Assessing the Impact of GPT-4 Turbo in Generating Defeaters for Assurance Cases

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

Mithila Sivakumar
msivakum@yorku.ca
York University
Toronto, Canada

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

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

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

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

ABSTRACT

Assurance cases (ACs) are structured arguments that allow verifying the correct implementation of the created systems' non-functional requirements (e.g., safety, security). This allows for preventing system failure. The latter may result in catastrophic outcomes (e.g., loss of lives). ACs support the certification of systems in compliance with industrial standards, e.g., DO-178C and ISO 26262. Identifying defeaters —arguments that challenge these ACs — is crucial for enhancing ACs' robustness and confidence. To automatically support that task, we propose a novel approach that explores the potential of GPT-4 Turbo, an advanced Large Language Model (LLM) developed by OpenAI, in identifying defeaters within ACs formalized using the Eliminative Argumentation (EA) notation. Our preliminary evaluation assesses the model's ability to comprehend and generate arguments in this context and the results show that GPT-4 turbo is very proficient in EA notation and can generate different types of defeaters.

CCS CONCEPTS

• **Computing methodologies → Modeling methodologies**; **Artificial intelligence**.

KEYWORDS

Large Language Models, assurance cases, assurance defeaters, system certification, FM for Requirement Engineering

ACM Reference Format:
Kimya Khakzad Shahandashti, Mithila Sivakumar, Mohammad Mahdi Mohajer, Alvine B. Belle, Song Wang, and Timothy C. Lethbridge. 2024. Assessing the Impact of GPT-4 Turbo in Generating Defeaters for Assurance Cases. In *AI Foundation Models and Software Engineering (FORGE '24), April 14, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3650105.3652291

FORGE '24, April 14, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0609-7/24/04
https://doi.org/10.1145/3650105.3652291

1 INTRODUCTION

An assurance case (AC) is a structured hierarchy of claims aiming at demonstrating that a given mission-critical system supports specific requirements (e.g., safety, security, and privacy) [1, 6, 16]. ACs can be presented in various formats, such as straightforward text like structured prose or through graphical representations. Graphical notations include GSN (Goal Structuring Notation) [14] and EA (Eliminative Argumentation) [13]. The presence of assurance weakeners in ACs reflects insufficient evidence, knowledge, or gaps in reasoning [17]. These weakeners can undermine confidence in assurance arguments, which may hamper the verification of mission-critical system capabilities and further result in catastrophic outcomes [21, 31, 32].

Khakzad et al. [19] classified these assurance weakeners by considering several categories, e.g., argument, aleatory, epistemic, and ontological uncertainty. Our focus is on argument uncertainty also referred to as defeaters. Inaccurate, incomplete, or inherently flawed reasoning regarding evidence can introduce defects known as argument uncertainty into safety arguments [26]. This may lead to overconfidence in a system and to the tolerance of certain faults, ultimately contributing to safety-related system failure [26]. Manually creating and challenging arguments are recognized as being labor-intensive, time-consuming, and prone to errors [24, 28].

A few approaches (e.g., [8, 15, 27, 35, 37]) allow identifying defeaters in ACs. However, they often fall short of an all-encompassing strategy that covers all types of assurance weakeners, highlighting the urgent requirement for a more integrated identification approach. To address that gap, we rely on LLMs to automatically generate defeaters in ACs represented using EA. In our work, we adopt GPT-4 Turbo, owing to its increased efficiency in producing responses and its capability to yield deterministic outputs [30].

2 BACKGROUND AND RELATED WORK

2.1 Assurance Cases

An assurance case (AC) is a *"set of auditable claims, arguments, and evidence created to support the claim that a defined system/service can satisfy particular requirements"* [33]. An AC is crucial for facilitating clear communication among different stakeholders in a system, such as suppliers and acquirers, and between operators and regulators [33]. Its primary role is to effectively convey information about the system's non-functional requirements (e.g., safety, security, and reliability) [1, 12, 16]. Employing an AC to demonstrate the

