EXAMINING THE EFFECTIVENESS OF GENERATIVE ARTIFICIAL INTELLIGENCE FOR THE IDENTIFICATION OF DEFEATERS IN ASSURANCE CASES

KIMYA KHAKZAD SHAHANDASHTI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

APRIL 2024

Abstract

Assurance cases are structured arguments that allow verifying the correct implementation of the created systems' non-functional requirements (e.g., safety, security, reliability). This allows for preventing system failure. The latter may result in loss of life, severe injuries, large-scale environmental damage, property destruction, and major economic loss. Assurance cases support the certification of systems in compliance with industrial standards (e.g., DO-178C, ISO26262). However, the presence of assurance weakeners - deficits and logical fallacies - signals gaps in evidence and reasoning. Addressing this, our research presents a comprehensive taxonomy for categorizing these assurance weakeners, alongside proposed management strategies. The taxonomy divides weakeners into four categories of uncertainty: aleatory, epistemic, ontological, and argumentation. It also categorizes management approaches into representation, identification, and mitigation. A critical aspect of strengthening assurance cases involves identifying argumentation uncertainty or defeaters. To automate this process, we explore the capabilities of GPT-4 Turbo, a sophisticated large language model by OpenAI. We focus on its application in detecting defeaters within assurance cases represented using Eliminative Argumentation(EA) notation. Our initial evaluation assesses GPT-4 Turbo's proficiency in understanding and applying this notation, a key factor in effectively generating defeaters. The results indicate that GPT-4 Turbo is highly adept in EA notation, demonstrating its potential to generate a diverse range of defeaters, thereby enhancing the robustness and reliability of assurance cases. Moreover, we used GPT-4 Turbo to identify defeaters which demonstrated effective proficiency.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr Alvine Boaye Belle, for her exceptional guidance, unwavering support, and expertise throughout my studies at York University. Her expertise, endless patience, and valuable feedback have not only improved the research presented in this thesis but have also had a significant impact on my professional and personal development. Working under such insightful supervision has been an invaluable experience, and I will always be grateful for the valuable lessons learned.

Furthermore, I would like to express my appreciation to my family, Zinat Yosefi Moghadam and Paknoush Khakzad Shahandashti for their enduring love, understanding, and encouragement. Their belief in me has been a constant source of motivation, pushing me to work harder toward my goals. Additionally, I owe special thanks to Amirhossein Khakzad Shahandashti for his unwavering emotional support and encouragement throughout my studies. I am genuinely thankful for the love and support they have generously provided.

Furthermore, I extend my gratitude to my friends Zahra Hosseini, Sara Mahmoudi, Mana Akbarzadegan, Reyhan Khooban, Pouria Poursarvi, Bita Ghoudsi and Sina Tavakoli for their emotional support throughout this journey.

Last but not least, I would like to acknowledge our lab members, Mohammad Mahdi Mohajer, Mithila Sivakumar and Oluwafemi Odu, whose collaborative spirit and shared passion for research have undoubtedly enhanced the depth and quality of this work. I would especially like to thank the formal and informal members of my supervisory committee Dr Timothy C. Lethbridge, Song Wang and Hadi Hemmati for their contributions and camaraderie during this research.

This thesis is dedicated to the memory of those who lost their lives on Flight 752. Their spirits and stories inspire us to strive for a better world, reminding us of the preciousness of life and the importance of pursuing our goals with determination and compassion.

Table of Contents

List of Tables

List of Figures

Glossary

Logical Fallacy A mistake or flaw in the reasoning of an argument.

Assurance Deficit Is a gap in knowledge that prohibits us from having complete confidence in the assurance case.

Assurance Weakener It encompasses of logical fallacies and assurance deficits.

Uncertainty Is the presence of imperfect or unknown information.

Defeater It challenges the credibility of claims, evidence, and inference rules of assurance cases.

Rebutting Defeater It presents doubt about the claims in assurance cases.

Undermining Defeater It presents doubt about the evidence in assurance cases.

Undercutting Defeater It presents doubt about the inference rules in assurance cases.

Chapter 1

Introduction

1.1 Context

Cyber-physical systems (CPSs) play pivotal roles in safety- or mission-critical applications, demanding rigorous assurance of their safe operation [1]. Many CPSs, such as autonomous vehicles and unmanned aerial vehicles, harness the power of Machine Learning. However, these systems sometimes struggle with accurate pattern recognition and prediction when confronted with unanticipated edge cases [2]. Given their multifaceted functionalities, these intricate systems can execute crucial operations across various domains, from healthcare to aviation. The immense responsibility these systems bear, particularly in scenarios directly influencing human safety, has intensified the emphasis on their dependability [3, 4, 5, 6, 2, 7]. Addressing this imperative, the field of system assurance has risen to the fore, leveraging assurance cases to provide evidence and articulate arguments verifying a system's adherence to its set specifications and aptitude for its designated role [1]. Establishing comprehensive industry standards and ensuring that autonomous technology makers adhere to them is vital for building consumer trust. Increasingly, the producers of these CPSs are using assurance cases to show regulatory bodies their adherence to current standards (e.g., ISO 26262 [8]). 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) [6, 9, 10]. 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) [11] and EA (Eliminative Argumentation) [12].

1.2 Motivation

Amidst the surge of attention toward the assurance process, it is essential to recognize the potential pitfalls that can undermine the reliability of assurance cases. One significant aspect that needs to be addressed is the presence of assurance *"weakeners"* (i.e., assurance deficits [13] and defeaters [12]) within the assurance argumentation structure. Figure 1.1 presents an example of a fragment of an AC using Eliminative Argumentation notation, along with its defeaters adapted from [12]. The claim "the light turns on" is the top-level claim. Defeaters R2.1, R2.2, and R2.3 represent doubts that challenge the validity of the claim C1.1.

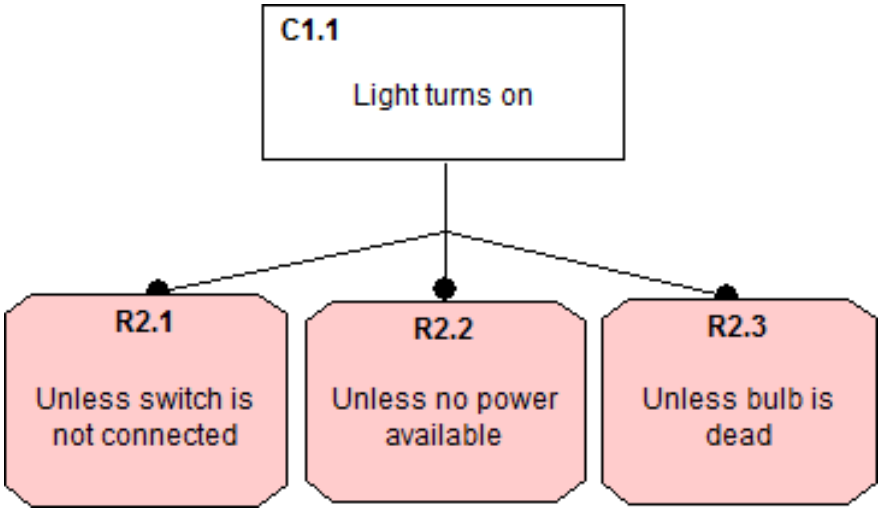

Figure 1.1: Example AC (adapted from [12])

The presence of fallacies in assurance cases can lead to false confidence in the resulting argumentation structure. A fallacious assurance case may, therefore, be inadequate to properly verify the correctness of the implementation of the requirements of the system under analysis. Hence, such a system can be prone to system failure. This can have severe consequences such as the death of people and financial losses [5, 14]. For example, automotive software failures have led to the loss of human lives and required automotive companies to recall their vehicles at significant costs to them [15, 16]. The fatal crash of the Nimrod military aircraft in Afghanistan in 2006 serves as a well-known example of how assurance weakeners can undermine an assurance case [17]. The post-crash investigation revealed that the Nimrod Safety Case was "fatally undermined by an assumption by all the organizations and individuals involved that the Nimrod was 'safe anyway'." This incident underscores the

importance of identifying and mitigating assurance weakeners in systems, as these factors contributed directly to the crash. Devising solutions to effectively manage (e.g., represent, detect, mitigate) assurance weakeners is therefore critical to yield more reliable systems.

This thesis first reports a taxonomy that categorizes assurance weakeners and approaches proposed to manage them at the modelling level. By utilizing this taxonomy, researchers and practitioners (e.g., corporate safety analysts and regulatory authorities) can therefore: 1) enhance their understanding of existing assurance weakeners solutions and 2) have a starting point to devise effective solutions to manage assurance weakeners, ultimately leading to more reliable mission-critical systems.

A few approaches (e.g., [18, 19, 20, 21, 22]) presented in the taxonomy allow identifying defeaters in ACs. However, despite these approaches, the current state of research predominantly offers methodologies that are not automatic, requiring significant manual intervention and expertise in identifying and addressing defeaters in ACs. By integrating the strengths of automation with the nuanced understanding of human experts, a semi-automatic approach holds the promise of significantly improving the efficiency and effectiveness of defeater identification in ACs, paving the way for more reliable and secure system certifications.

Maksimov et al. [23] highlight that assurance cases can significantly expand in size and complexity. The majority of the information within these documents is presented in natural language, which poses challenges for automated analysis. Consequently, manually creating and challenging arguments are recognized as being labour-intensive, time-consuming, and prone to errors [16, 24]. 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 [25, 26]. To address that gap in the research, we rely on Large language models (LLMs) to automatically generate defeaters in ACs represented using EA. LLMs are advanced generative Artificial Intelligence (AI) models that have become prominent in natural language processing (NLP) [27] such as GPT-4 Turbo. In our work, we adopt GPT-4 Turbo, owing to its increased efficiency in producing responses and its capability to yield deterministic outputs [28]. In this work, we first assessed GPT-4 Turbo efficiency and revealed its excellent proficiency in understanding and applying EA notation. Second, in this thesis, we aim to use GPT-4 Turbo to automatically generate the defeaters.

1.3 Research Objectives

The goal of our study is to answer the following research questions (RQs):

RQ1: What are the different assurance *"weakeners"* categories and what approaches allow managing them? What are the trends, patterns and relationships characterizing the literature on assurance *"weakeners"*?

In this research question, we aim to perform a qualitative synthesis to classify various forms of *"assurance weakeners"* found in the assurance case literature. That literature usually refers to assurance *"weakeners"* using several keywords: assurance deficits, uncertainty, fallacies, doubts, etc. We aim to unify that vocabulary by creating a map that structures it. This will provide future work with a controlled, unambiguous, and consistent vocabulary that can be used to refer to assurance weakeners. It also seeks to identify diverse approaches that can effectively handle these weakeners. Hence, the study aims to examine the approaches the literature has proposed to manage (e.g., represent, detect, fix/mitigate) assurance weakeners.

Moreover, we aim to perform a bibliometric synthesis to provide some descriptive statistics to identify trends, patterns and relationships among the primary studies, authors, venues, and related information we retrieved from the primary studies. This will notably highlight the thematic evolution of the literature on assurance weakeners and help identify the most influential themes characterizing that literature.

RQ2: Is GPT-4 Turbo sufficiently proficient in the EA notation? To investigate this research question, we formulated 22 specific questions. We rely on these questions to assess the proficiency of GPT-4 Turbo in understanding and applying the structural and semantic principles of EA notation. These questions also allow gauging the generative AI abilities of GPT-4 Turbo in terms of EA concepts.

RQ3: Is GPT-4 Turbo able to generate defeaters in EA notation? In pursuit of our research objective, we plan to conduct distinct experiments designed to evaluate the effectiveness of GPT-4 Turbo in accurately generating defeaters.

1.4 Contributions

The contributions of the thesis are the following:

- **Contribution 1**: We presented a taxonomy (map) that provides a uniform categorization of assurance weakeners and approaches proposed to manage them at the modeling

level. The taxonomy classifies weakeners into four broad categories of uncertainty: aleatory, epistemic, ontological, and argumentation uncertainty. Additionally, it classifies approaches supporting the management of weakeners in three main categories: representation, identification and mitigation approaches.

- **Contribution 2**: We published at FORGE 2024 (AI Foundation Models and Software Engineering) – FORGE is co-located with ICSE 2024 – a conference paper focusing on the assessment of the impact of GPT-4 Turbo in generating defeaters for assurance cases. 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.

- **Contribution 3**: We have leveraged GPT-4 Turbo to automate the generation of defeaters, which are critical for identifying and countering potential arguments or weaknesses in reasoning. This advancement enables the production of more deterministic outputs, significantly enhancing the reliability and predictability of the system's responses.

- **Contribution 4**: We leveraged the Chain of thought (CoT) prompting technique for prompting our GPT-4 Turbo model. Our application of the CoT technique represents a significant methodological innovation, allowing for the explicit delineation of reasoning steps involved in problem-solving processes. This technique not only facilitates a more transparent and interpretable decision-making process but also significantly improves the system's ability to tackle complex problems by structuring its reasoning in a sequential, step-by-step manner.

- **Contribution 5**: We conducted four distinct experiments to assess how contextual information and examples in the prompts influence the AI's reasoning process and its effectiveness. We designed these experiments to measure the impact of context and examples on the AI's output quality and relevance, exploring the ways in which different types of prompts affect the system's performance across various tasks.

1.5 Thesis Structure

This thesis, comprised of submitted and published papers, is structured into several chapters. Chapter 2 covers key background concepts such as system assurance and Large Language Models. Chapter 3 reviews related work. Chapter 4 details the methodology used for our research questions. Chapter 5 presents our findings. Chapter 6 reports threats to validity. Chapter 7 outlines future research directions and concludes the thesis.

Chapter 2

Background Concepts

2.1 What is System Assurance, and how to support it?

Ensuring the reliability and trustworthiness of software is a fundamental component of the software development process, aiming to reduce potential hazards and guarantee superior results [29]. Assurance cases allow supporting system assurance. Mansourov and Campara [29] define an assurance case as a *"collection of verifiable claims, arguments, and evidence assembled to confirm that a particular system/service aligns with given requirements"*. An assurance case is a document that streamlines communication among various system stakeholders (e.g., suppliers, acquirers), as well as between the operator and regulator. Its main purpose is to effectively convey insights about the system's prerequisites, such as safety, security, and reliability [10, 6, 30]. There are several types of assurance cases, each focusing on specific non-functional requirements, including dependability, safety, security, and specifically, reliability and trustworthiness [21, 31, 32, 25, 9].

Assurance cases are becoming increasingly popular, especially in sectors where safety is paramount. These include healthcare, railways, automotive, and aviation [33, 23]. As a result, creating compelling assurance cases often becomes crucial to secure the regulatory acceptance required to deploy such systems [34]. Hence, the use of assurance cases supports systems certification. This allows for preventing system failure. The latter may have catastrophic consequences such as life loss, major injuries, environmental threats, property damages, and financial losses [4, 35]. Noteworthy, various industry standards, such as DO-178C (in the avionics field) [36] and ISO 26262 (pertaining to automotive), advocate for the use of

assurance cases to support system certification.

Maksimov et al. [23] highlight that assurance cases can significantly expand in size and complexity. The majority of the information within these documents is presented in natural language, which poses challenges for automated analysis. Consequently, manually creating and challenging arguments are recognized as being labour-intensive, time-consuming, and prone to errors [16, 24].

2.2 What Are Assurance Weakeners?

In the literature on assurance cases, several expressions have been used to refer to *assurance weakeners*. For instance, an *assurance deficit* refers to *"a gap in knowledge that prohibits us from having complete confidence in the assurance case"* [13]. Additionally, according to Goodenough et al. [37], arguments about system properties are open to revision based on new information. In the realm of *defeasible reasoning*, these challenges are termed *defeaters*. Hence, *defeaters* are elements that challenge the validity of an argument [38]. *Defeaters* usually come in three forms [39, 37]: *rebutting*, which provides a counter-example; *undermining*, which raises doubts about evidence; and *undercutting*, which questions conclusions under specific conditions when initial premises are true.

In [25, 26], the notion of assurance weakeners takes the form of the *doubts* that may undermine the trust in the assurance of a system: *epistemic doubt* and *logic doubt*. *Epistemic doubt* refers to the completeness and accuracy of the knowledge regarding the various aspects of the system at hand. These aspects include the knowledge about its requirements, environment, implementation, and hazards. *Logic doubt* refers to the accuracy of the reasoning about the system's design at hand, given the knowledge we have about the system.

Burgueño et al. [40] defines *uncertainty* as *"the presence of imperfect and/or unknown information"*. That uncertainty applies to a system's predictions, estimations, measurements, and properties. According to Gansch et al. [41] as well as Schleiss et al. [42], there are three types of uncertainty: *aleatory uncertainty*, which means that it stems from inherent variation, *epistemic uncertainty*, which means that it arises from incomplete knowledge during the modelling process and *ontological uncertainty* which means as a condition of complete ignorance in the model of a relevant aspect of the system. Muram et al. [43] highlight that creating arguments is complex and error-prone. It mentions that an argumentation fallacy is

"a mistake or flaw in the reasoning of an argument". Such fallacies may lead to overconfidence in a system and the acceptance of certain faults, contributing to safety-related failures. In safety arguments, fallacies come in various forms. Greenwell et al. [44] categorized common *logical fallacies* into three types: *relevance fallacies*, which offer no meaningful evidence; *acceptability fallacies*, which provide contradictory or insufficient evidence; and *sufficiency fallacies*, which illustrate a lack of enough evidence to support claims.

Interestingly, studies such as [39, 31] use aforementioned terms interchangeably, including assurance deficits, defeaters, and rebuttals. For consistency's sake, we refer to these various terms as ***assurance weakeners*** since they usually undermine the confidence one may have in the assurance argumentation structure.

2.3 How to represent Assurance Cases?

Assurance cases can be represented using unstructured text (natural language) [6], semi-structured text (prose), and graphical notations. Graphical notations have gained popularity due to their ability to express clear and well-structured arguments [45]. Graphical notations include the GSN (Goal Structuring Notation) [46], CAE (Claims-Arguments-Evidence) [47] and EA(Eliminative Argumentation) [12]. GSN is currently the most widely used graphical notation for representing assurance cases [48, 49]. The GSN Standard Working Group specifies the GSN [46]. To enhance standardization and interoperability, the Object Management Group (OMG) recently introduced the Structured Assurance Case Metamodel (SACM) for assurance case representation [50]. Both GSN and CAE are aligned with SACM. The nature of SACM arguments varies: they can be boolean, probabilistic, qualitative, etc [51]. Probabilistic arguments can be based on simulations, while qualitative arguments may be based on regulations or even the specifications of a manufacturer [51].

2.3.1 Goal Structuring Notation

GSN (Goal Structuring Notation) enables the representation of an assurance case through a tree-like structure known as a *goal structure*. Its core elements are Goals, Strategies, Solutions, Contexts, Assumptions, and Justifications [46]. *Goals* articulate specific claims (a true or false statement) within the argument. *Strategies* outline the inference that exists between a goal and its supporting goals. *Solutions* reference specific pieces of evidence. *Contexts* provide

relevant contextual information or statements. Additionally, context nodes are used to express constraints on the scope of the domain over which the assurance case is intended to be valid. *Assumptions* describes intentionally unsubstantiated statements. *Justifications* rationalize the underlying assumptions or claims. The *Undeveloped Element* decorator signifies that further development is needed in the argument, applicable to both goals and strategies. Relationships between these elements are defined by *SupportedBy* and *InContextOf*. The *SupportedBy* relationship, represented as a line with a solid arrowhead, implies an inference from the target of the *SupportedBy* (the supporting element) to the element being supported. The *InContextOf* relationship, shown as a line with a hollow arrowhead, declares a contextual relationship. Figure 2.1 provides an example of a safety assurance case in the GSN developed for UAV (Unmanned Aerial Vehicle) Collision Avoidance [51].

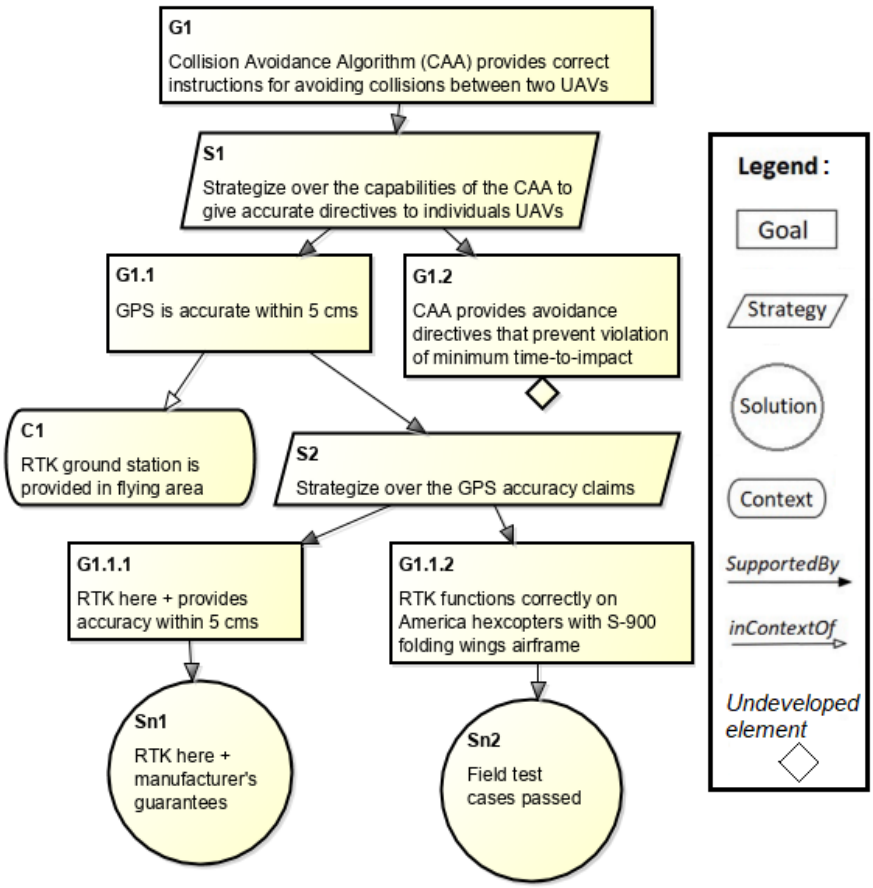

Figure 2.1: Partial safety case for UAV Collision Avoidance (adapted from [51])

Regarding the dialectic extension of GSN in V3 [46], the integration of a dialectic process

notably enhances the structure by introducing a methodical, truth-seeking approach to ACs. Central to this approach is the investigation of truth through the discovery and identification of doubt, which can be depicted and the residual doubt exposed. A significant aspect of this dialectic extension is the introduction of a status that can be assigned to elements and relationships within GSN, notably the "Defeated" status. In the dialectic extension of GSN, a notable limitation is the absence of explicit notations for distinguishing among the different types of defeaters. Consequently, we use Eliminative Argumentation which we describe in the next section.

