AUTOMATIC INSTANTIATION OF ASSURANCE CASES FROM PATTERNS USING LARGE LANGUAGE MODELS

OLUWAFEMI JOHN ODU

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

DECEMBER 2024

Abstract

Justifying the correct implementation of mission-critical systems' non-functional requirements (e.g., safety, and security) is crucial to prevent system failure. The latter could have severe consequences such as the death of people and financial losses. Assurance cases can be used to prevent system failure. They are structured sets of arguments supported by evidence, demonstrating that a system's non-functional requirements have been correctly implemented. Assurance case patterns serve as templates derived from previous successful assurance cases, aimed at facilitating the creation of new assurance cases. Despite the use of these patterns to generate assurance cases, their instantiation remains a largely manual and error-prone process that heavily relies on domain expertise. Thus, exploring techniques to support their automatic instantiation becomes crucial. To address this, our thesis explores the literature on assurance case patterns to understand recent advancements and trends characterizing that literature. Then we investigated the potential of Large Language Models (LLMs) in automating the generation of assurance cases that comply with specific assurance case patterns. Our findings suggest that LLMs can generate assurance cases that comply with the given patterns. However, this study also highlights that LLMs may struggle with understanding some nuances related to pattern-specific relationships. While LLMs exhibit potential in the automatic generation of assurance cases, their capabilities still fall short compared to human experts. Therefore, a semi-automatic approach to instantiating assurance cases may be more advisable at this time.

Acknowledgements

I am profoundly grateful to my supervisor, Dr. Alvine Boaye Belle, for her exceptional guidance, unwavering support, and expertise throughout my studies at York University.

I would like to extend my heartfelt appreciation to my co-supervisor Dr Song Wang, for his unwavering support and his invaluable suggestions. I am deeply grateful.

My sincere thanks to my defense committee chair, Dr. Marin Litoiu, for consistently providing invaluable feedback that significantly enhanced the quality of my thesis.

I would like to extend my gratitude to Dr. Mannar Jammal for graciously serving as the external member of my defense committee.

Finally, my deepest gratitude goes to my family, particularly Tolulope Odu and Jewel Odu, for their unwavering love, understanding, and encouragement. Their faith in me has been a relentless source of motivation, driving me to strive harder towards my goals.

This thesis is dedicated to the memory of my dad, Bamidele Odu, whose legacy continues to inspire me each day.

Table of Contents

List of Tables

List of Figures

Abbreviations

1. **ACP** - Assurance Case Pattern
2. **AC** - Assurance Case
3. **GSN** - Goal Structuring Notation
4. **CAE** - Claim Argument Evidence
5. **SACM** - Structured Assurance Case Metamodel
6. **LLM** - Large Language Model
7. **SE** - Software Engineering
8. **ML** - Machine Learning
9. **AI** - Artificial Intelligence
10. **PRISMA** - Preferred Reporting Items for Systematic Reviews and Meta-Analyses
11. **SEGRESS** - Software Engineering Guidelines for Reporting Secondary Studies
12. **AMLAS** - Assurance of Machine Learning for use in Autonomous Systems
13. **HARA** - Hazard Analysis and Risk Assessment
14. **ADS** - Autonomous Driving System

Chapter 1

Introduction

1.1 Context

Mission-critical systems are increasingly designed to be interoperable and interconnected. Their operational contexts usually change at runtime [8, 9, 10]. Hence, these systems usually operate under unpredictable conditions throughout their life cycle. Justifying and providing confidence in the essential non-functional requirements (e.g., security, safety) of these mission-critical systems is therefore crucial to prevent system failure. The latter could result in severe injuries, death of people, financial losses, and property destruction [11].

Assurance cases (ACs) are structured arguments that allow justifying that the non-functional requirements of a system have been correctly implemented [12]. Assurance cases are utilized across various domains (e.g., medicine [13, 14, 15], automotive [16, 17, 11], aerospace) to demonstrate the reliability of mission-critical systems and support their certification in compliance with industry standards (e.g., ISO 26262 [18], DO-178C [19]). In safety-critical domains, it is essential to ensure the system's safety when there is a change in its operational context [8]. Navigating these complexities and ensuring system reassurance with each change in the operating context can be time-consuming, tedious, and expensive [8, 20].

1.2 Motivation

Manually creating assurance cases can be time-consuming, especially for large, complex, and interconnected systems [21, 22]. For instance, Maksimov et al. [21] noted that an assurance case for an air traffic control system may consist of over 500 pages

and include references to 400 documents. This suggests that the manual creation and subsequent modifications of an initial assurance case draft can take several months [23]. To facilitate this creation process, practitioners use assurance case patterns (ACPs). These patterns are templates composed of evidence-based arguments derived from previous successful assurance cases. Practitioners instantiate assurance case patterns with system-specific information to create new assurance cases more efficiently. Several notations allow representing ACs and ACPs. These include the Goal Structuring Notation (GSN) [24] and the Claims-Arguments-Evidence (CAE) [12] notation.

Despite the use of assurance case patterns to create assurance cases, the instantiation of assurance cases usually remains tedious, error-prone, and time-consuming. This is primarily due to the heterogeneous nature of system artefacts and the complexity of mission-critical systems [4]. Thus, instantiating these patterns with system-specific information still requires domain expertise to efficiently extract the necessary system artefacts. Experts must then manually replace the abstract elements within these patterns with concrete values from the extracted artefacts to create an assurance case that complies with the given pattern(s).

Most of the existing approaches for instantiating assurance cases from patterns strongly depend on the model-based engineering approach that supports the extraction of information from system models (e.g., model-based design [4, 25, 26, 27, 28]). However, a strong dependence on model-based engineering approach can limit the application of assurance case patterns in creating assurance cases for systems that do not conform to this design approach. This highlights the need for new techniques to automatically instantiate assurance cases from patterns for any given system, irrespective of its design methodology.

The rapid adoption of generative AI technologies like OpenAI's GPT series has fostered the automatic generation of content and spurred their increasing use in the automation of several software engineering tasks [29]. To capitalize on this momentum, we propose a novel approach that utilizes Large Language Models (LLMs) to automatically instantiate assurance cases complying with a specified assurance case pattern(s).

1.3 Research Objectives

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

RQ1: What are the trends characterizing the research in assurance case patterns? This question aims to explore the research evolution, research structure, and significance of assurance case patterns over time. In this regard, we relied on various bibliometric tools to identify and visualize the most cited keywords and publications to unravel the factors for the adoption of assurance case patterns within the last two decades. We also identified the future directions and challenges in the field of assurance case patterns.

RQ2: How effective are Large Language Models in the automatic instantiation of assurance cases from assurance case patterns? In this research question, we assessed the effectiveness of large language models in instantiating assurance cases complying with a given pattern. We assessed the performance of LLMs by evaluating the accuracy, correctness, and quality of the generated assurance case against the ground-truth assurance case derived from the manually created assurance case pattern.

Also, we compared the performance of the two utilized LLMs (i.e. GPT-4 Turbo vs GPT-4o) under various experimental setups respectively leveraging various categories of software engineering knowledge: domain information, context information, predicate rules, and K-shot examples.

RQ3: Can we rely on an adaptation of AMLAS for the manual creation of the safety case of an open-source ML-enabled ADS? To address this research question, we adopted the AMLAS-based design approach we proposed in [30] for creating a safety case for an ML-enabled system that is already developed and operational. We applied this approach to develop a safety case for an ML-enabled component within Baidu Apollo, an open-source autonomous driving system. Our focus is specifically on the trajectory prediction component of Baidu Apollo, for which we developed a safety case to provide a set of structured arguments, supported by evidence, that the component meets safety requirements and effectively mitigates potential risks associated with trajectory prediction.

RQ4: Can an LLM be more performant than human experts when creating the safety case of an open-source ML-enabled ADS? In this research question, we leveraged GPT-4o for the automatic creation of an assurance case for an autonomous driving system, complying with specific assurance case patterns.

1.4 Contributions

The contributions of our work are as follows:

1. **Contribution 1:** We have conducted a bibliometric analysis on assurance case patterns spanning two decades and reported the evolutionary trends, and research structure in the field of assurance case patterns.

2. **Contribution 2:** We submitted a paper to JSS (Journal of Systems and Software). That paper proposes a novel method for formalizing assurance case patterns into predicate-based rules complying with GSN. This allows for capturing the internal structure of assurance case patterns more generically and uniformly.

3. **Contribution 3:** We conducted nine distinct experiments with two popular and very recent LLMs (i.e. GPT-4 Turbo and GPT-4o) to explore their ability to automatically generate assurance cases.

4. **Contribution 4:** We created a complete safety case for the ML-enabled trajectory prediction component of the Baidu Apollo autonomous driving system.

5. **Contribution 5:** We submitted a paper to CAIN (International Conference on AI Engineering). That paper focuses on evaluating GPT-4o's performance in the automatic creation of a safety case for the ML-enabled trajectory prediction component of the Baidu Apollo autonomous driving system.

6. **Contribution 6:** Co-authored three papers related to assurance case research:

    - *"A PRISMA-driven Systematic Mapping Study on System Assurance Weakeners"* published in IST (Information and Software Technology) Journal.
    - *"Using GPT-4 Turbo to Automatically Identify Defeaters in Assurance Cases"* published in AIRE (International Workshop on Artificial Intelligence and Requirements Engineering).
    - *"Design of the Safety Case of the Reinforcement Learning-enabled Component of a Quanser Autonomous Vehicle"* published in AIRE.

1.5 Thesis Structure

Chapter 2 presents key background concepts. Chapter 3 reviews related work. Chapter 4 provides details about the methodology we adopted in our thesis. Chapter 5 presents our findings. Chapter 6 discusses our findings. Chapter 7 reports threats to validity. Chapter 8 outlines future research directions and concludes the thesis.

Chapter 2

Background

2.1 System Assurance and Assurance Cases

The producers of mission-critical systems designed to perform essential tasks must usually demonstrate that their systems effectively support their intended non-functional properties (e.g., safety, and security). This allows certifying these systems comply with specific industrial standards and allows regulators to authorize the deployment of these systems [31]. System assurance supports that demonstration by relying on assurance cases [31]. System assurance management activities include: the construction of assurance cases, their assessment, their update, their formalization, and their formal verification [21, 32, 33].

An assurance case is a well-established, structured, reasoned, and auditable set of arguments designed to support a specific goal [34]. These arguments are often supported by evidence and aim at demonstrating that a system meets desirable non-functional requirements (e.g., safety, security). There are several types of assurance cases, each focusing on a specific non-functional requirement: safety cases [5, 15], security cases [6, 35], reliability cases [36], etc.

Assurance cases are utilized to prevent system failure in various domains, including medicine [13, 14, 15] and automotive domain [11, 16, 17]. They are also used to ensure the reliability of mission-critical systems and facilitate certification in line with industry standards (e.g., ISO 26262, DO-178C). Regulatory bodies like the Food and Drug Administration (FDA) advocate for the use of assurance cases to bolster the safety confidence of medical devices during their approval process [35].

An assurance case comprises three primary components [37, 38]: (1) a top claim (root claim) which is usually subdivided into sub-claims. This top claim serves as the fundamental statement indicating that the system fulfills a specific requirement. (2)

body of evidence supporting both the sub-claims and the root claim. (3) a collection of structured arguments that establish connections between the evidence and the sub-claims, linking all sub-claims to the top claim of the assurance case [39].

2.1.1 How to Represent AC?

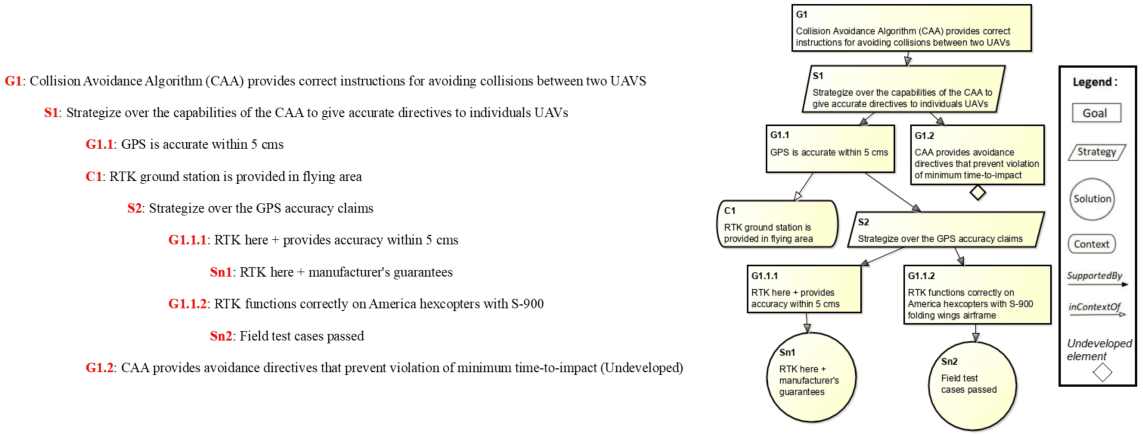

Figure 2.1: On the right, an example of a partial safety case (GSN diagram) adapted from Vierhauser et al. [1]; on the left, the equivalent of the safety case in structured prose

Several notations allow for representing assurance cases and can be broadly categorized into textual and graphical notations. Holloway [40] described five text-based notations for representing assurance cases, including normal (i.e. unstructured) prose representation and structured prose representation. The latter is a notation that introduces a structured format to address the typical verbosity, lack of structure, and ambiguity characterizing the normal prose representations of assurance cases [41, 42]. Kelly et al. [41] as well as Selviandro et al. [42] opined that unstructured textual notations for representing assurance cases often lack clarity, can be ambiguous, and may suffer from poor structure. Hence, the use of structured prose is a good alternative [40, 43].

Graphical notations also address the limitations of unstructured textual representations by improving the clarity and structure of assurance cases. Graphical notations include GSN (Goal structuring Notation) [24], CAE (Claim-Argument-Evidence) [12], and Eliminative Argumentation (EA) [44]. The Object Management Group (OMG) recently introduced SACM (Structured Assurance Case Metamodel) [37] to promote interoperability and standardization [9]. SACM aligns with existing assurance case

notations (e.g., GSN, CAE). Still, GSN is the most popular notation [9]. GSN supports the representation of an assurance case as a *goal structure*. The latter is a GSN diagram depicted as a tree-like structure. The GSN standard [24] proposes the following six main GSN elements to represent assurance cases:

- **A Goal** is depicted as a rectangle and represents the main claim or a sub-claim. Examples are G1 through G4 in Figure 2.2.
- **A Strategy** is depicted as a parallelogram and describes the inference between a goal and its sub-goals. See S1 in Figure 2.2.
- **A Solution** is depicted as a circle and represents the evidence supporting an argument or goal. See Sn1 and Sn2 in Figure 2.1.
- **A Context** is rendered as a rounded rectangle, presenting a contextual artefact. This can be a reference to contextual information, or a statement. See C1 in Figure 2.2.
- **An Assumption** is rendered as an ellipse with the letter 'A' at the top or the bottom right denoting an intentionally unsubstantiated statement.
- **A Justification** is rendered as an ellipse with the letter 'J' at the top or the bottom right and presents a statement of rationale for the inclusion or wording of a GSN element.

Assurance cases' claims, evidence, and arguments respectively map to GSN goals, solutions, and strategies [38]. It is possible to decorate GSN elements using the *Undeveloped* decorator. The latter allows indicating a GSN element has not been developed yet [24]. It is depicted as a hollow diamond applied to the bottom center of an element [24]. Also, the GSN standard [24] defines two main relationships between GSN elements: *SupportedBy* and *InContextOf*. *SupportedBy* is depicted as a line with a solid arrowhead and represents supporting relationships between GSN elements. *InContextOf* is depicted as a line with a hollow arrowhead and represents a contextual relationship between GSN elements.

The GSN standard [24] proposes guidelines to convert an assurance case represented in the textual format (e.g., structured prose) into a GSN diagram. Figure 2.1 shows an excerpt of an assurance case adapted from [1]. This excerpt is represented in the GSN and in the structured prose format.

2.2 Assurance Case Pattern

Similar to design patterns used in Software Engineering (SE), assurance case patterns are templates formed from common repeated structures and previous successful assurance cases [8]. These assurance case patterns contain placeholders filled with generic information that can be replaced with system-specific information during their instantiation [4, 8]. The application of assurance case patterns fosters the reuse and eases the creation of assurance cases. There are various types of assurance case patterns based on the non-functional requirements they target. Assurance case patterns are also used to mitigate assurance deficits [45, 46, 47]. Assurance deficits refer to *"any knowledge gap that prohibits perfect confidence"* in an assurance case [48].

2.2.1 How to Represent Assurance Case Patterns?

To represent an assurance case pattern, GSN has been extended to support several modeling elements such as multiplicity, optionality, and abstraction [49]. Hence, in addition to the elements we described in subsection 2.1.1, GSN and some reference literature on GSN patterns (e.g., [50, 51, 52]) also propose the following additional decorators to help represent assurance case patterns:

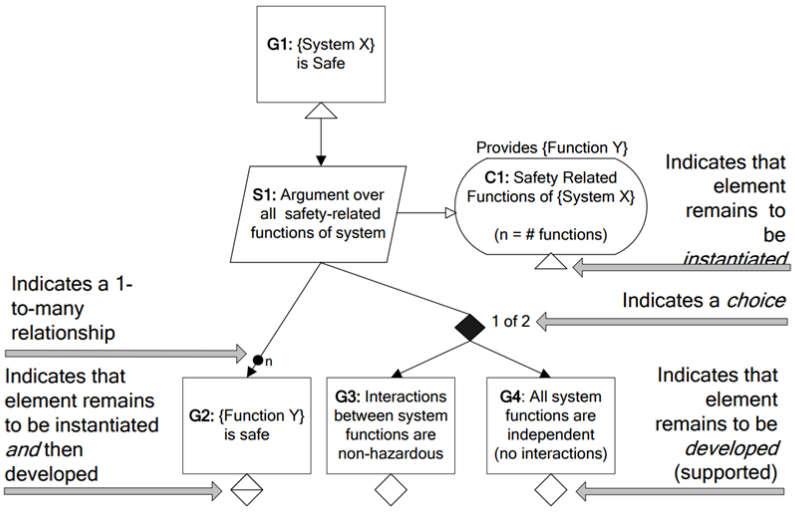

Figure 2.2: A Sample Assurance Case Pattern Adapted from Alexander et al. [2]

- **Uninstantiated** - This decorator is depicted with a small triangle applied to the bottom center of an element. It allows indicating that a GSN element is yet to be instantiated, i.e., an abstract element in a placeholder needs to be replaced by a concrete instance [24].
- **Undeveloped and Uninstantiated** - Both the *undeveloped* and *uninstantiated* decorators are overlaid to form this decorator. It denotes that a GSN element requires both further development and instantiation.
- **Parameterized expressions within Placeholders** - Parameterized expressions are abstract expressions inside placeholders that need to be replaced with concrete information [50, 51, 52].
- **Multiplicity** - Multiplicity symbols allow describing how many instances of one element type relate to another element. These symbols are generalized n-ary relationships between GSN element [24].
- **Optionality** - It represents optional and alternative relationships between GSN elements which generalizes n-of-m choices between GSN elements [24].
- **Choice**. This decorator is depicted as a solid diamond. The choice decorator denotes possible alternatives in satisfying a relationship [24].

Figure 2.2 shows a sample safety case pattern adapted from [2] and represented using GSN.

2.3 What is a Large Language Model?

Large Language Models (LLMs) are advanced artificial intelligence systems with computational ability to generate human language, with at least the appearance of understanding it [53, 54]. These models are complex neural network structures with massive parameter sizes and trained on vast amounts of data from diverse sources [54]. LLMs can generate new content, answer questions, and capture inherent rules of a domain [53, 55, 56]. These capabilities of LLMs have ensured their wide application across various fields including automated software engineering. Some examples of LLMs include the GPT series by OpenAI [29], BERT [57], T5 [58], and ERNIE [59].

Prompt engineering refers to the different techniques used to provide instructions and guidelines to an LLM to ensure a desired generated response [60]. It improves the efficiency and quality of LLMs responses. The most popular prompting techniques

include the Chain-of-Thought (CoT) prompting technique [61], Zero-shot prompting, [62], and Few-shot prompting [63]. CoT utilizes a series of intermediate reasoning steps to significantly improve the ability of LLMs to perform complex reasoning tasks [61]. Zero-shot prompting [62] — a technique whereby we prompt an LLM without any examples, attempting to take advantage of the reasoning patterns it has extracted. Few-shot prompting [63] is a technique whereby we prompt an LLM with several concrete examples of task performance so that the model can learn from these examples.

Rule distillation is a technique enabling LLMs to learn and acquire knowledge from rules or instructions. It involves extracting knowledge from defined textual rules and then explicitly encoding this knowledge into LLM parameters. This ensures that LLMs can effectively comprehend and apply the distilled rules [55, 64].

Chapter 3

Related Work

3.1 Assurance Case Patterns

In the wide and growing field of system assurance using assurance case patterns, a plethora of secondary studies have also been conducted to understand the current research and growth of the field. For instance, Szczygielska and Jarzębowicz [65] presented an online unified assurance case patterns catalog containing 45 patterns extracted from several sources. Their objective was to aid the quick creation of assurance cases in supporting tools like NOR-STA. Preschern et al. [66] surveyed the various methods to build or improve a system architecture using safety patterns. They provided an overview of 12 existing pattern-based safety development methods extracted from the literature. They analyzed and compared these methods based on the domain and pattern targeted by these methods, the approach to apply the patterns, the safety standard followed, and the method maturity. Their analysis offers insights into the pros and cons of each pattern-based safety development method.

Gleirscher and Kugele [67] conducted a comprehensive survey examining safety design and argument patterns employed in the construction and assurance of safety-critical systems spanning multiple domains. Their study involved addressing twelve survey questions, establishing the correlation between design and argumentation concepts crucial for ensuring system safety. The authors also identified challenges related to the efficient reuse of safety mechanisms within systems and proposed research directions aimed at resolving these challenges.

Driven by the widespread adoption of Model-Based Engineering (MBE) in crafting model-based assurance cases, Yan et al. [68] performed a survey to assess techniques for generating safety cases using MBE. They identified a crucial research gap characterized by the lack of automated processing of raw and unstructured instantiable data. To

address this gap, the authors suggested the implementation of a System Assurance Case Metamodel (SACM) compliant framework for generating assurance cases.

Our analysis of existing literature on assurance case patterns reveals that no previous study has analyzed the state of the art in ACP from a bibliometric analysis perspective. This hampers the identification of the emerging trends, research gaps, and potential research directions in the field of assurance case patterns. To address this gap, our thesis presents a bibliometric analysis as an initial step, evaluating the scientific landscape of ACPs and proposing future research directions.

3.2 Formalization of ACs and ACPs

Denney and Pai [69] proposed a formal definition of an assurance case pattern as a tuple characterized by a directed hypergraph with specific labeling functions to enhance pattern utilization. Matsuno [50] introduced an assurance case language based on GSN, validated through the D-Case Editor tool, which supports GSN patterns and modules. Expanding on the concept of assurance case languages, Beyene and Carlan [70] developed *'CyberGSN'*, integrating informal GSN elements with formal Cyberlogic to facilitate safety case creation and maintenance. Murugesan et al. [71] focused on semantic correctness and logical consistency in assurance cases. They developed a framework that converts assurance cases into Prolog predicates and utilizes Constraint Answer Set Programming (CASP) to ensure consistency and completeness of the arguments and evidence in assurance cases.

To assess the benefits associated with the formalization of assurance arguments about a system property, Graydon [72] conducted a survey of 20 studies focusing on proposed formal assurance arguments. Their result revealed that majority of these studies speculate on the advantages of formalism without presenting concrete proof to substantiate these presumed advantages.

Our work is similar to Shahandashti et al. [73], in which the authors extracted predicates from the structural rules embedded in the assurance case notation called EA. They then used these predicates to create predicate-based rules that they incorporated into GPT-4 Turbo prompts to investigate the effectiveness of that LLM in identifying defeaters within assurance cases represented in EA notation. Defeaters are a form of assurance deficits. Defeaters refer to *"arguments that can undermine the effectiveness of assurance cases by compromising the reliability and adequacy of these assurance cases in verifying a system's capabilities such as safety and security"* [73]. In contrast, our work introduces a novel pattern formalization method that uses formal predicates to capture the internal structure and relationships among elements within an assurance case pattern presented in GSN. By leveraging these formal predicates, we enable

LLMs including GPT-4 Turbo and GPT-4o, to automatically generate assurance cases that adhere to the specified formalized assurance case pattern(s).

Our approach focuses on GSN and utilizes a formalized pattern to systematically generate assurance cases. This key distinction sets our work apart from previous research, which did not use formalized assurance case patterns for the creation of assurance cases and did not assess the performance of various LLMs in generating these assurance cases.