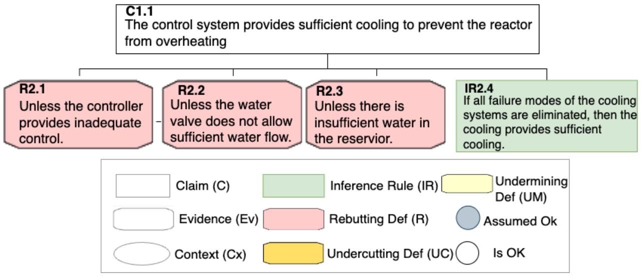
C1.1
The control system provides sufficient cooling to prevent the reactor from overheating
R2.1
Unless the controller provides inadequate control.
R2.2
Unless the water valve does not allow sufficient water flow.
R2.3
Unless there is insufficient water in the reservior.
IR2.4
If all failure modes of 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

Figure 1: Fragment of EA assurance case adapted from [10]

correct implementation of a system's requirements is crucial to prevent system failure. The latter could have severe consequences such as loss of lives and financial losses [7, 21]. Hence, several industry standards, including DO-178C in avionics [18] and ISO 26262 in the automotive sector, advocate for the use of ACs to support the certification of systems [11, 22]. ACs can be presented in various formats, such as straightforward text like structured prose, or through graphical representations [4] (e.g, GSN [14], CAE [2], and EA [13]).

2.2 Eliminative Argumentation

The EA notation allows constructing arguments and evaluating confidence in these arguments by relying on the notion of *defeasible reasoning* [10, 13]. The latter supports the recursive challenging of claims to progressively eliminate the doubts (defeaters) they may embed and, consequently, increase the confidence in these arguments [10].

An eliminative argument is comprised of five key components, i.e., Claims, Evidence, Inference Rules, Defeaters, and Argument Terminators [13]. Claims (C) are assertions that require further argumentative support to establish their credibility. Evidence (E) pertains to observations, data, or artifacts that bolster claims. Strategies (S), which outline the method for arranging a group of claims or defeaters, adopt a "top-down" perspective, encapsulating a comprehensive method for substantiating a claim. Inference Rules (IR) are guidelines for logically amalgamating multiple claims or defeaters to back a higher-level claim. An optional "context" element can be included to provide more details about a primary element. Defeaters challenge the credibility of claims, evidence, and inference rules.

There are three types of defeaters, classified based on the elements they target, i.e., **rebutting defeaters (R)** that offer reasons why a claim might be false, **undermining defeaters (UM)** that present arguments why evidence might be unreliable, and **undercutting defeaters (UC)** that pinpoint flaws in an inference rule such that the validity of its premises doesn't necessarily guarantee the truth of its conclusion. Regarding Argument Terminators, the "Assumed OK" terminator signifies that further argument or evidence is unnecessary for a defeater, as its resolution is considered obvious. Conversely, the "Is OK" terminator is used for an inference rule, indicating it's a tautology without undercutting defeaters, where the premise is deductively equivalent to the conclusion. Figure 1 provides a fragment of an AC in EA notation for a chemical reactor.

2.3 LLMs and Their Applications

Large language models (LLMs) are advanced AI models that have become prominent in natural language processing (NLP). Typically built as transformer models, like GPT-4, they are trained on extensive datasets, enabling them to generate text and respond to queries with notable accuracy. Key examples of LLMs include the GPT series by OpenAI, such as GPT-3.5 and GPT-4 [29], and Google's BERT [9]. The focus on GPT, particularly GPT-4 Turbo, stems from its advanced capabilities and widespread application potential. GPT-4 Turbo is an enhanced version of the GPT-4 model, known for its larger number of parameters and improved efficiency in generating responses. Note that GPT-4 Turbo also suffers from deterministic issues [29]. To mitigate these issues, techniques like fixing the seed in the random number generator or employing consistent prompting strategies can be used.

Recent studies have demonstrated various applications of LLMs to automate software engineering tasks [3, 5, 5, 25, 34, 35]. Chen et al. [5] focused on GPT-4's use in requirements engineering, specifically for generating goal-oriented models in compliance with the Goal-oriented Requirement Language (GRL). Their work highlighted GPT-4's substantial understanding of goal modeling. Chaaben et al. [3] utilized ChatGPT to generate UML models, introducing a novel method that employs few-shot prompt learning, thereby reducing the need for large datasets in domain modeling. Viger et al. [35] proposed using GPT-4 to identify defeaters in ACs to enhance their reliability. However, their research is still in its early stages and has not been empirically validated yet. Mahdi Mohajer et al. [25] introduced SkipAnalyzer which is a tool that leverages an LLM-powered agent for static code analysis. It autonomously detects and patches bugs filters out false positives, and is built on ChatGPT. Lastly, Sivakumar et al. [34] adapts the work of Chen et al. [5] to investigate the generation of safety cases using GPT-4, focusing specifically on its understanding of GSN.