2.3.2 Eliminative Argumentation

The Eliminative Argumentation (EA) notation allows constructing arguments and evaluating confidence in these arguments by relying on the notion of *defeasible reasoning* [52, 12]. The latter supports the recursive challenging of claims to progressively eliminate the doubts they may embed and, consequently, increase the confidence in these arguments [52]. EA notation builds on the foundational ideas of GSN. Just as in GSN, EA employs a directed, non-cyclic graph to lay out the structure of an argument. Both EA and GSN methodologies employ a hierarchical tree-like design where a principal system claim (positioned at the "root") gets broken down into more detailed sub-claims, eventually anchoring in concrete evidence. This offers a clear linkage between the evidence presented and the resulting claims. A distinctive feature of EA, as compared to GSN, is its capacity to capture and represent "doubts" or defeaters regarding the authenticity of claims, the backing evidence, or the derived logical conclusions [17]. An eliminative argument is comprised of five key components: Claims, Evidence, Inference Rules, Defeaters, and Argument Terminators [12]. Claims (C) are assertions that require further argumentative support to establish their credibility. Evidence (E) pertains to observations, data, or artifacts that bolster claims. 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 element they target:

- **Rebutting defeaters** offer reasons why a claim might be false. They directly challenge the truth of the conclusion itself.

- **Undermining defeaters** present arguments why evidence might be unreliable. They target the support structure of the argument, suggesting that the evidence or reasons provided do not hold up under scrutiny.
- **Undercutting defeaters** pinpoint flaws in an inference rule such that the validity of its premises doesn't necessarily guarantee the truth of its conclusion. They focus on the connection between premises and conclusion, suggesting that even if the premises are true, the conclusion might not be.

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 2.2 is adapted from [12]. That figure illustrates an EA assurance case (i.e., an assurance case complying with the EA notation) for the claim that the light turns on, bearing structural resemblance to Figure 5. The primary claim (C1.1: "Light turns on") and the corresponding evidence (Ev3.1: "Test results confirm the bulb doesn't make a rattling sound when shaken") remain consistent. The rebutting defeaters R2.1, R2.2, and R2.3, which suggest reasons why C1.1 might not hold true. Inference rules of IR2.4 and IR3.2 allow us to pinpoint scenarios where these rules may not be valid, specifically when their premises don't necessarily lead to their conclusions. For instance, the inference rule IR3.2 ("A non-rattling bulb implies it's not defective") becomes questionable if the bulb in question isn't an incandescent bulb, like an LED bulb, where the absence of rattling doesn't reliably indicate whether it's defective. This exception is noted in the undercutting defeater UC4.2, "Unless the bulb is of a non-incandescent type."

2.4 What Is A Large Language Model?

Large language models (LLMs) are advanced AI models that have become prominent in natural language processing (NLP) [27]. 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 [53]. Key examples of LLMs include the GPT series by OpenAI, such as GPT-3.5 and GPT-4 [54], and Google's BERT [55] (Bidirectional Encoder Representations from Transformers). The focus on GPT, particularly GPT-4 Turbo, stems from its advanced

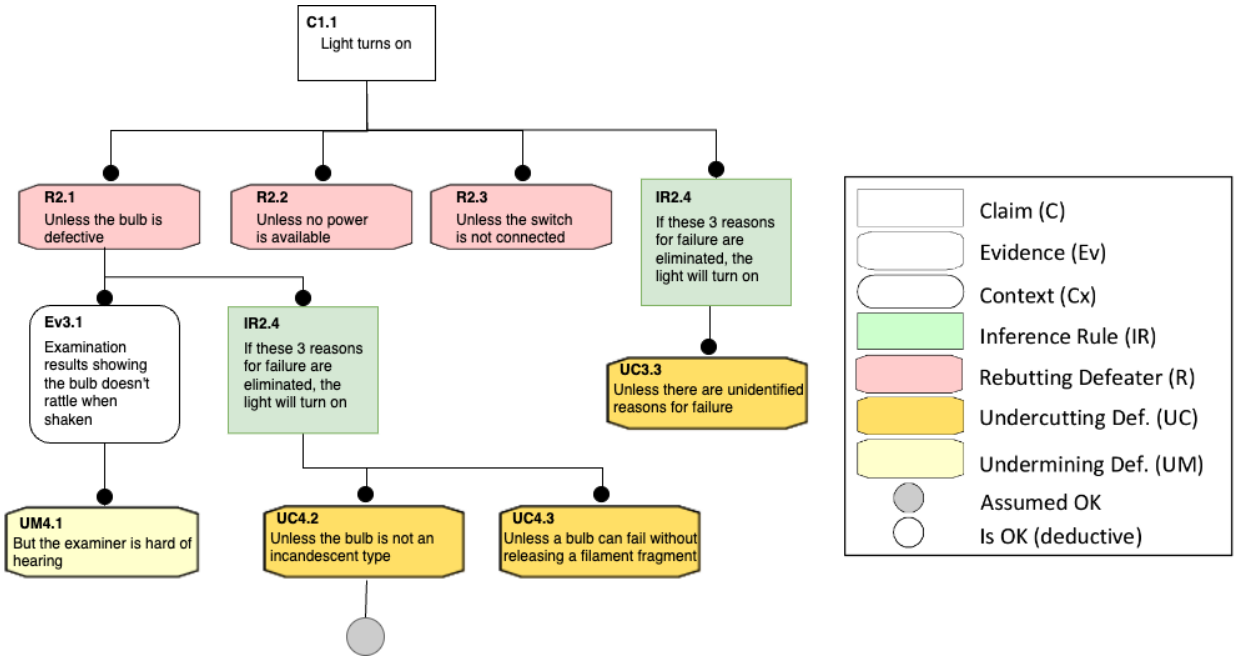

Figure 2.2: Sample EA Assurance Case (adapted from [12])

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 [56]. It works by processing input text and generating responses based on its training, employing a vast number of parameters. The seed parameter in GPT-4 Turbo enables reproducible outputs by making the model return consistent completions most of the time [28].

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 [57]. Chain of thought (CoT) prompting enhances reasoning by structuring complex problems into smaller, manageable sub-tasks [58, 57]. 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 [59]. K-shot prompting, which provides models with k examples to learn from, also sees benefits from incorporating CoT strategies [60]. 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 [61].

Chapter 3

Related work

This section discusses the related reviews that specifically addressed the concept of assurance weakeners to establish the rationale for conducting our systematic review. Moreover, it also discusses the related work that uses LLM in the Software Engineering domain.

3.1 Related Reviews On Assurance Weakeners

Table 3.1 specifies the focus of each corresponding review compared to our work. Greenwell et al. [62] introduced a classification system for logical fallacies found within safety cases. The purpose of this system was to provide a framework for developers and regulators, enabling them to pinpoint and rectify these logical fallacies. They grouped these into categories of relevance, acceptability, and sufficiency fallacies. A subsequent refinement by Greenwell et al. [44] expanded on this, outlining a taxonomy that comprised 33 typical logical fallacies in safety arguments. These fallacies were grouped into seven distinct categories: Circular Reasoning, Diversionary Arguments, Fallacious Appeals, Mathematical Fallacies, Unsupported Assertions, Omission of Key Evidence, and Linguistic Fallacies.

Ramirez et al. [63] shifted their gaze to uncertainties within Dynamically Adaptive Systems (DAS), especially in the context of intelligent vehicle systems. They crafted a taxonomy capturing potential sources of uncertainty throughout system development and spotlighted methods to tackle particular uncertainties. Contrarily, Duan et al. [33] categorized uncertainty as aleatory and epistemic and examined how these types were tackled in assurance cases. These studies predominantly emphasized uncertainty. Graydon and Holloway [64], on

Reference	**Publication Year**	**Focus**	**Title**
Greenwell et al. (2005) [62]	2005	Classification of Logical Fallacies	A Taxonomy of Fallacies in System Safety Arguments
Greenwell et al. (2006) [44]	2006	Classification of Logical Fallacies	A Taxonomy of Fallacies in System Safety Arguments
Ramirez et al. [63]	2012	Classification of Uncertainty	A Taxonomy of Uncertainty for Dynamically Adaptive Systems
Duan et al. [33]	2017	Classification and Assessment of Uncertainty	Reasoning About Confidence and Uncertainty in Assurance Cases: A Survey
Graydon and Holloway [64]	2017	Confidence Assessment	An Investigation of Proposed Techniques for Quantifying Confidence in Assurance Arguments
Maksimov et al. [23]	2019	Assessment of structure and content	A Survey of Tool-Supported Assurance Case Assessment Techniques
Mohamad et al. [65]	2021	Security Assurance Cases	Security Assurance Cases - State of the Art of an Emerging Approach
Khakzad et al. [66] (i.e., our work)	2023	Assurance Weakeners	A PRISMA-driven Systematic Review on Assurance Weakeners

Table 3.1: Comparison with Related Reviews

the other hand, delved into methodologies for gauging confidence in assurance arguments. In a similar vein, Maksimov et al. [23] undertook an analysis of how assurance cases (specifically measuring confidence and uncertainty) are evaluated within various software tools. Nevertheless, their primary focus remained on the evaluative dimensions of assurance cases. Mohamad et al. [65] embarked on a thorough review dedicated to Security Assurance Cases (SAC), a methodological approach to scrutinizing system security parameters. Their survey was notably centred around security assurance cases, bypassing other forms such as safety cases.

Therefore, the analysis of existing reviews indicates that none of them have proposed a comprehensive classification of the various assurance weakeners or a systematic overview of approaches proposed to address them. Our work, in contrast, aims to bridge this gap by focusing on the representation, detection, and mitigation of various types of assurance weakeners, including aleatory uncertainty, epistemic uncertainty, ontological uncertainty, and argument uncertainty (logical fallacies), as well as doubts (logic and epistemic doubt). Furthermore, our systematic review exclusively focuses on approaches addressing assurance weakeners at the model level. Addressing assurance weakeners at such a high level of abstraction enables more comprehensive and abstract reasoning about these weakeners, thereby simplifying their management. Our systematic review is, therefore, poised to significantly contribute to filling this crucial gap in the existing literature.

3.2 Approaches relying on LLM to Support Software Engineering tasks

Recent studies have demonstrated various applications of Large Language Models to automate Software Engineering (SE) tasks.

In requirements engineering in SE tasks, Zhang et al. [67] performed an initial assessment of ChatGPT's capabilities in zero-shot requirement retrieval across four distinct datasets and two analysis tasks. While these findings are in the early stages, they suggest the promising potential of Large Language Models to serve as aids in streamlining the requirements engineering process. Meanwhile, Luo et al. [68] engaged in prompt engineering using BERT to automate the categorization of requirements. Luitel et al. [69] targeted the aspect of requirements completeness, employing BERT to produce suggestions for completing partially

obscured requirements. Chen et al. [70] 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.

In the domain of software testing, Li et al. [71] implemented a combined approach using differential testing alongside ChatGPT to improve its capacity for generating test cases that reveal faults in programs with bugs. For testing graphical user interfaces, Sun et al. [72] translated descriptions of the user interface into textual form and then queried ChatGPT regarding the subsequent action it would opt for, based on the description. The response was then transformed into tangible GUI interactions. Additionally, Xie et al. [73] devised prompts aimed at generating tests by analyzing a project to produce a tailored contextual focus that encapsulates the method under examination along with its related dependencies. They also applied rule-based techniques to correct syntactical and straightforward compilation errors in the generated tests.

Regarding software maintenance and evolution, Kang et al. [74] examined the aptitude 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. Wu et al. [75] conducted an extensive exploration into the effectiveness, consistency, and clarity of both GPT-3.5 and GPT-4 in accurately locating software faults. Additionally, Feng and Chen [76] introduced AdbGPT, a methodology that employs prompt engineering with ChatGPT to recreate Android defects from reported bugs. Regarding documentation generation, Ahmed et al. [77] engaged in prompt engineering with GPT-3.5 to refine the process of generating code summaries. Meanwhile, Gent et al. [78] showcased that pre-trained Large Language Models are inherently equipped with enough contextual understanding to produce a variety of code summaries, each from distinct technical viewpoints.

In terms of other uses of LLMs in SE tasks, Chen Kua et al. [79] explored the use of LLMs like GPT3.5 and GPT4 for automating domain modeling in software engineering, assessing their ability to understand and generate domain concepts against expert solutions. A systematic literature review by Hou et al. [80] delves into how LLMs optimize processes and outcomes in SE, categorizing LLMs used in SE tasks and identifying strategies for optimizing LLM performance. This comprehensive review underscores the versatility of LLMs in enhancing software engineering practices. Mahdi Mohajer et al. [81] 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.

Weyssow et al. [82] 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. Chaaben et al. [83] 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. Lastly, Sivakumar et al. [84] adapts the work of Chen et al. [70] to investigate the generation of safety cases using GPT-4, focusing specifically on its understanding of GSN.

Chapter 4

Methodology

4.1 Methods used to answer RQ1

To carry out our systematic mapping study (RQ1), we followed: 1) the guidelines for reporting systematic reviews that PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analysis) 2020 [85]; 2) the guidelines that SEGRESS (Preferred Reporting Items for Systematic Reviews and Meta-Analysis) [86]; and 3) the guidelines for reporting systematic mapping studies that Petersen et al. [87] propose. We further describe our methodology below.

4.1.1 Information Sources

Database Sources

To conduct the survey, we utilized five common academic databases, namely Scopus [88], Google Scholar [89], IEEE Xplore [90], ACM [91] and Engineering Village [92] to gather relevant papers. To automatically search studies in Google Scholar, we used the *Publish or Perish* tool [93]. To manage the studies we collected from different databases, we utilized a reference management tool called EndNote [94].

Snowballing Sources

We complement the database-driven search by relying on backward and forward snowballing. Wohlin [95] describes in detail how to perform snowballing. We rely on a well-established

tool called *Connected Papers* [96] to automate the snowballing process. Our snowballing process uses as a start set the primary studies found through the database-driven search.

4.1.2 Identification Strategies

Database-driven Search

To increase the likelihood of capturing all relevant studies in the field, the search string employed in the search engines must incorporate commonly used keywords found in papers within this particular domain. Consequently, prior to constructing the search string, we thoroughly acquainted ourselves with the precise terminology commonly utilized by researchers in the field of assurance cases. We also refined the search string multiple times until we found a search string that captured a meaningful number of studies focusing on the surveyed topic.

Next, we created the search string for the mentioned databases to identify potentially relevant studies to this study. For that purpose, we used two groups of keywords. The first group relates to assurance weakeners, while the second one relates to assurance cases. Consequently, we formed the search string as the following:

("assurance deficits" OR "false assurance" OR "defeaters" OR "counter evidence" OR "counter-argument" OR "fallacies" OR "aleatory uncertainty" OR "aleatoric uncertainty" OR "epistemic uncertainty" OR "uncertainty reasoning" OR "uncertainty elicitation" OR "informal semantics" OR "doubt")
AND
("assurance case" OR "safety case" OR "trust case" OR "dependability case" OR "reliability case" OR "security case" OR "availability case")

Due to the variations in syntax among different databases for queries, we adjusted the query for each specific database. Table A.1 in Appendix A details the queries and parameters employed for the search across the five databases.

Snowballing

We leveraged the Connected Papers tool to execute the snowballing process automatically. We iteratively relied on Connected Papers to perform the snowballing process, using the primary studies found through the database-driven search as a start set. We stopped our

snowballing iterations when no additional studies were found.

4.1.3 Selection Strategy

Eligibility Criteria

To select our primary studies, we relied on a set of inclusion and exclusion criteria. Table 4.1 reports these eligibility criteria.

Table 4.1: Eligibility Criteria

Eligibility Criteria	**Description**
Inclusion	The paper should be in English.
	Published in a Journal, Conference, Workshop, or Magazine.
	Within ten years (January 2012 - October 2023).
	Papers addressing assurance deficits and logical fallacies when creating, designing, representing, reviewing, or arguing in the assurance case.
Exclusion	Books, Tutorials, and Book Chapters.
	Papers not available or accessible.
	Serial publications.
	Papers that do not discuss techniques to deal with assurance deficits or uncertainty.
	Secondary Studies (e.g., Surveys, Reviews, Systematic Reviews, Systematic Literature Review, Systematic Mapping Studies), and Tertiary Studies.
	Papers focusing on uncertainty assessment without reasoning about uncertainty at the modeling level.
	Papers that have not been peer-reviewed.

Database-driven Search

Table 4.2 outlines the selection strategy we used to narrow down the studies for inclusion in the search through the database-driven search. That selection strategy consists of 6 rounds to filter the studies that did not meet the specified eligibility criteria.

Table 4.2: Selection Strategy

Selection Round	**Round for the Selection of Primary Studies**
1	Importing in EndNote the references of studies found in the searched databases
2	Cleaning references (e.g., with no title, of study type not covered, not peer-reviewed)
3	Removing duplicates (i.e., identical references coming from different databases)
4	Excluding studies based on the titles, keywords, abstracts, venues, and inclusion and exclusion criteria
5	Excluding studies based on the introductions and conclusion, and inclusion and exclusion criteria
6	Excluding studies based on full-text reading and inclusion and exclusion criteria

Snowballing

To ensure the completeness of our search process, we use the Connected Papers tool to complete snowballing. The latter follows a similar process as the database-driven search, with slight modifications in the starting set and duplicate removal steps. Thus, in the first snowballing Iteration, we generate a graph for each primary study found through the database-driven search. Each graph generated by Connected Papers usually yields 41 studies. We then apply inclusion and exclusion criteria to each of these studies found using Connected Papers. This can result on a set of additional primary studies. We then iteratively use Connected Papers to apply snowballing on that set until no additional primary study is found.

4.1.4 Data Extraction Strategies

Data Extraction Strategy for the Bibliometric Synthesis

We reviewed the primary studies and compiled a CSV file using Notion [97] to document data such as publication year, venue, and number of citations, which were essential for our bibliometric synthesis. We sourced the citation count by manually exploring Google Scholar. We exported the primary studies from EndNote in RIS format to obtain data on influential research topics.

Data Extraction Strategy for the Qualitative Synthesis

We iterated over the primary studies to identify categories that could help us classify the qualitative information retrieved from these studies: categories of Assurance Weakeners and categories of approaches proposed to manage (e.g., represent, identify/detect and mitigate) assurance weakeners). We used these categories to create data extraction forms in Notion. We then used these forms to extract data from the primary studies.

Two researchers extracted data from the primary studies. The first researcher (a graduate student) extracted all the studies' data. The second researcher (a faculty member) randomly extracted data from 20 primary studies (half of the primary studies). We held meetings to resolve potential disagreements between researchers during the data extraction process.

4.1.5 Synthesis Strategies

Strategy for the Bibliometric Synthesis

The bibliometric synthesis is becoming increasingly popular in several research areas [98, 99, 100, 101, 102]. We, therefore, decided to use it to synthesize the various trends and patterns characterizing primary studies. To extract and synthesize the bibliometric data found in primary studies, we mainly use a popular bibliometric tool called VosViewer [103]. That tool allows us to automatically generate charts depicting the bibliometric information characterizing the primary studies. To generate additional charts, we employed Python, specifically using the Pandas library [104] for data manipulation and Matplotlib [105] for data visualization. Python is an industry-standard data analysis tool with robust libraries like Pandas.

Strategy for the Qualitative Synthesis

To perform our qualitative synthesis, we have used tables to capture and report the data extracted from primary studies. This is useful to present data in a tabular form in accordance with the research questions associated with the qualitative synthesis. Presenting data in a tabular form is a practice that is strongly recommended when completing systematic reviews [106].

4.2 High-Level Overview of our approach

Our work adapts the one of Chen et al. [70] and Sivakumar et al. [84] 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 [52] to better support requirements verification and validation. Figure 4.1 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.

In **Phase I**, like Sivakumar et al. [84], we conducted a thorough analysis of the documentation on EA (e.g., [12, 52]) to extract the structural and semantic rules its notation embodies [107]. We then derived structural and semantic-based questions from these rules. We combined these questions with EA generation-based questions. We used 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. We found that 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. Thus, it would be beneficial to employ specific prompting techniques to enhance its comprehension of EA semantics, which could lead to better defeaters.

Phase II centers around applying GPT-4 Turbo to identify potential defeaters within EA assurance cases. We guide GPT-4 Turbo through Chain-Of-Thought prompting techniques to 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 [57, 59]. We also use predicate-based structural rules since LLMs can effectively encode rule-based knowledge [108]. 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.

4.3 Methods used to answer RQ2

To answer this research question (RQ2), we focus on Phase I of the approach shown in Figure 4.1.

4.3.1 Extraction Of The Structural And Semantic Rules Of EA

The first step of our approach consists in analyzing existing documentation (e.g., [12, 52]) on EA to extract the set of structural and semantic rules that EA embodies. Table 4.3 reports the structural and semantic rules that we extracted. Extracting rules is a vital component of our research, forming the core foundation for crafting essential questions relevant to RQ. Semantic rules are focused on the text and meaning within EA elements. 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 connections with other elements.

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

We crafted an initial batch of 22 questions and categorized our evaluation into three distinct sections, i.e., structural, semantic, and generation-based questions. The structural questions aim to assess GPT-4 Turbo's comprehension of EA notation's core principles. In contrast, the semantic questions examine its understanding of the vital semantic rules in 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. Table 4.4 presents sample questions for each category.

4.3.3 GPT-4 Turbo Setting

In this research, interaction with GPT-4 Turbo was facilitated through the use of the OpenAI API[1]. This approach was chosen to leverage the advanced capabilities of the GPT-4 Turbo model in a controlled and reproducible manner. A key aspect of our setup involved making the model's responses more deterministic. To achieve this, we set the seed parameter to a

[1] `https://openai.com/api/`

Table 4.3: EA Structural and Semantic Rules

Category	**Name**	**Structural Rules**	**Semantic Rules**
Element	Claim	Connected to: Context, Rebutting Defeater	A claim is stated as a predicate, a true or false statement.
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.
Element	Context	Connected to: Claim	It gives additional information about the content of a fundamental element and is optional.
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.
Element	Undercutting Defeater	Connected to: Inference Rule	Is a doubt about the validity of an inference rule ($P \rightarrow Q$), preceded by "Unless"
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.
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 4.4: Sample Questions for assessing GPT-4 Turbo's Proficiency in EA

Category of Question	Sample Question
Structural Question	What are the different types of defeaters in Eliminative Argumentation?
Semantic questions	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 Question	Generate me a sample Claim and a Rebutting defeater that defeats it. Show it in structured prose.

specific number. The seed is a crucial component in the model's operation, as it ensures consistency in the responses generated by GPT-4 Turbo for the same prompt.

4.3.4 Prompting Process

In our study, we meticulously followed the best practices of prompt engineering as outlined in OpenAI's guide[2] 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 Assurance Cases, 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.

4.3.5 Assessment Process

GPT-4 Turbo generates each EA concept in the structured prose complying with EA. To objectively evaluate each of the GPT-4 Turbo's responses to the 22 questions, we had two

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

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. [70]. The value of the Kendall rank correlation coefficient progresses from zero to one. A value close to zero means the level of agreement between raters is almost close to none. A value close to one means the level of agreement is strong. We rely on an online tool called **Gigacalculator**[3] to automatically assess the values of the Kendall rank coefficient with a confidence level of 95%.

4.4 Methods used to answer RQ3

In this research question (RQ3), we focus on Phase II of the approach shown in 4.1.

4.4.1 Specification of predicate-based structural rules

We planned to use predicate-based rules for illustrating EA elements and their connections. The integration of predicate-based rules into the prompting process of AI language models has been a focus of recent research. As Yang et al. demonstrated in [108], LLMs can effectively encode rule-based knowledge into their parameters through a method known as rule distillation. This approach not only enhances the efficiency of the models compared to example-based learning but also improves their generalization capabilities. Similarly, [109] introduces 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.