3.3 Manual Creation of Assurance Cases

Creating assurance cases manually often involves a meticulous, step-by-step process. This process includes systematically assembling evidence and arguments. It also involves conducting hazard analysis and risk assessment to demonstrate that a system complies with safety, security, and reliability standards. Over the years, earlier studies predominantly focused on and explored the manual creation of assurance cases. For instance, Nguyen and Ellis [23] manually created an assurance case from scratch for a spacecraft safing system. They then generalized this assurance case by incorporating lessons learned from prior safing and spacecraft failures, to develop a domain-specific pattern. This pattern was subsequently applied in the creation of a second assurance case for another spacecraft safing system. Their work provides detailed insights into the use of assurance cases to mitigate risk, as well as the lessons learned in creating both the assurance cases and the assurance case pattern for a spacecraft safing system.

With the evolution of technology and the widespread integration of artificial intelligence and machine learning components in safety critical systems, new methodologies have begun to emerge to address the unique challenges presented by these advanced systems. Hawkins et al. [74] introduced a methodology known as AMLAS (Assurance of Machine Learning for use in Autonomous Systems). This methodology has two main objectives: (1) to systematically integrate safety assurance into the development life-cycle of ML components, concurrently with the development process of these components, and (2) to generate the evidence necessary to provide confidence in the safety of these components when integrated into an autonomous system. The AMLAS [74] framework comprises six iterative stages, each with argument patterns that can be instantiated to develop a safety case for the ML-enabled components in an autonomous system.

Recently, the focus has shifted towards manual assurance case creation for machine learning-based components in cyber-physical systems, such as autonomous vehicles. Borg et al. [75] undertook a significant industry-academia collaboration to manually create a comprehensive safety case for an ML-based component within a simulated

automotive environment. Their work focused on the creation of a safety case for SMIRK - a pedestrian automatic emergency braking (PAEB) system, by using the Assurance of Machine Learning for use in Autonomous Systems (AMLAS) framework. They followed a detailed process involving simulation-based evaluations, thorough documentation, hazard analysis, risk assessment, validation and verification activities, and iterative refinement to meticulously create a safety case that incorporates functional safety principles outlined in standards such as ISO 26262 and ISO 21448 SOTIF.

Motivated by the challenge of accessing complete safety cases for ML-enabled autonomous systems, Sivakumar et al. [30] proposed an approach that adapts and enhances the AMLAS [74] design methodology, aiming to simplify the assurance case creation process for systems that are already developed and operational. Their approach was structured around three phases: Hazard Analysis and Risk Assessment (HARA), the application of selected AMLAS stages, and a Change Impact Analysis to maintain safety case validity during system modifications. They applied their methodology in the manual creation of a complete safety case for a reinforcement learning algorithm utilized by the Quanser Qcar [76] to avoid collisions at an unsignalized 4-way intersection.

3.4 Automatic Instantiation of ACPs

Several approaches support the automatic instantiation of assurance case patterns (e.g., [77, 78, 79, 80]). For instance, some approaches (e.g., [77, 78]) used a weaving method with the model-based engineering approach for instantiating assurance case patterns. This method weaves assurance case patterns with system models, facilitating the extraction of system-specific information or artefacts from system models and mapping these artefacts to placeholders in the ACP to generate an assurance case.

Other approaches (e.g., [79, 80]) utilized reference tables to keep track of system artefacts, requirements, and the mapping of these artefacts to placeholders in the ACP. However, these approaches strongly depend on specific engineering methodologies that focus on extracting information from system models (e.g., model-based design [4, 25, 26, 27, 28]).

Additionally, the use and maintenance of reference tables can be complex and challenging, especially for large systems with numerous requirements and artefacts as evidence. This complexity can lead to the omission of important artefacts or evidence if the reference table is not well maintained.

Therefore, it is crucial to devise new techniques to automatically instantiate assurance case patterns and create assurance cases for systems regardless of their

design methodology.

3.5 Use of Assurance Cases in the Automotive Domain

Several studies in the literature have demonstrated the application of assurance cases to certify safety and other non-functional requirements in the automotive domain. Driven by the need to ensure the safety of increasingly complex modern vehicles, Wagner et al. [17] investigated the construction of safety cases for automotive systems, focusing specifically on the cruise control component. Their findings indicate that the use of safety case patterns, modules, and automotive models—such as functional models, platform models, and environment models— guides the construction of safety arguments. This approach aids in developing more consistent and comprehensive safety cases.

Recently, with the emergence of Artificial Intelligence (AI) and Machine Learning (ML)-based components being utilized to adjust system behaviors based on the surrounding environment, autonomous driving systems have garnered significant interest from both industry and academia [81].

Nearchou et al. [81] introduced a novel paradigm that integrates an assurance case into the software development life cycle of cyber-physical systems (CPS) to ensure safe, self-adaptive behaviors at runtime. Their approach was demonstrated through a case study involving the F1TENTH racing car. They developed an initial assurance case template during the system's design phase and integrated this template with the Monitor-Analyze-Plan-Execute under a Knowledge base (MAPE-K) control loop—a widely used framework for designing self-adaptive CPS. The template is linked to four MAPE-K assurance manager classes that update the assurance case at runtime based on changes in the environment, thereby guiding the runtime operation of the autonomous vehicle.

Safety-critical systems like autonomous vehicles are designed to be interoperable and interconnected, leading to changes in their operating conditions at runtime [8, 9]. Hence, operating conditions play a crucial role in the safety assurance of these systems. Weiss et al. [82] provided a different approach for arguing the safe operation of AI-based automated driving systems (ADS), focusing on the operating conditions known as the Operational Design Domain (ODD). They presented a safety case with eight heterogeneous sets of evidence to argue for the sufficient completeness and consistency of the ODD definition during the development of safety-critical AI-based ADS functionalities. They evaluated their approach on a driverless train in the

railway domain with a fixed route, demonstrating the feasibility of their approach in an industrial use case.

Burton et al. [83] demonstrated a safety assurance approach for machine learning-based road surface classification across various operational scenarios for chassis control functions. They illustrated this approach by creating an assurance case for a Tire Noise Recognition (TNR) system, which is essential for the vehicle's chassis control in determining, in real-time, whether the road is dry.

3.6 LLM for Software Modeling

LLMs are currently being utilized for a variety of downstream software engineering tasks, including software defect prediction [84], static code analysis [85], automated program repair [86], code generation [87], and software modeling [88, 89].

In the field of software modeling, Chen et al. [88] investigated the use of LLMs, specifically GPT-3.5 and GPT-4, to fully automate domain modeling. They concluded that including examples in prompts significantly improves the performance of LLMs. Also, Chen et al. [89] applied GPT-4 in goal-oriented modeling, focusing on its use with the Goal-oriented Requirement Language (GRL). Their results showed that GPT-4 can generate basic goal models, though its outputs often require manual domain-specific adjustments and validation. Chaaben et al. [90] utilized few-shot prompt learning to ease the completion of domain diagrams (eg., UML class and activity diagrams) without requiring extensive training data. They used semantic mappings to convert modeling formalisms into meaningful patterns of tokens that LLMs can understand to improve and complete modeling activities.

Sivakumar et al. [22] conducted an evaluation of GPT-4's proficiency in understanding and generating GSN elements and safety cases. They extracted intricate structural and syntactic rules from the GSN standard to formulate 19 evaluative questions divided into rule-based and generation-based questions. They assessed GPT-4's comprehension of these rules and its capability to generate GSN elements. Additionally, using both contextual and domain information, they performed experiments to evaluate GPT-4's ability in producing safety cases that are structurally, semantically, and reasonably accurate.

Shahandashti et al. [91] analyzed EA reference documents to extract both the structural and semantic rules that EA embodies. These rules served as the foundation for crafting the EA-based questions they used to evaluate GPT-4 Turbo's proficiency in understanding EA as well as its ability to generate EA elements such as defeaters. Unlike us, both Sivakumar et al. [22] and Khakzad Shahandashti et al. [91] did not

rely on formalized assurance case patterns to guide the generation of assurance cases and did not compare the performance of various LLMs in generating assurance cases.

Chapter 4

Methodology

4.1 Methods Used to Answer RQ1

To conduct our bibliometric analysis, we relied on SEGRESS (Software Engineering Guidelines for Reporting Secondary Studies) [92]. SEGRESS is an adaptation of PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) 2020 [93] for the software engineering field. We also relied on common bibliometric analysis techniques [94]. Figure 4.1 presents a high-level overview of our methodology. We further describe the steps of our methodology in the remainder of this section.

4.1.1 Information Sources

Database-driven Search

Our search for primary studies relating to assurance case patterns was conducted on five well-known scientific search engines; IEEE Xplore [95], Scopus [96], ACM Digital Library [97], Engineering Village [98], and Google Scholar [99]. To automatically search Google Scholar, we used the publish or perish tool [100].

Snowballing Technique

To ensure high coverage of primary studies related to assurance case patterns during our search process, we also performed snowballing [101]. We utilized the Connected Papers tool [102] to automatically perform the forward and backward snowballing technique. We used the primary studies found during our database-driven search as the start set for the snowballing process.

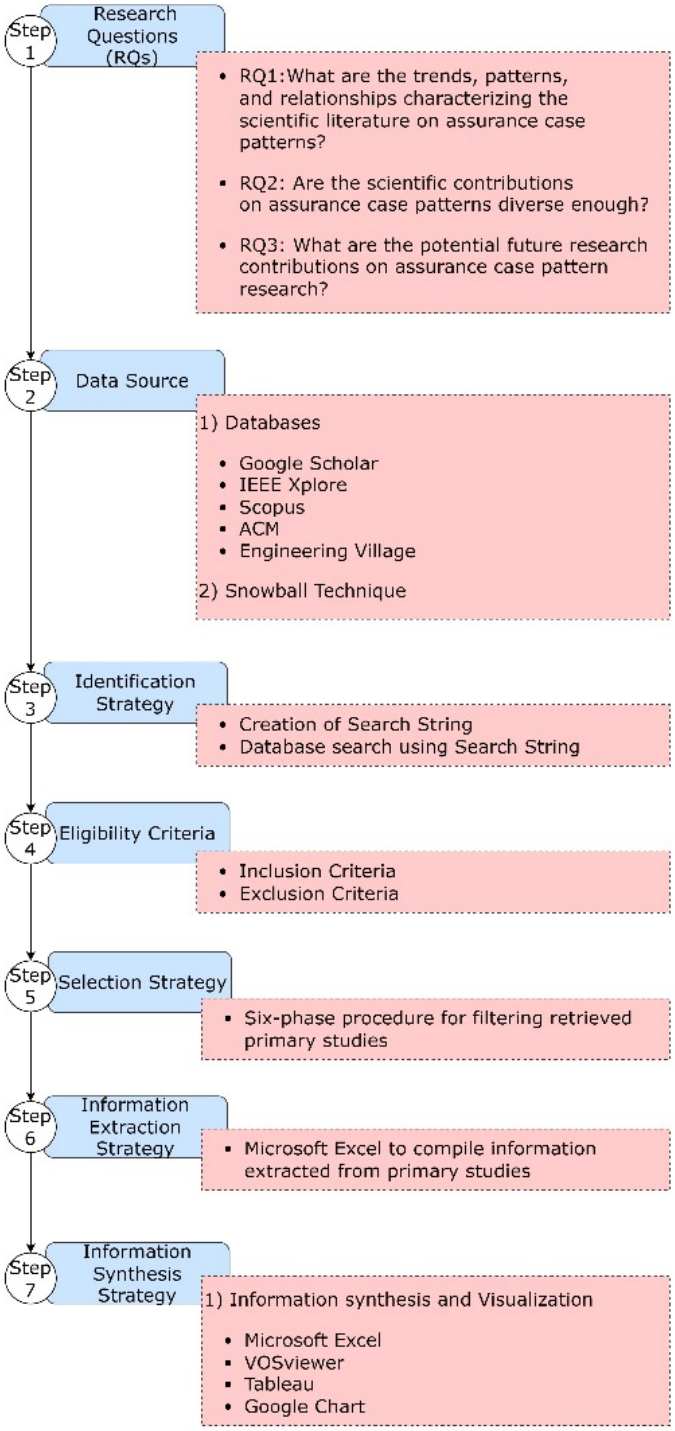

Figure 4.1: High-level overview of our Bibliometric Analysis methodology

4.1.2 Identification Strategies

To ensure the retrieval of all relevant studies, we first conducted a manual search for papers related to assurance case patterns to become acquainted with the keywords commonly used by researchers in this field. Using the insights gained from this initial search, we formulated the search strings provided in the text box below, and applied them across the five scientific databases.

1: *"safety case pattern" OR "security case pattern" OR "assurance case pattern" OR "argument pattern"*

2: *("safety case" OR "security case" OR "assurance case") AND ("pattern")*

The first part of the search string is aimed at increasing the breadth and extending the area of the search to include all occurrences of safety case pattern, security case pattern, assurance case pattern, and argument pattern. The second part of the search string is aimed at limiting the search string to patterns for common functional requirements. These search strings were used in the advanced search field of the five different databases.

4.1.3 Eligibility Criteria

Table 4.1 reports the inclusion and exclusion criteria we utilized to select primary studies. We carefully formed these criteria when designing our review protocol. This made it easy to filter the retrieved primary studies. For the inclusion criteria, we considered studies focusing on the representation, instantiation, creation, assessment, reuse, and formalization of assurance case patterns. We also considered studies that are peer-reviewed and studies written in English as this is the common language among authors in the literature. To limit outdated studies and maintain relevancy, we only considered studies published within the last two decades (January 2003 to October 2023). Also, we did not consider short papers (less than four pages) as well as non peer-reviewed papers. We made this choice to ensure that the studies utilized for our bibliometric analysis were of great quality.

4.1.4 Selection Strategy

Database-driven

Table 4.2 shows the strategy we utilized to narrow down the primary studies for our bibliometric analysis. After we completed our database-driven search, we proceeded with a six-phase approach to filter out unrelated studies from our retrieved primary

Table 4.1: Inclusion and Exclusion Criteria

Inclusion criteria	Exclusion criteria
1. Conference papers, Workshop papers, Journal papers, Peer reviewed papers	1. Books, posters, book chapters, grey literature (e.g., Computing Research Repository (CoRR), ArXiv, SSRN), theses, tutorials, technical reports, papers with less than 4 pages
2. The study is written in English, the study has been published in the last 20 years (between 2003 and October 2023)	2. Studies not in the English language
3. Papers discussing how assurance case patterns are represented.	3. Papers that do not propose techniques to design, model, represent, or generate assurance cases using patterns
4. Papers describing how patterns are used for assurance case creation.	4. Incomplete papers (only abstract is available)
5. Papers applying a case study in the use of assurance case generated from an assurance case pattern.	5. Non-context assurance, secondary studies (e.g., reviews, systematic literature reviews, systematic mapping studies, surveys)
6. Papers focusing on assessment, reuse, formalization, automation, maintenance, and instantiation of assurance case patterns.	6. Papers with paywall preventing free access to the paper and emailing the authors of the paper to get it was unfruitful
7. Papers focusing on patterns used in assurance cases	7. Papers with assurance case patterns, but no description of how the patterns were utilized

studies. In the first phase, we imported all retrieved studies from the five different databases into a reference manager tool called EndNote [103]. In the second phase, we filtered out studies that were not peer-reviewed and studies without titles. The third phase involved using the *"find duplicates"* feature in Endnote to automatically remove duplicate studies from the different database results. We filtered out studies based on titles, keywords, abstracts, venues, inclusion, and exclusion criteria in the fourth phase. Also, in the fifth phase, we filtered out studies by reading the introduction, and conclusion of the studies and using the inclusion and exclusion criteria. Finally, in the last phase, we eliminated unrelated studies based on full-text reading, and our

inclusion and exclusion criteria.

Snowballing

When completing the snowballing process, we also followed the same six-phase approach listed in Table 4.2 for filtering out primary studies with a slight modification in the duplicate removal step. In the first snowballing iteration, we used the connected paper tool [102] to generate a graph of related papers for each primary study found in the database-driven search. This is then followed by the six-phase approach listed in table 4.2.

During phase three of the first snowballing iteration, we removed all duplicate references in this phase and also compared them with references from phase one of the database-driven search to identify and further remove any duplicate studies. In the subsequent snowballing iteration, we used the result of the last filtering phase (phase six) in the previous snowballing iteration as a start set and followed the six-phase approach listed in table 4.2 for selecting primary studies.

During phase three of the subsequent snowballing iteration, we removed all duplicate references in this phase and also compared them with references from phase one of the database-driven search and the previous snowballing iterations phase one to identify and further remove any duplicate studies.

A sole researcher executed the phases listed in Table 4.2 for primary study selection. To minimize potential bias, a second researcher independently and randomly sampled studies in each phase for validation. Regular meetings were held to address and resolve any disagreements between the two researchers.

4.1.5 Data Extraction Strategies

In our bibliometric analysis, we used Excel sheets to compile data from the retrieved primary studies, including details such as publication year, authors, titles, venue, and the number of citations. The citation count for each primary study was manually obtained from Google Scholar. We extracted the primary studies in the EndNote library in RIS (Research Information Systems) format and used them as input to VosViewer [104] for data synthesis.

4.1.6 Synthesis Strategies

In our bibliometric analysis, we employed the widely used VosViewer tool [104] to synthesize data extracted from primary studies. We used VosViewer to facilitate the analysis, synthesis, and visualization of trends and patterns in published research

Table 4.2: Primary Studies Selection Strategy

Phase	Action Performed
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

works on assurance case patterns. For the generation of additional charts, we utilized Microsoft Excel, Google Charts [105], and Tableau [106].

4.2 Methods Used to Answer RQ2

The novelty of our approach in RQ2 lies in the use of LLMs to guide the automatic generation of assurance cases from formalized assurance case patterns. This ensures that the LLMs at hand leverage recurring argumentation structures (i.e. assurance case patterns) to guide the generation of assurance cases. Figure 4.2 shows a high-level overview of our approach. That approach consists of four phases that we describe below.

4.2.1 Phase I: Formalization of ACs and ACPs

LLMs can generate and comprehend textual information. However, assurance cases represented in graphical notations such as GSN have inherent structural rules that define the properties and relationships among their elements. These notations for representing either an assurance case or assurance case pattern are usually depicted in a hierarchical, tree-like structure. In this structure, the top goal is considered the root goal, sub-goals are likened to sub-trees, and the solutions can be likened to the leaves of the tree. Based on this observation and inspired by the foundational

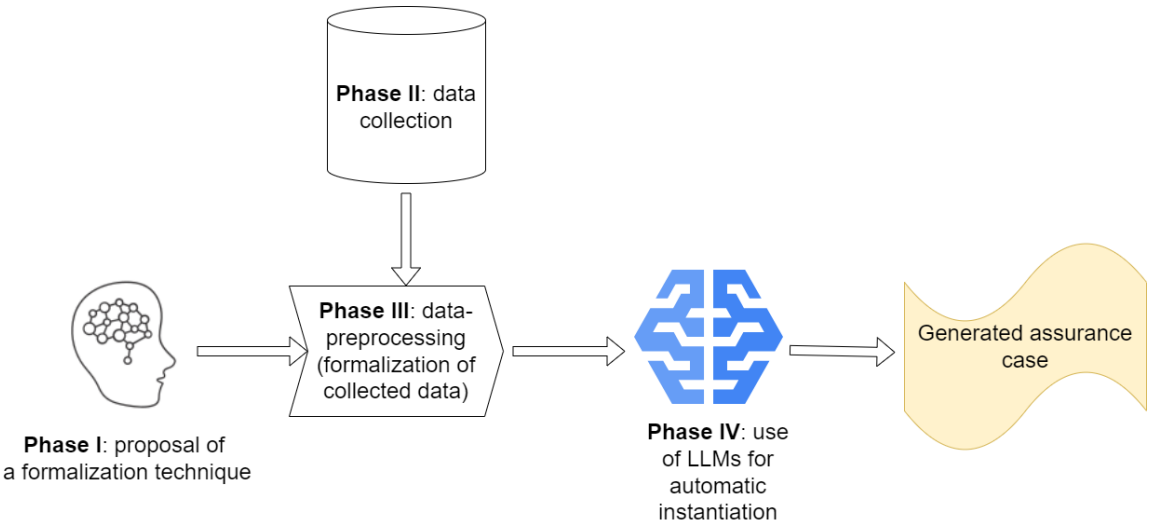

Figure 4.2: High-Level Overview of our Approach for RQ2

work of Denney and Pai [69] which formalizes ACs and ACPs, we propose a set of predicate-based rules to represent and therefore formalize assurance cases and assurance case patterns. The use of predicates for formalization allows capturing the properties and relationships among the elements of an assurance case and assurance case pattern [71].

To enhance the usability and understanding of LLMs within graphical notations like GSN, we propose integrating rule-based knowledge into LLMs using our predicate-based rules. For this purpose, we first create predicates allowing us to formalize assurance case patterns as a set of predicates rendered in a textual format complying with the very popular GSN. The predicate-based format is suitable for LLM ingestion and can be considered as an advanced and more formal structured prose. We construct our predicates based on the guidelines, elements, relationships, and decorators that the GSN Standard [24] outlines. Thus, we formulate a predicate for each GSN core element, relationship, and decorator represented in GSN. This results in three categories of predicates that we further discuss below.

Predicate for Formalizing an Assurance Case and Its Decorators

We propose the following predicates to formalize an assurance case complying with GSN:

- **Goal (G)**: True if G is a goal within the assurance case. This predicate can be

represented as Goal (ID, Description) where ID is the unique identifier for the goal, description is the textual description of the goal.

- **Strategy (S)**: True if S is a strategy within the assurance case. This predicate can be represented as Strategy (ID, Description) where ID is the unique identifier for the strategy and description is the textual description of the strategy.
- **Solution (Sn)**: True if Sn is a piece of evidence within the assurance case. This predicate can be represented as Solution (ID, Description) where ID is the unique identifier for the evidence and description is the textual description of the evidence.
- **Context (C)**: True if C is a context within the assurance case. This predicate can be represented as Context (ID, Description) where ID is the unique identifier for the context and description is the textual description of the context.
- **Assumption (A)**: True if A is an assumption within the assurance case. This predicate can be represented as Assumption (ID, Description) where ID is the unique identifier for the assumption and description is the textual description of the assumption.
- **Justification (J)**: True if J is a justification within the assurance case. This predicate can be represented as Assumption (ID, Description) where ID is the unique identifier for the assumption and Description is the textual description of the assumption.
- **Undeveloped (X)**: True if X is either a Goal or Strategy marked as undeveloped. This predicate is represented as Undeveloped(X), where X can be either a goal or strategy.

Predicates for Formalizing an Assurance Case Pattern

To formalize an assurance case pattern complying with GSN, we propose the following predicates:

- **Uninstantiated (X)**: True if element X (can be any GSN element) is marked as uninstantiated.
- **UndevelopStantiated(X)**: True if element X is either a Goal or Strategy and is marked both as uninstantiated and undeveloped.

- **HasPlaceholder (X)**: True if element X (can be any GSN element) contains a placeholder '{}' within its description that needs instantiation.
- **HasChoice (X, [Y], Label)**: True if an element X (either a Goal or Strategy) can be supported by selecting among any number of elements in [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between X and Y. A label is of the general form *m of n* (e.g. a label given as *1 of 3* implies an element in X can be supported by any one of three possible supporting elements in [Y]).
- **HasMultiplicity (X, [Y], Label)**: True if multiple instances of an element X (either a Goal or Strategy) relate to multiple instances of another element [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies how many instances of an element in X relate with how many instances of an element in [Y] (e.g. *m of n* implies m instances of an element in X must be supported by n instances of an element in Y).
- **IsOptional (X, [Y], Label)**: True if an element X (either a Goal or Strategy) can be optionally supported by another element [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between X and Y (i.e. an instance of an element in X may be supported by another instance of an element in [Y], but it is not required).

Predicates for Formalizing Relationships between GSN elements

The predicates we propose below are analogous to the two core GSN relationships i.e. *InContextOf* and *SupportedBy*.

- **IncontextOf (X, [N], D)**: True if element X at depth D has a neighbor [N] to the left or right at depth D, where N can be an assumption, justification, or context. X can be a goal or strategy and D represents the height or depth of the goal or strategy element and its neighbors in the GSN hierarchical structure.
- **SupportedBy (X, [C], D)**: True if element X at depth D has children [C] directly below it, where [C] can include Goal(G), Strategy(S), or Evidence(E) and X can be a goal(G), or strategy(S).

- If X is a Strategy, [C] can only be a Goal.
- If X is a Goal, [C] can be either Goal, Strategy, or Evidence.