3 APPROACH

Our work adapts the one of Chen et al. [5] and Sivakumar et al. [34] to the context of ACs formalized with EA. By using an LLM (i.e. GPT-4 Turbo) to automatically identify and mitigate defeaters in ACs, our objective is to emulate some argumentative and doubt-driven aspects of EA [10] to better support requirements verification and validation. Figure 2 shows a high-level overview of our proposed approach that leverages GPT-4 Turbo to generate (identify) and mitigate defeaters for ACs represented using EA. As shown in this figure, our approach consists of three phases. In this paper, we focus on Phase I. Future work will focus on Phase II and Phase III.

In **Phase I**, like Sivakumar et al. [34], we conduct a thorough analysis of the documentation on EA (e.g., [10, 13]) to extract the structural and semantic rules its notation embodies. We then derive structural and semantic-based questions from these rules. We combine these questions with EA generation-based questions. We use the resulting set of questions to assess GPT-4 Turbo's proficiency in EA by challenging its understanding of the syntax and semantics of EA, as well as its ability to generate EA concepts.

Phase II is centered around applying GPT-4 Turbo to identify potential defeaters within EA assurance cases. We plan to guide GPT-4 Turbo through Chain-Of-Thought prompting techniques to

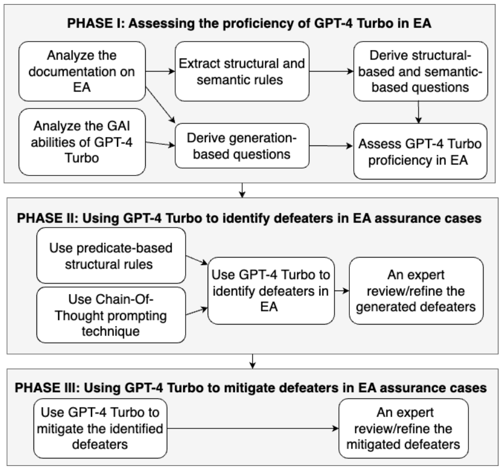
PHASE I: Assessing the proficiency of GPT-4 Turbo in EA
Analyze the documentation on EA
Extract structural and semantic rules
Derive structural-based and semantic-based questions
Analyze the GAI abilities of GPT-4 Turbo
Derive generation-based questions
Assess GPT-4 Turbo proficiency in EA
PHASE II: 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

Figure 2: Overview of the approach

clarify the reasoning behind defeater identification. This involves either integrating the reasoning steps into the examples or soliciting a detailed explanation of the thought process from the LLM [20, 36]. Moreover, we plan to incorporate a strategy similar to that of Zhu, Zhaocheng, et al. [38], who proposed the Hypotheses-to-Theories (HtT) framework to establish a rule library for structured reasoning with LLMs. This is because prompting methods relying on an LLM's implicit knowledge are prone to "hallucinations" [23], leading to incorrect answers. We plan to adopt this technique, forming predicate-based rules from the structural rules EA embodies to prompt GPT-4 Turbo such that it could generate better defeaters. Additionally, we involve an expert to review and refine the defeaters generated by GPT-4 Turbo to ensure their validity and applicability.

Phase III relies on GPT-4 Turbo to mitigate the defeaters identified in Phase II. Like Phase II, an expert is engaged to review/ refine the mitigation strategies proposed by GPT-4 Turbo, enhancing the reliability of ACs and ensuring they withstand rigorous scrutiny.

4 EXPERIMENTAL SETUP

This preliminary study focuses on Phase I and is designed to assess the proficiency of GPT-4 Turbo in understanding and applying EA.