These studies underscore the potential and 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*" [110], leading to incorrect answers. We adopt this technique, forming predicate-based rules from the structural rules EA embodies, to prompt GPT-4 Turbo in a way that enables it to generate more effective defeaters. This

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

Table 4.5: 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)#)

allowed us to extract the following general predicate-based rule for EA elements:

ELEMENT#({{TEXT}}, CONNECTEDELEMENT(S)#)

An `ELEMENT` can be any of the 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 the `ELEMENT`. Furthermore, the symbol # indicates the unique number for each element.

Utilizing the general rule and the semantic and structural rules outlined in Table 4.3, we derived specific predicate-based rules for generating defeaters. These rules, as detailed in Table 4.5, 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).

4.4.2 Experiments Setup

To investigate this RQ, we conducted 4 distinct experiments designed to evaluate the effectiveness of GPT-4 Turbo in accurately generating defeaters.

- **Experiment 1: Zero-shot with CoT and Predicate-based Rules, 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. The experiment leverages CoT and predicate-based rules but does not provide any contextual information about the system under analysis.
- **Experiment 2: One-shot with CoT and Predicate-based Rules, Without Context** - Similar to Experiment 1, this test uses CoT and predicate-based rules.

However, it 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 CoT and Predicate-based Rules, With Context** - This experiment also applies a zero-shot methodology. Unlike Experiment 1, it provides GPT-4 Turbo with context about the system, alongside utilizing CoT and predicate-based rules, to assess the model's capability in generating defeaters with background knowledge.

- **Experiment 4: One-shot with CoT and Predicate-based Rules, With Context** - Building upon the previous experiments, this test combines one-shot learning with context about the system, along with the use of CoT and predicate-based rules. The focus is on observing the effectiveness of providing both an example and system context in defeater generation.

4.4.3 Dataset

Our dataset consists of nine partial ACs (AC fragments) that we collected from the ACs respectively created for two systems. These two ACs comply with the EA notation. We selected these ACs from a variety of studies featured in our survey on assurance weakeners [66]. Our selection strategy was intentional, aiming to encompass ACs from diverse sectors, including nuclear and aviation, to validate the versatility and domain independence of our approach. Table 4.6 reports the characteristics of each of the nine AC fragments included in our dataset. We refer to each of them as case 1, case, case 2, case 3, case 4, case 5, case 6, case 7, case 8, and case 9, respectively.

In the remainder of this section, we describe the two ACs from which we collected the nine AC fragments, as well as the two systems for which these two ACs were created.

Table 4.6: Summary of the dataset

Case	**ID**	**Original defeater name**	**Category of the defeater**	**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.

Table 4.6 continued from previous page

Case	ID	Original defeater name	Category of the defeater	Content of the defeater
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.

Table 4.6 continued from previous page

Case	ID	Original defeater name	Category of the defeater	Content of the defeater
	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.

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 [111]. 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 AC for the MPS, developed using the EA methodology, comprises 509 nodes and is accessible publicly in PDF and JSON formats [112]. This AC focusing on the **nuclear** domain, identifies 105 potential defeaters. Since that AC is very large and spans more than one hundred pages, we randomly collected 8 AC fragments (i.e., case 1 to case 8) from it.

Case 1: We selected a fragment of the AC for the MPS. That fragment breaks down claim C0030 illustrated in Figure 4.2. This claim asserts that "The BIS will effectively communicate a beam permit loss to the BDS within a timeframe of under 100 microseconds". The decomposition occurs through Strategy S0654, which examines potential failure scenarios that might obstruct, postpone, or in any way 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.

Case 2: We also selected another fragment of the AC for the MPS. This second fragment focuses on C0196. The latter is a claim which states that the Surface Electronics will provide or withdraw a user permit and enable a beam dump signal to the BIC illustrated in Figure 4.3. This claim is defeated with the following defeaters: D0121 and D0252. These defeaters present doubt about the claim, so one can consider them rebutting.

Case 3: When conducting our experiments, we also chose to focus on a third fragment of the AC for the MPS. Figure 4.4 illustrates that fragment. That third fragment focuses on C0378 which states that the magnets used to dilute the beam (called the MKB magnets) activate within 18 μs of receiving a beam dump signal and do not cause losses as they are separated from the main LHC ring. This claim is followed by the E0379 evidence and a defeater D0380. This defeater presents doubt about the evidence, so one can consider them undermining.

Cases 4, 5, 6, 7, and 8: These partial ACs are also for MPS and their corresponding defeaters can be found in Table 4.6. Figure 4.5, Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9 illustrate these AC fragments.

Air Traffic Control System

This AC falls in the domain of **aviation**. Referenced from [17], this AC complies with EA notation and originates from Raytheon Canada's experience in developing a safety assurance case for Nav Canada's air traffic management system. This safety assurance 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 aircraft 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. We collected the ninth AC fragment (i.e., Case 9) from this AC. Noteworthy, Additional AC fragments exist but Diemert et al. [17] did not document most of them for brevity's sake.

Case 9: The AC in Figure 4.10 which is a fragment of the AC in [17], argues that discrete SSR codes assigned to aircraft in the controlled airspace are unique over a tolerable interval of X seconds (C0001). Moreover, two defeaters are presented: aircraft arriving from another airspace might be using a code that is already in use in the current airspace (D0003), or

communication errors between controllers and pilots might contribute to duplication events (D0004). As mentioned in [17], as these defeaters present doubt about the claim, one can consider them rebutting.

4.4.4 Data pre-processing

After collecting the nine AC fragments (i.e., cases 1-9) we did not immediately feed them to GPT-4 turbo. We first pre-processed them as follows:

- In EA, inference rules are a more structured version of GSN strategies [12]. For the sake of simplicity and to foster data reduction, we, therefore, removed strategies from the analyzed fragments because these fragments already comprised inference rules. This also allows obtaining AC fragments that align with the structural and semantic rules we extracted from EA.
- In the nine AC fragments we collected, defeaters' names all start with "D". Hence, they seem generic. We therefore 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".
- EA fragments are represented in EA which is a visual notation. We therefore converted the resulting fragments into the textual form (compliant with the predicate rule) suitable for GPT-4 turbo processing.

.

4.4.5 GPT-4 Turbo Setting

In our study, we utilized the OpenAI API[4] to interact with the GPT-4 Turbo model. A key methodological decision was ensuring more deterministic responses from the model. We achieved this by setting the `seed` parameter in the API. Setting the `seed` parameter is critical for generating more reproducible outputs from GPT-4 Turbo for the same input [28]. 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

[4] `https://openai.com/api/`

you're aiming for deterministic results [113]. 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 [114].

4.4.6 Prompting Process

In our study, we followed the best practices of prompt engineering as outlined in OpenAI's guide[5] 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.

We also leveraged from CoT (Chain of Thought) prompting technique. Chain of thought prompting is a technique used in natural language processing to enhance the problem-solving capabilities of language models [57]. It involves prompting the model to generate intermediate steps or reasoning paths when tackling complex questions or tasks [115, 57]. The technique has been shown to improve the performance of language models on tasks that require reasoning, such as arithmetic calculations, common sense reasoning, and text-based problem-solving [116].

In the following, we discuss the user and system prompt templates that we used for each experiment 4 since it is the more complex one. We annotated each part 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

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

about the case in the input after CONTEXT.②For 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), undermining (it can be only for Evidence), or undercutting (it can be only for an inference rule). 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 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 are describing the general rule for EA elements as described in 4.3.
4. In this section, we describe different types of defeaters and their structural and semantic rules defined in Table 4.5.
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 is the partial ACs described in Section 4.4.3 in the format of the general 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.

> @@@
> 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:

4.4.7 Assessment Measures

To assess the results of our experiments, we rely on various measures that we further describe below. We adapted these measures from the work of Chen et al. [70] and Sivakumar et al. [84].

Ground-truth Similarity

It evaluates the degree of lexical and semantic similarity between the defeaters generated by GPT-4 Turbo and the defeaters in the reference assurance case (ground-truth). We assess the ground-truth similarity based on three measures:

- **Exact match or string matching measure (using fuzzy logic)**. The score quantifies the similarity between two texts, considering partial matches and minor differences [117]. 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 [118] to assess it. The script we used to calculate this measure is the following:

```
from fuzzywuzzy import fuzz


for index, experiment_row in df.iterrows():
    for case in ['Case 1', 'Case 2', 'Case 3', 'Case 4', '
       ↪ Case 5', 'Case 6', 'Case 7', 'Case 8' , 'Case 9'
       ↪ ]:
        experiment_text = experiment_row[case]
        ground_truth_text = ground_truth_df.loc[
           ↪ ground_truth_df['Case'] == case, 'AC'].iloc
           ↪ [0]
        similarity_score = fuzz.ratio(experiment_text,
           ↪ ground_truth_text)/ 100.0

        print(f"Similarity score for {case} in {
           ↪ experiment_row['Experiment']}: {
           ↪ similarity_score}")
    print("-------")
```

- **Bleu score**: The Bleu (Bilingual Evaluation Understudy) score quantifies the similarity between the machine-generated text and the reference text based on the precision of n-grams in the generated text compared to the reference text [119]. 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 a Python library called sacrebleu which is specifically designed for computing BLEU scores in a standardized way, ensuring consistency and comparability of the scores across different evaluations. The script we used to calculate this measure is the following:

```
import sacrebleu

def calculate_bleu_score(experiment_text, ground_truth_text
   ↪ ):
```

```
    references = [[ground_truth_text]]
    system_output = [experiment_text]
    bleu = sacrebleu.corpus_bleu(system_output, references)
    # Scale the score to a range between 0 and 1 and round
      ↪ to two decimal points
    return round(bleu.score / 100, 2)

for case in ['Case 1', 'Case 2', 'Case 3', 'Case 4', 'Case
  ↪ 5', 'Case 6', 'Case 7', 'Case 8' , 'Case 9']:
    print(f"Calculating BLEU scores for {case}:")
    for index, row in df.iterrows():
        experiment_text = row[case]
        ground_truth_text = ground_truth_df.loc[
          ↪ ground_truth_df['Case'] == case, 'AC'].iloc
          ↪ [0]

        if pd.notnull(experiment_text) and pd.notnull(
          ↪ ground_truth_text):
            bleu_score = calculate_bleu_score(
              ↪ experiment_text, ground_truth_text)
            print(f"Experiment {index + 1} BLEU score: {
              ↪ bleu_score}")
        else:
            print(f"Experiment {index + 1} BLEU score: Data
              ↪  Missing")
    print("-------")
```

- **Semantic similarity**: In this context, utilizing cosine similarity enables us to quantitatively evaluate the semantic alignment between the reference defeaters within the AC and those produced by GPT-4 Turbo. By transforming textual content into numerical vectors through techniques like TF-IDF, we can measure the cosine of the angle between these vectors, thus obtaining a cosine similarity score. The cosine similarity

value ranges from -1 to 1, where -1 denotes complete dissimilarity and 1 signifies exact similarity. However, in text analysis, especially with TF-IDF, negative cosine similarity scores are practically impossible because you cannot have negative term frequencies or inverse document frequencies. Thus, the vectors generated from text data using TF-IDF will always be in the first quadrant of the vector space, where all components are non-negative, leading to cosine similarity scores ranging from 0 to 1. We rely on a Python library called scikit-learn [120] for both vectorization and cosine similarity computation. The script we used to calculate this measure is the following:

```
from sklearn.feature_extraction.text
import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

vectorizer = TfidfVectorizer()

for case in ['Case 1', 'Case 2', 'Case 3', 'Case 4', 'Case
   ↪ 5', 'Case 6', 'Case 7', 'Case 8', 'Case 9']:
    experiment_texts = df[case].tolist()
    ground_truth_text = ground_truth_df.loc[ground_truth_df
       ↪ ['Case'] == case, 'AC'].iloc[0]

    combined_texts = experiment_texts + [ground_truth_text]

    tfidf_matrix = vectorizer.fit_transform(combined_texts)

    experiment_vectors = tfidf_matrix[:-1]
    ground_truth_vector = tfidf_matrix[-1:]

    cosine_similarities = cosine_similarity(
       ↪ experiment_vectors, ground_truth_vector)

    for i, similarity_score in enumerate(
       ↪ cosine_similarities.flatten(), 1):
```

```
        # Round the score to two decimal places without
            ↪ multiplying by 100
        rounded_score = round(similarity_score, 2)
        print(f"Cosine similarity for {case}, Experiment {i
            ↪ }: {rounded_score}")
    print("-------")
```

Reasonability

Reasonable means the GPT-4 turbo generated EA "*element could reasonably be in the ground-truth but is not*" [70]. This measure is used in [70] and [84]. 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***. To assess the consistency and reliability of these ratings, we rely on the Kendall rank correlation coefficient as in Chen et al. [70]. 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 titled **Gigacalculator**[6] to automatically assess the value of the Kendall rank coefficient with a confidence level of 95%.

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

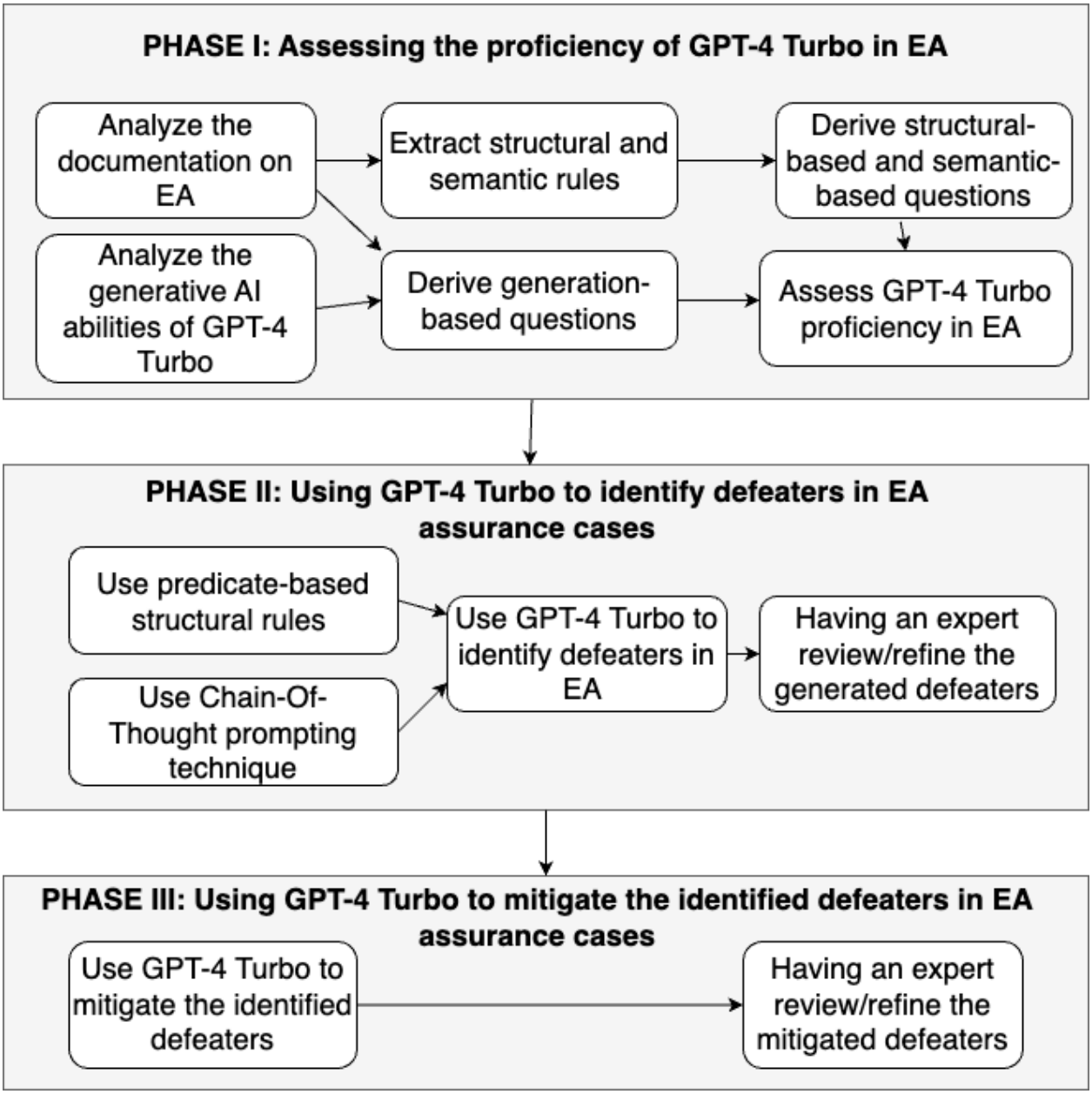

Figure 4.1: Overview of the Approach

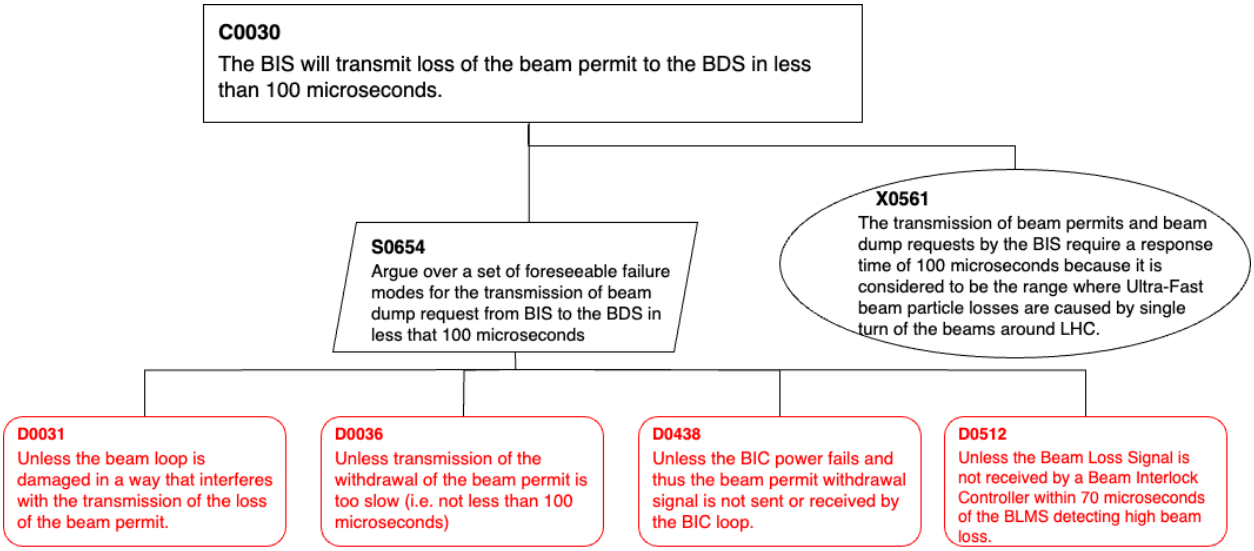

Figure 4.2: Case 1 (adapted from [111])

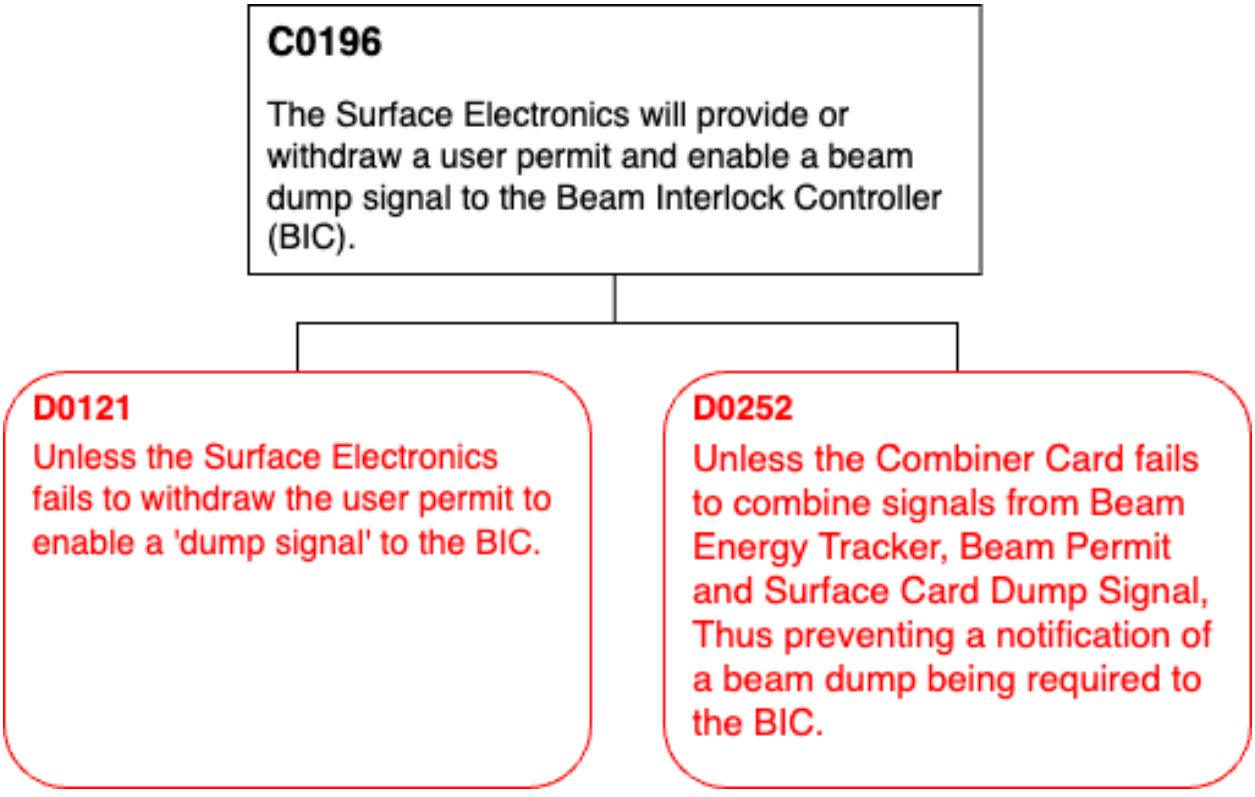

Figure 4.3: Case 2 (adapted from [111])

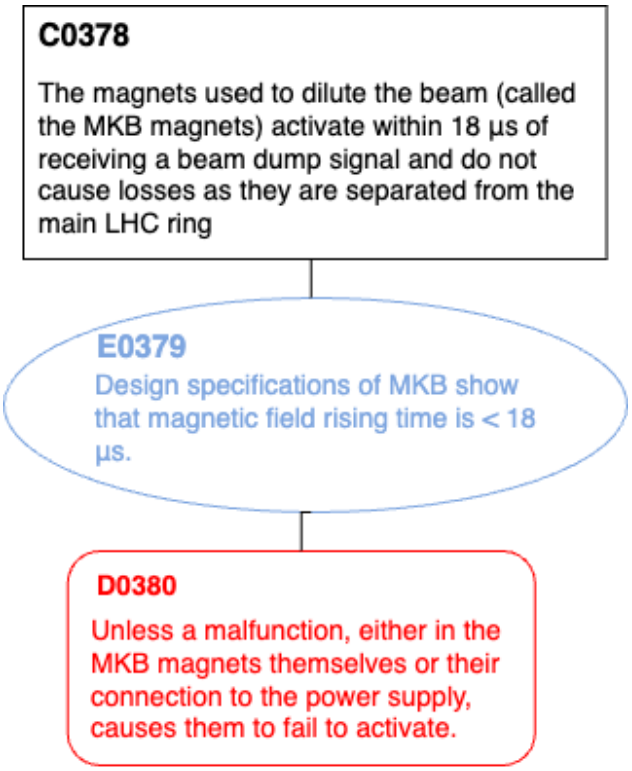

Figure 4.4: Case 3 (adapted from [111])