4.2.2 Phase II: Data Collection

In RQ1, we conducted a bibliometric analysis on assurance case patterns. This allowed us to collect 92 primary studies published within the past two decades which focus on assurance case patterns. To collect data relevant to our current study, we analyzed these 92 studies, then selected five of them provided they described a pattern(s) and assurance cases instantiated (derived) from that pattern(s). We also made sure the selected patterns and assurance cases covered various application domains. This allowed us to create a dataset consisting of a set of assurance case patterns together with assurance cases derived from them.

4.2.3 Phase III: Data Pre-pocessing

Based on the defined predicates for GSN elements, relationships, and decorators we proposed in Phase I, we converted each collected assurance case pattern in GSN to a corresponding predicate form that an LLM can understand. This conversion facilitates the LLMs' comprehension of the inherent tree-like structure and relationships among the GSN elements in our dataset. It also aids in the generation of assurance cases complying with specific assurance case pattern(s). Figure 4.3 illustrates a predicate-based representation of an assurance case pattern in our dataset.

4.2.4 Phase IV: Using LLM to Automatically Generate AC

To generate assurance cases from patterns, we rely on LLMs. Each LLM takes as input the predicate-based representations of assurance case patterns and uses this representation as rules to: 1) enhance its reasoning capabilities by learning the features of patterns that are typically used to manually generate assurance cases and; 2) guide the automatic instantiation of assurance cases from the formalized patterns specified as inputs to the LLM. Each LLM generates an assurance case in the traditional structured prose (not in the predicate-based format). This allows using GSN guidelines [24] to convert the generated assurance case into GSN diagrams.

To support the generation process, we rely on prompt engineering to provide instructions, and guidelines, and enforce rules to ensure a desired generated response. We describe the LLMs prompts in Section 4.2.9.

Goal (G0, {System} satisfies security requirements}
Goal (G1, {System} satisfies the asset protection requirements)
Goal (G2, {System} satisfies secure development requirements)
Goal (G3, Asset protection requirements are met during the architecture design phase)
Goal (G4, Asset protection requirements are met during other phases)
Goal (G5, {System} architecture is protected against identified security threats (STs))
Goal (G6, {System} architecture is validated)
Goal (G0.X, {System} architecture is protected against STX)
Strategy (S0, Argue through asset protection and secure development requirements)
Strategy (S1, Argue through the different stages of the system development life cycle)
Strategy (S2, Argue through derivating security threats from SRs)
Strategy (S3, Argue over each security threat)
Context (C0, Description of {system})
Context (C1, SR are requirements about protecting the system from malicious entities)
Context (C2, Description of the {architecture})
Context (C3, Description of {system} architecture model)
Justification (J0, The argumentation is based on satisfaction of SRs)
Justification (J1, Detection and mitigation of threats fulfill SRs)
Assumption (A0, System SRS are complete, adequate, and consistent}
Assumption (A1, Asset inventory is established}
Assumption (A2, All relevant threats have been identified}
Assumption (A3, {System} architecture model is well defined in {formal method}}
SupportedBy (G0, S0, 1)
SupportedBy (S0, [G1, G2], 2)
SupportedBy (G1, S1, 3)
SupportedBy (S1, [G3, G4], 4)
SupportedBy (G3, S2, 5)
SupportedBy (S2, [G5, G6], 6)
SupportedBy (G5, S3, 7)
SupportedBy (S3, G0.X, 8)
IncontextOf (G0, [C0, C1, J0, A0], 1)
IncontextOf (G1, A1, 3)
IncontextOf (G3, C2, 5)
IncontextOf (S2, J1, 6)
IncontextOf (G5, A2, 7)
IncontextOf (G6, [C3, A3], 7)
HasPlaceholder (G0)
HasPlaceholder (C0)
HasPlaceholder (G1)
HasPlaceholder (G2)
HasPlaceholder (C2)
HasPlaceholder (G5)
HasPlaceholder (G6)
HasPlaceholder (C3)
HasPlaceholder (A3)
HasPlaceholder (G0.X)
Uninstantiated (G0)
Uninstantiated (C0)
Uninstantiated (G1)
Uninstantiated (C2)
Uninstantiated (G5)
Uninstantiated (C3)
Uninstantiated (A3)
Undeveloped (G4)
UndevelopStantiated (G2)
UndevelopStantiated (G6)
UndevelopStantiated (G0.X)

Figure 4.3: A Simple Predicate-Based Representation of the Assurance Case Pattern Depicted in Figure B.1 (see appendix).

4.2.5 Description of Dataset

Our dataset comprises six assurance case patterns and five corresponding assurance cases that comply with these patterns, covering five distinct systems. These systems span various application domains, namely: the aviation, automotive, medical, and computing domains. We selected one assurance case pattern, presented in our formalized format, along with the assurance case complying with this pattern, presented in a structured prose format, as a one-shot example for our experiments. This example helps to illustrate to our LLM the concept of generating assurance cases from assurance case patterns. The remaining assurance case patterns, presented in a formalized format, are used in the LLM prompts. This helps LLMs to generate assurance cases that comply with these patterns. Table 4.3 provides various statistics, such as the count of decorators (e.g., *undeveloped*, *uninstantiated*, *choice*) in the assurance case patterns, and the count of relationships (e.g., *InContextOf*, *SupportedBy*) in the corresponding assurance cases that comply with these patterns. In the remainder of this section. We provide a detailed description of our dataset. For each system, we explain the assurance case and associated pattern(s) used to manually develop the assurance case. These manually created assurance cases serve as our ground-truth data for evaluating the LLM-generated assurance cases.

The GSN diagrams depicting the ACPs and ACs complying with these patterns for the systems in our dataset are available in the appendix B.

Table 4.3: Overview of our Dataset

System	Domain	Patterns (ACPs)		Assurance Cases (ACs)	
		Placeholder	**Element**	**Element**	**Relations**
ACAS XU	Aviation	10	22	24	23
BLUEROV2	Automotive	8	18	24	21
GPCA	Medical	21	23	27	26
IM SOFTWARE	Computing	9	15	24	23
DEEPMIND	Medical	26	17	23	23

ACAS XU and Its Assurance Framework

ACAS Xu (Airborne Collision Avoidance System Xu) is a collision avoidance system designed for use in unmanned aerial vehicles (UAVs), commonly known as drones [3]. The primary objective of ACAS Xu is to enhance the safety of drone operations

by preventing collisions between drones or between a drone and other objects in its environment [3]. The architecture of ACAS Xu contains four major components: the sensors to gather data on potential intruders; the processor to compute a suitable avoidance strategy; the planner that plans the trajectory to navigate safely while avoiding collisions; and the actuator that executes the planned trajectory [3]. To ensure that ACAS Xu is acceptably secure against security threats, Zeroual et al. [3] provided a threat identification assurance case pattern. They demonstrated the application of this pattern in creating a partial security case specific to the ACAS Xu system.

BlueROV2 and its Assurance Framework

The BlueROV2 system is an advanced Unmanned Underwater Vehicle (UUV) or underwater Remotely Operated Vehicle (ROV) [4]. Its main objective is to autonomously track pipelines on the seafloor while avoiding static obstacles such as plants and rocks [4]. Safety assurance for BlueROV2 is achieved through the identification of potential hazards and reduction of the risk posed by those hazards based on the ALARP (As Low As Reasonably Practicable) principle [4]. Hartsell et al. [4], generated an assurance case for BlueROV2 using the ALARP pattern sequentially composed with another pattern called the ReSonAte pattern. We utilized these two patterns and the generated assurance case in our dataset. Figure B.3 (located in the appendix) shows the combined pattern formed from both the ALARP pattern and the ReSonAte pattern. The *SupportedBy* relationship between G3 and S4 in that Figure links the two patterns together.

GPCA and its Assurance Framework

The Generic Patient-Controlled Analgesia (GPCA) system, also known as an *Infusion pump* is one of the most common safety critical systems in the medical domain. Some of the operational hazards faced by this system include "*Overinfusion*" and "*Underinfusion*" which can have dire consequences for patient safety [5]. To demonstrate the application of assurance cases for certifying the safety of critical systems, several studies [5, 26] have utilized the GPCA system as a case study.

Lin et al. [5] presented a safety case pattern and a safety case for the GPCA system complying with this pattern. Note that some GSN elements of this safety case have duplicated identifiers whereas GSN usually fosters uniqueness of identifiers. To mitigate that duplication, we systematically reassigned unique identifiers to each duplicated GSN element.

The Instant Messaging (IM) Server Software and its Assurance Framework

The Instant messaging (IM) server software is used for information exchange, with the data within the software forming the basis of user interaction [6]. That system is characterized by its independent behavioral features, known as its internal structure, as well as by its interactive relationships with external components, referred to as the External Manifestation (EM) [6]. The external manifestation encompasses the overall interaction of the software with the outside world, including the set of external environment entities, the interaction set between these entities and the software, and the direction of these interactions. The Internal Structure (IS) of software focuses on the internal functional processes and data transmission within the software. This includes the set of software functional processes, data storage, and internal interaction sets [6].

Ensuring that the IM server software is acceptably secure requires a demonstration to show that critical assets such as user account information, and authentication information are well-protected [6]. Thus, Xu et al. [6] presented a software security top-level argument pattern and utilized this pattern to create a software security case for the IM software. Some GSN elements of this security case are duplicated. To mitigate the duplication as explained above, we systematically renumbered and reassigned unique identifiers to each GSN element.

The DeepMind ML System for Retinal Disease Diagnosis and its Assurance Framework

The DeepMind system is an example of a safety-critical system that uses machine learning-based functionality. The DeepMind system utilizes two neural networks to predict retinal disease from eye scans. The first neural network processes a retinal scan to generate a tissue-segmentation map. This map is then analyzed by the second neural network, which provides a diagnosis and referral [7].

Ward and Habli [7] presented an assurance case pattern for justifying the sufficiency of the interpretability of ML in safety-critical systems. They demonstrated the application of this pattern in creating an assurance case for the interpretability of the machine learning component in the DeepMind system.

We utilized both this ACP and AC generated for the machine learning component of the DeepMind system as a one-shot example in our experiments. Our choice of that example is random.

4.2.6 LLM Setup

To carry out our experiments, we focused on two LLMs, namely: GPT-4o and GPT-4 Turbo. We chose them because they are powerful and incorporate the latest features. The non-deterministic nature of LLMs, motivated us to run each of our experiments multiple times i.e. K times, where $K = 5$. This allows for mitigating the potential inconsistencies in responses and ensuring reliable evaluation of our LLMs. To interact with both LLMs, we relied on the OpenAI API [107]. We set the default values for the following parameters when interacting with both LLMs:

- **The temperature**: it controls the randomness and creativity in the output of LLMs. By adjusting this parameter, users can balance creativity and coherence in the generated response [108]. In our experiments, we set the temperature parameter to its default value of 1.
- **The maximum number of tokens**: It controls the length of responses generated by the LLM. In our experiments, we consistently set its value to "4096" to accommodate longer text across all experiments.

4.2.7 Description of Experiments

In our experiments, we applied the Chain-of-Thought (CoT) [61] prompting technique. This allows for enhancing the reasoning capabilities of the LLMs and therefore improves their ability to perform a complex reasoning task, namely: the generation of assurance cases.

Table 4.4 provides a descriptive summary of the various experiments in our study. Each row, labeled from Exp 1 to Exp 9, describes the individual configuration of each experiment, while each column represents one of four distinct categories of software engineering knowledge: 1) Example; 2) Context Information; 3) Domain Information; and 4) Predicate Rules. The presence of each of these categories in a given experiment is denoted by an 'X' mark in the corresponding cell. For instance, Experiment 1 excludes all four categories of SE knowledge, Experiment 2 includes all four, while Experiment 9 only includes predicate rules.

Experiment without Software Engineering Knowledge Specified in the Prompt

In this experiment (i.e. Experiment 1), we want to assess the performance of GPT-4 Turbo and GPT-4o in generating assurance cases when no extra software engineering knowledge is included in their prompts. Hence, in this experiment, we do not provide

Table 4.4: Overview of our Experiments

	Example	Context Information	Domain Information	Predicate Rules
Exp 1				
Exp 2	x	x	x	x
Exp 3		x	x	x
Exp 4	x	x		x
Exp 5		x		x
Exp 6	x		x	x
Exp 7			x	x
Exp 8	x			x
Exp 9				x

context information, domain information, examples, and predicate rules to both LLMs.

Experiments with Software Engineering (SE) Knowledge Specified in the Prompt

Building on previous work from literature (e.g., [22], [73], and [89]) and to answer our RQ2, we conducted eight additional experiments (i.e. Experiments 2 to Experiments 9), each leveraging at least one category of SE knowledge. These categories include the presence or absence of domain information, contextual information, and K-shot examples (i.e. providing the LLMs with some examples (shots) and then querying the model to complete the task.) paired with our predicate rules. Note that all the patterns we use as input are represented in the predicate-based format that we specified. To allow both LLMs to digest that format, we therefore specify predicate rules in each of the eight experiments described above.

4.2.8 Description of S.E Knowledge

As stated previously, we can specify several categories of SE knowledge in our prompts. Like in the literature (e.g., [22], [73], and [89]), to allow each LLM to perform effectively and generate outputs (assurance cases) close to the ground-truth, we notably rely on the following two categories of SE knowledge:

- **Contextual Information** – We define *'contextual information'* as the background details conveying the fundamental information about the structure and

representation of the different elements and decorators in the assurance case and assurance case pattern represented in GSN. The contextual information also provides instructions on how to derive an assurance case from an assurance case pattern. It aims to allow the LLM to enhance its ability to interpret and understand the general structure, content, and guidelines necessary to generate assurance cases complying with a given pattern effectively. It remains the same across all our experiments. The contextual information we used in our experiments is available in the Appendix (see Section D).

- **Domain Information** – In our experiments, *'domain information'* refers to the specialized knowledge, terminology, and facts specific to the domain or system for which an assurance case is being automatically created. Examples of this information include details about the mode of operation, test results, and verification activities within that domain or system. The domain information enables the LLM to select from a variety of artefacts (arguments, evidence) necessary to replace the generic information found in placeholders within the assurance case pattern. The domain information can vary from one application domain to another. Hence, in our experiments, we utilized different domain information for each system in our dataset. We extracted this domain information from the cited references [3, 4, 5, 6] that describe our dataset.

4.2.9 Description of Prompts

The OpenAI API supports three types of prompts: the system prompt, the user prompt, and the assistant prompt. These can be categorized into two main groups that we further explain below.

Input passed to the LLM

The input passed to the LLM is the LLM prompt. The latter results from the combination of two other prompts:

- **System Prompt**: This consists of instructions and guidelines provided to the LLM to ensure it responds appropriately. Depending on the experiment, our system prompt may contain all or a combination of the various categories of SE knowledge. Figure 4.4 depicts the system prompt given to each LLM for Experiment 1. That system prompt deliberately excludes any SE knowledge. This allows us to evaluate the inherent performance of the LLMs without the influence of SE knowledge.

> You are an assistant who assists in developing an assurance case in a tree structure using Goal Structuring Notation (GSN). Your role is to create an assurance case.

Figure 4.4: A Sample System Prompt for Experiment 1

- **User Prompt**: This is the input or query from the user interacting with the model, requesting the model to complete a specific task. In the user prompts specified for the experiment without SE knowledge (i.e. Experiment 1), we do not specify any assurance case pattern, as we consider this to be a form of SE knowledge. Figure 4.5 shows a sample of our user prompt for Experiment 1.

> Create a security case for ACAS Xu (Airborne Collision Avoidance System Xu) and display it in a hierarchical tree format using dashes (-) to denote different levels.

Figure 4.5: A Sample User Prompt for Experiment 1

Output generated by the LLM

- **Assistant Prompt**: is the output that the LLM generates. More specifically, the assistant prompt refers to the response generated by the model based on the system and user prompts provided to the model. Thus, in our work, the assistant prompt provides the assurance case that the LLM generates.

Description of System Prompts with S.E Knowledge

Figure 4.6 depicts the template (i.e. the generic structure) of the system prompts we used in our experiments with SE knowledge.

Figure 4.6: Generic Structure of our System Prompts for Experiments with SE Knowledge

Description of Zero-shot System Prompts with SE Knowledge In our zero-shot experiments involving SE knowledge (i.e. Experiments 3, 5, 7, and 9), the system prompts used to query both LLMs may consist of various categories of SE knowledge. However, these prompts exclude the example category, which is indicated by red dotted lines in Figure 4.6.

Description of One-shot System Prompts with SE Knowledge In our one-shot experiments involving SE knowledge (i.e. Experiments 2, 4, 6, and 8), the system prompts used to query both LLMs may consist of various categories of SE knowledge. This includes the example category, which is indicated by red dotted lines in Figure 4.6.

Description of User Prompts with S.E Knowledge

In the user prompts specified for experiments involving SE knowledge (i.e., Experiments 2 to Experiment 9), we include the assurance case pattern in our predicate-based format for each system. We then prompt the model to create an assurance case that complies with this pattern for a given system in our test dataset. Figure 4.7 shows an excerpt of a sample user prompt in our experiments with SE knowledge.

4.2.10 Evaluation Measures

Each evaluation metric we outline below allows for assessing the similarity between the experiment results (i.e. LLM-generated assurance cases) and the ground-truth.

Exact Match

As in Chang et al. [109], we employ this metric to gauge the accuracy with which our LLM-generated output matches the ground-truth, character by character, without discrepancies. The scoring typically ranges from 0 to 1, where 0 signifies no similarity and 1 indicates a perfect or near-perfect match. To compute this measure, we rely on the Python library called *FuzzyWuzzy* [110].

BLEU Score

As in Hou et al. [111], we utilize the BLEU score to assess the similarity between the generated text and the ground-truth. The latter is one of the widely used metrics in NLP (Natural Language Processing) and one of the most commonly used evaluation metrics for natural language texts [112]. It ranges between 0 and 1, where 0 indicates

Based on the predicates given below for a security case pattern for threat identification, use this pattern to create a security case for ACAS Xu (Airborne Collision Avoidance System Xu and display in a hierarchical tree format using dashes (-) to denote different levels.

Goal (G0, {System} satisfies security requirements}
Goal (G1, {System} satisfies the asset protection requirements)
Goal (G2, {System} satisfies secure development requirements)
Goal (G3, Asset protection requirements are met during the architecture design phase)
Goal (G4, Asset protection requirements are met during other phases)
Goal (G5, {System} architecture is protected against identified security threats (STs))
Goal (G6, {System} architecture is validated)
Goal (G0.X, {System} architecture is protected against STX)
Strategy (S0, Argue through asset protection and secure development requirements)
Strategy (S1, Argue through the different stages of the system development life cycle)
Strategy (S2, Argue through derivating security threats from SRs)
Strategy (S3, Argue over each security threat)
Context (C0, Description of {system})
Context (C1, SR are requirements about protecting the system from malicious entities)
Context (C2, Description of the {architecture})
Context (C3, Description of {system} architecture model)
Justification (J0, The argumentation is based on satisfaction of SRs)
Justification (J1, Detection and mitigation of threats fulfill SRs)
Assumption (A0, System SRS are complete, adequate, and consistent}
Assumption (A1, Asset inventory is established}
Assumption (A2, All relevant threats have been identified}
Assumption (A3, {System} architecture model is well defined in {formal method}}
SupportedBy (G0, S0, 1)
SupportedBy (S0, [G1, G2], 2)
SupportedBy (G1, S1, 3)
SupportedBy (S1, [G3, G4], 4)
SupportedBy (G3, S2, 5)
SupportedBy (S2, [G5, G6], 6)
SupportedBy (G5, S3, 7)
SupportedBy (S3, G0.X, 8)
IncontextOf (G0, [C0, C1, J0, A0], 1)
IncontextOf (G1, A1, 3)
IncontextOf (G3, C2, 5)
IncontextOf (S2, J1, 6)
IncontextOf (G5, A2, 7)
IncontextOf (G6, [C3, A3], 7)
HasPlaceholder (G0)
HasPlaceholder (C0)
HasPlaceholder (G1)
HasPlaceholder (G2)
HasPlaceholder (C2)
HasPlaceholder (G5)
HasPlaceholder (G6)
HasPlaceholder (C3)
HasPlaceholder (A3)
HasPlaceholder (G0.X)
Uninstantiated (G0)
Uninstantiated (C0)
Uninstantiated (G1)
Uninstantiated (C2)
Uninstantiated (G5)
Uninstantiated (C3)
Uninstantiated (A3)
Undeveloped (G4)
UndevelopStantiated (G2)
UndevelopStantiated (G6)
UndevelopStantiated (G0.X)

Figure 4.7: A Sample User Prompt for Experiments with SE Knowledge.

no match between the generated text and the ground-truth text, while 1 indicates a perfect match between both the generated text and the ground-truth text. We use a Python library called *Sacrebleu* [113] for its assessment.

Semantic Similarity

In this metric, we evaluate how closely the texts in the GSN elements of the assurance cases generated by our LLMs relate in meaning to the ground-truth assurance cases. To assess that measure, we rely on the cosine similarity measure [114, 115]. Cosine similarity values range from -1 to 1, where -1 indicates no similarity or completely dissimilar texts, and 1 indicates identical texts. To compute this measure, we first transform both our ground-truth text and LLM-generated text to real-valued vectors using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. Since TF-IDF vectors are non-negative, the angle between both vectors ranges from 0 to 90 degrees. Hence, the cosine similarity, which measures the cosine of the angle between both vectors, ranges from 0 to 1. We rely on a Python library called *scikit-learn* [116] to automatically compute the values of that metric.

4.3 Methods Used to Answer RQ3

To address RQ3, we adopted the design approach we proposed in [30] for creating a safety case for an ML-enabled system that is already developed and operational. We applied this approach to develop a safety case for an ML-enabled component within Baidu Apollo, an open-source autonomous driving system. Our focus is specifically on the trajectory prediction component of Baidu Apollo, for which we developed a safety case to provide a set of structured arguments, supported by evidence, that the component meets safety requirements and effectively mitigates potential risks associated with trajectory prediction.

4.3.1 Steps to Manually Create an Assurance Case for an ML-Enabled Autonomous Driving System

Figure 4.8 presents a high-level overview of the design methodology we introduced in [30]. That methodology consists of three phases, which we describe in detail below.

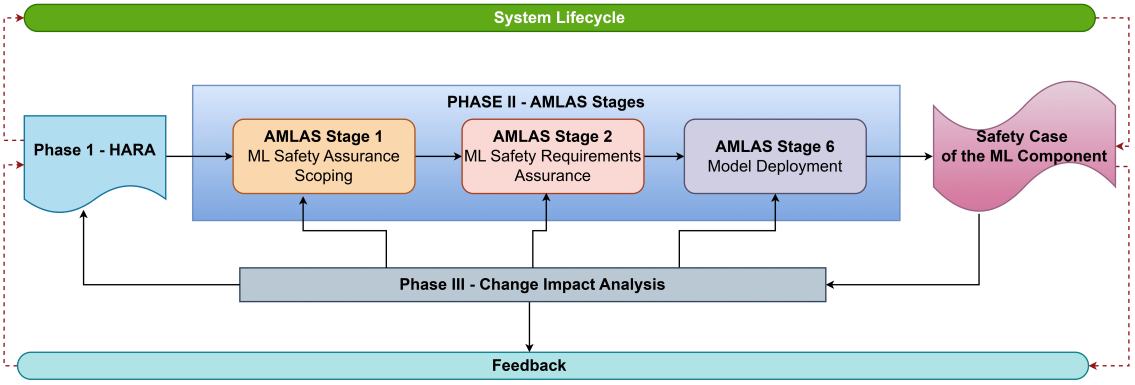

Figure 4.8: RQ3 High-Level Approach Overview Adapted from [30]

Phase I: Hazard Analysis and Risk Assessment (HARA)

Following the design approach we proposed in [30], we conducted a Hazard Analysis and Risk Assessment (HARA) of the trajectory prediction component to identify its potential hazards and determine its safety requirements. HARA is a systematic approach used to identify and evaluate potential hazards associated with a system and to determine the associated risks [30, 117]. For safety-critical systems like trajectory prediction components, HARA is essential in identifying possible failure modes and understanding their impact on system safety. In this work, the use of HARA is particularly relevant as it ensures a thorough understanding of the potential risks involved in the trajectory prediction process. The HARA process also facilitated the creation of several artefacts used in Phase II for instantiating the selected argument patterns from AMLAS [74]. The results of our HARA process are available on GitHub[1].

Phase II: Implementation of Selected AMLAS Stages

In this phase, following Sivakumar et al. [30], we implemented three stages from the AMLAS [74] methodology that are relevant to the creation of our safety case. These three stages include stages 1,2 and 6. We provide a brief description of each of these stages.