4.1 Research Objective

The goal of our preliminary study is to answer this research question (RQ): **Is GPT-4 Turbo sufficiently proficient in the EA notation?** Like Chen et al. [5] and Sivakumar et al. [34], to investigate this RQ, we formulated 22 specific questions based on the EA documentation. We rely on them to assess the proficiency of GPT-4 Turbo in understanding and applying the structural and semantic rules of EA and its ability to generate EA concepts (e.g., defeaters).

4.2 Extraction Of The Structural and Semantic Rules Of EA

The first step of our approach consists in analyzing existing documentation (e.g., [10, 13]) on EA to extract the set of structural and semantic rules that EA embodies. Table 1 reports the structural and semantic rules that we extracted. Extracting rules is a critical component of our research, forming the core foundation for crafting essential questions relevant to RQ. Semantic rules are focused on the text within EA elements and the meaning of that text. On the other hand, structural rules address the arrangement and design of EA elements, conveying the appropriate structure of each element and the nature of its relationships with other elements.

4.3 Generation Of Questions To Assess GPT-4 Turbo's Proficiency In EA

We crafted an initial batch of 22 questions categorized in three sections, i.e., structural, semantic, and generation-based questions. The structural questions aim to assess GPT-4 Turbo's comprehension of EA notation i.e. EA structural rules. In contrast, the semantic questions examine its understanding of the semantic rules of EA. Lastly, the generation-focused questions are designed to test GPT-4 Turbo's capability in effectively generating EA elements, with a specific emphasis on defeaters. We developed eight structural, seven semantic, and seven generation-based questions. The complete list of these questions together with supplemental material is available on GitHub[1]. Table 2 presents sample questions for each category.

4.4 GPT-4 Turbo Setting

In our study, we utilized the OpenAI API[2] to interact with the GPT-4 Turbo model. A key methodological decision was ensuring deterministic responses from the model. This was achieved by setting the seed parameter in the API. The seed parameter is vital for generating consistent outputs from GPT-4 Turbo for the same input. It initializes the model's internal random number generator to a fixed state, thereby ensuring reproducibility and consistency of the responses generated by GPT-4 Turbo for identical prompts.

4.5 Prompting Process

In our study, we followed the best practices of prompt engineering as outlined in OpenAI's guide[3] to interact with GPT-4 Turbo. The process involved careful construction of both 'system' and 'user' prompts to guide the model effectively. The 'user' prompts were the direct questions posed to the model, designed to extract specific information or analysis. Meanwhile, for 'system' prompts, the primary objective was to orient GPT-4 Turbo appropriately, ensuring it understood its role as an assistant in addressing our inquiries. The system prompt that we used is provided in the box below:

System Prompt: You are an assistant that helps me answer questions about Eliminative Argumentation. Eliminative Argumentation is a method used in ACs, particularly in the fields of software engineering and system safety. It focuses on systematically identifying and eliminating potential causes of failure to strengthen the assurance of system safety and reliability. Answer each question separately and try to generate the samples even if they are simple. Your answers should be concise and to the point. It should not be more than 2-3 lines.

[1] Supplemental material link: https://github.com/anonymousforge2024/Forge2024
[2] https://openai.com/api/
[3] https://platform.openai.com/docs/guides/prompt-engineering

Table 1: EA Structural and Semantic rules

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 → 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 → 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.

Table 2: Sample Questions for assessing GPT-4 Turbo's Proficiency in EA

Category	Sample Question
Structural	What are the different types of defeaters in Eliminative Argumentation?
Semantic	How should a claim be structured in Eliminative Argumentation? i.e., mention whether it can be in the form of noun-phrase, verb-phrase or predicate.
Generation-based	Generate me a sample Claim and a Rebutting defeater that defeats it. Show it in structured prose.

4.6 Assessment Process

GPT-4 Turbo generates each EA concept in the structured prose complying with EA. To evaluate each of the GPT-4 Turbo's responses to the 22 questions, we had two researchers with extensive expertise in EA to assess these responses. They independently rated each of GPT-4 Turbo's answers on a linear scale ranging from one (totally correct) to five (incorrect). To assess the consistency and reliability of these ratings, we rely on the Kendall rank correlation coefficient as in Chen et al. [5]. The value of that correlation coefficient progresses from - 1 to 1. A value close to -1 means the level of agreement between raters is almost close to none. A value close to 1 means the level of agreement is strong. We rely on an online tool i.e. **Gigacalculator**[4] to automatically assess the values of the Kendall rank coefficient with a confidence level of 95%.

