this threat to validity in future work.

VIII. CONCLUSION AND FUTURE WORK

In previous work, we introduced an approach to identify and mitigate defeaters embedded in assurance cases. In this paper, we have refined that approach by proposing a set of predicate-based rules that leverage GPT-4 Turbo to automate the generation of defeaters embedded in assurance cases. To achieve this, we relied on the Eliminative Argumentation

[6]https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo

notation to represent assurance cases and their defeaters. We have also conducted experiments to evaluate the extent to which contextual information and examples in the prompts influence GPT-4 turbo's effectiveness in generating defeaters. The quality of our experiments results is moderate. Still, the analysis of these results provides some key insights on the effectiveness of GPT-4 Turbo in generating defeaters.

In future work, we plan to incorporate into our approach a module that will automate the translation of assurance cases into a predicate-based textual format compliant with Eliminative Argumentation. This will make our approach less error-prone. Additionally, we will conduct further experiments on a broader set of assurance cases that comply with EA. Furthermore, we intend to explore the use of additional techniques (e.g., Retrieval Augmented Generation [23]) to improve the quality of our results.

REFERENCES

[1] Lhc mps argument. https://cds.cern.ch/record/2854725, 2023.

[2] B. BELLE, A., AND ZHAO, Y. Evidence-based decision-making: On the use of systematicity cases to check the compliance of reviews with reporting guidelines such as prisma 2020. *ESWA 217* (2023), 119569.

[3] BAGHERI, M., LAMP, J., ZHOU, X., FENG, L., AND ALEMZADEH, H. Towards developing safety assurance cases for learning-enabled medical cyber-physical systems. *arXiv preprint arXiv:2211.15413* (2022).

[4] BLOOMFIELD, R. E., LITTLEWOOD, B., AND WRIGHT, D. Confidence: its role in dependability cases for risk assessment. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)* (2007), IEEE, pp. 338–346.

[5] CHAABEN, M., BURGUEÑO, L., AND SAHRAOUI, H. Towards using few-shot prompt learning for automating model completion. In *ICSE-NIER* (2023), IEEE, pp. 7–12.

[6] CHEN, B., CHEN, K., HASSANI, S., YANG, Y., AMYOT, D., LESSARD, L., MUSSBACHER, G., SABETZADEH, M., AND VARRÓ, D. On the use of gpt-4 for creating goal models: an exploratory study. In *REW* (2023), IEEE, pp. 262–271.

[7] CHEN, K., YANG, Y., CHEN, B., LÓPEZ, J. A. H., MUSSBACHER, G., AND VARRÓ, D. Automated domain modeling with large language models: A comparative study. In *MODELS* (2023), IEEE, pp. 162–172.

[8] DAT NGO, M. S. A. A. A. A. Mastering the openai api: Tips and tricks. https://arize.com/blog-course/mastering-openai-api-tips-and-tricks/, August 2023.

[9] DIEMERT, S., GOODENOUGH, J., JOYCE, J., AND WEINSTOCK, C. Incremental assurance through eliminative argumentation. *Journal of System Safety 58*, 1 (2023), 7–15.

[10] DIEMERT, S., AND JOYCE, J. Eliminative argumentation for arguing system safety-a practitioner's experience. In *SysCon* (2020), IEEE, pp. 1–7.

[11] FINNEGAN, A., AND MCCAFFERY, F. A security argument pattern for medical device assurance cases. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops* (2014), IEEE, pp. 220–225.

[12] FOSTER, S., NEMOUCHI, Y., GLEIRSCHER, M., WEI, R., AND KELLY, T. Integration of formal proof into unified assurance cases with isabelle/sacm. *Formal Aspects of Computing 33*, 6 (2021), 855–884.

[13] FOUNDATION, H. Evidence: Using safety cases in industry and healthcare, 2012.

[14] GE, J., LUO, H., QIAN, S., GAN, Y., FU, J., AND ZHAN, S. Chain of thought prompt tuning in vision language models. *arXiv preprint arXiv:2304.07919* (2023).