1. Stage 1: ML Safety Assurance Scoping

[1]https://github.com/Oluwafemi17/LLM-Based-Safety-Case-Generation-for-Baidu-Apollo-Are-We-There-Yet-/blob/main/Hazard_Analysis_Risk_Assessment.xlsx

In this stage, we aim to establish a clear framework for assuring the safety of the ML component. We define the scope of the safety assurance process for the ML component, determine the scope of the safety case, and create the top-level safety assurance claim while specifying the relevant contextual information. This is achieved by instantiating the ML Safety Assurance Scoping Argument pattern shown in Figure 5.13.

2. Stage 2: ML Safety Requirements Assurance

   In this stage, we focus on developing and validating the machine learning safety requirements. We derive the ML safety requirements from the allocated system safety requirements, ensuring they are aligned with the specific safety needs of the ML component. We validate these requirements against the allocated safety requirements, the system and software architecture, and the operational environment to ensure completeness. This process is achieved by instantiating the ML Safety Requirements Argument Pattern shown in Figure 5.15.

3. Stage 3: Model Deployment

   In this stage, our goal is to provide the necessary arguments and evidence to demonstrate the successful integration of the machine learning component into the autonomous driving system. We aim to show that the system safety requirements continue to be satisfied post-integration. This is achieved by instantiating the ML Deployment Argument Pattern shown in Figure 5.17, ensuring the ML model continues to meet its safety requirements within the ADS.

Each AMLAS stage consists of a list of input artefacts, output artefacts, and activities that must be performed in that stage when analyzing a specific system. We describe them in Section 5.3.3, in the light of a real-world system.

Phase III: Change Impact Analysis

This phase addresses how modifications — such as changes to the ML component, other parts of the autonomous driving system, or the operating environment — affect system artefacts and the existing safety case. Carlan and Gallina [8] opined that *"in safety-critical domains, a change in the operating context triggers the need for impact analysis on the artefacts generated during the safety life-cycle"*. Autonomous driving systems, being part of safety-critical domains, often encounter changes in their operating conditions, such as transitioning between urban and rural environments. This highlights the importance of change impact analysis in ensuring the highest

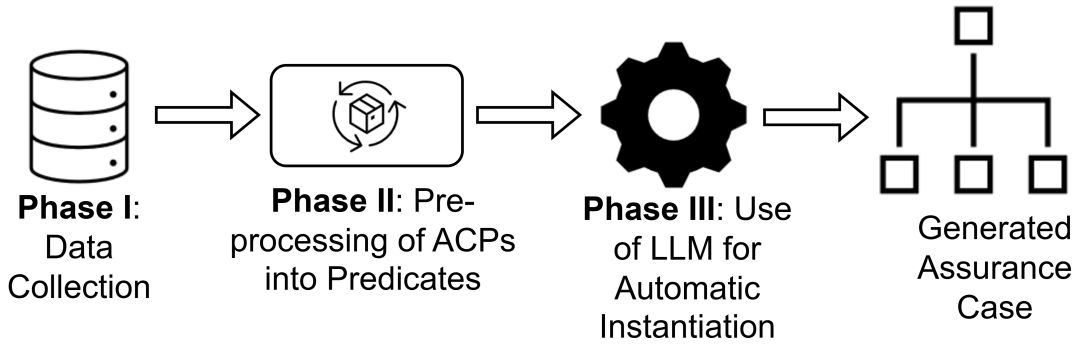

Figure 4.9: RQ4 High-Level Approach Overview Adapted from [12]

level of system safety and functionality. We can only focus on Phases I and II of the design methodology if we do not have access to real-time driving and simulation data needed to complete a comprehensive change impact analysis.

4.4 Methods Used to Answer RQ4

We adopted a similar approach to the approach used in section 4.2 to facilitate the automatic creation of assurance cases using LLMs. We provide a brief description of this approach below.

4.4.1 Description of the LLM-powered Approach

We adopted the four-phase steps proposed in section 4.2 to instantiate assurance cases from assurance case patterns using LLMs. The details of these steps are as follows.

Figure 4.9 shows a high-level overview of this approach. We further describe that approach below.

4.4.2 Phase I: Data Collection

To collect relevant data for the automatic instantiation of assurance cases from patterns, we followed a structured approach centered on key steps essential for effective assurance case development. We began by focusing on safety-critical systems that include specialized components, such as those employing machine learning technologies. These systems present unique assurance requirements due to their

complexity making them ideal candidates for analysis. We then identified well-established and validated assurance case patterns from peer-reviewed studies, which have been used to guide the structuring and development of assurance cases.

Our dataset comprises these assurance case patterns and the assurance cases developed from them. We divided the dataset into two parts: one for use as a one-shot example and the other for evaluating our approach's performance. The one-shot example includes an assurance case pattern and the corresponding derived assurance case, provided to our LLM as an example to enhance its understanding of instantiating an assurance case from a pattern.

The data used to evaluate the performance of our LLM-based approach also consists of assurance case patterns and assurance cases derived from these patterns. In this case, we input the assurance case patterns as prompts to the LLM for it to generate an assurance case complying with this pattern. The originally derived assurance case then serves as ground-truth to evaluate the reliability and completeness of the newly instantiated assurance cases generated by the LLM.

4.4.3 Phase II: Pre-processing of ACPs into Predicates

In this phase, we utilized the predicate-based rules proposed in Section 4.2.1 for the formalization of assurance cases, assurance case patterns, and their relationships and decorators. We converted the assurance case pattern from our dataset into this predicate-based format. The latter is an advanced structured prose format that complies with GSN. This conversion enables our LLM to use it as input for generating an assurance case that complies with this pattern.

4.4.4 Phase III: Using LLM to Automatically Generate Assurance Cases

In this phase, we utilize a Large Language Model to automatically generate an assurance case for the analyzed component of Baidu Apollo ADS, using the formalized pattern provided as input to that model. These patterns are the AMLAS argument patterns.

4.4.5 Case Study: Automatic Generation of a Safety Case for Baidu Apollo

Dataset description

1. The DeepMind System and its Assurance Framework
   This represents the same system and its assurance framework used as a one-shot example in Section 4.2.5. We re-used this system and its assurance framework as a one-shot example in RQ4 since it is the only ML-enabled system in our dataset in section 4.2.5.

2. Safety Argument Pattern and Derived Safety Case for Baidu Apollo ADS

   Our dataset also comprises the safety case we manually developed in Section 5.3.1 and the ACP resulting from the combination of the three argument patterns we used in Section 5.3.1 to create that safety case. To validate the experiment results associated with RQ4, we use that safety case as the ground-truth.

4.4.6 LLM Description and Settings

In our experiments, we chose GPT-4o as our LLM due to its robust capabilities and the advanced features it offers as a recent addition to the OpenAI GPT series. To address the non-deterministic behavior of our LLM, we conducted each experiment five times (K=5) following the literature (e.g., [22, 88]). We interacted with our LLM using the OpenAI API [107], while maintaining default settings for the different parameters used in our interaction with GPT-4o.

- **Temperature**: This parameter controls the randomness and creativity in the outputs of our LLM, allowing users to find a balance between creativity and coherence in the generated text. In our experiments, we set the temperature parameter to its default value of 1.

- **Number of Tokens**: This parameter determines the length of the responses generated by our LLM. We consistently set this parameter to "4096" to handle longer text across all experiments.

4.4.7 Description of the Experiments

To investigate RQ4, we conducted two different experiments to evaluate the effectiveness of GPT-4o in generating an assurance case for Baidu Apollo. Our experiments utilized the Chain of Thought (CoT) prompting technique [61] combined with software

engineering knowledge. As in RQ2, that knowledge includes: 1) Predicate-based rules, 2) Domain Information, 3) Context Information, and 4) One-shot Example.

- **Experiment 1: One-shot, with context information, with domain information, and with predicate rules** - This experiment evaluates our LLM ability to instantiate an assurance case using all four categories of SE knowledge. Thus, this experiment evaluates our LLM ability to instantiate an assurance case using a one-shot approach where the model is provided with a given example, together with domain information about the system, contextual information, and our predicate-based rules.
- **Experiment 2: Zero-shot, without context information, without domain information, and without predicate rules** - In this experiment, we evaluate our LLM's intrinsic ability to instantiate an assurance case without any SE knowledge in its prompts.

Similar to RQ2, we used the same structure of our system and user prompts to query the LLM for the experiment with SE knowledge and the experiment without any SE knowledge. We also utilized the same assessment measures such as the Exact Match, BLEU score, and Semantic similarity measure.

Chapter 5

Results

5.1 Results of RQ1

Figure 5.1 shows our PRISMA flow chart diagram. The total number of primary studies identified, filtered out, and included using our database-driven search and snowballing technique is depicted in this diagram. We identified and utilized a total of **92** primary studies to answer our proposed RQ1.

Table A.1 in the Appendix reports the details about our list of primary studies. That information consists of: the authors of the study, its publication year, its title, its venue acronym, and the search method (i.e. snowballing, database-driven search) we used to select that study.

5.1.1 Annual Distribution of Publications

Figure 5.2 reports the number of publications and their distribution over time within the last two decades. From this distribution, the number of publications per year before 2009 is very low. An explanation for this might be due to the novelty of assurance case patterns and the existence of less complex interconnected systems. We observed that the number of publications started to increase in 2009 which could be attributed to the introduction and popularization of model-based systems engineering (relevant in the development and conceptualization of assurance case patterns) in 2007 by the International Council on Systems Engineering (INCOSE).

The number of publications fluctuated between 2009 and 2013 leading to an average of four publications between 2009 (inclusive) and 2013 (inclusive). However, between 2014 and 2018, there was an average of eight publications per year. A notable observation in the distribution is the peak of 12 publications in 2014. This peak

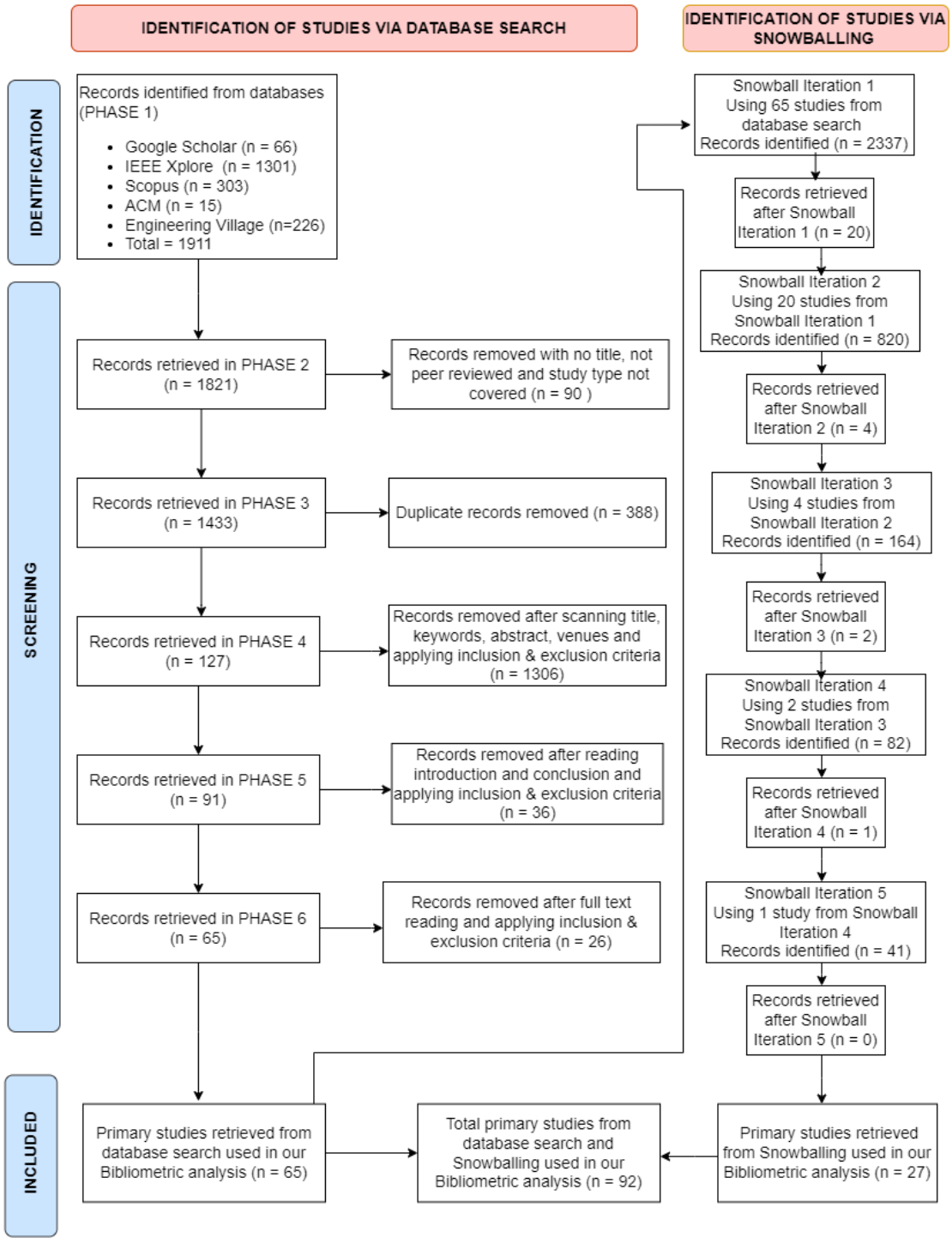

Figure 5.1: PRISMA Flow Diagram Illustrating the Identification and Selection of Primary Studies

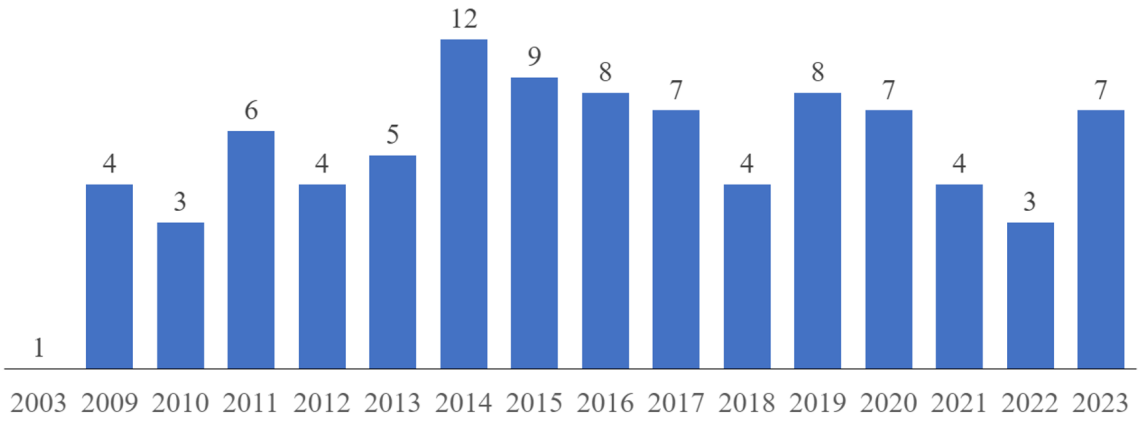

Figure 5.2: Publication-Year Distribution

could be attributed to the increase in the adoption of assurance case patterns. These patterns encapsulate and support the re-use of evidence and arguments.

In 2021 and 2022, there was a decrease in the number of publications which may be due to the COVID-19 pandemic that disrupted activities across the world and a decline in opportunities for collaboration. It is crucial to point out that our search for primary studies ended in early October 2023 which means that our search might miss a few studies published between early October 2023 and December of 2023. However, our publication-year distribution for 2023 shows an increase in the number of publications compared to the two previous years which signifies ongoing interest and relevance of research on assurance case patterns.

5.1.2 Distribution of Publication by Venue

Gaining knowledge about the focus of each conference, journal, or workshop is not only important for assisting new researchers decide the right venue for sharing their research, but also for identifying which venue to search and explore when performing a literature review. To this end, Figure 5.3 provides information about the most influential venues for publishing articles on assurance case patterns. That chart shows that SAFECOMP (International Conference on Computer Safety, Reliability, and Security) is the most preferred venue since it yields the highest number of publications. This corroborates the commitment of SAFECOMP to the promotion of research

Figure 5.3: Publication Venue Distribution

focusing on the dependability and reliability of safety-related and safety-critical systems.

A noteworthy observation is ISSREW (International Symposium on Software Reliability Engineering Workshops) which is a workshop of ISSRE (International Symposium on Software Reliability Engineering) ranks second with seven publications which shows that new researchers with viable ideas can publish their research in workshops and gather feedback. Subsequently, the International Symposium on High-Assurance Systems Engineering (HASE) has four publications. However, the last occurrence of this conference was in 2019. Also, SafeAI (The AAAI's Workshop on Artificial Intelligence Safety) being part of the top nine venues shows an ongoing focus on ethics assurance of machine learning systems and safety assurance of systems with machine learning components.

5.1.3 Top Publications by Citations

To identify some of the top publications that have a significant impact in the field of assurance case pattern, we utilized Google Scholar [99] to retrieve the count of citations for each of the primary studies used in our bibliometric analysis. Table 5.1 shows the top 16 most influential publications based on the number of citations retrieved from Google Scholar [99] as of December 12, 2023. All citations per publication in the table exceed 30, with an average citation value of 64.

The publications by Hawkins et al. [48] and Hawkins et al. [77] topped the list with 206 and 116 total citations respectively and were published by the same

first author. This is followed by Denney and Pai [118] and Denney and Pai. [69] with 115 and 69 total citations and published by the same first author. Hawkins et al. [48] proposed a separation of concerns approach for creating a clear assured safety argument that distinguishes the safety argument from its associated confidence arguments to ensure easy identification and mitigation of assurance deficits. In their work, they proposed ACPs to demonstrate sufficient confidence that assurance deficits have been identified, and mitigated and that any residual deficits are acceptable. Also, Hawkins et al. [77] proposed a model-based assurance approach that uses a weaving model to link argument patterns with information extracted from design models, analysis models, and development models of a system. This approach aimed to automate the generation and analysis of assurance cases. It also aimed to improve traceability between the assurance case artefacts and the design models.

Denney and Pai [118] developed AdvoCATE a robust multi-functional tool-set for the automatic creation of assurance cases, instantiating argument patterns, support for integrating evidence from formal methods into assurance cases, and support for hierarchy and modular organization of argument structures. The publications by Hawkins [48, 77] and Denney [69, 118] performed well in terms of the number of citations which shows its strong relevance and high influence in assurance case patterns research. Thus, new researchers should pay more attention to these publications.

A glance of Table 5.1 shows the terms ***"Patterns"***, ***"model-based approach"***, ***"automotive"***, ***"machine learning"***, and ***"formal"*** as the most common keywords in the titles of these publications. This suggests the use of assurance case patterns in the automotive industry, the need for ACPs to provide argument structure for systems with machine learning components, the formalization of ACPs, and the need for model-based assurance cases using SACM.

5.1.4 Most Cited Keyword Analysis

To understand the knowledge and conceptual details of the primary research topics within the field of assurance case patterns, we used VosViewer to generate the keyword network that Figure 5.4 illustrates. VosViewer synthesizes frequently used keywords and analyzes the keyword co-occurrence network across the field of ACP. Based on our dataset of 92 studies in ACP and a total of 224 keywords retrieved from these studies, we set the minimum number of occurrences of a keyword as two in VosViewer and eliminated some inconsistencies by merging semantically identical keywords like ***"GSN"*** and ***"Goal Structuring Notation"***. Based on these criteria, a total of 45 keywords met this threshold. We used these 45 keywords to generate the keyword network.

Table 5.1: Top 15 publications based on the number of citations on Google Scholar as of December 12, 2023

No	Author	Title	# Citation
1	Hawkins et al. (2011)	A New Approach to Creating Clear Safety Arguments	206
2	Hawkins et al. (2015)	Weaving an Assurance Case from Design: A Model-Based Approach	116
3	Denney & Pai. (2018)	Tool support for assurance case development	115
4	Denney & Pai. (2013)	A formal basis for safety case patterns	69
5	Yamamoto et al. (2013)	An evaluation of argument patterns to reduce pitfalls of applying assurance case	63
6	Wei et al. (2019)	Model Based System Assurance Using the Structured Assurance Case Metamodel	62
7	Weaver et al. (2003)	A Pragmatic Approach to Reasoning about the Assurance of Safety Arguments	50
8	Ayoub et al. (2012)	A Systematic Approach to Justifying Sufficient Confidence in Software Safety Arguments	47
9	Ayoub et al. (2012)	A safety case pattern for model-based development approach	46
10	Wagner et al. (2010)	A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models	45
11	Picardi et al. (2019)	A Pattern for Arguing the Assurance of Machine Learning in Medical Diagnosis Systems	41
12	Palin et al. (2010)	Assurance of Automotive Safety - A Safety Case Approach	40
13	Burton et al. (2019)	Confidence Arguments for Evidence of Performance in Machine Learning for Highly Automated Driving Functions	36
14	Hawkins et al. (2010)	A Systematic Approach for Developing Software Safety Arguments	34
15	Sljivo et al. (2014)	Generation of Safety Case Argument-Fragments from Safety Contracts	32
16	Matsuno et al. (2011)	Parameterised Argument Structure for GSN Patterns	32

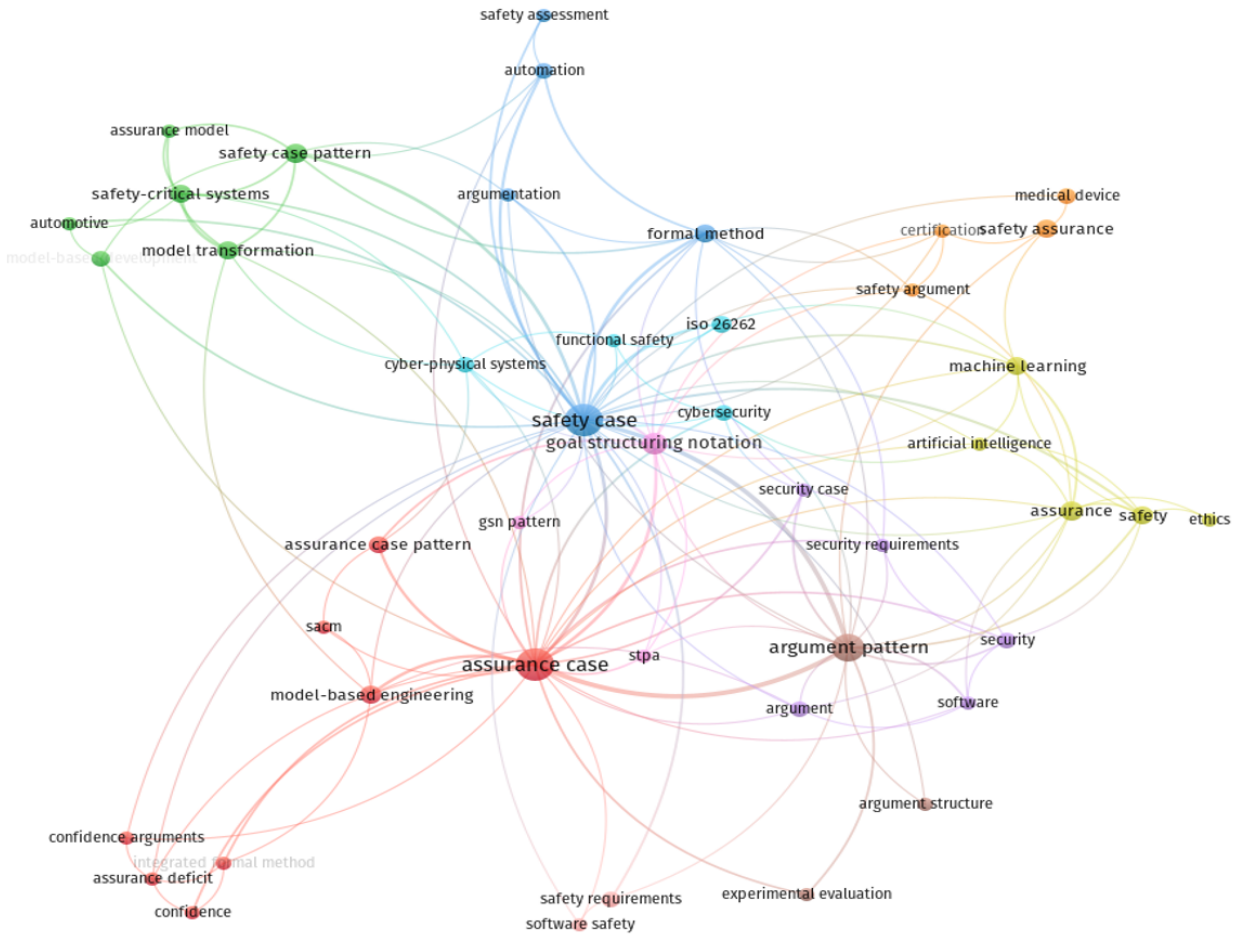

Figure 5.4: Author Keyword Analysis Chart

In a keyword network, a node represents a keyword, and the bigger the node, the more frequently the keyword appears. A link between two nodes means the co-occurrence of two keywords. Similarly, each color in our keyword network represents a cluster of keywords with a similar theme, and there are ten clusters in Figure 5.4.