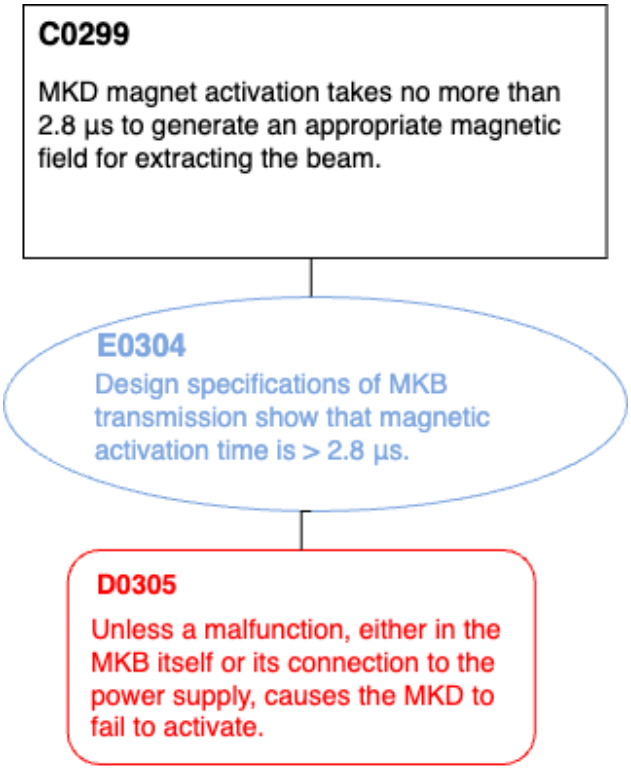

Figure 4.5: Case 4 (adapted from [111])

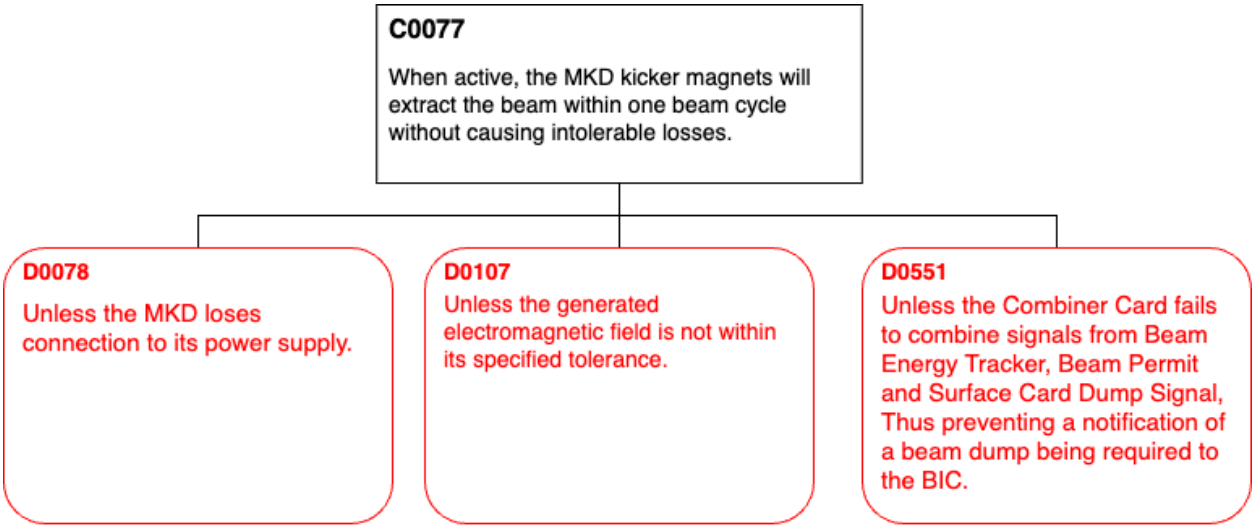

Figure 4.6: Case 5 (adapted from [111])

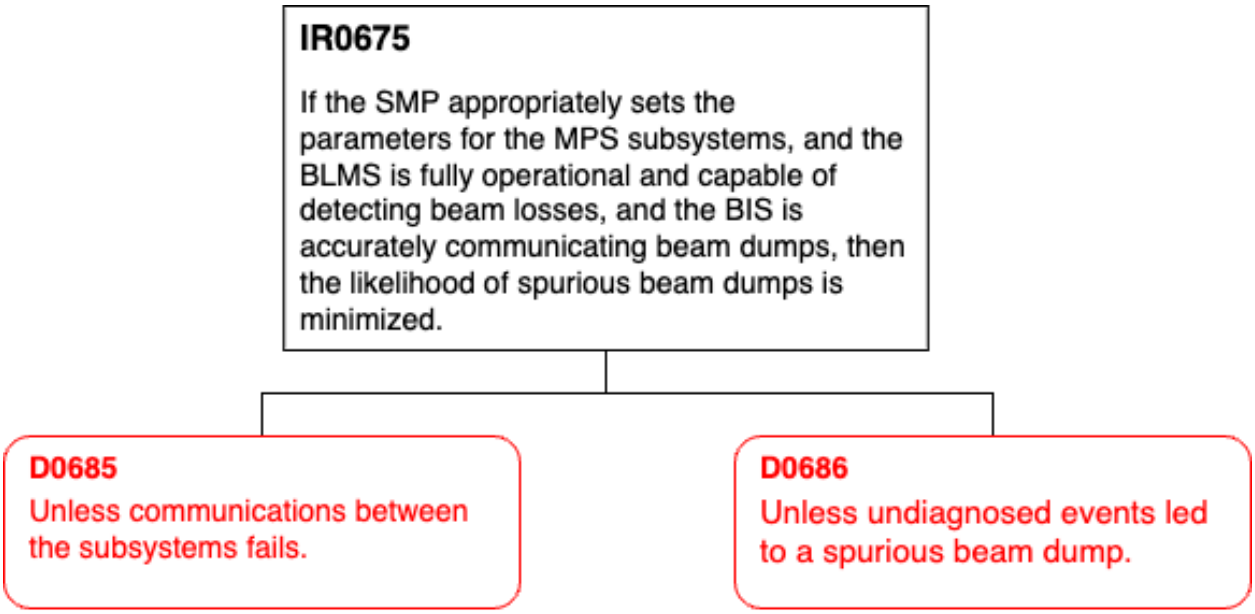

Figure 4.7: Case 6 (adapted from [111])

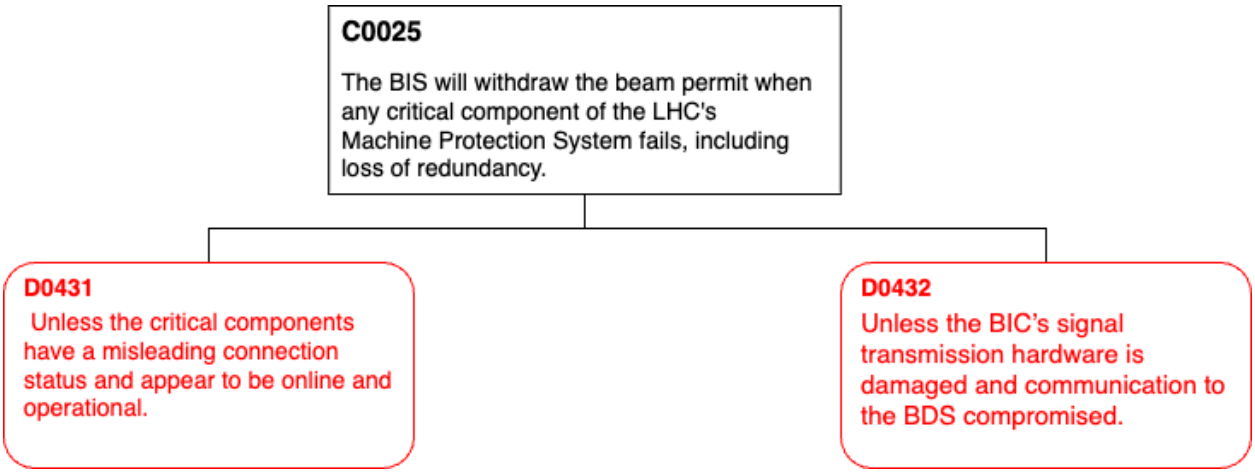

Figure 4.8: Case 7 (adapted from [111])

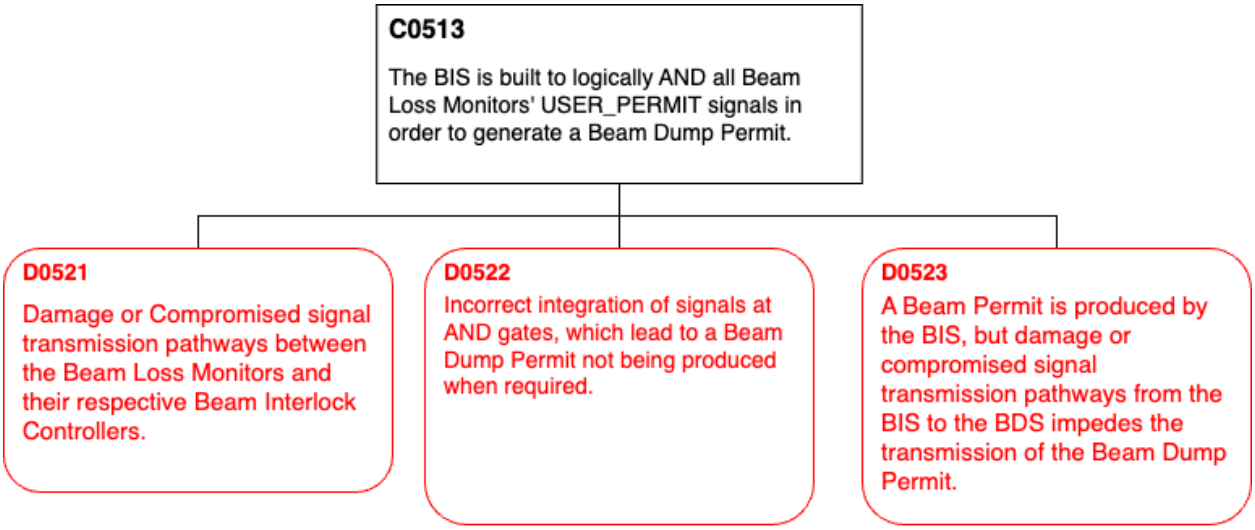

Figure 4.9: Case 8 (adapted from [111])

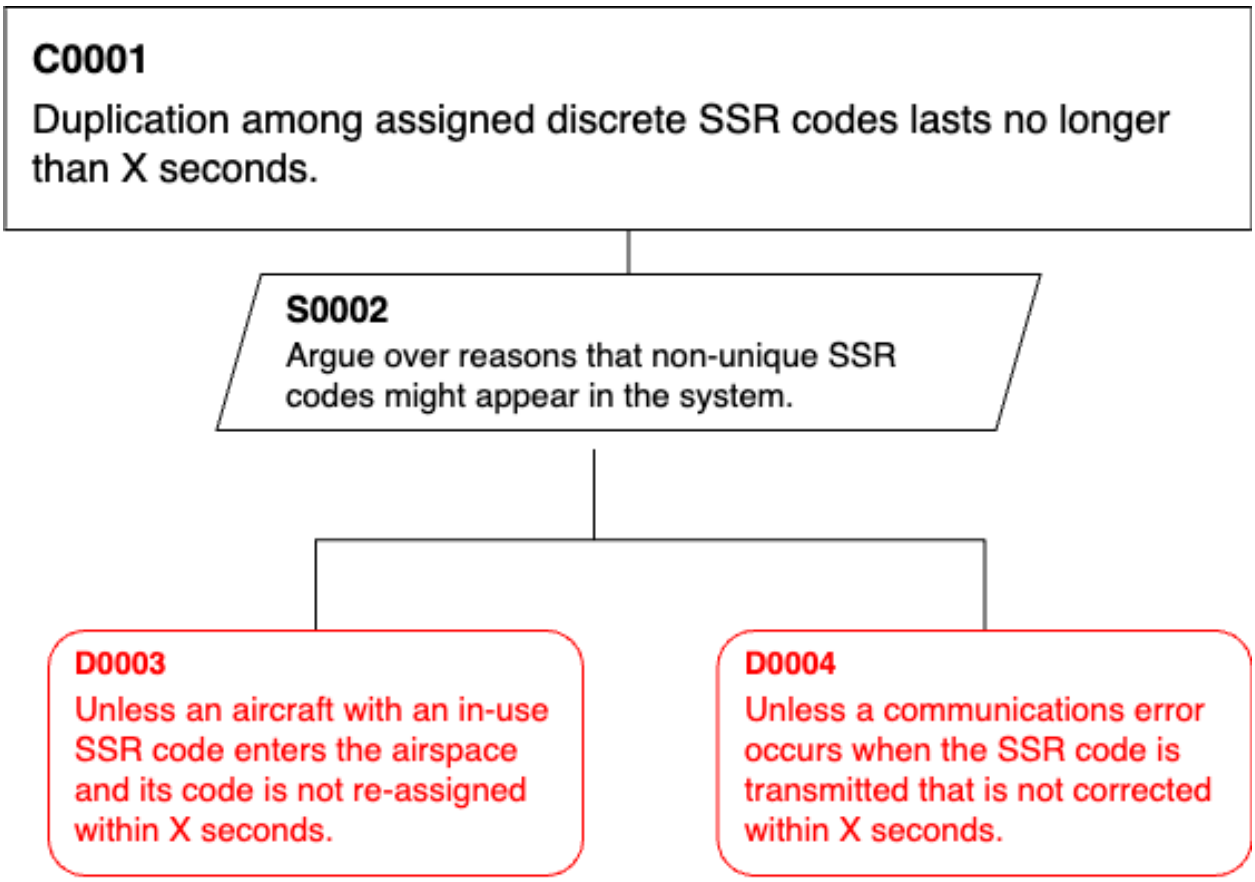

Figure 4.10: Case 9 (adapted from [17])

Chapter 5

Results

5.1 Results of RQ1

5.1.1 Study Selection

Figure 5.1 shows our PRISMA flow chart diagram. This diagram illustrates the selection results that both our database-driven search and snowballing strategies yield. This diagram maps out the number of records identified, screened and included from different information sources. Overall, 39 primary studies were selected from the database and snowballing search. Noteworthy, as of October 10, 2023, Connected Papers could not snowball some publications (i.e., [111], [17] and [2]) due to their recency.

5.1.2 Study Characteristics

Appendix B reports the characteristics of the primary studies. The columns of that table provide information about the authors of the primary studies, their publication year, their title, venue, and the type of search (e.g., database-driven search, snowballing) used to select them. The venues' acronyms are also shown in Appendix C.

5.1.3 Bibliometric Synthesis Results

For this purpose, we generate charts embedding bibliometric information and comment on them. We further describe our bibliometric synthesis in the remainder of this section.

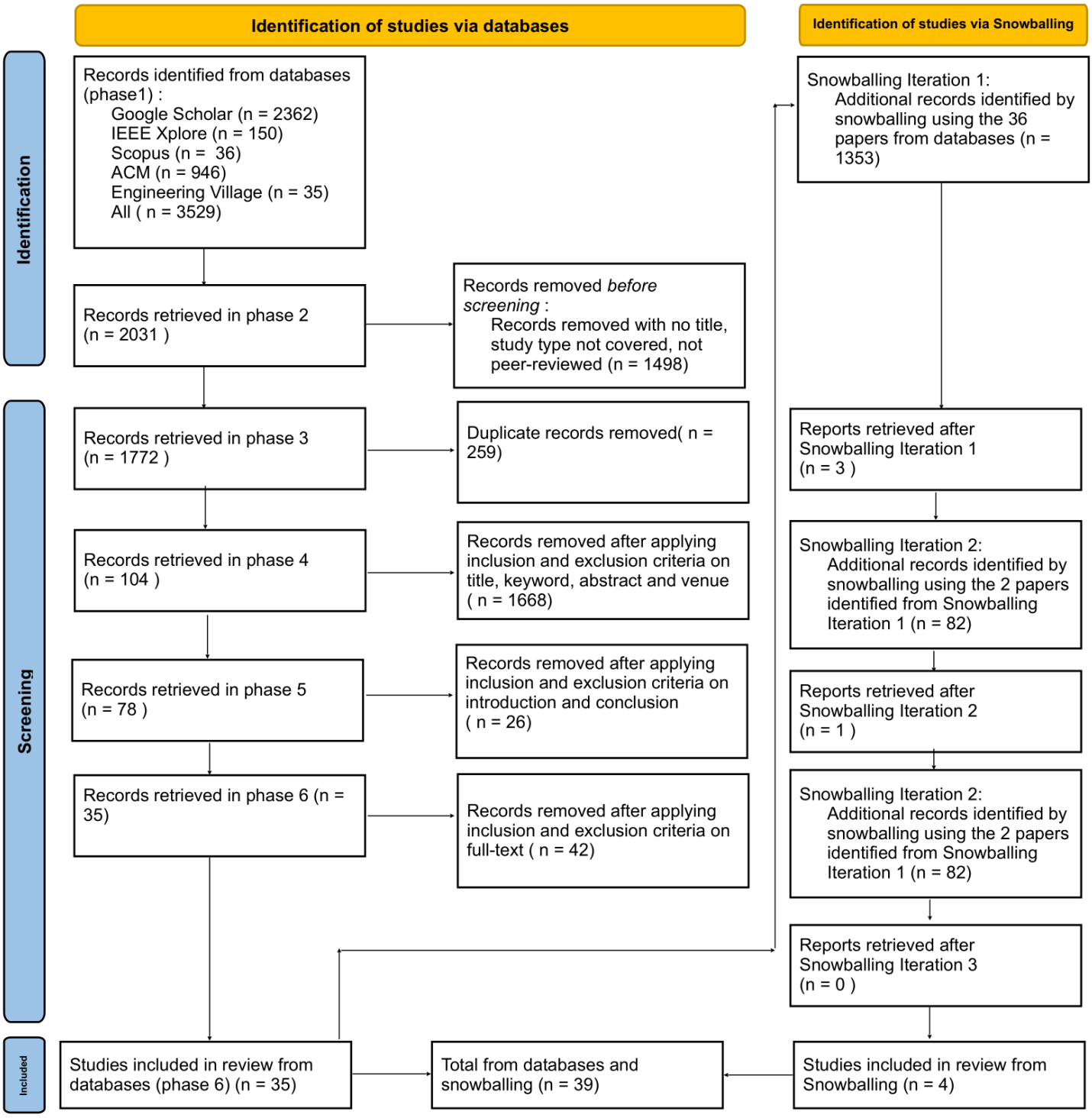

Figure 5.1: PRISMA flow diagram illustrating the identification and search processes

Publication Year Trend

Figure 5.2 illustrates the publication year trend for the primary studies. A notable peak of five publications in 2013 signalled increased interest in the field compared to the previous year. Subsequently, the number of studies displayed fluctuations until a notable dip to a single study in 2018. However, a subsequent recovery was observed leading up to 2020. The decline after 2020 could be attributed to various factors, including possible repercussions from the global COVID-19 pandemic. The pandemic's disruptive effects on research activities, ranging from practical limitations and resource reallocation to financial uncertainties, likely contributed to this decrease. Nonetheless, in 2023, the trend has seen an unprecedented surge. This is possibly due to the introduction of innovative methods, increased funding, or the growing topicality and cruciality of the surveyed theme, especially regarding safety-critical systems such as cyber-physical systems. This recent spike in studies publications underscores assurance weakeners' growing significance and relevance, reflecting its prominence in the research community and the industrial sector.

Venue Distribution

Figure 5.3 presents a detailed visualization of the distribution of studies across various venues. The most striking observation is the dominance of SAFECOMP, which has the most publications among the venues considered. SAFECOMP's preeminence in primary studies can be attributed to its comprehensive focus on developing, assessing, operating, and maintaining safety-related and safety-critical computer systems. This wide-ranging purview harmonizes seamlessly with the core research theme on assurance weakeners. Moreover, SAFECOMP's esteemed reputation in the fields of safety, reliability, and security further underpins its high publication count. Next in line, the International Symposium on Software Reliability Engineering (ISSRE) has a considerable number of studies, which can be attributed to its strong alignment with the domain of system assurance. ISSRE's emphasis on software reliability engineering closely resonates with the fundamental concerns of system assurance and the mitigation of assurance weakeners. Following ISSRE, the International Conference on Software Engineering (ICSE) also stands out with a significant concentration of studies. ICSE's prominence in the illustrated distribution can be attributed to its status as a leading conference in the software engineering domain.

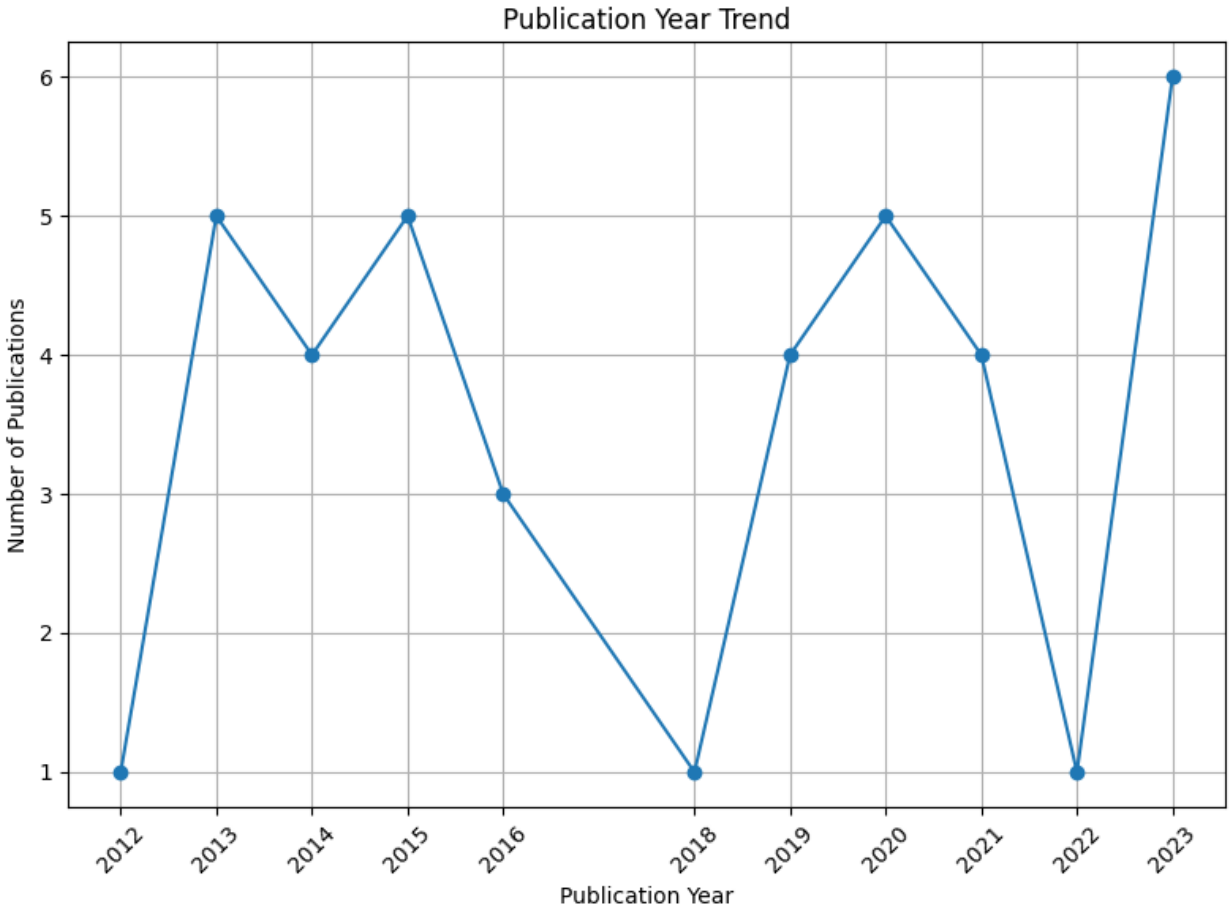

Figure 5.2: Publication Year Trends

Most influential research topics

The visual representation captured in Figure 5.4 sheds light on the pivotal topics shaping the assurance weakeners domain. We generated this chart with VosViewer to create a term co-occurrence map with a minimum of 2 occurrences of a term on the title and abstracts field. We also removed the noisy keywords.