5 PRELIMINARY RESULTS

Two researchers evaluated the answers of GPT-4 to the 22 questions. The resulting correlation between their ratings is **0.75**. That strong correlation level underscores a robust agreement between the two assessors and a high level of consistency in their ratings.

Table 3: Average ratings of questions in RQ

Structural	Semantic	Generation-based	Avg
1.125	1.78	1.35	1.40

Table 3 reports the average ratings of the answers GPT-4 Turbo provided to structural, semantic, and generation-based. The average of the ratings achieved by GPT-4 Turbo is **1.40**, which is close to 1. That value reflects its strong grasp of the essential elements of EA notation. In the context of the grading systems used by many universities, that average of ratings equates to a **grade of A**.

GPT-4 Turbo achieved excellent proficiency when answering structural-based questions. It also showed commendable performance when answering generation-based questions, effectively creating EA elements, particularly various types of defeaters. However, GPT-4 Turbo's understanding of the semantics of EA elements was less robust. It would be beneficial to employ specific prompting techniques to enhance its comprehension of EA semantics.

6 CONCLUSION

Our investigation into the capabilities of GPT-4 Turbo has revealed its excellent proficiency in understanding and applying EA notation. This underscores the model's potential as a valuable tool for future endeavors in the identification (Phase II) and mitigation of defeaters (Phase III) within ACs represented using EA. We are optimistic about the role of GPT-4 Turbo in enhancing the robustness of ACs, particularly in mission-critical systems where the assurance of non-functional requirements is paramount.

[4] https://www.gigacalculator.com/calculators/correlation-coefficient-calculator.php

REFERENCES

[1] A. B. Belle and Y. Zhao. 2023. Evidence-based decision-making: On the use of systematicity cases to check the compliance of reviews with reporting guidelines such as PRISMA 2020. *Expert Systems with Applications* 217 (2023), 119569.

[2] P. Bishop and R. Bloomfield. 2000. A methodology for safety case development. In *Safety and Reliability*, Vol. 20. Taylor & Francis, 34–42.

[3] M. Chaaben, L. Burgueño, and H. Sahraoui. 2023. Towards using few-shot prompt learning for automating model completion. In *ICSE-NIER*. IEEE, 7–12.

[4] M. Chelouati, A. Boussif, J. Beugin, and E. El Koursi. 2023. Graphical safety assurance case using Goal Structuring Notation (GSN)—challenges, opportunities and a framework for autonomous trains. *Reliability Engineering & System Safety* 230 (2023), 108933.

[5] B. Chen, K. Chen, S. Hassani, Y. Yang, D. Amyot, L. Lessard, G. Mussbacher, M. Sabetzadeh, and D. Varró. 2023. On the use of GPT-4 for creating goal models: an exploratory study. In *REW*. IEEE, 262–271.

[6] E. Cioroaica, B. Buhnova, D. Schneider, I. Sorokos, T. Kuhn, and E. Tomur. 2022. Towards the Concept of Trust Assurance Case. In *TrustCom*. IEEE, 1581–1586.

[7] J. L. de La Vara, M. Borg, K. Wnuk, and L. Moonen. 2016. An industrial survey of safety evidence change impact analysis practice. *TSE* 42, 12 (2016), 1095–1117.

[8] E. Denney, G. Pai, and I. Habli. 2015. Dynamic safety cases for through-life safety assurance. In *ICSE*, Vol. 2. IEEE, 587–590.

[9] J. Devlin, M. Chang, K. Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] S. Diemert and J. Joyce. 2020. Eliminative Argumentation for Arguing System Safety-A Practitioner's Experience. In *SysCon*. IEEE, 1–7.

[11] Simon Foster, Yakoub Nemouchi, Mario Gleirscher, Ran Wei, and Tim Kelly. 2021. Integration of formal proof into unified assurance cases with Isabelle/SACM. *Formal Aspects of Computing* 33, 6 (2021), 855–884.

[12] Health Foundation. 2012. Evidence: Using Safety Cases in Industry and Healthcare.

[13] J. B. Goodenough, C. B. Weinstock, and A. Z. Klein. 2015. Eliminative argumentation: A basis for arguing confidence in system properties. *SEI, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005* (2015).