In Figure 5.4, we can see the terms ***"safety case"***, ***"assurance case"***, ***"goal structuring notation"***, ***"argument pattern"***, ***"safety case pattern"***, and ***"assurance case pattern"*** as the most prominent keywords taking notable portions in the chart. This is not surprising as the majority of these keywords are present in the search strings used to retrieve relevant primary studies. In addition, these keywords form the fundamental background terms in the field of assurance case patterns. The presence of Goal Structuring Notation (GSN) - a graphical argumentation notation [24] – among these prominent keywords, shows the widespread use of GSN in the representation of assurance cases and assurance case patterns.

In the keyword network, the ***"SACM"*** keyword stands for Structured Assurance Case Metamodel [101]. SACM is a specification defined by the Object Management Group (OMG) for representing structured assurance cases [48]. Wei et al [9], provided an exposition on the usage and robust features of SACM, and the relationship that exists between SACM and other notations like CAE (Claim-Argument-Evidence) and GSN for representing assurance cases. Also, Selviandro [123] proposed an extension of SACM in SACM notation (SACMN) which is a syntax of visual vocabularies and compositional rules for representing assurance cases and assurance case patterns. Both Wei et al. [9] and Selviandro [123] aimed to simplify the complexity of SACM and drive the adoption of its use for representing assurance cases and assurance case patterns.

Other prominent sets of keywords include ***"model-based engineering"***, ***"model transformation"***, ***"model-based development"***, ***"assurance model"***, and ***"automation"***. These keywords provide evidence of the direct link that exists between assurance case patterns and system design models. Numerous studies in assurance case patterns [4, 9, 77] have proposed different approaches to automate the generation of assurance cases from assurance case patterns by instantiating ACPs using extracted information from system design models. Wagner et al. [17] investigated the structure and application of models (functional models, platform models, environment model) as "information and requirement sources to guide the construction of safety cases using ACP along the architecture of a system" in the automotive domain. In another recent study, Hartsell et al. [4] also proposed an automated method for constructing assurance cases for complex cyber-physical systems based on the instantiation of ACP using information extracted directly from a set of interconnected models. Their focus was on generating assurance cases that maintain and ensure traceability for

each argument and evidence used in the assurance case back to the artefact it was sourced from in the system design model.

Keywords such as ***"Safety critical systems"***, ***"Automotive", "Medical device"***, ***"cyber-physical systems"*** embody some application domains and type of systems that the literature on assurance case patterns focuses on. In the automotive domain, assurance case patterns in the form of safety case patterns have been proposed to generate safety cases for cruise control systems [17], braking systems [27, 124], and airbag systems [8]. In the medical domain, we have patterns that provide a reusable argument structure for safety compliance of Generic Patient Controlled Analgesic (GPCA) [25, 26]. Keywords like ***"ISO 26262"***, ***"certification"*** show that assurance case patterns can be used to support the compliance of an industry standard and certification of systems.

Assurance case patterns target different functional requirements to support their assurance. The keywords ***"functional safety"***, ***"security requirement"***, ***"safety requirement"***, ***"Software safety"***, ***"security"***, ***"cybersecurity"*** provide credence to some of the requirements that research in ACP targets. From these keywords, it is evident that safety and security currently rank highest as the requirements supported by assurance case patterns. In addition, keywords like ***"safety assessment"***, ***"confidence arguments"***, ***"assurance deficit"***, ***"confidence"***, ***"integrated formal method"***, ***"formal methods"*** in the red cluster of Figure 5.4 represent another area in the field of ACP that is concerned with formalization of ACP and the use of formal methods in system assurance to eliminate assurance deficits.

Finally, keywords in the yellow cluster are, ***"Machine learning"***, ***"artificial intelligence"***, ***"assurance"***, ***"safety"***, ***"ethics"***. These keywords reflect the emergence of new assurance case patterns to support various emerging machine learning-enabled systems and evolving system requirements, including those related to ethics.

5.1.5 Distribution of Author's Affiliation

Figure 5.5 shows the top institutions with the highest number of authors contributing to the field of assurance case pattern. The University of York ranks first with 65 authors followed by Fortiss Gmbh research institute in Germany with 22 authors. Ranked third in this distribution is Malardalen University in Sweden with 19 authors. This is closely followed by the University of Pennsylvania with 18 authors and Beihang University in China in fifth position with 14 authors. In addition, a noteworthy mention is the SGT/NASA Ames Research Center with 8 authors ranked in the sixth position.

The author affiliation distribution corresponds with Table 5.1 as six publications

from the top 15 publications have their first authors affiliated with the University of York which shows the commitment and dedication of the University of York to the field of assurance case patterns. Also, a key observation is that the USA has the highest number of diverse institutions within this chart. However only one of its institutions emerged in the top five for author's affiliation.

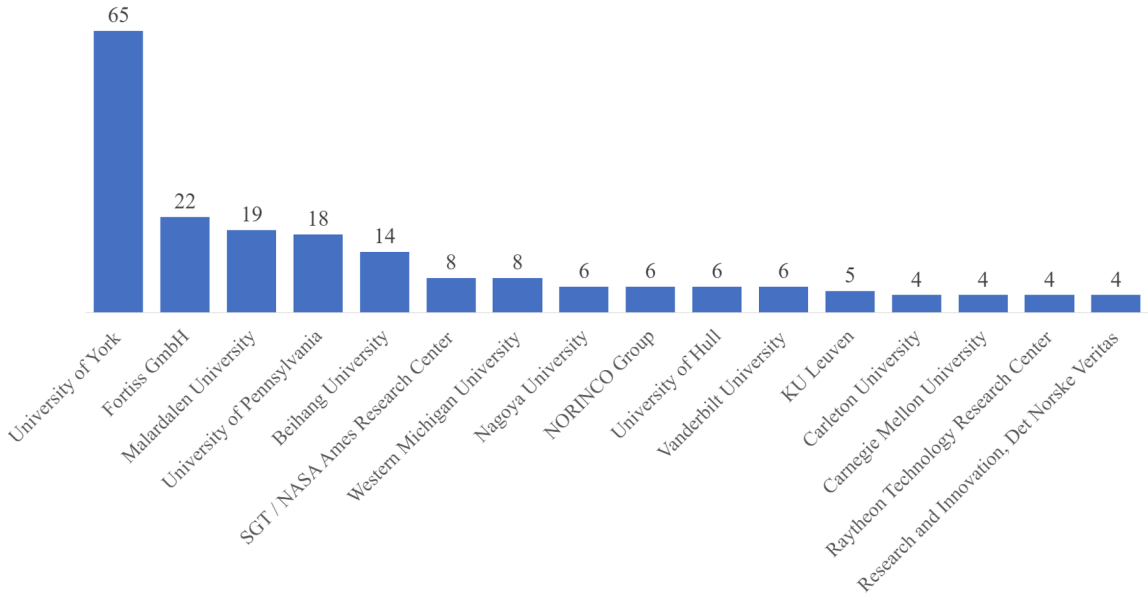

Figure 5.5: Author's Affiliation Distribution

5.1.6 Distribution of Author's Country of Affiliation

Researchers worldwide have tried to expand the body of knowledge in ACP. To better understand this research diversity across countries of the world, we generate a chart to show the distribution of country of affiliations of authors in our study. In Figure 5.6, the United Kingdom emerged first with the highest number of authors in the field of assurance case pattern. This can be attributed to the presence of the University of York a pacesetter, pioneer, and leader in the field of assurance case patterns in the United Kingdom.

The next prominent country in the chart is the United States of America (USA) with 68 authors. This aligns with Figure 5.5, which shows the USA has the highest number of institutions within that chart. Also, agencies like the Food and Drug Agency (FDA) within the USA are concerned with the safety of medical devices, and

NASA which is concerned with the safety of space shuttles and unmanned vehicles might have contributed to the USA being highly ranked in this chart.

Also, the presence of Germany as the third-ranked country could be due to Fortis Gmbh. The latter is a research center that supports safety and security in software and system development. In addition, popular car brands like BMW, and Volkswagen are based in Germany, and from their perspectives, ensuring the safety of automotive vehicles is important.

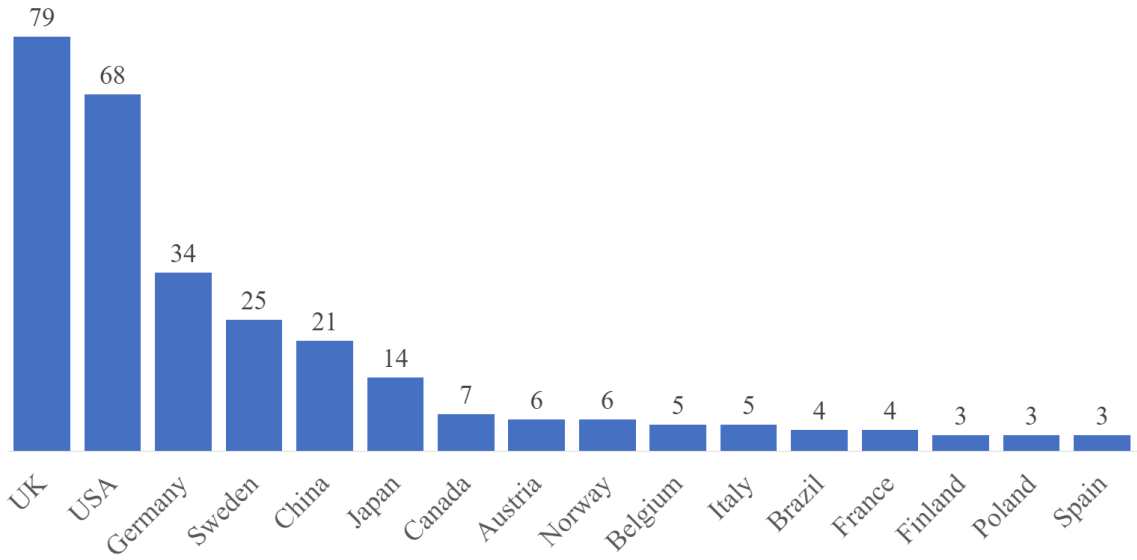

Figure 5.6: Distribution of Author's Country

5.1.7 Geographical Landscape of Assurance Case Pattern

This section focuses on the geographical distributions of the authors' country of affiliation. This analysis allows us to better understand the geographic distribution of researchers contributing to the field of assurance case patterns. We used Tableau [13] to create the world map distribution [2] in Figure 5.7. From the world map, each highlighted area represents an author's country of affiliation.

European countries are the main contributors in the field of assurance case patterns, having 11 countries represented in the top 16 countries. The three other representative

[2]In this Figure, UK = United Kingdom; USA = United States of America; DE = Germany; SE = Sweden; CN = China; JP = Japan; CA = Canada; AT = Austria; NO = Norway; BE = Belgium; IT = Italy; BR = Brazil; FR = France; FI = Finland; PL = Poland; ES = Spain

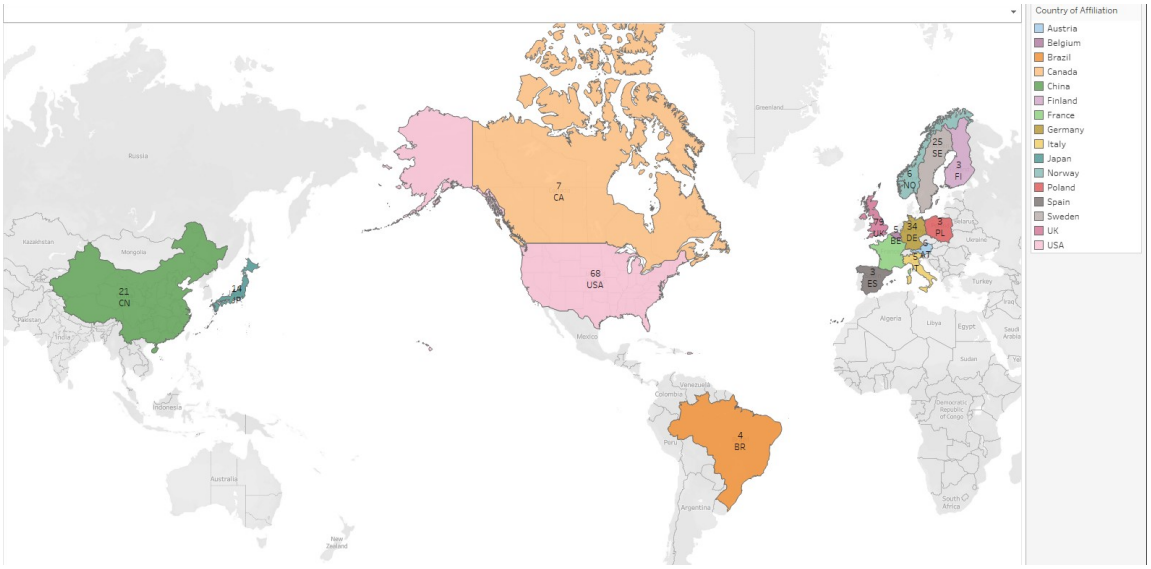

Figure 5.7: World Map Showing Country of Affiliation

continents are North America, Asia, and South America. Only two countries (i.e. Canada and the United States of America) are represented in North America while China and Japan are representatives from Asia. There is only one representation for South America which is Brazil.

The reason for the absence of some countries in Asia, South America, Europe (e.g., Russia), and North America, and the total absence of countries from Africa and Oceania could be due to the language barrier. This makes sense since the studies we analyzed in our bibliometric analysis are exclusively written in English. This absence could also be due to the possibility that researchers with nationalities of these countries are affiliated with organizations in foreign countries prominent in the field of assurance case patterns.

Also, the strong prominence of European countries in the field of assurance case patterns can be attributed to the common standards, policies, and rules that apply automatically and uniformly to all European Union countries. Based on these findings, the geographical location of countries tends to influence the degree of research and development of the field of assurance case patterns. Hence, there is a need for active collaboration and cooperation among researchers from all countries or continents to speed up the development of the field of assurance case patterns, especially in under-represented regions.

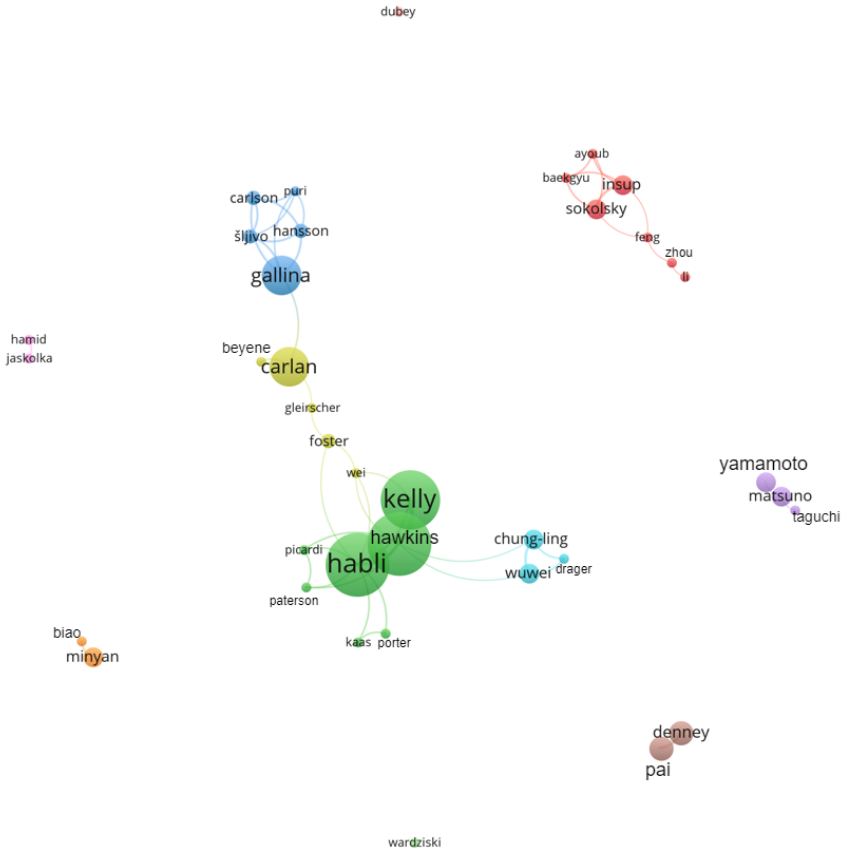

Figure 5.8: Co-author Network

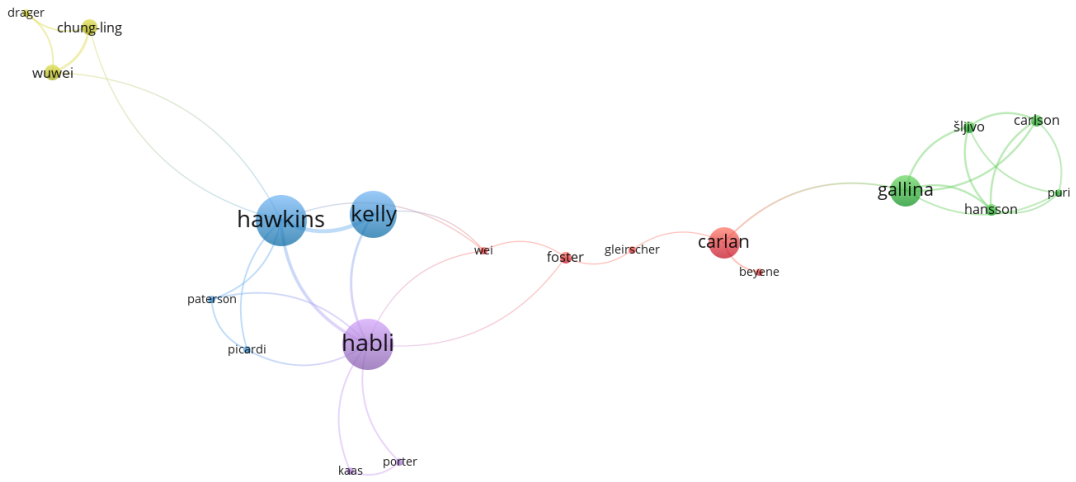

Figure 5.9: Connected Co-author Network

5.1.8 Analysis of Co-Author Network

To analyze the authors in our primary studies by publications made together, we created two co-author networks using the visualization tool VOSviewer to reveal the collaboration pattern of global researchers. We set the minimum number of publications for an author to two. Based on this criteria, 38 authors met this threshold out of a total of 192 authors. Figure 5.8 contains some unconnected nodes representing authors with a minimum of two publications but without collaboration with any of the other 37 authors. Figure 5.9 shows in turn the chart with all nodes connected representing only authors with a collaboration.

In both Figure 5.8 and Figure 5.9, each node represents an author. A link between two nodes represents a co-author relationship. To be specific, the bigger the node, the broader and more contributions made by the author. To obtain a better view of all the authors in our collaboration network, we analyze Figure 5.8 containing all 38 authors. In Figure 5.8, there are various subgraphs with different colors, each subgraph represents a cluster and there are 11 clusters. Authors with the same color show that they are in the same cluster and may have more cooperation with each other. Based on the size of nodes, the most prominent authors with a high number of collaborations and contributions include Richard Hawkins, Ibrahim Habli, Tim Kelly from the University of York, Barbara Gallina from Malardalen University, and Carmen Carlan from Fortiss GmbH research institute.

Cluster 1, marked in red, represents the research team from the University of Pennsylvania. It indicates that the majority of publications in assurance case patterns have been produced by this team, with limited collaboration with authors outside the university. Cluster 2, shown in blue, represents the research team from Mälardalen University. This cluster indicates a connection between Barbara Gallina in Cluster 3 and Carlan in Cluster 4 (Fortiss GmbH), highlighting the collaboration between different organizations. Cluster 3, depicted in green, represents the research team from the University of York. This cluster includes authors with the highest level of collaboration with peers from other universities. Meanwhile, Cluster 5 illustrates the collaborative efforts among Shuichiro Yamamoto (Nagoya University), Yutaka Matsuno, and Kenji Taguchi, all of whom are affiliated with institutions in Japan. A noteworthy observation is Cluster 6, which highlights the collaboration between Ewen Denney and Ganesh Pai, both affiliated with SGT/NASA Ames Research Center. This cluster notably shows no links or collaborations with other authors or affiliations.

Based on this generated co-author chart that Figure 5.8 depicts, the interconnections between nodes and clusters are a bit limited, which suggests that the majority of the authors in assurance case patterns might not be collaborating enough with other authors from other institutions. Hence there is a need to encourage collaborative work to ensure the development of newer approaches and advancement in the field of assurance case patterns.

5.1.9 Evolution of Influential Keywords

To explore the past, present, and future research directions in assurance case pattern research, we utilized a Sankey diagram created with Google Charts [105]. Figure 5.10 depicts this diagram. The Sankey diagram shows the thematic evolution of the assurance case pattern field over the past two decades. The blocks in the diagram represent various paradigms and timelines in the ACP field. As depicted, researchers have explored a wide range of ACP-related themes over the years.

In the Sankey diagram, the early years of 2003 - 2007 show the keywords ***"software safety"***, ***"safety argument"*** and ***"certification"*** which suggests a focus on the use of safety arguments to support the certification of software systems. In the subsequent timeline of 2007 - 2011, the length of the block for ***"assurance case"*** and ***"safety case"*** signifies a wide adoption of assurance cases specifically safety cases to support assurance of system safety requirements. Also, the keyword ***"goal structuring notation"*** (GSN) shows the popularity of GSN as a notation for representing assurances cases which led to its standardization in 2011.

Around 2011 - 2015, the chart shows ***"argument patterns"***, ***"safety case***

pattern", ***"assurance case patterns"*** have become mainstream approaches to support the reuse of argument structures to ease the generation of assurance cases. During this period argument patterns were also proposed to identify and mitigate assurance deficits in safety arguments. Additionally, the keyword ***"security"*** during this timeline indicates the emergence of assurance case patterns that address security as a key property for system assurance. The introduction of the ISO 26262 standard in 2011, which provides functional safety guidelines for the automotive industry, coincides with the focus of ACPs during this period to support and enhance the safety assurance of automotive vehicles.

The period between 2015 - 2019 is characterized by keywords like ***"safety-critical system"***, ***"cyber-physical system"***, ***"security case"***, ***"functional safety"***, ***"safety assurance"*** and ***"model-based engineering"***. These keywords highlight a continuous focus on the safety assurance of safety-critical systems. It also presents the rise in both security and safety assurance of more complex systems, such as cyber-physical systems, developed using model-based engineering. The majority of the studies in assurance case patterns during this period proposed different approaches to extract arguments and evidence from design models of complex systems to instantiate assurance case patterns.

Recently, the timeline of 2019 - 2023, highlights standout keywords such as ***"Medical Devices"***, ***"Machine Learning"***, ***"Artificial intelligence"***,***"Ethics"***, ***"Confidence"***, ***"Cybersecurity"***, ***"Model Transformation"***, and ***"SACM"***. The wide application of artificial intelligence, specifically machine learning in various domains has brought about various systems having a machine learning component. Modern medical devices used in health care are ML-enabled and to this end, assurance case patterns are being proposed to provide the framework to argue assurance of a desired property and improve confidence in medical devices and other ML-enabled systems.

Moreover, the machine learning components of most ML-enabled systems are often black boxes, lacking transparency in their decision-making processes. This has led to emerging ethical considerations for ML-enabled systems, different from traditional safety and security assurance concerns. Additionally, the keyword ***"cybersecurity"*** suggests the emergence of cybersecurity requirement assurance and data privacy, likely driven by the continuous rise of web-based systems. The terms ***"model transformation"*** and ***"SACM"*** signify the current approach of transforming information obtained from system design models into evidence and arguments, which are used to instantiate an assurance case pattern. This process facilitates the automatic creation of model-based assurance cases.

Finally, from the keywords shown in the timeline of 2019 - 2023, we identify some

of the potential future research directions for assurance case patterns.