The term **"assurance case"** emerges prominently as a central pivot in the discussion, emphasizing its critical importance. This concept serves as an anchor for other interrelated terms, reflecting the intricate network of these themes. Notably, nestled close to **"assurance case"** is the **"safety case"**, which underscores the significant emphasis the research community places on safety assurance. When compared to other types of assurance cases like security or reliability cases, the prominence of **"safety case"** further highlights the supreme significance

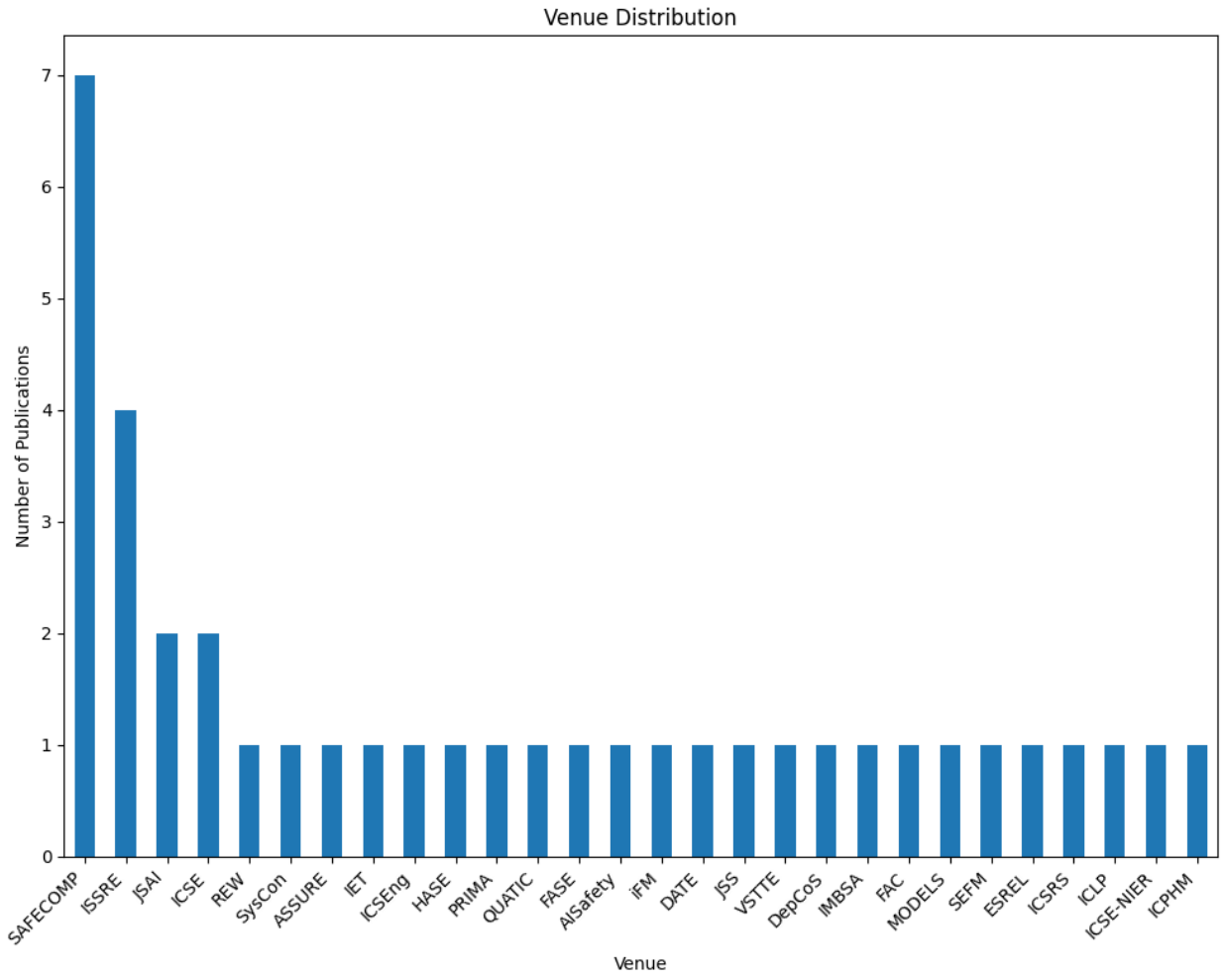

Figure 5.3: Venue Distribution

of safety within the realm of assurance weakeners. The prioritization of safety is rational, considering that system safety is crucial to prevent disastrous consequences, ranging from the loss of lives to substantial economic or environmental damages. Noteworthy, the term **"safety case"** frequently appears in the contexts of **"safety-critical domains"** such as **"railway"**. This association underscores the potentially catastrophic implications of system failures in these domains, reinforcing the mandatory requirement of a **"safety case"** for certification. Additionally, the presence of terms such as **"assurance deficit"**, **"defeater"**,**"uncertainty"** and **"fallacy"** emphasize assurance weakener-related terminology. The latter brings to light potential shortcomings or lapses in the assurance process. Their prominence underscores the importance of addressing these weakeners to achieve the highest safety standards. There's also

a clear linkage between **"eliminative argumentation"** and **"confidence"**. This connection suggests that eliminative argumentation is a prevalent mitigation strategy used to tackle assurance weakeners and increase system confidence. To add, keywords such as **"gsn"** and **"sacm"** are among the most recurring keywords in Figure 5.4. Both notations are among the most used to represent assurance weakeners.

Most Cited References

Table 5.1 provides an overview of the 15 most frequently cited references, offering a glimpse into the scholarly works that have significantly influenced the field. Among these notable references, the contributions of Denney et al. [19], Yamamoto et al. [121], and Goodenough et al. [37] stand out as pivotal and widely acknowledged.

The prominence of Denney et al.'s work [19] can be attributed to its pioneering introduction of dynamic safety cases, an assurance case for operationalizing through-life safety assurance. Published in 2015, this seminal contribution addresses critical safety concerns and advocates proactive safety management practices. Furthermore, its early publication date may have given it sufficient time to disseminate across the research community, which significantly contributed to its significant citation count. Yamamoto et al.'s work [121] occupies a prominent position due to its insightful exploration of assurance case development challenges and innovative solutions through assurance case pattern methods. The contribution of Goodenough et al. [37] holds significance for its pioneering application of eliminative induction and defeasible reasoning principles. Introducing the concept of a "confidence map" allows for providing a structured approach to addressing doubts and reinforcing confidence in system property assertions.

The extensive citations of renowned authors such as Ewen Denney, Ganesh Pai, Marsha Chechik, and Patrick John Graydon may stem from their established reputation as influential researchers in the assurance case domain. As well-established figures, their work naturally becomes reference points for other scholars, further amplifying their citation counts.

5.1.4 Qualitative Synthesis Results

To achieve this, we have developed Figure 5.5, which visually represents the taxonomy that we constructed from the primary studies. This taxonomy encompasses various **categories** of

Table 5.1: Top 15 Most Cited Papers as of October 18, 2023

Ranking	**Authors (Reference)**	**Title**	**Citations**
1	Denney et al. (2015a) [19]	Dynamic safety cases for through-life safety assurance	99
2	Yamamoto et al. [121]	An evaluation of argument patterns to reduce pitfalls of applying assurance case	61
3	Goodenough et al. [37]	Eliminative induction: A basis for arguing system confidence	48
4	McDermid et al. [122]	Towards a framework for safety assurance of autonomous systems	40
5	Rushby [25]	Logic and epistemology in safety cases	33
6	Denney and Pai. [123]	Evidence arguments for using formal methods in software certification	30
7	Muram et al. [43]	Preventing omission of key evidence fallacy in process-based argumentation	24
8	Rushby [26]	Mechanized support for assurance case argumentation	21
9	Denney et al. (2015b) [124]	Formal foundations for hierarchical safety cases	19
10	Chechik et al. (2019) [125]	Software assurance in an uncertain world	18
11	Graydon [38]	Towards a clearer understanding of context and its role in assurance argument confidence	16
12	Nemouchi et al. [126]	Isabelle/sacm: Computer- assisted assurance cases with integrated formal methods	15
13	Gansch et al. [41]	System theoretic view on uncertainties	15
14	Groza et al. [18]	A formal approach for identifying assurance deficits in unmanned aerial vehicle software	13
15	Cârlan et al. (2016b) [127]	On using results of code-level bounded model checking in assurance cases	11

assurance weakeners and the corresponding **approaches** proposed in the literature for their management.

Categories of Assurance Weakeners

The taxonomy that Figure 5.5 depicts allows us to classify assurance weakeners into several categories of uncertainty. Uncertainty can stem from several sources: the system's specifications, the environment in which the system operates, the system itself, the extent to which the arguments about the system's capabilities can be trusted, etc. [125].

One notable thing in our review was that some primary studies, such as the work of Yuan et al. [21] and Liu et al. [3], use different classifications of logical fallacies, which pertain to evidence, claims, and arguments. Essentially, they all point to shortcomings in claims, arguments, or evidence. These types of doubts are often termed undermining, undercutting, and rebutting defeaters. Given this established terminology, we chose to use these terms to describe logical fallacies, logic doubts, and argument uncertainties. Consequently, our categorization is structured accordingly.

According to Gansch et al. [41], Denney et al. [19], Chechik et al. [125], and Schleiss et al. [42], uncertainties can be classified into the following categories: aleatory, epistemic, ontological, and argument (logical).

- **Aleatory Uncertainty**: it pertains to the inherent unpredictability of a particular event or situation. Such uncertainties are often described as *"known unknowns"* and can be measured using probability distributions [41, 19, 125, 42, 2, 33]. Examples of aleatory uncertainties include an overdose from an infusion pump or mistakes in computing medication dosages [33]. Additionally, one often-overlooked type of aleatory uncertainty is the residual kind: even when a risk is thought to be addressed, there's a slight possibility it may not have been completely mitigated [33].

- **Epistemic Uncertainty/ Epistemic Doubt**: also referred to as epistemic doubt, epistemic uncertainty arises from a deficiency in knowledge or data, often referred to as the *"unknown unknowns."*[41, 19, 125, 42, 2, 33]. Examples encompass unnoticed flaws in logical thinking or unanticipated input sequences during the design and development phases [33].

- **Ontological Uncertainty**: it delves into a state of complete unfamiliarity with a vital aspect of the system's representation [41, 42]. Particularly in the initial phases of research and development for new systems, the accuracy and entirety of models often face scrutiny. For example, in autonomous systems functioning in open environments, one cannot entirely eliminate ontological uncertainty [41, 128].

- **Argument Uncertainty / Logical Fallacies / Logic Doubt:** the endeavor of constructing argumentations has been acknowledged as resource-intensive, time-consuming, and prone to errors [16, 24]. Inaccurate, incomplete, or inherently flawed reasoning regarding evidence can introduce defects known as logical fallacies into safety argumentations, leading to overconfidence in a system and tolerating certain faults, ultimately contributing to safety-related system failures [43]. According to Chechik et al. [125], using safety arguments to convince that software-intensive systems are safe enough raises questions about the extent to which these arguments can be trusted. These questions relate to the confidence one may have in the completeness of the verification and validation processes used to support the contention that the software at hand is indeed safe. These questions embody argument uncertainties and relate to the extent to which the evidence sufficiently backs arguments and the thoroughness of the arguments. Several primary studies (i.e., [129, 128, 126]) use the term *logical fallacies* to refer to argument uncertainties. A few primary studies use the expression *logic doubt* to refer to argument uncertainties (i.e., [25, 26]).

  A Logic doubt is also called *a defeater*. There are three main categories of defeaters: *undermining*, *undercutting*, and *rebuttal* defeaters [39, 37, 130, 131, 52, 132]. When "defeaters" represent a level of uncertainty deemed permissible as remaining doubt (without additional justification), they are labelled as "residual" and add to the total lingering doubt linked to the case [133]. Jarzębowicz and Wardziński [39] further divide these three categories of defeaters into five categories of defeaters, namely: D1, D2, D3, D4 and D5. We provide below the definitions of these three defeaters and explain their mapping with the five categories of defeaters that Jarzębowicz and Wardziński [39] proposed.

  - **Undermining Defeaters**: Invalidate one or more premises, thereby diminishing the basis for accepting the associated claim, even if the inference rule remains

sound [39, 37, 130, 131, 52, 133, 132]. Undermining defeater is also called **D1** (i.e., provision of arguments against the evidence)[39].

- **Rebuttal Defeaters**: provide a counterexample to a claim by introducing information that contradicts them and challenge their validity [39, 37, 130, 131, 52, 132]. Rebutting defeater is also called **D2** (direct refutation of the claim, typically through counterevidence)[39].
- **Undercutting Defeaters**: Focus on contesting the inference rule itself, presenting additional information about scenarios in which the claim might not hold true even with true premises [39, 37, 130, 131, 52, 133, 132]. The more detailed aspects of the Undercutting defeater are split into three variations [39]: 1) **D3** (attack on the inference rule's validity); 2) **D4** (disruption of the link between premises and conclusion, often due to misuse of the inference rule); and 3) **D5** (challenge to the applicability of the inference rule).

Key Takeaway: Although there are several categories of uncertainties, most of the research work centred around addressing assurance weakeners at the modeling level focuses on argument uncertainty. This may indicate that argument uncertainty may explicitly be tied to the flawed argument structure and may, therefore, be reasoned away by improving that argument structure.

Approaches to Represent Assurance Weakeners

Several approaches focus on the representation of assurance weakeners. They commonly depict assurance weakeners using assurance cases or confidence maps, as shown in Figure 5.5. Table 5.2 reports the primary studies that focused on representing assurance weakeners using these two structures. Table 5.2 also maps these primary studies to the different notations they used when representing assurance weakeners. These notations include GSN, SACM, CAE and TCL, as well as their extensions (when applicable).

GSN-based representation approaches: The most common notation for representing assurance weakeners is the Goal Structuring Notation (GSN) [125]. However, it's important to note that GSN alone is sometimes insufficient to effectively capture weakeners. So, extending that notation is often required to properly capture assurance weakeners. GSN primarily serves as a notation for explicitly presenting assurance cases without inherently judging the quality

Table 5.2: Approaches for Representing Assurance Weakeners

References	Representation structure	Notation
[38, 18, 124, 130]	Assurance Case	GSN Extension
[111, 16, 17, 132]	Assurance Case	Eliminative Argumentation
[39]	Assurance Case	TCL
[134, 135]	Assurance Case	SACM
[20]	Assurance Case	CAE
[37, 52]	Confidence Map	Eliminative Argumentation

of an argument [131]. Graydon [38] advocates for assured safety arguments, encompassing two sub-arguments: one that documents the argument and evidence for system safety and another that justifies the sufficiency of confidence in this safety argument, considering plausible defeaters. Groza et al. [18] formally represent argumentative-based GSN and employ description logic for identifying assurance deficits in GSN models. Denney et al. [124] introduce hierarchical safety cases in GSN to enhance the comprehension of safety argument structures and mitigate assurance deficits, albeit noting the need for a formal specification to verify tool operations to eliminate potential deficits. In addressing uncertainty, Yuan et al. [21] propose a predicate-based representation of GSN elements, contributing to considering uncertainty within safety arguments. Moreover, Takai and Kido [130] propose a supplemental notation for GSN, rooted in defeasible logic and argumentation theory, to address unexpected changes and rebut arguments explicitly.

Eliminative Argumentation-based approaches: the Eliminative Argumentation (EA) notation allows constructing arguments and evaluating confidence in these arguments by relying on the notion of *defeasible reasoning* [52]. The latter supports the recursive challenging of claims to progressively eliminate the doubts they may embed and, consequently, increase the confidence in these arguments [52]. EA notation builds on the foundational ideas of GSN. Just as in GSN, EA employs a directed, non-cyclic graph to lay out the structure of an argument. Both EA and GSN methodologies employ a hierarchical tree-like design where a principal system claim (positioned at the "root") gets broken down into more detailed sub-claims, eventually anchoring in concrete evidence. This offers a clear linkage between the evidence

presented and the resulting claims. A distinctive feature of EA, as compared to GSN, is its capacity to capture and represent "doubts" or defeaters regarding the authenticity of claims, the backing evidence, or the logical conclusions derived [17]. Therefore, in EA, defeaters can highlight uncertainties related to claims, the adequacy of evidence in supporting its preceding claim, and the robustness or comprehensiveness of the derived conclusions. Defeaters in EA notation are represented in rectangles with chopped-off corners, and each defeater has its own color: red for rebutting, yellow for undermining, and orange for undercutting and inference rules are also represented using green rectangles [12]. Menghi et al. [16], Diemert et al. [17], Millet et al. [111] and Cobos et al. [132] use EA notation to represent different types of defeaters.

An eliminative argument can be visualized using a **Confidence map** [12]. Accordingly, Goodenough et al. [37] rely on Confidence maps to notably represent defeaters. These maps are grounded in eliminative induction philosophy and defeasible reasoning. Confidence maps explicitly represent reasons for doubt relevant to an argument, aiding in its development and evaluation. Diemert et al. [52] also employ the concept of confidence maps, providing an abstract framework for constructing and assessing arguments based on defeasible reasoning. According to Goodenough et al. [37], a confidence map is a graphical argumentation structure that is mainly constituted from a set of claims, defeaters, inference rules, and evidence. Their work focuses on three defeaters: rebutting, undermining, and undercutting. They depict claims using clear rectangles, they depict inference rules using green-shaded rectangles, and they depict defeaters using red-shaded octagons. They label claims with a “C” and inference rules with an “IR”. They respectively label rebutting, undercutting, and undermining defeaters with an “R”, “UC,” and a “UM”. To depict the completion of the doubt refinement, they rely on the concept of argument terminators represented by a shaded circle.

SACM-based representation approaches: Selviandro et al. [134] as well as Foster et al. [135] discuss the Structured Assurance Case Metamodel (SACM), a rich specification for structured assurance cases, offering features that surpass existing notations. For instance, SACM includes the concept of "Defeated Assertion," indicating when an assertion is defeated by counter-evidence or argumentation and visualizes it as a Claim with a cross. Moreover, a line representing a *Defeated AssertedContext* is depicted with a solid square close to one end and a cross positioned at its center. Noteworthy, V2.3 of the SACM specification [136], provides more details on the various three categories of defeaters discussed in this review.

Overall, with such notations and concepts, SACM therefore supports the representation of all the defeaters and their relationships. Still, very few approaches have adopted SACM so far. This implies that more effort should be made to promote SACM in the research community and in the industry.

TCL-based representation approaches: another approach to represent is presented in Jarzębowicz and Wardziński [39]'s work, which is for integrating confidence and assurance arguments using the Trust Case Language (TCL) and providing a checklist of defeaters to build confidence arguments.

CAE-based representation approaches: Murugesan et al. [20] propose a Claims-Arguments-Evidence approach, emphasizing the evidence presented, the logical reasoning utilized, and the exploration and assessment of counter-claims. Doubts, concerns, or counter-claims on any part of the case are captured as defeaters and represented as red ellipses.

Key Takeaway: Over the years, various notations have emerged to represent assurance weakeners. However, many of these notations, while valuable in their own right, have had limitations in terms of expressiveness, adaptability, and their ability to holistically represent complex interrelationships between arguments and weakeners. From the result, it can be seen that GSN extensions and EA are the most used notation for representing assurance weakeners. However, SACM is the "relatively new kid on the block", as it is a standard that unifies several notations such as GSN and CAE [126]. SACM offers a more comprehensive range of capabilities than current assurance case notations. It lays the groundwork for model-driven system assurance, showing immense potential in that emerging specification [45]. Its superior interoperability means it can seamlessly integrate with other systems and notations, making it a preferred choice for diverse applications. Moreover, its enhanced expressiveness allows for more intricate detailing of arguments and their potential pitfalls. This adaptability ensures a comprehensive and intuitive representation of arguments, their interrelations, and any associated weakeners. Esteemed institutions like the University of York in the UK and Carnegie Mellon University have already vouched for SACM [136]. Another critical takeaway is the absence of an approach representing aleatory, epistemic and ontological uncertainties, as current approaches focus on representing argument uncertainty. This is primarily because these types of assurance weakeners stem from inherent unpredictability, incomplete knowledge or unfamiliarity of the system representation(unknowns in the system). Consequently, it may be challenging to explicitly represent such types of assurance weakeners.

Approaches to Detect Assurance Weakeners

In this section, we explore approaches used to detect (identify) assurance weakeners, as seen in Figure 5.5. Table 5.3 reports the primary studies focusing on detecting assurance weakeners.

Groza et al. [18] propose using reasoning in description logic (DL) to identify assurance deficits in the GSN model. Their solution has two steps: First, they check with hybrid logic if the evidence nodes from the GSN representation have their corresponding formulas validated against the Kripke model. Second, by reasoning in DL, they identify which goals in the GSN model are not supported by verified evidence. Moreover, Denney et al. [19] suggest that assurance deficits (i.e., aleatory uncertainty and epistemic uncertainty) can weaken confidence in the safety and as the system and its safety argument change, so will the

assurance deficits. Consequently, they claim there is a need for a dynamic safety case lifecycle to identify such deficits by proactively computing confidence and updating the reasoning about ongoing operations' safety.

Murugesan et al. [20] propose using semantic analysis to identify counter-claims (defeaters) and counter-evidence. Moreover, Muram and Javed. [133] introduces ATTEST, an assurance framework rooted in natural language processing (NLP). Initially, the framework processes text through various NLP procedures. It then formulates rules that encapsulate both syntactic attributes obtained via NLP tasks and semantic features discerned from model structures and their interconnections. These rules are subsequently activated to understand arguments, ensure their correctness, verify their adequacy, identify potential defeaters, and select counter-evidence. Regarding detecting argument fallacies, Yuan et al. [21] examine a subset of them and how they can be detected automatically via predicate-based representation. They build an ontology that contains a set of constant symbols, function symbols and predicate symbols, which form the vocabulary for the expressions of GSN nodes.

Key Takeaway: In our exploration of assurance weakener detection approaches, several methods have emerged. Predominantly, semantic analysis and reasoning, as discussed in [20, 133], have become the most utilized techniques. However, there's an evident limitation in current approaches. Many tend to concentrate on specific issues, either pinpointing argument fallacies or highlighting assurance deficits. This segmented view means we lack a comprehensive method capable of encompassing all assurance weakeners types. It underscores a pressing need for a more holistic detection approach in the field.