[15] GOODENOUGH, J. B., WEINSTOCK, C. B., AND KLEIN, A. Z. Eliminative argumentation: A basis for arguing confidence in system properties. *SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005* (2015).

[16] GROUP, T. A. C. W. Goal structuring notation standard version 3, 2021.

[17] GROZA, A., LETIA, I. A., GORON, A., AND ZAPOROJAN, S. A formal approach for identifying assurance deficits in unmanned aerial vehicle software. In *ICSEng* (2015), Springer, pp. 233–239.

[18] HAWKINS, R., HABLI, I., KOLOVOS, D., PAIGE, R., AND KELLY, T. Weaving an assurance case from design: a model-based approach. In *HASE* (2015), IEEE, pp. 110–117.

[19] HOLLOWAY, C. M. The friendly argument notation (fan).

[20] KANG, S., AN, G., AND YOO, S. A preliminary evaluation of llm-based fault localization. *arXiv preprint arXiv:2308.05487* (2023).

[21] KHAKZAD S., K., BELLE, A. B., LETHBRIDGE, T. C., ODU, O., AND SIVAKUMAR, M. A prisma-driven systematic mapping study on system assurance weakeners. *arXiv preprint arXiv:2311.08328* (2023).

[22] KOJIMA, T., GU, S. S., REID, M., MATSUO, Y., AND IWASAWA, Y. Large language models are zero-shot reasoners. *NeuRIPS 35* (2022), 22199–22213.

[23] LEWIS, P., PEREZ, E., PIKTUS, A., PETRONI, F., KARPUKHIN, V., GOYAL, N., KÜTTLER, H., LEWIS, M., YIH, W.-T., ROCKTÄSCHEL, T., RIEDEL, S., AND KIELA, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 9459–9474.

[24] LI, T.-O., ZONG, W., WANG, Y., TIAN, H., WANG, Y., AND CHEUNG, S.-C. Finding failure-inducing test cases with chatgpt. *arXiv preprint arXiv:2304.11686* (2023).

[25] LIU, Y., HE, H., HAN, T., ZHANG, X., LIU, M., TIAN, J., ZHANG, Y., WANG, J., GAO, X., ZHONG, T., ET AL. Understanding llms: A comprehensive overview from training to inference. *arXiv preprint arXiv:2401.02038* (2024).

[26] MATSUNO, Y., TAKAI, T., AND YAMAMOTO, S. Facilitating use of assurance cases in industries by workshops with an agent-based method. *IEICE TRANSACTIONS on Information and Systems 103*, 6 (2020), 1297–1308.

[27] MCHUGH, M. L. Interrater reliability: the kappa statistic. *Biochemia medica 22*, 3 (2012), 276–282.

[28] MCINTOSH, T. R., LIU, T., SUSNJAK, T., WATTERS, P., NG, A., AND HALGAMUGE, M. N. A culturally sensitive test to evaluate nuanced gpt hallucination. *TAI 1*, 01 (2023), 1–13.

[29] MILLET, L., DIEMERT, S., REES, C., VIGER, T., CHECHIK, M., MENGHI, C., AND JOYCE, J. Assurance case arguments in the large: The cern lhc machine protection system. In *SafeComp* (2023), Springer, pp. 3–10.

[30] MOHAJER, M. M., ALEITHAN, R., HARZEVILI, N. S., WEI, M., BELLE, A. B., PHAM, H. V., AND WANG, S. Skipanalyzer: An embodied agent for code analysis with large language models. *arXiv preprint arXiv:2310.18532* (2023).

[31] MURAM, F. U., GALLINA, B., AND RODRÍGUEZ, L. G. Preventing omission of key evidence fallacy in process-based argumentations. In *QUATIC* (2018), IEEE, pp. 65–73.

[32] MURAM, F. U., AND JAVED, M. A. Attest: Automating the review and update of assurance case arguments. *JSA 134* (2023), 102781.