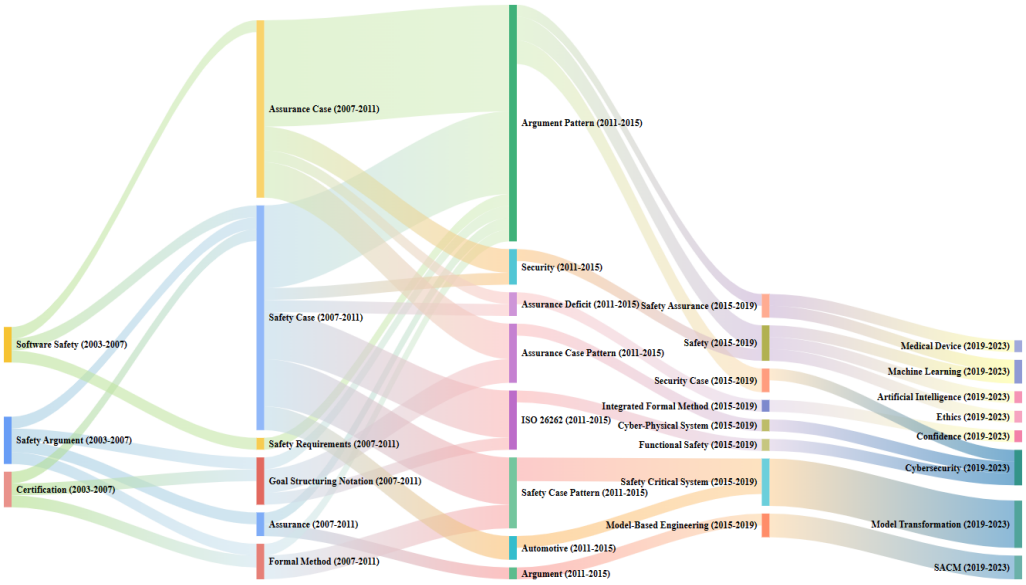

Figure 5.10: Evolution of Influential Keywords in Assurance Case Pattern from January 2003 - October 2023

Fine-tuning assurance case patterns

With the recent advancements in Large Language Models that drive Generative AI, various domains are increasingly utilizing LLMs to solve complex tasks. In software engineering, LLMs are employed for tasks such as software defect prediction [84], automated program repair [86], and code generation [87]. In the field of system assurance, we foresee the application of LLMs in capturing the internal structure of assurance case patterns and the connections between its elements. This could aid the fine-tuning of the argumentation structure of an assurance case pattern for a given system version and facilitate the automatic generation of assurance cases for subsequent system versions. As a result, leveraging LLMs' fine-tuning capabilities, it may become feasible to manually create an assurance case for one version of a system and then automatically infer assurance cases for all subsequent versions. This advancement holds the potential to eliminate the need for manual creation and maintenance of assurance cases as systems evolve.

Assurance case pattern to argue Fairness, Bias, and Explainability in ML-enabled systems

The widespread integration of artificial intelligence, particularly machine learning, across various domains has led to the inclusion of machine learning components in numerous systems. Figure 5.10 underscores the recent emphasis on ensuring the safe and secure performance of these systems, especially in the medical domain where machine learning models are used to diagnose patients and make decisions for medical treatment [15, 14]. However, due to the complex and black-box nature of machine learning components, ensuring confidence in the properties of machine learning-enabled systems is a non-trivial task. Recently, a few studies [125, 126] have proposed the assurance of ethical considerations in machine learning-enabled systems. This suggests a growing need for the development of new assurance case patterns to support and provide sufficient confidence in the assurance of additional properties such as the absence of bias, fairness in decision-making, and clarity in the decision-making process of ML-enabled systems.

Assurance case pattern with Prediction and Advice

Assurance deficits refer to "any knowledge gap that prohibits total or perfect confidence" [48] in an assurance case. Different studies [45, 46] in the literature have proposed the use of assurance case patterns to identify and mitigate these deficits. To measure the confidence in assurance cases created by assurance case patterns, we foresee the development of a "*trust parameter*" which would serve as a measure to predict and estimate the level of confidence in assurance cases generated from assurance case patterns.

5.2 Results of RQ2

5.2.1 On the Ability of LLMs to Create Well-formed and Semantically Correct Assurance Cases when no SE Knowledge is Specified

Metric Results

Table 5.2 reports the median of the Exact match, BLEU scores, and Cosine similarity results we obtained when running each LLM five times in Experiment 1. The standard deviations of these results are very close to zero. This indicates our results are stable across multiple runs. For brevity's sake, we do not report these deviations.

The metric values Table 5.2 reports are extremely low and therefore mediocre. More specifically, the exact match values in Table 5.2 are notably very low, with the highest median exact match value being 0.05. On the other hand, both the BlueROV2 system under both models and the IM software system under GPT-4o yield the lowest median exact match value of 0.02. When it comes to the BLEU scores Table 5.2 reports, we observe that GPT-4o yields the highest median BLEU score value of 0.03 across all runs for the ACAS XU system. Finally, as Table 5.2 shows, the semantic similarity results are slightly better compared to both the Exact match and BLEU score values but remain below average (i.e. 0.5). Note that the ACAS XU system under GPT-4o and DeepMind system under GPT-4o yields the highest semantic similarity median value of 0.23 across all runs in Experiment 1.

Table 5.2: Median Exact Match, BLEU Score, and Semantic Similarity Results for Experiments without SE Knowledge.

System	**Model**	**Exact Match**	**BLEU Score**	**Semantic Similarity**
ACAS XU	GPT-4o	0.04	0.03	0.23
	GPT-4 Turbo	0.03	0.02	0.22
BlueROV2	GPT-4o	0.02	0	0.08
	GPT-4 Turbo	0.02	0.01	0.07
GPCA	GPT-4o	0.04	0	0.04
	GPT-4 Turbo	0.04	0.01	0.11
IM Software	GPT-4o	0.02	0.01	0.09
	GPT-4 Turbo	0.05	0.01	0.13
DeepMind	GPT-4o	0.05	0.01	0.23
	GPT-4 Turbo	0.05	0.01	0.22

Reasons explaining Experiment 1 results

The manual analysis of the LLM-generated assurance cases obtained with Experiment 1 allowed us to identify some reasons explaining the poor results it yields:

- **Lack of a Clear Goal Structure**: In Experiment 1, the majority of LLM-generated assurance cases lack the goal-structured hierarchy required for clarity. These assurance cases list GSN elements in an unclear and unstructured manner, failing to capture the relationships between them. This random organization of elements deviates from the structural rules defined in the GSN standard [24],

causing confusion about the statements in goals, the strategies for achieving these goals, and the supporting evidence.

- **Inconsistent GSN Element Names and IDs:** In some of the LLM-generated assurance cases Experiment 1 yields, there are inconsistencies in GSN element names and the symbols the LLMs use to identify them. For instance, when generating the assurance case for "ACAS XU" in Run 1, GPT-4 Turbo used the same symbol *"S"* to represent both a strategy and a solution, whereas assurance case developers usually use two different symbols/abbreviations to name and distinguish strategies from solutions. Likewise, when generating the assurance case for "DeepMind" in Run 2, GPT-4o used a variety of inconsistent abbreviations (i.e. *"su", "sp", "st", "sd", "sn", "sm", "sl"*) to represent solutions, leading to ambiguity in the assurance cases. Assurance case developers usually use a unique symbol/abbreviation to name solutions.

  Also, when generating the assurance case of BlueROV2 for Run 2, the assurance case that GPT-4o generated included an element called *"Evidence"* in addition to a *"Solution"* element. This can lead to confusion as *"Evidence"* is not a GSN concept. Similarly, when generating the assurance case of the BlueROV2 system for Run 2, GPT-4 Turbo termed an element *"Argument"*, whereas *"Argument"* is not a GSN concept. Likewise, when generating the assurance case for the GPCA system for Run 5, GPT-4 Turbo generated an assurance case that includes an unexpected element called *"Inference"*. Both *"Argument"* and *"Inference"* are not recognized as GSN elements (i.e. concepts) within the GSN standard. Thus, this further contributes to inconsistency and potential misinterpretation of the elements within an assurance case.

- **Absence of GSN Element Identifier:** In some of the LLM-generated assurance cases, there are no element IDs. For instance, when generating the assurance case of "BlueROV2" for Run 2, GPT-4o included in that assurance case a list of elements having no IDs. This absence of unique identifiers can impede the understanding of inferential links between GSN elements. It can also hinder the understanding of how the evidence (through solutions) supports the arguments within an assurance case.

In summary, our analysis of Experiment 1 for **RQ2** reveals that without SE knowledge, LLMs cannot generate reliable assurance cases. The exact match values are very low, with the highest median exact match value being only 0.05. The highest median BLEU score is 0.03, and the best semantic similarity value is 0.23. These values indicate that the generated assurance cases significantly differ from the ground-truth assurance cases, rendering them ineffective for supporting system assurance. This emphasizes the need for incorporating SE knowledge to improve the reliability of LLM-generated assurance cases.

5.2.2 On the Ability of LLMs to Automatically Instantiate Assurance Cases from Assurance Case Patterns by Leveraging SE knowledge

Metric results

Similar to the methodology used in [89, 127], we performed each of our experiments five times (K = 5) for each of the four analyzed systems. Therefore, across the eight experiments involving SE knowledge, both GPT-4o and GPT-4 Turbo generated a total of 320 assurance cases for our test systems. Tables 5.3, 5.4, and 5.5 respectively report the median of the Exact match results, BLEU scores, and Semantic similarity results we obtained across our eight experiments involving SE knowledge (i.e., Experiment 2 to Experiment 9), using the AC and ACP of DeepMind as the one-shot example. Each row in these tables corresponds to a system in our test dataset under both GPT-4o and GPT-4 Turbo. In the eight experiments, the standard deviations associated with metrics values are close to zero, indicating stability in these values. Thus, for brevity's sake, we do not report them. We discuss the results reported in these three tables in the remainder of this section.

Table 5.3 shows that the median Exact match results are quite high for both the ACAS XU and BlueROV2 systems. However, these results are quite low for the GPCA system, while the IM software system has the lowest Exact match values.

As shown in Table 5.4, the median of BLEU score results ranges from moderately high to moderate for both the ACAS XU and BlueROV2 systems. However, the scores are relatively low for both the GPCA system and the IM software system.

Table 5.5 reports the semantic similarity results obtained for the systems in our test dataset. Thus, for each experiment, the semantic similarity results are outstanding for the ACAS XU system and high for the BlueROV2 system. This indicates the assurance cases the LLMs generated for both systems are semantically close to the ground-truth assurance cases. For both the GPCA and IM Software systems, the

Table 5.3: Median Exact Match Result for Experiments with Software Engineering Knowledge.

System	Model	E2	E3	E4	E5	E6	E7	E8	E9
ACAS XU	GPT-4o	0.79	**0.85**	0.84	0.82	0.77	0.74	0.83	0.82
	GPT-4 Turbo	0.65	0.59	**0.83**	0.81	0.56	0.65	0.52	0.69
BlueROV2	GPT-4o	0.65	0.78	0.75	0.75	**0.8**	0.75	0.77	0.71
	GPT-4 Turbo	0.76	0.76	0.66	0.58	**0.81**	0.78	0.64	0.61
GPCA	GPT-4o	0.26	0.3	0.26	0.34	0.22	0.28	**0.35**	**0.35**
	GPT-4 Turbo	0.25	0.31	0.3	0.21	0.23	0.32	**0.32**	0.25
IM Software	GPT-4o	0.1	0.1	0.12	0.12	0.1	**0.28**	0.12	0.27
	GPT-4 Turbo	0.09	0.1	0.07	0.09	0.08	**0.14**	0.08	0.08

Table 5.4: Median BLEU Score Result for Experiments with Software Engineering Knowledge.

System	Model	E2	E3	E4	E5	E6	E7	E8	E9
ACAS XU	GPT-4o	0.73	0.75	0.68	0.68	**0.76**	0.72	0.68	0.64
	GPT-4 Turbo	**0.71**	0.4	0.69	0.6	0.7	0.41	0.67	0.65
BlueROV2	GPT-4o	0.54	0.53	0.55	0.52	**0.57**	0.56	0.55	0.53
	GPT-4 Turbo	0.58	0.46	0.5	0.4	**0.62**	0.56	0.55	0.51
GPCA	GPT-4o	**0.32**	0.28	0.27	0.23	0.21	0.31	0.26	0.23
	GPT-4 Turbo	0.16	0.22	0.19	0.17	**0.25**	0.2	0.22	0.15
IM Software	GPT-4o	0.27	0.27	**0.3**	0.29	0.27	0.29	0.22	0.23
	GPT-4 Turbo	0.17	0.19	0.18	0.16	**0.21**	0.18	0.16	0.17

results vary between moderate and low depending on the type of experiment and the LLM utilized. It is worth noting that the semantic similarity results are relatively higher compared to the BLEU score and Exact match measure for the majority of our experiments. This is probably because the semantic similarity measure mainly

Table 5.5: Median Semantic Similarity Result for Experiments with Software Engineering Knowledge.

System	Model	E2	E3	E4	E5	E6	E7	E8	E9
ACAS XU	GPT-4o	**0.92**	**0.92**	0.91	0.91	**0.92**	0.9	0.91	0.85
	GPT-4 Turbo	**0.91**	0.89	**0.91**	**0.91**	0.9	0.85	0.87	0.9
BlueROV2	GPT-4o	0.88	0.9	0.64	0.67	**0.92**	0.83	0.73	0.59
	GPT-4 Turbo	0.89	0.89	0.57	0.58	**0.9**	0.87	0.53	0.63
GPCA	GPT-4o	**0.81**	0.76	0.37	0.28	0.73	0.76	0.37	0.27
	GPT-4 Turbo	0.56	**0.57**	0.28	0.27	**0.57**	0.55	0.31	0.27
IM Software	GPT-4o	0.7	0.7	0.58	0.59	**0.71**	**0.71**	0.57	0.54
	GPT-4 Turbo	0.65	**0.66**	0.5	0.52	0.65	0.64	0.52	0.5

considers the meaning of the texts associated with the elements comprised in the generated assurance cases. Thus, if two texts have similar meanings but different wording, they can still have a high cosine similarity.

5.2.3 Performance Analysis of Different LLMs for the Automatic Instantiation of Assurance Cases from Assurance Case Patterns

Tables 5.3 to 5.5 highlight in bold the higher median value for a given system-model combination under the different metrics. Evaluating which model has the higher median value for each system-model combination across the different metrics and experiments with SE knowledge shows that GPT-4o achieves 77% of the highest median scores across the various metrics and experiments for the different systems. On the other hand, GPT-4 Turbo achieves 17% of the highest median scores across the various metrics and experiments for the different systems. The remaining 6% represents the single instance where both GPT-4o and GPT-4 Turbo are tied for having the same highest median value in an experiment across the various metrics for the different systems. Based on this analysis, GPT-4o outperforms GPT-4 Turbo in the median values for the Exact Match, BLEU score, and Semantic similarity results across the majority of our experiments for each system.

5.2.4 Effects of Different Software Engineering Knowledge

Comparative Analysis of One-Shot vs Zero-Shot Experiments

The results reported in Table 5.3 to 5.5 show that when the median values are ranked from highest to lowest for a given system-model-metric combination across different experiments (i.e. Experiments 2 to 9), the one-shot experiments (i.e. Experiments 2, 4, 6, 8) tend to achieve higher median values across various metrics compared to the zero-shot experiments (i.e. Experiments 3, 5, 7, 9). Experiment 6, in particular, consistently yields the highest metric values.

Based on these results, we can conclude that the one-shot prompting approach is more effective for the automatic instantiation of assurance cases from assurance case patterns. Providing one example gives the LLMs a specific reference point to learn from, which aids in understanding the pattern instantiation task better and ensures that the generated assurance cases are closer to the ground-truth assurance cases.

Impact of Domain Information

The results presented in Tables 5.3 to 5.5 indicate that experiments incorporating domain information (i.e. Experiments 2, 3, 6, 7) yielded significantly higher median values across the three evaluation metrics compared to those without domain information (Experiments 4, 5, 8, 9). GPT-4o shows significant improvements when domain information is included in its prompts, thus achieving higher scores across all metrics. GPT-4 Turbo also benefits from this inclusion but to a lesser degree.

These findings highlight the crucial role of domain information in maintaining high performance across most metrics, especially semantic similarity, where the highest median semantic similarity values for different system-model combinations were observed in experiments with domain knowledge — with the only exception being Experiment 4 and 5 under GPT-4 Turbo for the ACAS XU system that is tied for highest with Experiment 2.

Impact of Contextual Information

Experiments leveraging contextual information are Experiments 2, 3, 4, and 5. The results reported in Tables 5.3 to 5.5 indicate that experiments with and without context information performed relatively well. However, a notable distinction emerges when ranking the performance of experiments without context information (Experiments 6, 7, 8, 9). Experiments 8 and 9, which also lack domain information, often yield the lowest median of metric values compared to other experiments for a given system-model-metric combination. Interestingly, when counting the frequency or number of

times an experiment performs the best or gives the highest results across all system-model-metric combinations, Experiment 6 (one-shot, without context information, with domain information, and with predicate rules) emerges with the highest frequency of top values across different metrics, models, and systems.

Initially, we expected Experiment 2 (one-shot, with context information, with domain information, and with predicate rules) to yield the highest frequency of top values and be the best experiment overall. This expectation was based on the assumption that having comprehensive context information would significantly enhance the model's performance. However, the results indicate otherwise. Experiment 6 outperformed all other experiments, including Experiment 2, consistently achieving the highest frequency of top metric values.

One possible reason for this outcome could be our predicate-based rules, formulated following the guidelines, elements, and decorators outlined in the GSN Standard. These GSN characteristics also informed our context information. The use of predicate-based rules that comply with the GSN standard may have contributed to the success of Experiment 6, even in the absence of explicit context information. Additionally, it is possible that both GPT-4o and GPT-4 Turbo, have a prior understanding of the information embedded within our context information, and hence, were able to perform effectively despite the lack of explicit context information in Experiment 6.

In conclusion, although incorporating context information was anticipated to enhance performance, its absence, particularly when combined with a lack of domain information, predicate-based rules, and one-shot examples, can lead to poorer results. While context information alone does not drastically affect performance, the presence of predicate-based rules, domain information, and one-shot examples significantly mitigates the negative impact of missing context information.

Impact of Predicate-based Rules

Recall from above that, in each of the eight experiments with SE knowledge, we specify the predicate-based rules in LLMs prompts. The results presented in Tables 5.3 to 5.5 indicate that Experiment 9 (zero-shot, without context information, without domain information, and with predicate-based rules) frequently yields the lowest median values compared to all other experiments when these median values are ranked from highest to lowest for a given system-model-metric combination. This suggests that predicate-based rules are not meant to be utilized alone for the automatic instantiation of assurance case patterns. Rather, they should be used in combination with other categories of SE knowledge (e.g., domain information, context information, and one-shot examples) to ensure the generation of assurance cases that are close to

the ground-truth.

5.2.5 Potential Factors that Could Affect the Performance of LLMs for the Automatic Instantiation of Assurance Cases

Potential reasons for the low results associated with GPCA and IM software are mainly ACP-related and may include the following:

- **Cardinality Ambiguity in Multiplicity Relationship**: In an assurance case pattern, cardinality specifies the required number of instances of a particular element that must be associated with other elements within the pattern. However, the common use of generic labels such as (*"0...*", "1....*", "N"*) to specify the cardinality of the multiplicity relationship in a pattern allows for various interpretations by LLMs. Hence, if the LLM at hand is not able to properly interpret that cardinality, this may lead to mismatches in the number of branches or relationships between elements of the generated assurance case compared to the ones in the ground-truth assurance cases. Also, due to the ambiguity in the cardinality of multiplicity relationships within patterns, we observed that LLMs occasionally generate duplicated GSN elements across the goal structure. For example, the same goal may be generated multiple times with identical IDs and descriptions, or nearly identical descriptions. This may result in a discrepancy between the GSN elements generated by LLMs and the ground-truth GSN elements.

- **Mismatch in Instantiating Abstract Parameters with Multiple Available Values**: In an assurance case pattern, a single abstract parameter can be instantiated with multiple concrete values, depending on the multiplicity within the pattern structure [4]. For elements with generic placeholders that can be replaced with diverse information from the available domain information, a mismatch in the number of branches or relationships between elements of the generated assurance case can occur. This mismatch is caused by the generic label for the cardinality of the multiplicity relationship and may result in a mismatch when instantiating multiple branches with multiple concrete values.

- **The complexity of the pattern:** that complexity might affect the performance and efficiency of the LLMs. For example, the number of placeholders in the assurance case pattern for ACAS XU and BLUEROV2 systems are 10 and 8, respectively while the number of placeholders in the assurance case pattern for

the GPCA system and IM software system are 21 and 9, respectively. This could impact the performance of LLMs in generating ACs close to the ground-truth ACs especially when there is a cardinality ambiguity in the multiplicity relationships in the input assurance case pattern.

Figures C.1 and C.2 (see Appendix) both illustrate the effects of cardinality ambiguity in multiplicity relationships. Figure C.2 illustrates a graphical notation (i.e., a GSN representation) of the assurance case that GPT-4 Turbo generated for the BlueROV2 system. Figure C.1 illustrates a graphical notation of the assurance case that GPT-4o generated for the same system and for the same experiment (i.e., Experiment 2) [3]. To facilitate the discussion and reasoning about these two figures, we compare both figures with the ground-truth assurance case of the BlueROV2 system that Figure B.4 depicts.

The differences between the assurance cases generated by GPT-4o and GPT-4 Turbo for the BLUEROV2 system are significant. Figures C.1 and C.2 highlight these differences: GPT-4o generated an assurance case with 42 GSN elements, whereas GPT-4 Turbo generated an assurance case with only 18 GSN elements. In contrast, the ground-truth assurance case, depicted in Figure B.4, consists of 24 GSN elements.

This discrepancy in the number of GSN elements generated by GPT-4o and GPT-4 Turbo, as compared to the ground-truth assurance case, can be attributed to the ambiguity in the cardinality (*"1....*"*, *"0....*"*, *"1....*"*) of the multiplicity relationships that the pattern in Figure B.3 depicts. This pattern contains the following multiplicity relationships: *HasMultiplicity (S1, G3, 1 of *), HasMultiplicity (S4, A1, 0 of *), HasMultiplicity (G5, G10, 1 of *)*. According to these multiplicity rules, the ground-truth assurance case contains three instances of *Goal (G3)*, three instances of *Assumption (A1)*, and one instance of *Goal (G10)*.

However, as illustrated in Figure C.2, GPT-4 Turbo generated only one instance of each element (G3, A1, and G10), which is fewer than specified in the ground-truth assurance case. On the other hand, GPT-4o generated three instances of each element (G3, A1, and G10), adhering more closely to the ground-truth assurance case. Also, GPT-4o further developed the other two instances of G3 that the ground-truth assurance case left undeveloped. This may explain the higher number of GSN elements (i.e. 42 elements) that GPT-4o generated.

The discrepancy in the number of generated elements between both LLMs and the ground-truth assurance case can thus be partly explained by the ambiguity in

[3]GPT-4 Turbo and GPT-4o generated both figures C.1 and C.2 in structured prose format. Thus, following the GSN standard guidelines [24], we have converted both figures into a graphical notation (GSN).

the cardinality of the multiplicity relationships in the pattern illustrated in Figure B.3. We should point out that, occasionally, both LLMs might overlook removing the pattern decorators after instantiating a given pattern to create an assurance. Removing the pattern decorators is critical because their presence introduces ambiguity about whether each element in the generated assurance case is fully developed and instantiated. This ambiguity can lead to doubts about the completeness and reliability of the assurance case, potentially undermining its use for certification and compliance purposes.

5.2.6 Analyzing Varying One-Shot Example

As stated in Section 4.2.5, to conduct our four one-shot experiments with SE knowledge, we randomly picked DeepMind's assurance case together with its assurance case pattern as an example. Still, in this section, we aim to determine if picking a specific example over another when running one-shot experiments has a significant impact on the quality of the results. For brevity's sake, we only focus in this section on Experiment 2, i.e., the experiment that leverages all the combinations of SE knowledge.

Methodology

To assess the impact of varying the one-shot example on Experiment 2, we adapted the popular Leave One Out Cross-Validation (LOOCV) [128, 129] method to split our example and test data. This ensures the utilization of all systems in our dataset for both the one-shot example and testing data. Given that the number of systems (N) in our dataset is 5, we iterated and picked one system from N to use as a one-shot example while we used the remaining *N-1* systems as test data to validate the LLMs performance. We iterated over N until all systems in N have been utilized as a one-shot example. Besides, we ran Experiment 2, $K{=}5$ times, and we calculated the median metric values across the 5 runs for the different one-shot examples.

Results

Tables 5.6, 5.7, and 5.8 report the comparison of the median metric results the LLMs yield when we vary the one-shot example in Experiment 2 across five runs. Each column under the *"One-shot Example"* header specifies each system used as a one-shot example. Each row under the *"Test System"* header represents each system used as test data. *"Null"* values indicate cases where the same system is meant to serve as both the test and one-shot example, which we do not evaluate. The values highlighted

Table 5.6: Exact Match Comparison of Varying One-Shot Example

Test System	**Model**	**One-Shot Example**				
		ACAS XU	**Blue-ROV2**	**DeepMind**	**GPCA**	**IM Software**
ACAS XU	GPT-4o	Null	**0.87**	0.79	0.78	0.75
	GPT-4 Turbo	Null	0.6	0.65	**0.66**	0.5
BlueROV2	GPT-4o	0.2	Null	**0.65**	0.2	0.21
	GPT-4 Turbo	**0.78**	Null	0.76	0.72	0.75
DeepMind	GPT-4o	0.3	**0.35**	Null	0.27	0.29
	GPT-4 Turbo	0.25	**0.3**	Null	0.28	0.23
GPCA	GPT-4o	**0.31**	0.22	0.26	Null	0.27
	GPT-4 Turbo	**0.28**	**0.28**	0.25	Null	0.24
IM Software	GPT-4o	**0.36**	0.28	0.1	0.29	Null
	GPT-4 Turbo	0.09	0.17	0.09	**0.25**	Null

in bold indicate the highest median value for a given test system across the different one-shot examples.