Table 5.3: Approaches for Detecting Assurance Weakeners

References	Categories of Approaches
[18]	Reasoning in Description Logic
[19]	Dynamic Safety assurance-based Lifecycle
[20, 133]	Semantic analysis and reasoning
[21]	Predicate-based Approach

Approaches to Mitigate Assurance Weakeners

Once assurance weakeners have been detected, we can decide to tolerate them (i.e., "live with them") or to mitigate them [137]. Mitigating assurance weakeners involves reducing them

[137] or even completely eliminating (removing) them. In accordance with the map that Figure 5.5 depicts, we present in this section the different categories of approaches proposed to mitigate assurance weakeners. Table 5.4 reports the corresponding primary studies.

Table 5.4: Approaches for Mitigating Assurance Weakeners

References	Category of Approaches
[3, 129, 43, 138, 137, 133, 132, 111, 17, 37, 131, 52]	Argumentation
[126, 25, 26, 123, 124, 139, 127, 125, 5, 140]	Formalization
[41]	Bayesian analysis
[42, 2, 122]	Runtime monitoring

- **Argumentation**: several approaches support argument decomposition, refinement, or review to reason away detected assurance weakeners or prevent them. For instance, Liu et al. [3] discuss a novel approach to analyzing failures of digital systems based on the concept of safety arguments. The safety argument of a failed system assists them in developing hypotheses concerning how the system might have failed, eliciting the evidence necessary to confirm or refute those hypotheses, documenting lessons, and developing recommendations to prevent the failure from recurring. Sun et al. [129] suggest considering logical appeals in safety arguments such as Argument by Causation, Argument by Comparison, Argument from Two Sides, Argument from Authority or Expert and Argument by Eliminative Induction. These logic appeals help refine argument structures to mitigate potential logical fallacies.

  Muram et al. [43] present an approach that validates the process models and prevents the occurrence of fallacy, specifically, the omission of key evidence in process-based argumentations. If fallacies are detected in the process models, the approach develops the recommendations to resolve them; afterwards, the process and/or safety engineers modify the process models based on the provided recommendations. Finally, the approach generates the safety argumentations (compliant with Structured Assurance Case Metamodel) from the modified process models by using model-driven engineering

principles free from fallacies. Matsuno et al. [138] investigate uncertainty in implementations constructed by machine learning regarding continuous argument engineering with a granular performance evaluation over the expected operational domain. They employ an attribute testing method for evaluating an implemented model in terms of explicit specification. Chechik et al. [137] suggest addressing uncertainty by modifying the design, the system operation, and the safety argument and producing additional evidence to increase confidence. They also propose to support argument weakening and evidence composition (i.e., provide pieces of evidence of higher quality by composing them to obtain evidence that yields lower uncertainty levels).

A particular form of argumentation is called ***Eliminative Argumentation*** [12]. Eliminative argumentation involves reasoning away argument uncertainties by relying on *defeaters* that challenge arguments [133]. Cobos et al. [132] present a method that combines Attack-Defence Trees used in cybersecurity risk assessments with eliminative argumentation techniques. This combination is showcased within a GSN-based framework, aiming to establish a more targeted risk assurance case for automotive cybersecurity. Millet et al. [111] employ the eliminative argumentation approach to address defeaters. They opted for EA because it distinctly allows for the articulation (and handling) of uncertainties regarding the soundness of the argument by incorporating defeaters. Diemert et al. [17] introduce an incremental assurance approach. Their suggested "syntactic pattern" clearly details the manner in which distinct pieces of evidence, or assertions validated by such evidence, can be integrated to fortify trust in addressing a defeater within an EA discourse.

Eliminative induction [12] is a particular form of induction that allows building confidence in arguments when performing eliminative argumentation. Some approaches rely on the Baconian philosophy of eliminative induction and the use of defeasible reasoning to generate assurance weakeners and reason them away. Goodenough et al. [37], Grigorova and Maibaum [131], as well as Diemert et al. [52] suggest using eliminative induction and defeasible reasoning for mitigating defeaters (doubts). In eliminative induction, a claim is justified only to the extent that reasons for doubting its truth have been eliminated. As grounds for doubt are eliminated (through evidence or argument), confidence in the truth of the claim increases. In addition, they use defeasible reasoning concepts to generate reasons for doubting the truth of a claim (Potential

reasons for doubt are called defeaters) and reason these doubts away. These reasons are usually documented within a confidence map that allows the explicit reasoning about sources of doubt in an argument [125] and eliminating these doubts.

- **Formalization**: Other approaches rely on formal methods to enable verification of argument consistency and well-formedness. This is the case of Nemouchi et al. [126], who suggest using formalization in a machine-checked logic to enable verification of consistency and Well-formedness of assurance cases which might suffer from logical fallacies and inadequate evidence. Rushby [25, 26] suggests using formal verification systems, which provide tools that can be adapted to represent, analyze, and explore the logic of the case, thereby mitigating logic doubt. Moreover, to mitigate epistemic doubt, they suggest building models in logic that describe the elements of our knowledge (e.g., the behaviour of the environment), and these models should be described as systems of constraints. These can be explored and validated using tools based on SMT (satisfiability modulo theories) solvers.

  Denney and Pai [123] discuss that an evidence argument including additional information, such as the tool-specific claims made and the assumptions/reasoning underlying the formal method/tool, provides a richer view of tool-based evidence. This argument can be independently scrutinized to mitigate the assurance deficit and improve the assurance that can be provided. The evidence argument could be created from the output of a theorem prover-based verification tool and then integrated automatically into the wider assurance case. Denney et al. [124] proposed hierarchical safety cases, *hicases*, to aid the comprehension of safety case argument structures. The broad goal is to make safety cases amenable to formal analysis, thereby providing greater assurance and mitigating any potential deficit.

  Cârlan et al. [139] suggest using integrated formal methods in verification activities since deficits in the verification process may negatively impact the confidence of verification results. They use integrated formal methods as evidence in assurance cases, which are used to certify safety-critical systems. They first present two workflows that employ integrated formal methods – code review workflow and code coverage workflow – corresponding to two of the most important activities of the verification phase. Cârlan et al. [127] also proposed an assurance case pattern which addresses the disciplined use of successful but possibly incomplete verification results obtained through C-level

bounded model checking as evidence in certification. They propose a strategy to express confidence in incomplete verification results by complementing them with classical testing and mitigating the assurance deficits with additional tests.

Chechik et al. [125] propose that integrating a certain level of formality within assurance cases can significantly enhance its validity, highlight hidden uncertainties, prevent errors, and also bolster its modularity, adaptability, and reusability. Additionally, they recommend addressing specific assurance deficits through a combination of verification methods. Viger et al. [5] and Murphy et al. [140] suggest using the interactive theorem prover Lean to bridge the gap between safety arguments and rigorous model-based reasoning. They generate formal, model-based, machine-checked AC arguments, taking advantage of the traceability between model and safety artifacts and mitigating errors that could arise from manual argument assessment.

- **Bayesian analysis**: Some approaches employ a Bayesian network approach coupled with evidence theory to support assurance weakeners mitigation. Gansch et al. [41] recommend adopting fault tree analysis (FTA) to address uncertainties. FTA is a graphical representation rooted in Boolean fault propagation, assisting in pinpointing system vulnerabilities such as single-point flaws. Although FTA is widely recognized and frequently employed, it's not without limitations. Especially for autonomous systems, FTA's emphasis on failures restricts its ability to consider human aspects of the system's standard performance. To address FTA's limitations concerning diverse uncertainties, the authors propose a new approach that utilizes evidence theory coupled with Bayesian networks (BN). Evidence theory encompasses various uncertainties, including epistemic and ontological.

- **Runtime monitoring**: Other approaches dynamically monitor assurance weakeners at runtime to provide continuous assurance. To provide continuous (dynamic) assurance, Schleiss et al. [42] advocate for adopting real-time monitoring techniques. These methods allow a system to detect its vulnerability to unforeseen hazardous conditions autonomously. Once identified, the system can swiftly initiate safety measures and embark on a redesign and updating phase to guarantee its continued safety. Belle et al. [2] propose using DACs (Dynamic Assurance Cases) to support real-time safety assurance in autonomous driving systems. They advocate for addressing aleatory

uncertainty during the design phase and relying on Machine Learning to manage epistemic uncertainty at runtime (during system operation). McDermid et al. [122] propose a framework for safety assurance that uses machine learning to provide evidence for a system safety case and thus enables the safety case to be updated dynamically as system behaviour evolves and mitigates assurance deficits.

Key Takeaway: Findings from our analysis suggest that argumentation methodologies, particularly EA, have emerged as the foremost and most frequently employed techniques for mitigating assurance weakeners. These techniques offer a systematic method of addressing and eliminating assurance weakeners, thereby strengthening the overall assurance case. Along with this, runtime monitoring mechanisms—exemplified by the implementation of Dynamic Assurance Cases and the integration of machine learning to furnish real-time evidence—appear to be a potent strategy for mitigating uncertainties.

5.2 Results of RQ2

Two researchers evaluated the answers of GPT-4 to the 22 questions. We then computed the concordance between the two assessors' ratings using the Kendall rank correlation. The resulting correlation between their ratings is **75%**. Such a strong correlation level underscores a robust agreement between the assessors. Thus, there is a high level of consistency in their ratings.

Table 5.5: Average ratings of questions in RQ2

Structural	Semantic	Generation-based	Avg
1.125	1.78	1.35	1.40

Table 5.5 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**.

In the following table 5.6, we can see the 22 questions and the answers GPT-4 Turbo generated.

Table 5.6: Questions and answers generated by GPT-4 Turbo

Type	**Question**	**Answer Generated by GPT-4 Turbo**
Structural	1. What is the Eliminative Argumentation notation?	Eliminative Argumentation Notation is a structured approach to demonstrating and assessing claims about the safety and reliability of a system, where potentially falsifying conditions are systematically evaluated and eliminated.
Structural	2. Which elements are used in the Eliminative Argumentation Notation?	Elements used in Eliminative Argumentation Notation include claims (C), evidence (E), contexts (Ctx), inference rules (IR), and defeaters which can be rebutting (RD), undermining (UM), or undercutting (UC).
Structural	3. What are the different types of defeaters in Eliminative Argumentation?	The different types of defeaters in Eliminative Argumentation are rebutting defeaters (RD), which directly counter the claim; undermining defeaters (UM), which challenge the evidence; and undercutting defeaters (UC), which question the inference rule.
Structural	4. What are the consequences if a defeater in an eliminative argument is not eliminated?	If a defeater in an eliminative argument is not eliminated, the validity of the claim remains uncertain or refuted, potentially undermining confidence in the system's assurance.

Continued on next page

Table 5.6 – continued from previous page

Type	Question	Answer Generated by GPT
Structural	5. Why does an eliminative argument typically not contain a true defeater, and what should be done if a true defeater is discovered?	An eliminative argument typically does not contain a true defeater because the method's goal is to refute all potential defeaters. If a true defeater is discovered, the argument must be restructured, potentially modifying the claim or improving the system.
Structural	6. What are argument terminators in Eliminative Argumentation Notation? Explain Assumed ok, Is ok.	Argument terminators in Eliminative Argumentation Notation indicate the status of a claim or argument element: "Assumed ok" suggests no further analysis is needed, while "Is ok" indicates that the item has been shown to meet its safety or assurance requirements.
Structural	7. What is structured prose for demonstrating Eliminative Argument Notation? How is the numbering and indentation when showing relationships between elements i.e., each level of connection is an indentation?	Structured prose for demonstrating Eliminative Argumentation Notation organizes arguments into a clear, readable format with indentation to show relationships. Each level of connection (e.g., claim to evidence) is indicated by an increased indentation.
Structural	8. How arguments of Eliminative Argumentation are depicted in structured prose?	In structured prose for Eliminative Argumentation, arguments are depicted with "C" for Claim, "RD" for Rebutting Defeater, "UM" for Undermining defeater, and "UC" for Undercutting defeater.

Continued on next page

Table 5.6 – continued from previous page

Type	Question	Answer Generated by GPT
Semantic	9. How should a claim be structured in Eliminative Argumentation? i.e., clearly mention whether it can be in a form of noun-phrase, verb-phrase or predicate.	A claim in Eliminative Argumentation should be structured as a predicate, a statement that something is the case. It often takes the form of a noun-phrase or a predicate expressing a property or relation.
Semantic	10. How should an evidence be structured in Eliminative Argumentation? i.e., clearly mention whether it can be in a form of noun-phrase, verb-phrase or predicate.	Evidence in Eliminative Argumentation should also be structured as a predicate, a statement of fact that supports the claim. It typically takes the form of a noun-phrase or a verb-phrase providing observed data or test results.
Semantic	11. How should a context be structured in Eliminative Argumentation? i.e., clearly mention whether it can have a noun-phrase or a verb-phrase or predicate.	A context in Eliminative Argumentation can be in the form of a noun-phrase or a predicate, setting the scene or conditions under which the claim and evidence are valid.
Semantic	12. How should an inference rule be structured in Eliminative Argumentation? i.e., clearly mention whether they can be in a form of noun-phrase, verb-phrase or predicate.	An inference rule in Eliminative Argumentation should be in the form of a predicate relating claim and evidence, often expressed as a conditional (if-then) statement.

Continued on next page

Table 5.6 – continued from previous page

Type	Question	Answer Generated by GPT
Semantic	13. How should a Rebutting Defeater be structured in Eliminative Argumentation? i.e., is a predicate associated with a claim, preceded by "Unless".	A Rebutting Defeater in Eliminative Argumentation is structured as a predicate that challenges the claim, preceded by "Unless..." to indicate the condition under which the claim would not hold.
Semantic	14. How should an Undercutting defeater be structured in Eliminative Argumentation? i.e., is a doubt about the validity of an inference rule (P → Q), preceded by "Unless".	An Undercutting Defeater in Eliminative Argumentation is a doubt about the validity of an inference rule, structured as a predicate, often in the form "Unless there is doubt about (P → Q)...
Semantic	15. How should an Undermining defeater be structured in Eliminative Argumentation? i.e., is a predicate associated with evidence, preceded by "But".	An Undermining Defeater in Eliminative Argumentation is a predicate associated with evidence, structured as "But (evidence) might not be valid because...
Generation Based	16. Generate me a sample Claim.	Sample Claim: "The software module is free from race conditions."
Generation Based	17. Generate me a sample Evidence.	Sample Evidence: "All concurrent processes have been tested with a lock mechanism."
Generation Based	18. Generate me a sample Context.	Sample Context: "Tests were conducted under maximum load scenarios."
Generation Based	19. Generate me a sample Inference rule.	Sample Inference Rule: "If all concurrent processes function correctly with a lock mechanism during maximum load, then the software module has no race conditions."

Continued on next page

Table 5.6 – continued from previous page

Type	Question	Answer Generated by GPT
Generation Based	20. Generate me a sample Claim and a Rebutting defeater that defeats it. Show it in structured prose.	Structured Prose Example for a Claim and a Rebutting Defeater: C: The bridge supports a load of 50 tons. RD: Unless the material fatigue is greater than expected.
Generation Based	21. Generate me a sample inference rule and an Undercutting defeater that defeats it. Show it in structured prose.	Structured Prose Example for an Inference Rule and an Undercutting Defeater: IR: If a bridge passes a load test, it can support 50 tons. UC: Unless the load test does not simulate real-world conditions accurately.
Generation Based	22. Generate me a sample evidence and an Undermining defeater that defeats it. Show it in structured prose.	Structured Prose Example for an Evidence and an Undermining Defeater: E: The sensor accurately measures temperatures up to 150°C. UM: But the sensor has not been tested in high-humidity environments.

GPT-4 Turbo demonstrated particular proficiency in responding to questions about structural accuracy. It also showed commendable performance in generation-based queries, effectively creating EA elements, particularly various types of defeaters. However, it is important to note that GPT-4 Turbo's understanding of the semantics of EA elements was less robust. Therefore, it would be beneficial to employ specific prompting techniques to enhance its comprehension of EA semantics, which in turn could lead to the generation of better defeaters.

5.3 Results of RQ3

5.3.1 Ground-truth similarity results

Exact match or string matching results

Table 5.7 shows the fuzzy similarity scores. Based on the mean similarity scores, Experiments 1 and 2 outperform Experiments 3 and 4, suggesting 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's 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.

Case	Experim. 1	Experim. 2	Experim. 3	Experim. 4
1	0.58	0.58	0.57	0.57
2	0.48	0.55	0.51	0.37
3	0.78	0.71	0.52	0.60
4	0.65	0.65	0.56	0.67
5	0.50	0.52	0.25	0.26
6	0.78	0.66	0.07	0.63
7	0.56	0.57	0.15	0.59
8	0.37	0.37	0.36	0.30
9	0.40	0.47	0.48	0.45
Mean	**0.57**	**0.56**	**0.39**	**0.49**

Table 5.7: Fuzzy Similarity Scores

BLEU score

Table 5.8 presents the 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 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 pattern reaffirms 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.

Case	Experim. 1	Experim. 2	Experim. 3	Experim. 4
1	0.21	0.35	0.34	0.45
2	0.18	0.32	0.37	0.48
3	0.58	0.73	0.46	0.58
4	0.69	0.69	0.58	0.69
5	0.11	0.37	0.20	0.19
6	0.53	0.62	0.44	0.54
7	0.25	0.65	0.11	0.63
8	0.06	0.07	0.23	0.45
9	0.06	0.31	0.32	0.33
Mean	**0.30**	**0.46**	**0.34**	**0.48**

Table 5.8: BLEU Scores

Semantic similarity results

Table 5.9 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. Contrary to initial expectations, the introduction of context in Experiment 3 leads to a reduction in mean similarity scores, suggesting that context alone, without tailored examples, might not always enhance text generation quality. However, Experiment 4, which combines one-shot learning with context, shows improved performance over Experiment 3, underlining the importance of a balanced integration of both context and example-based learning.

The analysis of ground-truth similarity across different metrics—fuzzy logic similarity, BLEU scores, and cosine similarity—reveals nuanced insights into the effectiveness of the experimental setups. Experiment 2, which utilized a one-shot approach with CoT and predicate-based rules without context, consistently showed high performance, indicating that minimal guidance (a single example) can significantly impact the model's ability to generate defeaters closely aligned with the expected outcomes.

Case	Experim. 1	Experim. 2	Experim. 3	Experim. 4
1	0.79	0.80	0.77	0.71
2	0.70	0.70	0.62	0.67
3	0.78	0.72	0.68	0.70
4	0.59	0.54	0.55	0.61
5	0.53	0.57	0.42	0.54
6	0.86	0.74	0.74	0.80
7	0.58	0.63	0.50	0.49
8	0.50	0.58	0.47	0.49
9	0.31	0.45	0.34	0.41
Mean	**0.63**	**0.64**	**0.57**	**0.60**

Table 5.9: Cosine Similarity Scores

5.3.2 Reasonability results

As we stated above, two assessors reviewed the defeaters GPT-4 generated and rated their reasonability. The corresponding value of the Kendall rank coefficient is 0.89. This indicates a strong agreement between the two raters. Table 5.10 reports the average of the reasonability results for the four experiments we performed on the 9 AC 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 5.10: Average Reasonability Scores

5.3.3 Discussion

Our dataset predominantly consists of rebutting defeaters, with a notable scarcity of undercutting and undermining defeaters. This causes a rather significant imbalance in our dataset

and makes it challenging to assess if our approach is more performant for a specific type of defeaters.

Contrary to our expectations, Experiment 4 results did not systematically outperform Experiment 2 results. This suggests that our context might not be well structured enough. However, the addition of context in Experiments 3 and 4 did not consistently enhance performance, as demonstrated by the cosine similarity scores where Experiment 3 registered the lowest average. Despite Experiment 4's attempt to meld one-shot learning with added context, it failed to significantly surpass the performance of the one-shot approach devoid of context. The mean scores across all experiments suggest that although context and examples hold value, their integration and the model's capacity to effectively leverage them can differ significantly. In this regard, improving the context for GPT-4 turbo prompts could potentially enhance its performance.

Additionally, GPT-4 turbo's performance in generating the right number of defeaters sometimes fell short of the ground truth, highlighting the need to determine the optimal number of defeaters to be generated. To tackle this issue, we may explore the possibility of developing a heuristic that enables the model to estimate and then generate the right number of defeaters that GPT-4 needs to generate.

Another significant challenge we faced was the scarcity of assurance cases complying with EA. Their scarcity is further compounded by the sensitive and critical nature of the data they encompass, which often restricts their public availability. Having access to more assurance cases complying with EA would have help us generalize our findings.