[14] The Assurance Case Working Group. 2021. Goal Structuring Notation Standard Version 3. https://scsc.uk/r141C:1?t=1

[15] A. Groza, I. A. Letia, A. Goron, and S. Zaporojan. 2015. A formal approach for identifying assurance deficits in unmanned aerial vehicle software. In *ICSEng*. Springer, 233–239.

[16] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly. 2015. Weaving an assurance case from design: a model-based approach. In *HASE*. IEEE, 110–117.

[17] R. Hawkins, T. Kelly, J. Knight, and P. Graydon. 2011. A new approach to creating clear safety arguments. In *SSS*. Springer, 3–23.

[18] L. A. Johnson et al. 1998. DO-178B: Software considerations in airborne systems and equipment certification. *Crosstalk, October* 199 (1998), 11–20.

[19] K. Khakzad S., Alvine B. Belle, T. C. Lethbridge, O. Odu, and M. Sivakumar. 2023. A PRISMA-driven systematic mapping study on system assurance weakeners. *arXiv preprint arXiv:2311.08328* (2023).

[20] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. 2022. Large language models are zero-shot reasoners. *NeuRIPS* 35 (2022), 22199–22213.

[21] Z. Langari and T. Maibaum. 2013. Safety cases: a review of challenges. In *ASSURE*. IEEE, 1–6.

[22] Yutaka Matsuno, Toshinori Takai, and Shuichiro Yamamoto. 2020. 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.

[23] T. R. McIntosh, T. Liu, T. Susnjak, P. Watters, A. Ng, and M. N. Halgamuge. 2023. A culturally sensitive test to evaluate nuanced gpt hallucination. *TAI* 1, 01 (2023), 1–13.

[24] C. Menghi, T. Viger, A. Di Sandro, C. Rees, J. Joyce, and M. Chechik. 2023. Assurance case development as data: A manifesto. In *ICSE-NIER*. IEEE, 135–139.

[25] M. M. Mohajer, R. Aleithan, N. S. Harzevili, M. Wei, A. B. Belle, H. V. Pham, and S. Wang. 2023. SkipAnalyzer: An Embodied Agent for Code Analysis with Large Language Models. *arXiv preprint arXiv:2310.18532* (2023).

[26] Faiz UL Muram, Barbara Gallina, and Laura Gómez Rodríguez. 2018. Preventing omission of key evidence fallacy in process-based argumentations. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. IEEE, 65–73.

[27] A. Murugesan, I. Hong Wong, R. Stroud, J. Arias, E. Salazar, G. Gupta, R. Bloomfield, S. Varadarajan, and J. Rushby. 2023. Semantic Analysis of Assurance Cases using s (CASP). In *GDE Workshop in ICLP*.

[28] S. Nair, J. L. De La Vara, M. Sabetzadeh, and L. Briand. 2014. An extended systematic literature review on provision of evidence for safety certification. *IST* 56, 7 (2014), 689–717.

[29] OpenAI. 2023. GPT 4. https://openai.com/research/gpt-4

[30] OpenAI. 2023. New Models and Developer Products Announced at DevDay. https://openai.com/blog/new-models-and-developer-products-announced-at-devday. Accessed: 2024-01-14.

[31] J. Rushby. 2013. Logic and epistemology in safety cases. In *SafeComp*. Springer, 1–7.

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

[33] SACM 2021. *Structured Assurance Case Metamodel*. SACM.

[34] M. Sivakumar, A. B. Belle, J. Shan, and K. Khakzad S. 2023. GPT-4 and Safety Case Generation: An Exploratory Analysis. *arXiv preprint arXiv:2312.05696* (2023).

[35] T. Viger, L. Murphy, S. Diemert, C. Menghi, A. Di, and M. Chechik. 2023. Supporting Assurance Case Development Using Generative AI. In *SAFECOMP 2023*.

[36] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Quoc V Le, D. Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeuRIPS* 35 (2022), 24824–24837.

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

[38] Z. Zhu, Y. Xue, X. Chen, D. Zhou, J. Tang, D. Schuurmans, and H. Dai. 2023. Large Language Models can Learn Rules. *arXiv preprint arXiv:2310.07064* (2023).