The results in Tables 5.6 to 5.8 indicate that when analyzing the frequency of the top metric results across the various metrics and test systems, the ACs and ACPs of both GPCA and DeepMind might be the most effective as one-shot examples. For instance, when including counts of ties, the GPCA system emerges as the best one-shot example while when excluding the six tie counts, the frequencies shift, making DeepMind the best one-shot example.

Tables 5.6 to 5.8 also indicate that using the AC and ACP of either ACAS Xu or IM Software as a one-shot example produces the lowest count of top values across all metrics. Specifically, the IM Software as an example fails to achieve any top values under the Exact Match metric while the ACAS Xu system as an example fails to achieve any top values under the BLEU score metric. This indicates that, among all the systems in our dataset, both the IM Software system and ACAS Xu system might be the least effective as a one-shot example.

Table 5.7: BLEU Score Comparison of Varying One-Shot Example

Test System	Model	One-Shot Example				
		ACAS XU	**Blue-ROV2**	**DeepMind**	**GPCA**	**IM Software**
ACAS XU	GPT-4o	Null	0.76	0.73	**0.81**	0.74
	GPT-4 Turbo	Null	0.7	**0.71**	0.69	0.58
BlueROV2	GPT-4o	0.44	Null	**0.54**	0.4	0.4
	GPT-4 Turbo	0.6	Null	0.58	**0.63**	0.58
DeepMind	GPT-4o	0.21	**0.22**	Null	0.17	0.21
	GPT-4 Turbo	0.16	0.19	Null	0.2	**0.22**
GPCA	GPT-4o	0.26	0.23	**0.32**	Null	0.29
	GPT-4 Turbo	0.24	0.28	0.16	Null	**0.3**
IM Software	GPT-4o	0.29	**0.31**	0.27	0.29	Null
	GPT-4 Turbo	0.19	**0.31**	0.17	**0.31**	Null

5.2.7 Are Human Experts Still Needed for Assurance Case Creation in the Age of LLMs?

We have manually assessed the best results the two LLMs produced to determine if they are equivalent to the ones generated by human experts and more specifically to the ones assurance case developers usually create. We further discuss that work below.

Methodology used for the manual assessment

As we stated in Section 5.2.4, Experiment 6 yields the best results for both LLMs. We therefore decided to focus on the manual assessment of that specific experiment result. To manually assess these results, we relied on a metric called ***reasonability*** [73, 88, 22, 127]. A reasonable GSN element is a GSN element that "*could reasonably be in the ground-truth but is not*" [22, 127]. The reasonability metric allows assessing the degree to which the assurance cases generated by LLMs are useful, coherent, and contain GSN elements that are valid but that are not present in the ground truths,

Table 5.8: Semantic Similarity Comparison of Varying One-Shot Example

Test System	**Model**	**One-Shot Example**				
		ACAS XU	**Blue-ROV2**	**DeepMind**	**GPCA**	**IM Software**
ACAS XU	GPT-4o	Null	**0.93**	0.92	**0.93**	0.9
	GPT-4 Turbo	Null	0.9	**0.91**	0.9	0.86
BlueROV2	GPT-4o	**0.89**	Null	0.88	0.86	**0.89**
	GPT-4 Turbo	0.86	Null	**0.89**	0.85	0.85
DeepMind	GPT-4o	0.59	0.59	Null	**0.61**	**0.61**
	GPT-4 Turbo	0.54	0.52	Null	0.55	**0.59**
GPCA	GPT-4o	0.59	0.74	**0.81**	Null	0.67
	GPT-4 Turbo	0.55	0.58	0.56	Null	**0.59**
IM Software	GPT-4o	**0.71**	0.7	0.7	**0.71**	Null
	GPT-4 Turbo	0.65	0.69	0.65	**0.71**	Null

Table 5.9: Average Reasonability Rating of Assurance Cases that Experiment 6 yields

Model	**System**	**Average Reasonability Ratings**	
		Rater 1	**Rater 2**
GPT-4o	ACAS XU	2	2
	BLUEROV2	3.2	2.4
	GPCA	3.8	3.2
	IM SOFTWARE	3	2.4
GPT-4 Turbo	ACAS XU	2.2	2.4
	BLUEROV2	3.4	2.6
	GPCA	3.6	3
	IM SOFTWARE	3.8	3.4

i.e., GSN elements that the human experts have not thought of but that could have enriched the assurance case.

Two researchers (i.e., raters) with strong experience in system assurance and GSN – independently assessed the reasonability of the forty assurance cases the two

LLMs collectively generated for Experiment 6. These researchers utilized a linear scale to assess the reasonability of each of the corresponding assurance cases. In that scale, 1 equals *Totally reasonable*, 2 signifies *Mostly reasonable*, 3 means *Moderately reasonable*, 4 represents *Slightly reasonable*, and 5 denotes *Unreasonable*. To assess the inter-rater reliability, we relied on Kendall's Tau [130] as in the literature (e.g., [88, 22, 91]). Kendall's Tau is a correlation coefficient that varies between - 1 and 1. A value equal to 1 indicates a strong level of agreement between raters. A negative value indicates there is no agreement between raters. As in [22, 91], we relied on an online tool called ***GIGA calculator*** [4] to automatically compute the value of that coefficient with a 95% confidence interval.

Note that to help in the automatic conversion of the assurance cases Experiment 6 yields, we relied on *SmartGSN*[5].

Discussion of the reasonability results

The Kendall's Tau value obtained from reasonability ratings is equal to **0.69** when considering all the ratings the two raters provided for both the assurance cases that both GPT-4o and GPT-4 Turbo generated. This indicates a good agreement between the two researchers who graded Experiment 6 results. This means that the two researchers who graded Experiment 6 results produced consistent, reliable, and similar ratings.

Table 5.9 reports for each rater, for each LLM, and each system, the average of the reasonability ratings. That Table also aggregates these ratings for each LLM at hand. Hence, the averages of the reasonability scores across all the forty assurance cases that Experiment 6 yields and across the two raters are respectively 2.75 for GPT-4o and 3.05 for GPT-4 Turbo. These averages are close to 3, i.e., Moderately reasonable. Thus, the reasonability results demonstrate that LLMs like GPT-4o and GPT-4 Turbo do quite well at generating assurance cases from assurance case patterns. Particularly in one-shot settings such as Experiment 6, where domain information and predicate-based rules are leveraged, these models showcase remarkable proficiency in assurance case generation.

These findings suggest that LLMs can effectively perform a significant portion of the pattern instantiation process, offering invaluable time and resource savings,

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

[5]SmartGSN is the prototype of a web-based tool whose main features include converting assurance cases from structured prose to GSN diagrams. The core technologies used to develop *SmartGSN* are ReactJS and Google FireBase. More specifically, *SmartGSN* relies on React Flow, a customizable React component, for the implementation of its node-based editors and interactive diagrams. (https://smartgsn.vercel.app/)

especially in generating initial drafts of assurance cases — a task that can be laborious, error-prone, and time-consuming when performed manually [4, 131].

However, despite the performance of LLMs in assurance case generation, human expertise remains currently indispensable. Assurance cases often demand adherence to specific standards or regulatory requirements, as well as an understanding of subtle pattern-related nuances (e.g., an inferred cardinality for a multiplicity relationship) that LLMs may not fully grasp or overlook. There is also the potential for LLMs to hallucinate, especially when complex context and domain information are involved, raising concerns about the reliability of generated assurance cases.

To address these limitations, we believe a semi-automatic approach may be more suitable to create assurance cases. In this regard, one can leverage LLMs for the automatic instantiation of assurance case patterns to create initial assurance case drafts. Subsequently, human experts (i.e. assurance case developers) can refine and adjust these drafts, ensuring they meet necessary standards, address potential gaps or inconsistencies, and enhance the overall quality of the assurance process. This is in accordance with Chen et al. [88] who concluded that LLMs are still not able to fully automate the domain modeling task. However, a human modeler can continuously provide feedback to the model to improve the model's output incrementally. This also aligns with Sivakumar et al. [127] who concluded that while LLMs can significantly accelerate the development of safety cases, the expertise and oversight of human safety case developers remain indispensable, particularly for ensuring the highest levels of safety assurance.

In summary, for **RQ2**, our experiments demonstrate that incorporating SE knowledge into LLMs significantly enhances their ability to generate assurance cases that comply with given patterns and closely align with ground-truth assurance cases. Experiment 6, which leverages a one-shot example, domain information, and predicate-based rules, consistently showed superior performance compared to other experiments. For instance, it achieved a median exact match value of 0.81 for the BlueROV2 system and a median BLEU score of 0.76 for the ACAS XU system.

Cardinality ambiguity in multiplicity relationships posed challenges, particularly for the GPCA and IM Software systems, resulting in lower Exact match and BLEU score values across all experiments, with the highest median Exact match for both systems being equal to 0.35 and 0.28 respectively. However, domain information significantly improved performance, as evidenced in experiments incorporating it, where models achieved higher semantic similarity scores. For example, ACAS XU under GPT-4o achieved a score of 0.92 in Experiments 2, 3, and 6, while GPCA saw a score of 0.81 in Experiment 2 under GPT-4o, and IM Software achieved 0.71 in Experiments 6 and 7.

Notably, semantic similarity scores were generally higher compared to Exact match and BLEU scores across most experiments, indicating that while text wordings might differ, the generated assurance cases were still meaningfully close to the ground-truth assurance cases. Comparative analysis also revealed that the one-shot approach generally produced better results than the zero-shot method.

Also, while we initially expected that having comprehensive context information would significantly enhance model performance, experiments without explicit context information but with predicate-based rules, domain information, and one-shot examples still performed exceptionally well. Experiment 6, which lacked context information yet incorporated these other categories of SE knowledge, frequently outperformed experiments that included context information.

Finally, incorporating various categories of SE knowledge helps achieve reliable and effective LLM-generated assurance cases that comply with given patterns, while addressing complexities like ambiguous cardinality in patterns further ensures that LLM-generated assurance cases closely align with ground-truth cases.

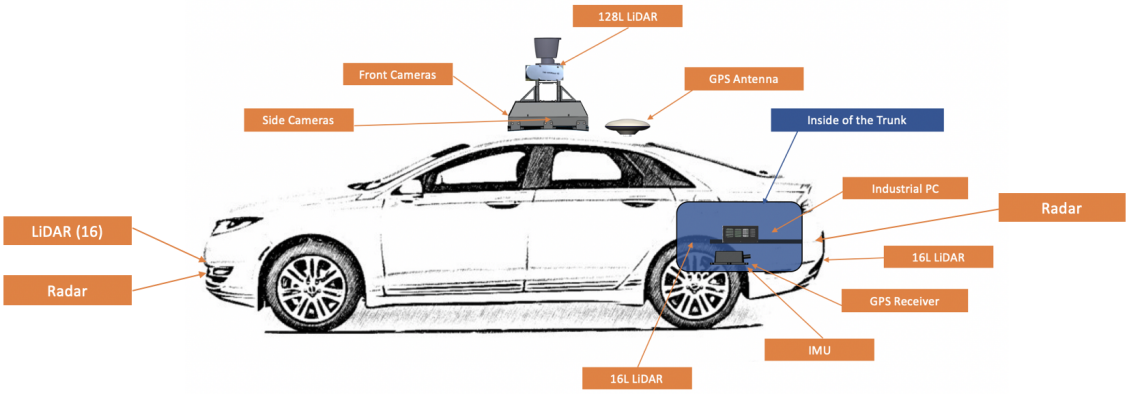

Figure 5.11: Baidu Apollo Vehicle Overview taken from [132]

5.3 Results of RQ3

5.3.1 Case study: Manual Construction of a Safety Case for Baidu Apollo

The Baidu Apollo open-source ADS consists of several components, each containing smaller sub-components that utilize ML models to make critical decisions (e.g., traffic light detection, collision avoidance, and trajectory prediction). Figure 5.11 illustrates the hardware overview of the Baidu Apollo autonomous driving system.

In this work, we focus on Baidu Apollo version 9.0. Baidu Apollo is developed using multiple programming languages, with C++ being the main one, accounting for nearly 76% of its codebase, including approximately 4.3k C++ files. The entire system consists of over 6k files and incorporates over 28 ML models across its various components [132, 133]. Baidu Apollo is recognized as one of the most advanced ADSs and is widely utilized in both academic and industrial settings.

In this case study, we focus on manually creating a safety case for the ML-enabled trajectory prediction component of Baidu Apollo. To create that safety case, we first relied on several information sources freely available online and describing that ADS: Baidu Apollo GitHub project [132], and scientific papers (i.e. [133], [134], [135], [136]). Then we applied Sivakumar et al. [30]'s methodology. Thus, in this section, we describe all the activities and artefacts resulting from the application of that methodology to the trajectory prediction component of Baidu Apollo. The instantiated AMLAS patterns, along with the arguments, activities performed, and the evidence gathered, collectively form the safety case for that component.

Phase III of the design methodology [30] focuses on change impact analysis. In this paper, we do not complete this phase because we do not have access to the real-time driving and simulation data needed for a comprehensive change impact analysis. Hence, we generate a static safety case.

5.3.2 Phase I: Hazard Analysis and Risk Assessment (HARA)

When completing the HARA process, we defined five key system functions of the trajectory prediction component as follows:

1. SF1: Predict the future position of an obstacle.
2. SF2: Outputs the probability value for a predicted obstacle trajectory.
3. SF3: Select the most appropriate machine learning model for prediction.
4. SF4: Identifies the scenario of an obstacle.
5. SF5: Assigns a priority to an obstacle.

To determine system malfunctions, we applied the guide word *"Wrongly"* combining it with each system function to derive potential malfunctions, which are listed below.

1. MF1: Wrongly predict the future position of an obstacle
2. MF2: Wrongly outputs the probability value for a predicted obstacle trajectory
3. MF3: Wrongly selects the most appropriate machine learning model for prediction
4. MF4: Wrongly identifies the scenario of an obstacle.
5. MF5: Wrongly assigns a priority to an obstacle.

Based on reference literature [132, 133, 134] on Baidu Apollo's trajectory prediction component, we defined several operational scenarios:

1. OS1: A vehicle in Cruise scenario
2. OS2: A vehicle in Junction scenario

3. OS3: Obstacle is a Pedestrian
4. OS4: Obstacle is a Vehicle
5. OS5: Obstacle priority is set as Ignore
6. OS6: Obstacle priority is set as Caution
7. OS7: Obstacle priority is set as Normal

At the end of our HARA process, we defined the following four safety goals, which form the basis of our system safety requirements for the Baidu Apollo autonomous driving system.

1. SG1: Rear-end collisions due to miscalculated obstacle halt or acceleration shall be prevented.
2. SG2: Collision with obstacles at an intersection shall be prevented
3. SG3: Collision due to wrong obstacle priority shall be prevented.
4. SG4: Collision with obstacles wrongly classified shall be prevented.

The detailed results of our HARA process are available on Github[6].

5.3.3 Phase II: Implementation of Selected AMLAS Stages

Stage 1: ML Safety Assurance Scoping

We now provide and describe each input artefacts, output artefacts and activities performed in this stage.

1. Input Artefacts
    - Artefact [A] – System safety requirements:
      We derived the system safety requirements from the safety goals established during the hazard analysis and risk assessment process. These requirements are as follows:

[6] https://github.com/Oluwafemi17/LLM-Based-Safety-Case-Generation-for-Baidu-Apollo-Are-We-There-Yet-/blob/main/Hazard_Analysis_Risk_Assessment.xlsx

  - SR1: The Ego vehicle will automatically initiate deceleration to prevent rear-end collisions upon the detection of sudden stops or unexpected changes in acceleration from nearby obstacles.
  - SR2: The Ego vehicle will halt at intersections if the predicted trajectory indicates potential collision with cross-traffic or obstacles.
  - SR3: The Ego vehicle will dynamically adjust the priority of obstacles using real-time environmental data to prevent collisions that may result from incorrect obstacle priority settings.
  - SR4: The Ego vehicle will utilize appropriate ML models to accurately detect and classify obstacles, preventing collisions due to misclassification.

- Artefact [B] - Description of Operating Environment of System

  The Baidu Apollo autonomous driving system operates within a complex environment composed of several critical elements. The road and traffic infrastructure are defined by High Definition (HD) maps, which provide details of road layouts, lane markings, traffic signs, traffic lights, and crosswalks. To achieve accurate perception of the surrounding environment, the system employs a diverse array of sensors and localization tools, including cameras, LiDAR, and radars. These are complemented by the Global Navigation Satellite System (GNSS), which ensures precise localization capabilities.

  The operating environment includes a variety of traffic situations, such as highway cruising and intersection navigation while adhering to traffic laws and regulations. We make the following assumptions about the operating environment: that perception and localization data used for trajectory prediction remain accurate and current; obstacles within the environment are classified as either vehicles or pedestrians, and all obstacles act independently.

- Artefact [C] - System Description

  The Baidu Apollo autonomous driving system is structured around five key components: Localization, Perception, Prediction, Planning, and Control. The Localization component automatically determines the exact position of the Ego vehicle, gathering relevant data about the road and surrounding environment. It employs vehicle cameras, Global Navigation Satellite Systems (GNSS), and High Definition (HD) Maps, which provide road details within a designated area to ensure safe navigation. Once the Ego vehicle's location is identified on the HD map, the ADS leverages the

Perception component, with sensors like cameras, LiDAR, and radars, to detect surrounding obstacles and dynamic objects, measuring their distance from the Ego vehicle. It employs machine learning models for the processing and classification of these detected objects. Based on the perception information, the ADS uses the Trajectory prediction component to predict the future trajectories of all detected obstacles. The motion planning component then uses the compiled information from the localization, perception, and prediction components to generate a smooth path according to different driving scenarios, and sends this to the control component, which then controls the Ego vehicle's movement relative to its environment to avoid collisions. Figure 5.12 shows an excerpt of the structure of the Baidu Apollo ADS highlighting its five major components.

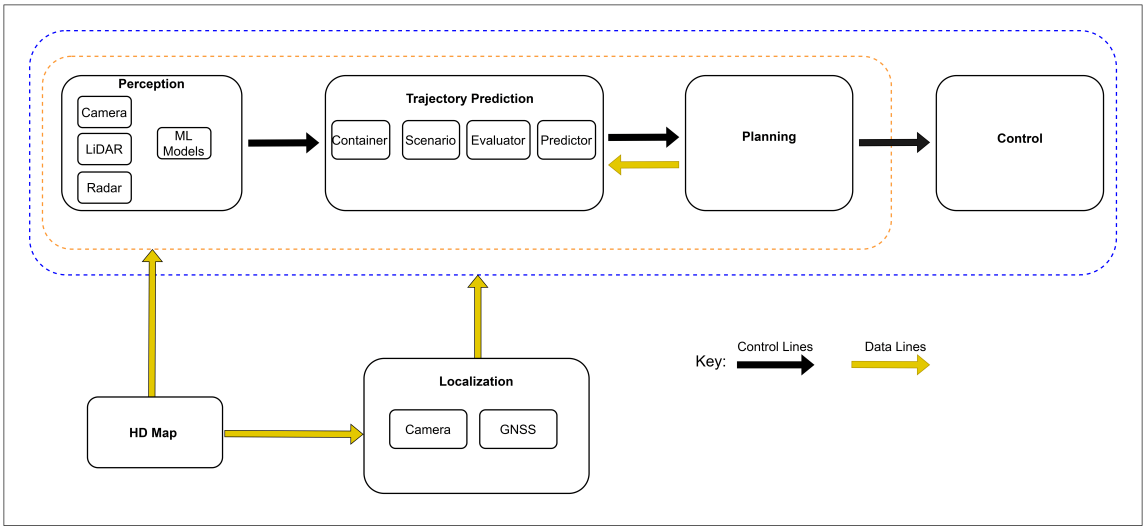

Figure 5.12: Major Components of the Baidu Apollo Autonomous Driving System Software Adapted from [132]

- Artefact [D] - ML Component Description

  The trajectory prediction component of the Baidu Apollo ADS predicts the behavior and future positions of all obstacles (e.g., vehicles, pedestrians) identified by the Perception component. It relies on information such as road boundaries, lane markings, traffic light locations and statuses, crosswalks, distances, velocities, and accelerations to generate predicted trajectories with probability estimates for these obstacles. The Trajectory

Prediction component consists of four sub-components: Container, Scenario, Evaluator, and Predictor. These sub-components contain various machine learning models designed for specific tasks or unique scenarios to predict obstacle trajectory probabilities. The trajectory prediction component's configuration file specifies these ML models and automatically selects the most suitable model for a task based on the perception information. The *Container* sub-component stores all perception and localization information, including data on the surrounding environment and obstacle history. The *Scenario* sub-component analyzes all possible scenarios involving the Ego vehicle, which include cruise or junction scenarios. The *Evaluator* sub-component evaluates a path and speed for any given obstacle by outputting a probability for it using a suitable model depending on the scenario type and obstacle priority. The *Predictor* sub-component is used to generate the predicted trajectories for all obstacles.

- Artefact [F] - ML Assurance Scoping Argument Pattern
  Figure 5.13 presents the ML assurance scoping argument pattern adapted from AMLAS [74]

2. Activities Performed

   We utilized the input artefacts ([A], [B], [C], [D]) to determine the safety requirements that are allocated to the trajectory prediction component and instantiate the ML assurance scoping argument pattern ([F]).

3. Output Artefacts: The outputs for this stage include

   - Artefact [E] - Safety Requirements Allocated to ML Component
   - Artefact [G] - Instantiated ML Safety Assurance Scoping Pattern
     The instantiated ML safety assurance scoping pattern is presented in Figure 5.14.

Stage 2: ML Safety Requirements Assurance

1. Input Artefact

   - Artefact [E] - Safety Requirements Allocated to ML Component
     Following the completion of the HARA process for the trajectory prediction component, we identified four safety goals specific to that component.

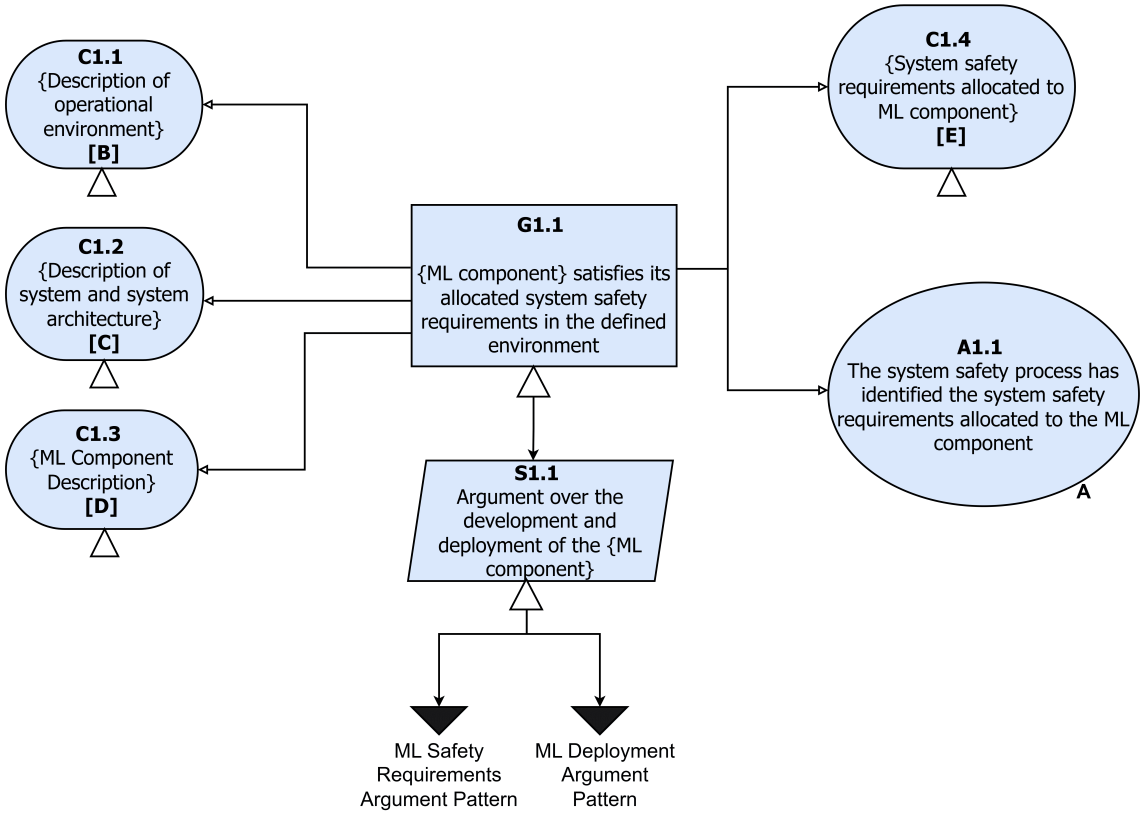

Figure 5.13: ML Assurance Scoping Argument Pattern [F] Adapted from AMLAS [137] and modified for our System

These four safety goals outlined below now serve as the safety requirements allocated to the ML component.