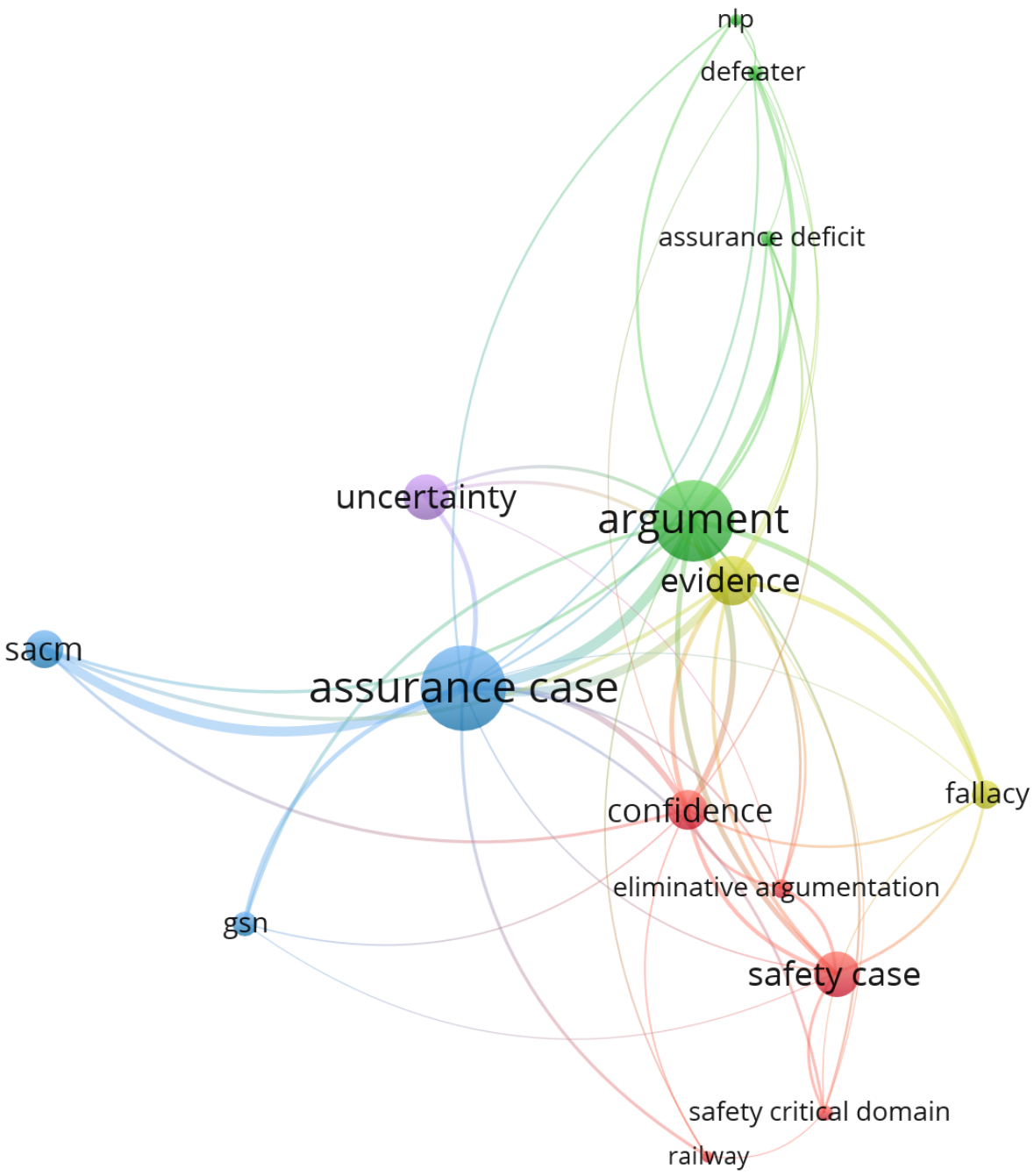

Figure 5.4: Most Influential Research Topics

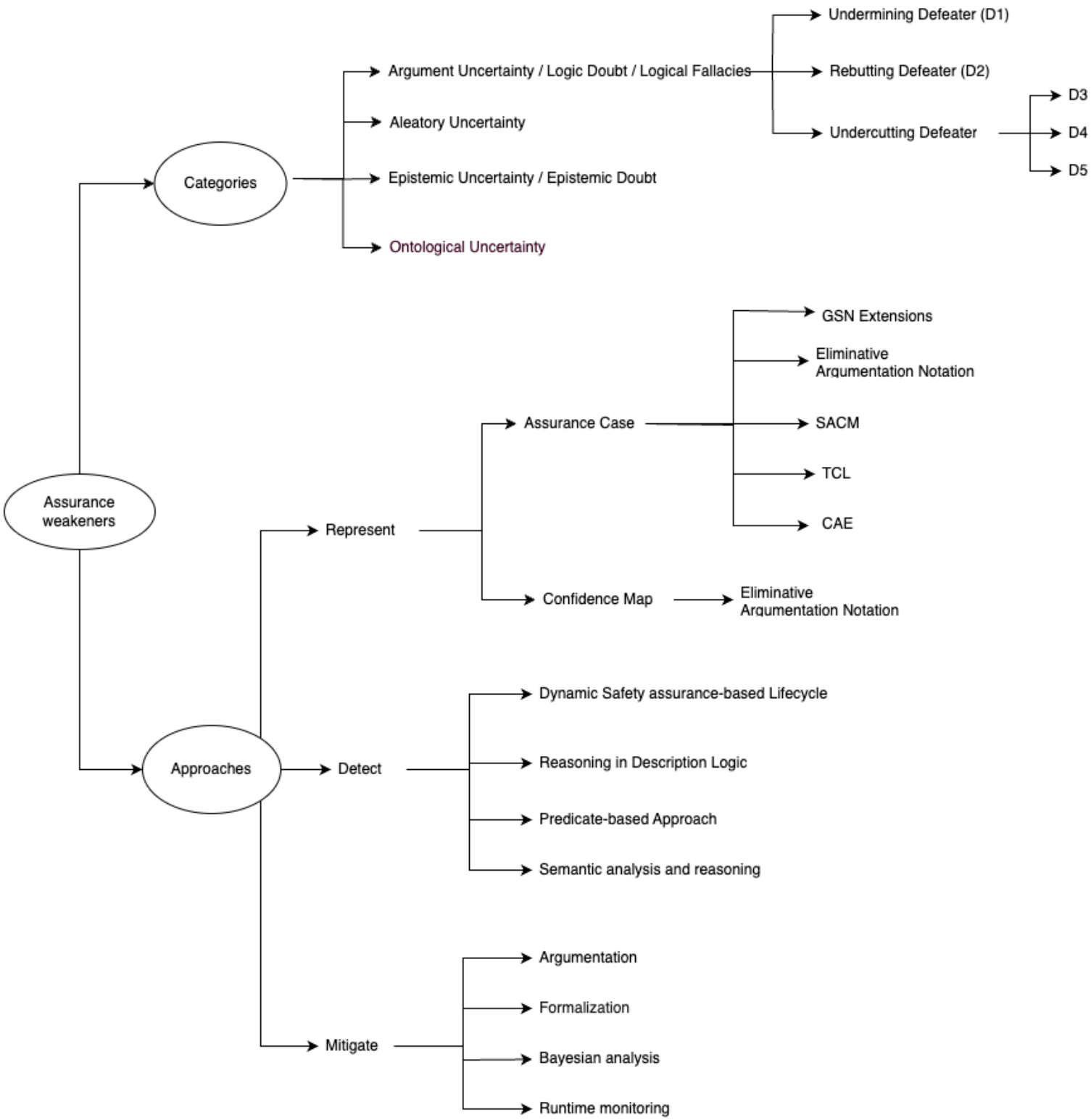

Figure 5.5: Taxonomy of assurance weakeners and their management approaches

Chapter 6

Threats To Validity

We adopt the framework that Wohlin et al. [141], and Zhou et al. [142] propose to discuss the limitations of our work.

6.1 Threats regarding RQ1

6.1.1 Internal validity

In our selection and data extraction processes, the potentially subjective analysis of the reviewers could introduce bias and the risk of missing relevant results. To minimize these potential issues, a rigorous approach was adopted. Two reviewers were involved in the selection of primary studies and data extraction. During selection, one researcher initially chose the papers, while the other conducted random sampling to ensure relevance with the eligibility criteria. Data extraction was also conducted by both reviewers, with the first reviewer covering all papers and the second reviewer extracting data from half of them. Reviewers discussed any disagreements during meetings to reach a consensus.

6.1.2 Construct validity

While our database-driven search employed a query that may not have been initially exhaustive, we took measures to enhance its completeness. Through iterative refinement based on the existing literature, we progressively developed a query that effectively captured a substantial body of relevant work. Moreover, the limitation of searching within five digital libraries

may have introduced the risk of overlooking relevant studies. To address this limitation, we employed a snowballing technique to identify additional studies, especially those not present in the databases of the five libraries. This ensures the completeness of our search process.

6.1.3 Conclusion validity

By adhering to systematic review guidelines and transparently reporting our methodology, we have ensured the reproducibility of our study. Researchers with access to the same libraries and resources we mention in this study can replicate our study and expect to obtain similar results. This transparency enhances the overall reliability and validity of our research. It's worth noting that they may uncover additional findings, especially for papers published after the completion of our search. For instance, when it comes to studies published in 2023, we searched for studies published between January 2023 and the beginning of October 2023. So, our search may miss the few studies published between October and December of the same year, if there are any. Thus, surveying studies published that year may have yielded a partial and potentially biased overview of the scientific contributions made in 2023 regarding the surveyed topic.

6.2 Threats regarding RQ2 and RQ3

6.2.1 External Validity

A threat to the external validity of this research comes from the sensitivity of our findings to the specific case studies examined. These cases were selected to showcase the potential of LLMs in identifying and mitigating defeaters in ACs. Although they effectively demonstrate the applicability of LLMs, they may not adequately represent the diverse range of ACs. Furthermore, the results of RQ2 and RQ3 are highly dependent on the capabilities and limitations of the specific LLM used, namely GPT-4 Turbo. Different LLMs might possess varying proficiencies in managing the complexities of assurance cases and defeater identification, potentially influencing the reproducibility and broader applicability of our findings. Additionally, the prompts utilized in this study were carefully designed to maximize the LLM's effectiveness in addressing the research questions. This tailored approach, suggests that the LLM's success may greatly depend on the precision and quality of the prompts

developed by users.

Despite these concerns impacting external validity, they do not undermine the demonstrated potential of LLMs in the field of ACs and the identification and mitigation of defeaters. Rather, they underscore the need for further research and caution when generalizing these findings to other contexts. Future studies could enhance the generalizability of these outcomes by incorporating a wider array of LLMs, expanding the selection of case studies, and employing less customized prompting techniques.

6.2.2 Construct Validity

Regarding RQ2, this threat raises the question of whether the set of 22 questions is complete, consistent, and can accurately capture the structural and semantic rules EA embodies. To address this threat, we engaged experts in EA to review and provide feedback on the questions. This process was iterative, allowing for multiple rounds of refinement based on the experts' insights, which ensured that the questions were complete, consistent, and effectively represented the complexities of EA.

Concerning RQ3, the dataset's significant imbalance, with a predominance of rebutting defeaters over undercutting and undermining defeaters, directly impacts construct validity. This imbalance challenges the generalizability of our findings to all categories of defeaters. This imbalance primarily stems from the scarcity of publicly available ACs, which restricts the variety of defeaters we can analyze. To address this limitation, we plan to expand our search for ACs beyond publicly available sources. Collaborating with industry partners and academic institutions may provide access to a broader range of cases, potentially enriching our dataset with a more diverse array of defeaters and enhancing the robustness of our conclusions.

6.2.3 Conclusion Validity

Our dataset consists of nine assurance case fragments: cases 1-9 (see Section 4.4.3). The work of Millet et al. [111] documents cases 1-8 and was published in 2023 by the SAFECOMP conference. Furthermore, the work of Diemert et al. [17] documents case 9 and was also published in 2023. Since the cut-off date of GPT-4 turbo is quite recent (i.e., 2023[7]), it may

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

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) when applicable. This means that the GPT-4 turbo has never seen most of the resulting pre-processed fragments. Still, to tackle that issue in future work, we will sample significantly more recent data (i.e., assurance case fragments) that is not publicly available.

Chapter 7

Future Work and Conclusion

7.1 Future Work

The outcomes of our thesis allow us to conclude that several future directions still need to be explored to better deal with assurance weakeners and increase the assurance of systems. Moreover, we discuss using GPT-4 Turbo to mitigate defeaters.

7.1.1 Improvement of existing Assurance Case Notations to better represent Assurance Weakeners

The findings of our systematic review suggest that we may need to extend existing graphical notations (e.g., GSN, CAE) to better present assurance weakeners. In Section 5.1.4, we specifically demonstrated that only **argument uncertainty** is depicted by the current notations, while aleatory, epistemic, and ontological uncertainties lack explicit representation. Such omissions complicate modeling-level reasoning and the formulation of effective solutions for their detection and mitigation. One might question the logic of such representations.

Besides, as Chechik et al. [125] point out, arguments may lack rigour. Such arguments usually miss several critical properties, including completeness, independence, relevance, or a clear statement of assumptions. So, it may be interesting to analyze the specifications of existing assurance case notations to determine if they need to be extended to better support these properties and provide a better representation of assurance weakeners in arguments, when applicable. Besides, explicit modelling and management of uncertainty in evidence, specifications and assumptions, and the clear justification of each step of the development of

a system, can go a long way toward making arguments valid, reusable, and generally helpful in helping produce high-quality software systems.

Moreover, extending well-established notations could help support a better representation of 1) different types of weakeners; 2) relationships between assurance weakeners and existing notations concepts, allowing the representation of the traditional argumentation structure made of claims, arguments and evidence; 3) assurance weakeners decorators. Such extensions could facilitate the communication of concerns and limitations about the system being audited [52].

Still, in Section 5.1.4, we showed that SACM may be superior to other notations, especially when it comes to representing weakeners. This is because it is a standard that unifies GSN and CAE [126] and supports all types of defeaters and their relationships. Consequently, there should be a heightened emphasis on further research on SACM which will probably witness a greater adoption in the upcoming years.

7.1.2 Using GPT-4 Turbo to mitigate defeaters

Future work focuses on the **Phase III** of the high-level overview of our proposed approach shown in 4.1. The primary objective is to use the capabilities of GPT-4 Turbo in effectively mitigating defeaters identified in Phase II. This phase is critical for enhancing the reliability and robustness of ACs by ensuring that they can withstand rigorous scrutiny and support requirements verification and validation more comprehensively.

Leveraging the expertise of domain specialists, we aim to refine and enhance the mitigation strategies proposed by GPT-4 Turbo. This collaborative effort ensures that the AI-generated solutions are not only theoretically sound but also practically applicable and aligned with the nuanced requirements of EA. Moreover, building on the findings from Phases I and II, Phase III will focus on developing sophisticated prompting techniques that guide GPT-4 Turbo more effectively in generating and refining mitigation strategies. These techniques will be designed to encourage deeper reasoning and a more nuanced understanding of the domain-specific challenges presented by defeaters in ACs.

7.2 Conclusion

Cyber-physical systems (CPSs) are usually mission-critical systems used in several sectors such as healthcare, aviation, nuclear energy and automotive. Their complexity increases when they incorporate machine learning. Their failure can have severe consequences such as the death of people, injuries, and major economic losses. CPSs therefore carry a lot of responsibilities, especially for human safety. Using convincing assurance cases to verify the correct implementation of their capabilities and therefore support their certification in compliance with industrial standards is critical to foster their acceptance by regulators. One of the biggest challenges in these systems is assurance weakeners, which can reduce our trust in their safety and reliability.

Our thesis reports a taxonomy of assurance weakeners and classifies them. This helps us better understand and manage their potential effects. Our thesis also reports a classification of approaches proposed to represent, identify and manage assurance weakeners. Additionally, the experiments reported in our thesis also show that GPT-4 Turbo is effective at understanding and using Eliminative Argumentation, a notation used to formally represent assurance cases. We also used GPT-4 Turbo to identify defeaters in ACs. We believe generative AI and more specifically large language models such as GPT-4 Turbo can play a crucial role in making cyber-physical systems more robust, especially when it comes to their safety, reliability and security.

Appendix A

Queries for Each Database

Table A.1: Queries searched in each database

Database name	**Query(ies)**	**Parameters**
Scopus	TITLE-ABS-KEY(("assurance deficits" OR "false assurance" OR "defeaters" OR "counter evidence" OR "counter-argument" OR "fallacies" OR "aleatory uncertainty" OR "aleatoric uncertainty" OR "epistemic uncertainty" OR "uncertainty reasoning" OR "uncertainty elicitation" OR "informal semantics" OR "doubt") AND ("assurance case" OR "safety case" OR "trust case" OR "dependability case" OR "reliability case" OR "security case" OR "availability case")) AND PUBYEAR AFT 2011	Advanced search on title/abstract/keyword, Published after 2011

Continued on next page

Table A.1 – continued from previous page

Database name	**Query(ies)**	**Parameters**
IEEE Xplore	("assurance deficits" OR “false assurance” OR "defeaters" OR "counter evidence" OR "counter-argument" OR "fallacies" OR "aleatory uncertainty" OR "aleatoric uncertainty" OR "epistemic uncertainty" OR "uncertainty reasoning" OR “uncertainty elicitation” OR “informal semantics” OR “doubt”) AND (“assurance case” OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”)	Command search on metadata
ACM	("assurance deficits" OR “false assurance” OR "defeaters" OR "counter evidence" OR "counter-argument" OR "fallacies" OR "aleatory uncertainty" OR "aleatoric uncertainty" OR "epistemic uncertainty" OR "uncertainty reasoning" OR “uncertainty elicitation” OR “informal semantics” OR “doubt”) AND ("assurance case" OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”)	Advanced search on the title and abstract

Continued on next page

Table A.1 – continued from previous page

Database name	**Query(ies)**	**Parameters**
Engineering Village	("assurance deficits" OR “false assurance” OR "defeaters" OR "counter evidence" OR "counter-argument" OR "fallacies" OR "aleatory uncertainty" OR "aleatoric uncertainty" OR "epistemic uncertainty" OR "uncertainty reasoning" OR “uncertainty elicitation” OR “informal semantics” OR “doubt”) AND (“assurance case” OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”)	Search on the subject, title and abstract

Continued on next page

Table A.1 – continued from previous page

Database name	Query(ies)	Parameters
Google Scholar	• **Query 1:** ("assurance deficits" OR “false assurance” OR "defeaters") AND (“assurance case” OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”) • **Query 2:** ("counter evidence" OR "counter-argument" OR "fallacies") AND (“assurance case” OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”) • **Query 3:** ("aleatory uncertainty" OR "aleatoric uncertainty" OR "epistemic uncertainty") AND (“assurance case” OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”) • **Query 4:** ("uncertainty reasoning" OR “uncertainty elicitation” OR “informal semantics” OR “doubt”) AND (“assurance case” OR “safety case” OR “trust case” OR “dependability case” OR “reliability case” OR “security case” OR “availability case”)	Keyword, 2012-2023

Appendix B

Study Characteristics

Table B.1: List of Studies

No.	Authors (Publication Year)	Study Title	Venue	Type of Search
1	Liu et al. (2012) [3]	A safety-argument based method to predict system failure	ICPHM	Database-driven
2	Denney and Pai. (2013) [123]	Evidence arguments for using formal methods in software certification	ISSRE	Database-driven
3	Goodenough et al. (2013) [37]	Eliminative induction: A basis for arguing system confidence	ICSE	Database-driven
4	Rushby (2013) [25]	Logic and epistemology in safety cases	SAFECOMP	Database-driven
5	Yamamoto et al. (2013) [121]	An evaluation of argument patterns to reduce pitfalls of applying assurance case	ASSURE	Database-driven
6	Graydon (2014) [38]	Towards a clearer understanding of context and its role in assurance argument confidence	SAFECOMP	Database-driven
7	Grigorova and Maibaum (2014) [131]	Argument evaluation in the context of assurance case confidence modeling	ISSRE	Database-driven
8	Rushby (2014) [26]	Mechanized support for assurance case argumentation	JSAI	Database-driven

Continued on next page

Table B.1 – continued from previous page

No.	Authors (Publication Year)	Study Title	Venue	Type of Search
9	Sun et al. (2014) [129]	Rethinking of Strategy for Safety Argument Development	SAFECOMP	Database-driven
10	Takai and Kido (2014) [130]	A supplemental notation of GSN aiming for dealing with changes of assurance cases	ISSRE	Database-driven
11	Bandur and McDermid (2015) [128]	Informing assurance case review through a formal interpretation of GSN core logic	SAFECOMP	Database-driven
12	Denney et al. (2015) [19]	Dynamic safety cases for through-life safety assurance	ICSE	Database-driven
13	Denney et al. (2015) [124]	Formal foundations for hierarchical safety cases	HASE	Database-driven
14	Groza et al. (2015) [18]	A formal approach for identifying assurance deficits in unmanned aerial vehicle software	ICSEng	Database-driven
15	Jarzębowicz and Wardziński (2015) [39]	Integrating confidence and assurance arguments	IET	Database-driven
16	Cârlan et al. (2016) [139]	Integrated formal methods for constructing assurance cases	ISSRE	Database-driven
17	Cârlan et al. (2016) [127]	On using results of code-level bounded model checking in assurance cases	SAFECOMP	Database-driven
18	Yuan et al. (2016) [21]	Automatically detecting fallacies in system safety arguments	PRIMA	Database-driven
19	Muram et al. (2018) [43]	Preventing omission of key evidence fallacy in process-based argumentations	QUATIC	Database-driven
20	Chechik et al. (2019) [125]	Software Assurance in an Uncertain World	FASE	Snowballing
21	Matsuno et al. (2019) [138]	Tackling Uncertainty in Safety Assurance for Machine Learning: Continuous Argument Engineering with Attributed Tests	SAFECOMP	Snowballing
22	McDermid et al. (2019) [122]	Towards a Framework for Safety Assurance of Autonomous Systems	AISafety	Snowballing

Continued on next page

Table B.1 – continued from previous page

No.	Authors (Publication Year)	Study Title	Venue	Type of Search
23	Nemouchi et al. (2019) [126]	Isabelle/SACM: Computer-assisted assurance cases with integrated formal methods	iFM	Database-driven
24	Chechik et al. (2020) [137]	Uncertainty, modeling and safety assurance: towards a unified framework	VSTTE	Database-driven
25	Diemert et al. (2020) [52]	Eliminative Argumentation for Arguing System Safety-A Practitioner's Experience	SysCon	Database-driven
26	Gansch et al. (2020) [41]	System theoretic view on uncertainties	DATE	Database-driven
27	Jarzębowicz and Markiewicz (2020) [31]	Representing process characteristics to increase confidence in assurance case arguments	DepCoS	Database-driven
28	Selviandro et al. (2020) [134]	A visual notation for the representation of assurance cases using sacm	IMBSA	Database-driven
29	Foster et al. (2021) [135]	Integration of Formal Proof into Unified Assurance Cases with Isabelle/SACM	FAC	Database-driven
30	Murphy et al. (2021) [140]	Validating safety arguments with lean	SEFM	Database-driven
31	Viger et al. (2021) [5]	A lean approach to building valid model-based safety arguments	MODELS	Database-driven
32	Cobos et al. (2021) [132]	Cybersecurity Assurance Challenges for Future Connected and Automated Vehicles	ESREL	Snowballing
33	Schleiss et al. (2022) [42]	Towards Continuous Safety Assurance for Autonomous Systems	ICSRS	Database-driven
34	Belle et al. (2023) [2]	Position paper: a vision for the dynamic safety assurance of ML-enabled autonomous driving systems	REW	Database-driven
35	Diemert et al. (2023) [17]	Incremental Assurance Through Eliminative Argumentation	JSS	Database-driven
36	Menghi et al. (2023) [16]	Assurance case development as data: A manifesto	ICSE-NIER	Database-driven

Continued on next page

Table B.1 – continued from previous page

No.	Authors (Publication Year)	Study Title	Venue	Type of Search
37	Millet et al. (2023) [111] (2023)	Assurance Case Arguments in the Large: The CERN LHC Machine Protection System	SAFECOMP	Database-driven
38	Muram and Javed (2023) [133]	ATTEST: Automating the review and update of assurance case arguments	JSA	Database-driven
39	Murugesan et al. (2023) [20]	Semantic Analysis of Assurance Cases using s(CASP)	ICLP	Database-driven

Appendix C

Venue Names and Acronyms

Table C.1: Venue Names and Acronyms

Venue Name	**Acronym**
IEEE International Symposium on Software Reliability Engineering Workshops	ISSRE
Prognostics and System Health Management Conference	ICPHM
Fundamental Approaches to Software Engineering	FASE
International Conference on Software Engineering and Formal Methods	SEFM
Formal Aspects of Computing	FAC
ACM/IEEE International Conference on Model Driven Engineering Languages and Systems	MODELS
Verified Software. Theories Tools and Experiments	VSTTE
Design Automation & Test in Europe Conference & Exhibition	DATE
International Conference on Dependability and Complex Systems	DepCoS
IEEE International Systems Conference	SysCon
International Symposium on Model-Based Safety and Assessment	IMBSA
Artificial Intelligence Safety	AISafety
Integrated Formal Methods	iFM
International Workshop on Assurance Cases for Software-Intensive Systems	ASSURE
International Conference on the Quality of Information and Communications Technology	QUATIC
Principles and Practice of Multi-Agent Systems	PRIMA

Continued on next page

Table C.1 – *Continued from previous page*

Venue Name	**Acronym**
IET System Safety and Cyber-Security Conference	IET
International Symposium on High Assurance Systems Engineering	HASE
International Conference on Software Engineering	ICSE
International Conference on Systems Engineering	ICSEng
JSAI International Symposium on Artificial Intelligence	JSAI
International Conference on System Reliability and Safety	ICSRS
European Safety and Reliability Conference	ESREL
International Requirements Engineering Conference Workshops	REW
Journal of System Safety	JSS
International Conference on Software Engineering: New Ideas and Emerging Results	ICSE-NIER
Journal of Systems Architecture	JSA
International Conference on Logic Programming	ICLP

Bibliography

[1] Charles Hartsell et al. "Automated method for assurance case construction from system design models". In: *2021 5th International Conference on System Reliability and Safety (ICSRS)*. IEEE. 2021, pp. 230–239.