[33] MURUGESAN, A., WONG, I. H., STROUD, R., ARIAS, J., SALAZAR, E., GUPTA, G., BLOOMFIELD, R., VARADARAJAN, S., AND RUSHBY, J. Semantic analysis of assurance cases using s (casp). In *GDE Workshop in ICLP* (2023).

[34] NAVEED, H., KHAN, A. U., QIU, S., SAQIB, M., ANWAR, S., USMAN, M., BARNES, N., AND MIAN, A. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435* (2023).

[35] OPENAI. Introducing gpt-4 turbo. https://help.openai.com/en/articles/8555510-gpt-4-turbo, 2023.

[36] OPENAI. New models and developer products announced at devday. https://openai.com/blog/new-models-and-developer-products-announced-at-devday, 2023. Accessed: 2024-01-14.

[37] OPENAI. Reproducible outputs. https://platform.openai.com/docs/guides/text-generation/reproducible-outputs, 2023.

[38] OPENAI. Reproducible outputs openai. https://cookbook.openai.com/examples/reproducible_outputs_with_the_seed_parameter, 2023.

[39] RUSHBY, J. Mechanized support for assurance case argumentation. In *New Frontiers in Artificial Intelligence: JSAI-isAI 2013 Workshops* (2014), Springer, pp. 304–318.

[40] SEATGEEK. Fuzzywuzzy: Fuzzy string matching in python. https://github.com/seatgeek/fuzzywuzzy, Year of Latest Release.

[41] SHAHANDASHTI, K. K., SIVAKUMAR, M., MOHAJER, M. M., BELLE, A. B., WANG, S., AND LETHBRIDGE, T. C. Evaluating the effectiveness of gpt-4 turbo in creating defeaters for assurance cases. *Accepted by FORGE* (2024).

[42] SINGLA, N., AND GARG, D. String matching algorithms and their applicability in various applications. *International journal of soft computing and engineering 1*, 6 (2012), 218–222.

[43] Sivakumar, M., B. Belle, A., Shan, J., and Khakzad S., K. Gpt-4 and safety case generation: An exploratory analysis. *arXiv preprint arXiv:2312.05696* (2023).

[44] Song, L., Zhang, J., Cheng, L., Zhou, P., Zhou, T., and Li, I. The impact of advanced prompting strategies on the natural language processing capabilities of large language models. *arXiv preprint arXiv:2309.15630* (2023).

[45] Viger, T., Murphy, L., Diemert, S., Menghi, C., Di, A., and Chechik, M. Supporting assurance case development using generative ai. In *SAFECOMP 2023* (2023).

[46] 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. *NeuRIPS 35* (2022), 24824–24837.

[47] Wei, M., Harzevili, N. S., Huang, Y., Yang, J., Wang, J., and Wang, S. Demystifying and detecting misuses of deep learning apis. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (2024), pp. 1–12.

[48] Weyssow, M., Sahraoui, H., and Syriani, E. Recommending metamodel concepts during modeling activities with pre-trained language models. *SoSym 21*, 3 (2022), 1071–1089.

[49] Yang, W., Lin, Y., Zhou, J., and Wen, J. Enabling large language models to learn from rules. *arXiv preprint arXiv:2311.08883* (2023).

[50] Yuan, T., Manandhar, S., Kelly, T., and Wells, S. Automatically detecting fallacies in system safety arguments. In *PRIMA Workshops* (2016), Springer, pp. 47–59.

[51] Zhang, Z., Zhang, A., Li, M., and Smola, A. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493* (2022).

[52] Zhu, S., Lu, M., and Xu, B. Software reliability case development method based on the 4+ 1 principles. In *ICRMS* (2018), IEEE, pp. 197–202.

[53] Zhu, Z., Xue, Y., Chen, X., Zhou, D., Tang, J., Schuurmans, D., and Dai, H. Large language models can learn rules. *arXiv preprint arXiv:2310.07064* (2023).