[2] Alvine Boaye Belle, Hadi Hemmati, and Timothy C Lethbridge. "Position paper: a vision for the dynamic safety assurance of ML-enabled autonomous driving systems". In: *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE. 2023, pp. 297–301.

[3] Qixing Liu et al. "A safety-argument based method to predict system failure". In: *Proceedings of the IEEE 2012 Prognostics and System Health Management Conference (PHM-2012 Beijing)*. IEEE. 2012, pp. 1–5.

[4] Zarrin Langari and Tom Maibaum. "Safety cases: a review of challenges". In: *2013 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*. IEEE. 2013, pp. 1–6.

[5] Torin Viger et al. "A lean approach to building valid model-based safety arguments". In: *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2021, pp. 194–204.

[6] Richard Hawkins et al. "Weaving an assurance case from design: a model-based approach". In: *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE. 2015, pp. 110–117.

[7] Marwa Zeroual et al. "Constructing security cases based on formal verification of security requirements in alloy". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2023, pp. 15–25.

[8] Shreyas Ramakrishna et al. "Automating Pattern Selection for Assurance Case Development for Cyber-Physical Systems". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2022, pp. 82–96.

[9] Emilia Cioroaica et al. "Towards the Concept of Trust Assurance Case". In: *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE. 2022, pp. 1581–1586.

[10] Alvine Boaye Belle and Yixi Zhao. "Evidence-based decision-making: On the use of systematicity cases to check the compliance of reviews with reporting guidelines such as PRISMA 2020". In: *Expert Systems with Applications* 217 (2023), p. 119569.

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

[12] John B Goodenough, Charles B Weinstock, and Ari Z Klein. "Eliminative argumentation: A basis for arguing confidence in system properties". In: *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005* (2015).

[13] Richard Hawkins et al. "A new approach to creating clear safety arguments". In: *Advances in Systems Safety: Proceedings of the Nineteenth Safety-Critical Systems Symposium, Southampton, UK, 8-10th February 2011*. Springer. 2011, pp. 3–23.

[14] Chung-Ling Lin. *Support Assurance-Based Software Development for Mission Critical Domains Using the Model Driven Architecture*. Western Michigan University, 2018.

[15] *Tesla Crashes*. Available online at https://www.latimes.com/business/story/2023-02-16/tesla-recall-cars-full-self-driving-beta-crash-risk. URL: https://www.latimes.com/business/story/2023-02-16/tesla-recall-cars-full-self-driving-beta-crash-risk.

[16] Claudio Menghi et al. "Assurance case development as data: A manifesto". In: *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE. 2023, pp. 135–139.

[17] Simon Diemert et al. "Incremental Assurance Through Eliminative Argumentation". In: *Journal of System Safety* 58.1 (2023), pp. 7–15.

[18] Adrian Groza et al. "A formal approach for identifying assurance deficits in unmanned aerial vehicle software". In: *Progress in Systems Engineering: Proceedings of the Twenty-Third International Conference on Systems Engineering*. Springer. 2015, pp. 233–239.

[19] Ewen Denney, Ganesh Pai, and Ibrahim Habli. "Dynamic safety cases for through-life safety assurance". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. IEEE. 2015, pp. 587–590.

[20] Anitha Murugesan et al. "Semantic Analysis of Assurance Cases using s (CASP)". In: (2023).

[21] Tangming Yuan et al. "Automatically detecting fallacies in system safety arguments". In: *Principles and Practice of Multi-Agent Systems: International Workshops: IWEC 2014, Gold Coast, QLD, Australia, December 1-5, 2014, and CMNA XV and IWEC 2015, Bertinoro, Italy, October 26, 2015, Revised Selected Papers 5*. Springer. 2016, pp. 47–59.

[22] Torin Viger et al. "Supporting Assurance Case Development Using Generative AI". In: *SAFECOMP 2023, Position Paper*. 2023.

[23] Mike Maksimov, Sahar Kokaly, and Marsha Chechik. "A survey of tool-supported assurance case assessment techniques". In: *ACM Computing Surveys (CSUR)* 52.5 (2019), pp. 1–34.

[24] Sunil Nair et al. "An extended systematic literature review on provision of evidence for safety certification". In: *Information and Software Technology* 56.7 (2014), pp. 689–717.

[25] John Rushby. "Logic and epistemology in safety cases". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2013, pp. 1–7.

[26] John Rushby. "Mechanized support for assurance case argumentation". In: *New Frontiers in Artificial Intelligence: JSAI-isAI 2013 Workshops, LENLS, JURISIN, MiMI, AAA, and DDS, Kanagawa, Japan, October 27–28, 2013, Revised Selected Papers 5*. Springer. 2014, pp. 304–318.

[27] Humza Naveed et al. "A comprehensive overview of large language models". In: *arXiv preprint arXiv:2307.06435* (2023).

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

[29] Nikolai Mansourov and Djenana Campara. *System assurance: beyond detecting vulnerabilities*. Elsevier, 2010.

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

[31] Aleksander Jarzębowicz and Szymon Markiewicz. "Representing process characteristics to increase confidence in assurance case arguments". In: *Engineering in Dependability of Computer Systems and Networks: Proceedings of the Fourteenth International Conference on Dependability of Computer Systems DepCoS-RELCOMEX, July 1–5, 2019, Brunów, Poland*. Springer. 2020, pp. 245–255.

[32] Robin E Bloomfield, Bev Littlewood, and David Wright. "Confidence: its role in dependability cases for risk assessment". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE. 2007, pp. 338–346.

[33] Lian Duan et al. "Reasoning about confidence and uncertainty in assurance cases: A survey". In: *Software Engineering in Health Care: 4th International Symposium, FHIES 2014, and 6th International Workshop, SEHC 2014, Washington, DC, USA, July 17-18, 2014, Revised Selected Papers 4*. Springer. 2017, pp. 64–80.

[34] Sunil Nair et al. "Classification, structuring, and assessment of evidence for safety–a systematic literature review". In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE. 2013, pp. 94–103.

[35] Jose Luis de La Vara et al. "An industrial survey of safety evidence change impact analysis practice". In: *IEEE Transactions on Software Engineering* 42.12 (2016), pp. 1095–1117.

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

[37] John B Goodenough, Charles B Weinstock, and Ari Z Klein. "Eliminative induction: A basis for arguing system confidence". In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 1161–1164.

[38] Patrick John Graydon. "Towards a clearer understanding of context and its role in assurance argument confidence". In: *Computer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings 33*. Springer. 2014, pp. 139–154.

[39] A Jarzbowicz and Andrzej Wardziński. "Integrating confidence and assurance arguments". In: (2015).

[40] Loli Burgueño et al. "Belief uncertainty in software models". In: *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*. IEEE. 2019, pp. 19–26.

[41] Roman Gansch and Ahmad Adee. "System theoretic view on uncertainties". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 1345–1350.

[42] Philipp Schleiss, Francesco Carella, and Iwo Kurzidem. "Towards continuous safety assurance for autonomous systems". In: *2022 6th International Conference on System Reliability and Safety (ICSRS)*. IEEE. 2022, pp. 457–462.

[43] Faiz UL Muram, Barbara Gallina, and Laura Gómez Rodríguez. "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. 2018, pp. 65–73.

[44] William S Greenwell et al. "A taxonomy of fallacies in system safety arguments". In: *24th International System Safety Conference*. 2006.

[45] Ran Wei et al. "Model based system assurance using the structured assurance case metamodel". In: *Journal of Systems and Software* 154 (2019), pp. 211–233.

[46] *GSN (Goal Structuring Notation) v3*. URL: `https://scsc.uk/gsn`.

[47] Peter Bishop and Robin Bloomfield. "A methodology for safety case development". In: *Safety and Reliability*. Vol. 20. Taylor & Francis. 2000, pp. 34–42.

[48] Tim Kelly and Rob Weaver. "The goal structuring notation–a safety argument notation". In: *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*. Vol. 6. Citeseer. 2004.

[49] Elisabeth A Strunk and John C Knight. "The essential synthesis of problem frames and assurance cases". In: *Proceedings of the 2006 international workshop on Advances and applications of problem frames*. 2006, pp. 81–86.

[50] *Structured assurance case metamodel (SACM), version 1.0, 2013*. URL: `www.omg.org/spec/SACM/`.

[51] Michael Vierhauser et al. "Interlocking safety cases for unmanned autonomous systems in shared airspaces". In: *IEEE transactions on software engineering* 47.5 (2019), pp. 899–918.

[52] Simon Diemert and Jeff Joyce. "Eliminative Argumentation for Arguing System Safety-A Practitioner's Experience". In: *2020 IEEE International Systems Conference (SysCon)*. IEEE. 2020, pp. 1–7.

[53] Yiheng Liu et al. "Understanding llms: A comprehensive overview from training to inference". In: *arXiv preprint arXiv:2401.02038* (2024).

[54] OpenAI. *GPT 4*. 2023. URL: `https://openai.com/research/gpt-4`.

[55] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[56] OpenAI. *Introducing GPT-4 Turbo*. `https://help.openai.com/en/articles/8555510-gpt-4-turbo`. 2023.

[57] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[58] Zhuosheng Zhang et al. "Automatic chain of thought prompting in large language models". In: *arXiv preprint arXiv:2210.03493* (2022).

[59] Takeshi Kojima et al. "Large language models are zero-shot reasoners". In: *Advances in neural information processing systems* 35 (2022), pp. 22199–22213.

[60] Jiaxin Ge et al. "Chain of Thought Prompt Tuning in Vision Language Models". In: *arXiv preprint arXiv:2304.07919* (2023).

[61] ML Solutions Architect at Arize AI Dat Ngo. *Mastering the OpenAI API: Tips and Tricks*. `https://arize.com/blog-course/mastering-openai-api-tips-and-tricks/`. 2023.

[62] William S Greenwell1 C Michael Holloway and John C Knight. "A Taxonomy of Fallacies in System Safety Arguments". In: (2005).

[63] Andres J Ramirez, Adam C Jensen, and Betty HC Cheng. "A taxonomy of uncertainty for dynamically adaptive systems". In: *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2012, pp. 99–108.

[64] Patrick J Graydon and C Michael Holloway. "An investigation of proposed techniques for quantifying confidence in assurance arguments". In: *Safety science* 92 (2017), pp. 53–65.

[65] Mazen Mohamad, Jan-Philipp Steghöfer, and Riccardo Scandariato. "Security assurance cases—state of the art of an emerging approach". In: *Empirical Software Engineering* 26.4 (2021), p. 70.

[66] Kimya Khakzad Shahandashti et al. "A PRISMA-driven systematic mapping study on system assurance weakeners". In: *arXiv preprint arXiv:2311.08328* (2023).

[67] Jianzhang Zhang et al. "A preliminary evaluation of chatgpt in requirements information retrieval". In: *arXiv preprint arXiv:2304.12562* (2023).

[68] Xianchang Luo et al. "Prcbert: Prompt learning for requirement classification using bert-based pretrained language models". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–13.

[69] Dipeeka Luitel, Shabnam Hassani, and Mehrdad Sabetzadeh. "Improving requirements completeness: Automated assistance through large language models". In: *Requirements Engineering* (2024), pp. 1–23.

[70] Boqi Chen et al. "On the Use of GPT-4 for Creating Goal Models: An Exploratory Study". In: *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE. 2023, pp. 262–271.

[71] Tsz-On Li et al. "Finding failure-inducing test cases with chatgpt". In: *arXiv preprint arXiv:2304.11686* (2023).

[72] Weisong Sun et al. "Automatic code summarization via chatgpt: How far are we?" In: *arXiv preprint arXiv:2305.12865* (2023).

[73] Zhuokui Xie et al. "ChatUniTest: a ChatGPT-based automated unit test generation tool". In: *arXiv preprint arXiv:2305.04764* (2023).

[74] Sungmin Kang, Gabin An, and Shin Yoo. "A preliminary evaluation of llm-based fault localization". In: *arXiv preprint arXiv:2308.05487* (2023).

[75] Yonghao Wu et al. "Large language models in fault localisation". In: *arXiv preprint arXiv:2308.15276* (2023).

[76] Sidong Feng and Chunyang Chen. "Prompting Is All You Need: Automated Android Bug Replay with Large Language Models". In: *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 2024, pp. 1–13.

[77] Toufique Ahmed et al. "Improving few-shot prompts with relevant static analysis products". In: *arXiv preprint arXiv:2304.06815* (2023).

[78] Mingyang Geng et al. "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning". In: *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 2024, pp. 1–13.

[79] Kua Chen et al. "Automated Domain Modeling with Large Language Models: A Comparative Study". In: *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2023, pp. 162–172.

[80] Xinyi Hou et al. "Large language models for software engineering: A systematic literature review". In: *arXiv preprint arXiv:2308.10620* (2023).

[81] Mohammad Mahdi Mohajer et al. "SkipAnalyzer: An Embodied Agent for Code Analysis with Large Language Models". In: *arXiv preprint arXiv:2310.18532* (2023).

[82] Martin Weyssow, Houari Sahraoui, and Eugene Syriani. "Recommending metamodel concepts during modeling activities with pre-trained language models". In: *Software and Systems Modeling* 21.3 (2022), pp. 1071–1089.

[83] Meriem Ben Chaaben, Lola Burgueño, and Houari Sahraoui. "Towards using few-shot prompt learning for automating model completion". In: *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE. 2023, pp. 7–12.

[84] Mithila Sivakumar et al. "GPT-4 and Safety Case Generation: An Exploratory Analysis". In: *arXiv preprint arXiv:2312.05696* (2023).

[85] Matthew J Page et al. "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews". In: *International journal of surgery* 88 (2021), p. 105906.

[86] Barbara Kitchenham, Lech Madeyski, and David Budgen. "SEGRESS: Software engineering guidelines for reporting secondary studies". In: *IEEE Transactions on Software Engineering* 49.3 (2022), pp. 1273–1298.

[87] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. "Guidelines for conducting systematic mapping studies in software engineering: An update". In: *Information and software technology* 64 (2015), pp. 1–18.

[88] *Scopus*. Available online at: `https://www.scopus.com`. URL: `https://www.scopus.com`.

[89] *Google Scholar*. Available online at: `https://scholar.google.com`. URL: `https://scholar.google.com`.

[90] *IEEE Xplore*. Available online at: `https://ieeexplore.ieee.org`. URL: `https://ieeexplore.ieee.org`.

[91] *ACM Digital Library*. Available online at: `https://dl.acm.org`. URL: `https://dl.acm.org`.

[92] *Engineering Village*. Available online at: `https://www.engineeringvillage.com`. URL: `https://www.engineeringvillage.com`.

[93] *Publish or Perish*. Available online at: `https://harzing.com/resources/publish-or-perish`. URL: `https://harzing.com/resources/publish-or-perish`.

[94] Clarivate. *EndNote*. Available online at: `https://endnote.com`. URL: `https://endnote.com`.

[95] Claes Wohlin. "Guidelines for snowballing in systematic literature studies and a replication in software engineering". In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014, pp. 1–10.

[96] *Connected Papers*. Available online at: `https://www.connectedpapers.com`. URL: `https://www.connectedpapers.com/`.

[97] Inc Notion Labs. *Notion*. Platform for comprehensive note-taking and project management. 2023. URL: `https://www.notion.so`.

[98] Martina K Linnenluecke, Mauricio Marrone, and Abhay K Singh. "Conducting systematic literature reviews and bibliometric analyses". In: *Australian Journal of Management* 45.2 (2020), pp. 175–194.

[99] Antonio V Silva Neto et al. "Safety Assurance of Artificial Intelligence-Based Systems: A Systematic Literature Review on the State of the Art and Guidelines for Future Work". In: *IEEE Access* (2022).

[100] Batista Dala Catumba et al. "Sustainability and challenges in hydrogen production: An advanced bibliometric analysis". In: *International Journal of Hydrogen Energy* 48.22 (2023), pp. 7975–7992.

[101] Fengxia Deng, Jizhou Jiang, and Ignasi Sirés. "State-of-the-art review and bibliometric analysis on electro-Fenton process". In: *Carbon Letters* 33.1 (2023), pp. 17–34.

[102] Sayantan Khanra, Amandeep Dhir, and Matti Mäntymäki. "Big data analytics and enterprises: a bibliometric synthesis of the literature". In: *Enterprise Information Systems* 14.6 (2020), pp. 737–768.

[103] *VOSviewer*. Available online at `https://www.vosviewer.com`. URL: `https://www.vosviewer.com`.

[104] *pandas: a powerful data analysis library for Python*. Available online at `https://pandas.pydata.org/`. 2023. URL: `https://pandas.pydata.org/`.

[105] *Matplotlib: Visualization with Python*. Available online at `https://matplotlib.org/`. 2023. URL: `https://matplotlib.org/`.

[106] Barbara Kitchenham. "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26.

[107] Kimya Khakzad Shahandashti et al. "Evaluating the Effectiveness of GPT-4 Turbo in Creating Defeaters for Assurance Cases". In: *arXiv preprint arXiv:2401.17991* (2024).

[108] Wenkai Yang et al. "Enabling Large Language Models to Learn from Rules". In: *arXiv preprint arXiv:2311.08883* (2023).

[109] Zhaocheng Zhu et al. "Large Language Models can Learn Rules". In: *arXiv preprint arXiv:2310.07064* (2023).

[110] Timothy R McIntosh et al. "A culturally sensitive test to evaluate nuanced gpt hallucination". In: *IEEE Transactions on Artificial Intelligence* 1.01 (2023), pp. 1–13.

[111] Laure Millet et al. "Assurance Case Arguments in the Large: The CERN LHC Machine Protection System". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2023, pp. 3–10.

[112] *LHC MPS Argument*. `https://cds.cern.ch/record/2854725`. 2023.

[113] OpenAI. *Reproducible Outputs*. `https://platform.openai.com/docs/guides/text-generation/reproducible-outputs`. 2023.

[114] OpenAI. *Reproducible Outputs OpenAI*. `https://cookbook.openai.com/examples/reproducible_outputs_with_the_seed_parameter`. 2023.

[115] Zihan Yu et al. "Towards better chain-of-thought prompting strategies: A survey". In: *arXiv preprint arXiv:2310.04959* (2023).

[116] Linxin Song et al. "The Impact of Advanced Prompting Strategies on the Natural Language Processing Capabilities of Large Language Models". In: *arXiv preprint arXiv:2309.15630* (2023).

[117] Nimisha Singla and Deepak Garg. "String matching algorithms and their applicability in various applications". In: *International journal of soft computing and engineering* 1.6 (2012), pp. 218–222.

[118] SeatGeek. *FuzzyWuzzy: Fuzzy String Matching in Python*. `https://github.com/seatgeek/fuzzywuzzy`.

[119] Kishore Papineni et al. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[120] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[121] Shuichiro Yamamoto and Yutaka Matsuno. "An evaluation of argument patterns to reduce pitfalls of applying assurance case". In: *2013 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*. IEEE. 2013, pp. 12–17.

[122] John Alexander McDermid, Yan Jia, and Ibrahim Habli. "Towards a framework for safety assurance of autonomous systems". In: *Artificial Intelligence Safety 2019*. CEUR Workshop Proceedings. 2019, pp. 1–7.

[123] Ewen Denney and Ganesh Pai. "Evidence arguments for using formal methods in software certification". In: *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2013, pp. 375–380.

[124] Ewen Denney, Ganesh Pai, and Iain Whiteside. "Formal foundations for hierarchical safety cases". In: *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE. 2015, pp. 52–59.

[125] Marsha Chechik et al. "Software assurance in an uncertain world". In: *Fundamental Approaches to Software Engineering: 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings 22*. Springer International Publishing. 2019, pp. 3–21.

[126] Yakoub Nemouchi et al. "Isabelle/SACM: Computer-assisted assurance cases with integrated formal methods". In: *Integrated Formal Methods: 15th International Conference, IFM 2019, Bergen, Norway, December 2–6, 2019, Proceedings 15*. Springer. 2019, pp. 379–398.

[127] Carmen Cârlan, Daniel Ratiu, and Bernhard Schätz. "On using results of code-level bounded model checking in assurance cases". In: *Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings 35*. Springer. 2016, pp. 30–42.

[128] Victor Bandur and John McDermid. "Informing assurance case review through a formal interpretation of GSN core logic". In: *Computer Safety, Reliability, and Security: SAFECOMP 2015 Workshops, ASSURE, DECSoS. ISSE, ReSA4CI, and SASSUR, Delft, The Netherlands, September 22, 2015, Proceedings 34*. Springer. 2015, pp. 3–14.

[129] Linling Sun, Nuno Silva, and Tim Kelly. "Rethinking of Strategy for Safety Argument Development". In: *Computer Safety, Reliability, and Security: SAFECOMP 2014 Workshops: ASCoMS, DECSoS, DEVVARTS, ISSE, ReSA4CI, SASSUR. Florence, Italy, September 8-9, 2014. Proceedings 33*. Springer. 2014, pp. 384–395.

[130] Toshinori Takai and Hiroyuki Kido. "A supplemental notation of gsn aiming for dealing with changes of assurance cases". In: *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE. 2014, pp. 461–466.

[131] Silviya Grigorova and TSE Maibaum. "Argument evaluation in the context of assurance case confidence modeling". In: *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE. 2014, pp. 485–490.

[132] Luis-Pedro Cobos, Alastair R Ruddle, and Giedre Sabaliauskaite. "Cybersecurity Assurance Challenges for Future Connected and Automated Vehicles". In: *Proceedings of the 31 st European Safety and Reliability Conference (ESREL 2021)*. 2021.

[133] Faiz Ul Muram and Muhammad Atif Javed. "ATTEST: Automating the review and update of assurance case arguments". In: *Journal of systems architecture* 134 (2023), p. 102781.

[134] Nungki Selviandro, Richard Hawkins, and Ibrahim Habli. "A visual notation for the representation of assurance cases using sacm". In: *Model-Based Safety and Assessment: 7th International Symposium, IMBSA 2020, Lisbon, Portugal, September 14–16, 2020, Proceedings 7*. Springer. 2020, pp. 3–18.

[135] Simon Foster et al. "Integration of formal proof into unified assurance cases with Isabelle/SACM". In: *Formal Aspects of Computing* 33.6 (2021), pp. 855–884.

[136] Object Management Group (OMG). *Structured Assurance Case Metamodel (SACM) Version 2.3 Beta 1*. `https://www.omg.org/spec/SACM/2.3/Beta1/About-SACM`. Accessed: [Your Date of Access]. 2022.

[137] Marsha Chechik et al. "Uncertainty, modeling and safety assurance: towards a unified framework". In: *Verified Software. Theories, Tools, and Experiments: 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13–14, 2019, Revised Selected Papers 11*. Springer. 2020, pp. 19–29.

[138] Yutaka Matsuno, Fuyuki Ishikawa, and Susumu Tokumoto. "Tackling uncertainty in safety assurance for machine learning: continuous argument engineering with attributed tests". In: *Computer Safety, Reliability, and Security: SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, September 10, 2019, Proceedings 38*. Springer. 2019, pp. 398–404.

[139] Carmen Cârlan, Tewodros A Beyene, and Harald Ruess. "Integrated formal methods for constructing assurance cases". In: *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2016, pp. 221–228.

[140] Logan Murphy et al. "Validating safety arguments with lean". In: *Software Engineering and Formal Methods: 19th International Conference, SEFM 2021, Virtual Event, December 6–10, 2021, Proceedings 19*. Springer. 2021, pp. 23–43.

[141] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[142] Xin Zhou et al. "A map of threats to validity of systematic literature reviews in software engineering". In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2016, pp. 153–160.