- SG1: Rear-end collisions due to miscalculated obstacle halt or acceleration shall be prevented.
- SG2: Collision with obstacles at an intersection shall be prevented
- SG3: Collision due to wrong obstacle priority shall be prevented.
- SG4: Collision with obstacles wrongly classified shall be prevented.

- Artefact [I] - ML Safety Requirements Argument Pattern

  Figure 5.15 presents the ML Safety Requirements Argument Pattern adapted from AMLAS [74] for our given system.

2. Activities Performed

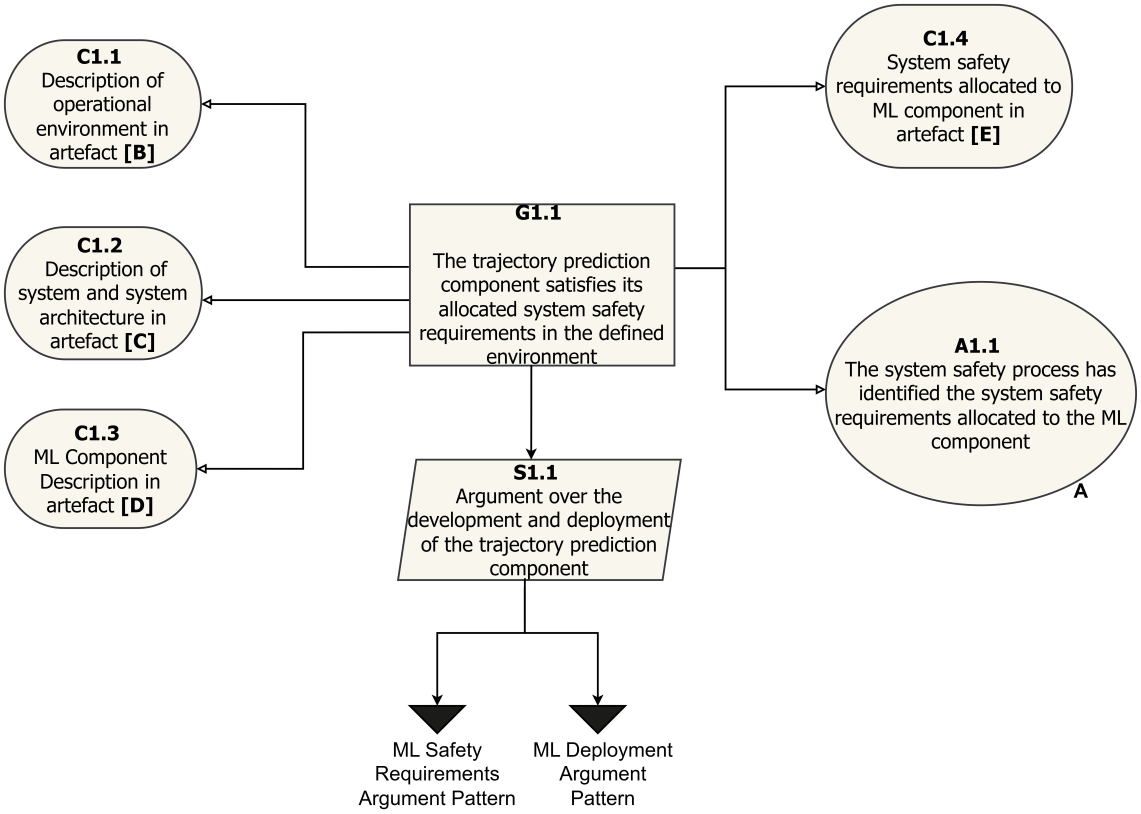

Figure 5.14: Instantiated ML assurance scoping argument [G]

In this stage, we use the output from the previous stage (Stage I), namely the safety requirement allocated to the ML component, and include it as an input in Stage II to develop the actual ML safety requirement. Subsequently, we validate this ML safety requirement against the system safety requirements. We use the results from this validation process, along with other artefacts, to instantiate the ML safety requirement argument pattern. The artefacts used in this stage are listed below.

3. Output Artefact

   - Artefact [H] - ML Safety Requirements
     We have developed the safety requirements for our trajectory prediction component (ML safety requirements) based on the safety requirements allocated to the ML component and its operating environment. Consequently, we have identified the following specific ML safety requirements:

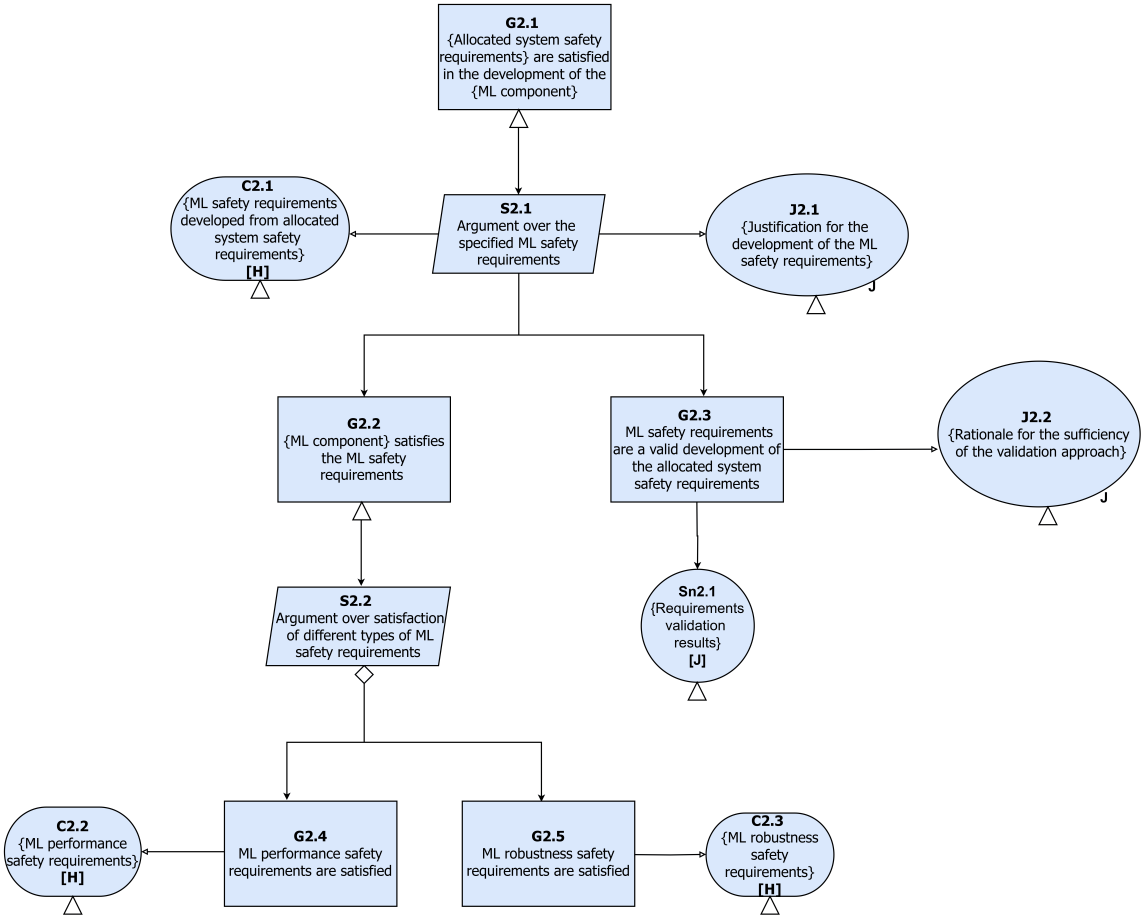

Figure 5.15: ML Safety Requirements Argument Pattern [I] Adapted from AMLAS [137] and modified for our System

- ML-SR1: The Trajectory Prediction component will accurately predict the path, velocity, and acceleration of obstacles to prevent rear-end collisions.
- ML-SR2: The Trajectory Prediction component will provide reliable predictions of obstacle movements at intersections.
- ML-SR3: The Trajectory Prediction component will correctly assign priorities to obstacles based on perceived information from the environment.
- ML-SR4: The Trajectory Prediction component will continuously re-evaluate obstacle trajectories and priorities based on new information

to ensure predictions remain accurate.

Following the approach outlined in AMLAS [74] and implemented in [75, 30], we have derived additional safety requirements focusing on the performance and robustness of the ML component.

ML Performance Requirement:

(a) The trajectory prediction component will accurately predict the future position of an obstacle, maintaining an Average Displacement Error (ADE) threshold of 0.001 meters.

ML Robustness Requirement:

(a) The Trajectory Prediction component will accurately assign priorities to obstacles under various weather conditions.

(b) The trajectory prediction component will ensure high accuracy for both short-term and long-term prediction horizons.

- Artefact [J] - ML Safety Requirements Validation Results

  This artefact presents the outcomes of domain expert reviews conducted to verify the validity of the established ML safety requirements ([H]).

- Artefact [K] - ML Safety Requirements Argument

  This output artefact represents the instantiated ML Safety Requirements Argument Pattern [I] specific to our system. The instantiated pattern is illustrated in Figure 5.16. We instantiated this pattern using artefacts [E], [H], and [J].

Stage 6: Model Deployment

1. Input Artefact: The input artefact needed for this stage includes the following

   - Artefact [A] - System Safety Requirements
   - Artefact [B] - Environment Description
   - Artefact [C] - System Description

     We previously described these three artefacts in stage 1. The additional input artefacts for this stage include

   - Artefact [V] - ML Model

     This artefact describes the list of all ML models, including the MLP, RNN, and LSTM models, used within the trajectory prediction component of

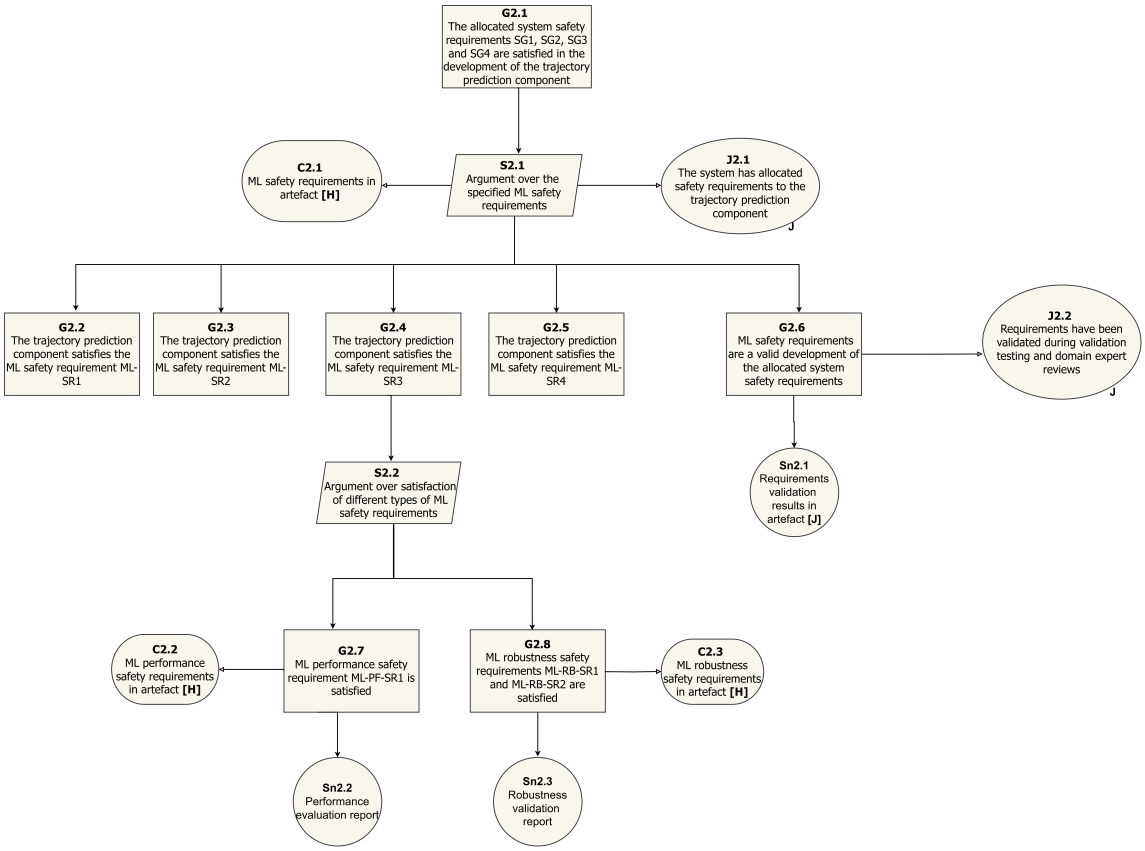

Figure 5.16: Instantiated ML Safety Requirements Argument [K]

the Baidu Apollo ADS. A complete list of these models, along with their specific use cases for various scenarios, obstacle types, and obstacle priority types, can be found in [132, 133].

- Artefact [EE] - Operational scenarios

  The operational scenario describes the set of real scenarios that may be encountered during the operation of the system. [74]. In our case study, we have narrowed down the operational scenarios to the following:

  – OS1: A vehicle in Cruise scenario - when a vehicle is moving along a lane
  – OS2: A vehicle in Junction scenario - when a vehicle approaches a road junction
  – OS3: Obstacle is a Pedestrian

  - OS4: Obstacle is a Vehicle
  - OS5: Obstacle priority is set as Ignore (i.e., Obstacles that do not affect the Ego vehicle's trajectory and can be disregarded)
  - OS6: Obstacle priority is set as Caution (i.e., Obstacles with a high possibility to interact with the Ego vehicle.)
  - OS7: Obstacle priority is set as Normal (i.e., when an obstacle does not fit into the "ignore" or "caution" categories.)
- Artefact [GG] - ML Deployment Argument Pattern

  Figure 5.17 presents the ML Deployment Argument Pattern adapted from AMLAS [74] for our given system.

2. Activities Performed: In this stage, we instantiate the ML Deployment Argument Pattern [GG] to generate the instantiated ML Deployment Argument Pattern [HH].

3. Output Artefact

- Artefact [DD] - Erroneous Behaviour Log

  In our case, we assume that no erroneous behavior is detected in the operation of the trajectory prediction component or its integration with the ADS. Therefore, we do not reference this artefact in the instantiated ML Deployment Argument Pattern.
- Artefact [FF] - Integration Testing Results

  This includes integration test case results indicating satisfactory system safety in the defined operational scenarios after integration with the ML component.
- Artefact [HH] - ML Deployment Argument

  This represents the instantiated ML Safety Requirements Argument Pattern [GG] specific to our system, demonstrating that the safety requirements assigned to the trajectory prediction component are still met when the component is deployed within the ADS. The instantiated pattern is illustrated in Figure 5.18.

Finally, the three instantiated argument patterns derived from the AMLAS stages, along with all the described and referenced artefacts, collectively form the first draft of our safety case for the trajectory prediction component of the Baidu Apollo autonomous driving system.

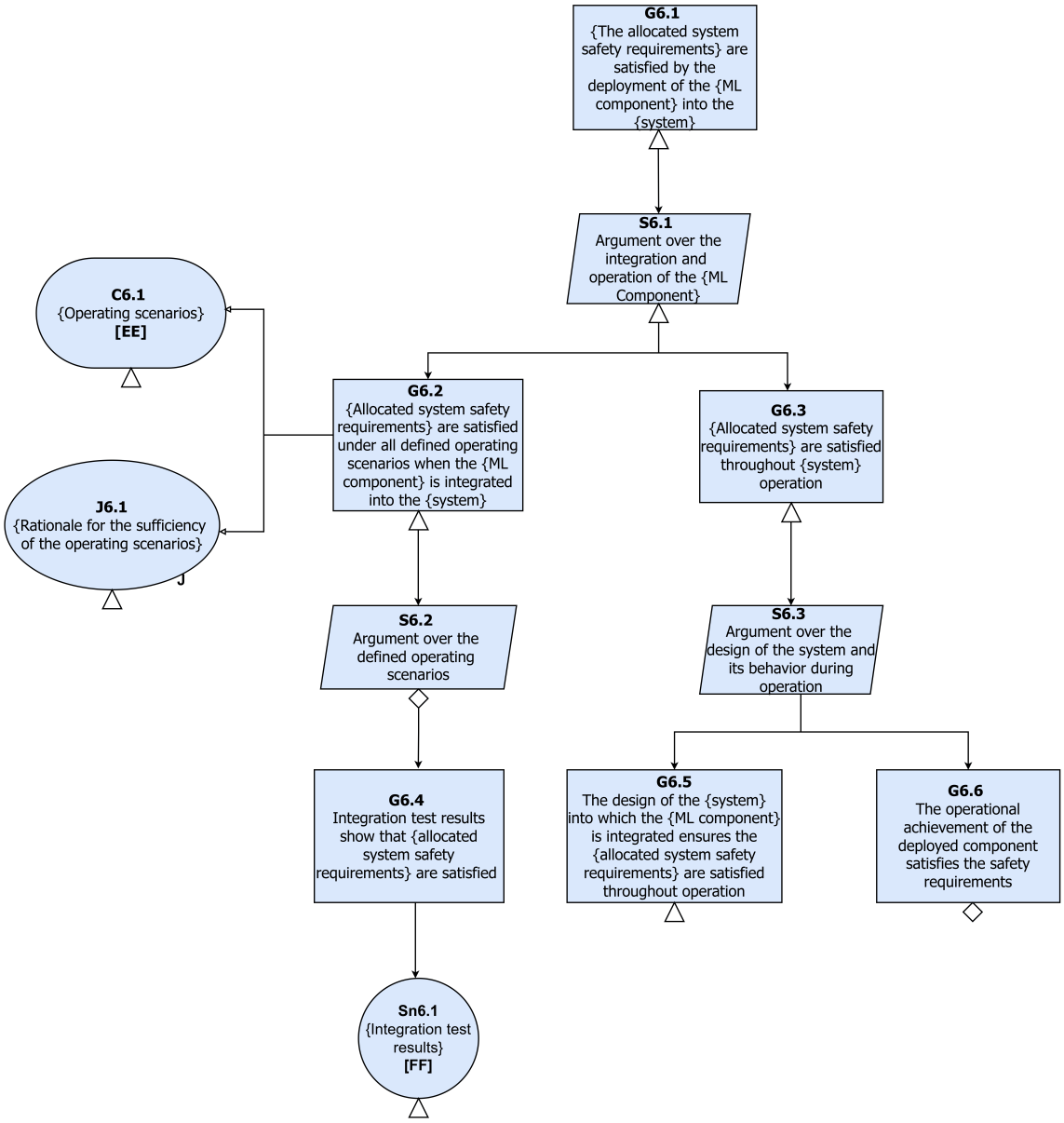

Figure 5.17: ML Deployment Argument Pattern [GG] Adapted from AMLAS [137] and modified for our System

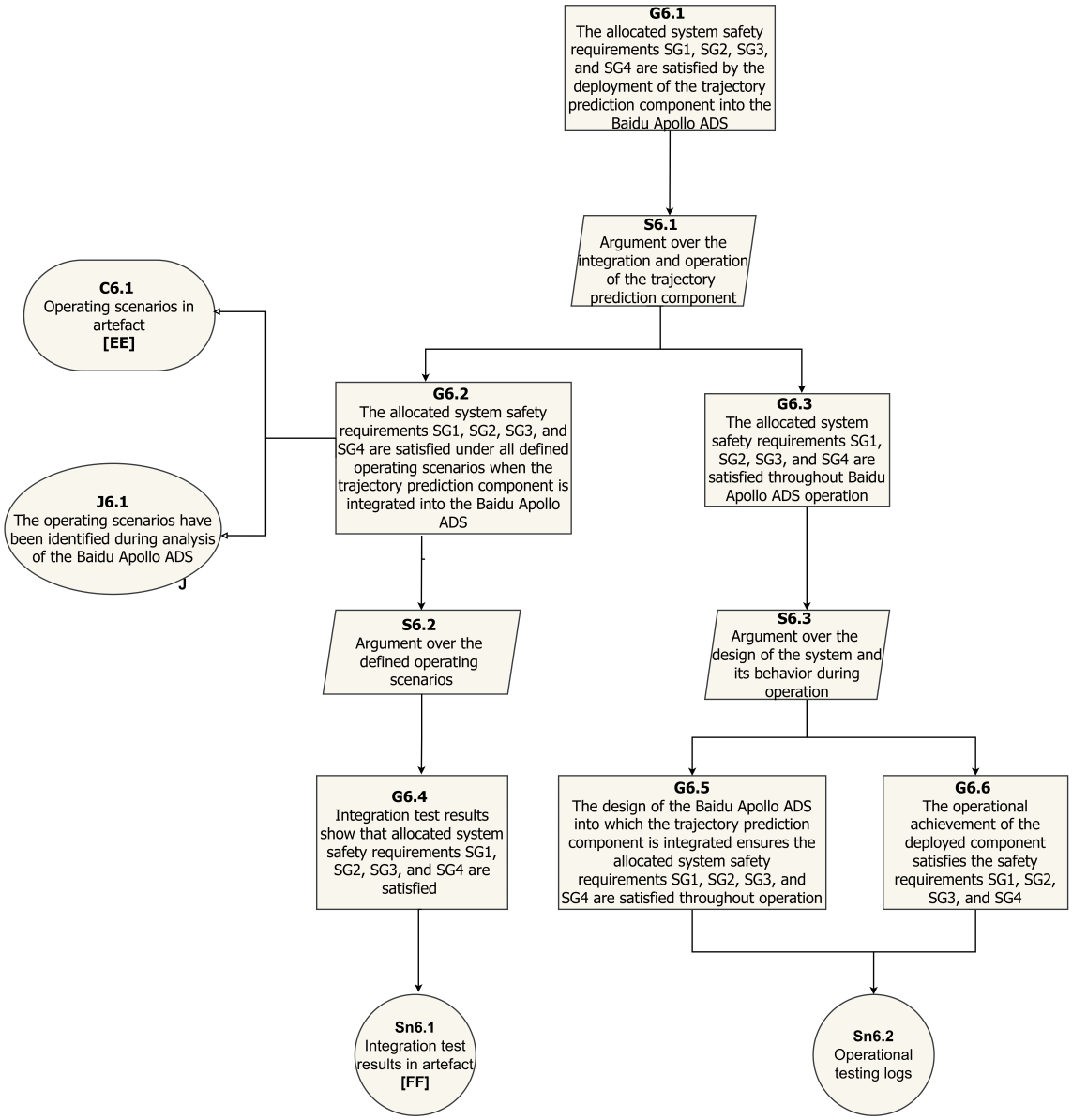

Figure 5.18: Instantiated ML Deployment Argument [HH]

5.3.4 Manual Evaluation and Refinement of the Safety Case

To review the first draft of the safety case, two raters with at least six years of experience in system assurance and/or SE followed the assurance case review guidelines

that the literature (e.g., [138, 139, 32, 24]) recommends. This allowed them to review that safety case by assessing nine review criteria. These criteria are: 1) Argument comprehension; 2) Well-formedness; 3) Expressive sufficiency; 4) Presence of Defeaters and Counter-evidence; 5) Clarity; 6) Atomicity; 7) Ambiguity; 8) Vagueness; and 9) Appropriate use of elements.

During the review process, the two raters independently rated each of these criteria on a linear scale going from 1 to 5, with 1 = Very High, 2 = High, 3 = Average, 4 = Low, and, 5 = Very Low. For each of these criteria, when the associated rating was suboptimal (i.e. higher than 1), the rater also commented on how to refine the safety case by improving that criteria. The Table reporting the nine assessment criteria, the short explanation of these criteria, and the independent ratings made by each of the two raters are available online[7].

To assess the consistency of the ratings made by the two raters, we relied on Kendall's Tau [130]. The latter is a correlation coefficient that varies between - 1 (lack of agreement between raters) and 1 (strong agreement between raters). We used an online tool [8] to automatically compute the value of Kendall's Tau with a 95% confidence interval, using the ratings from the table reporting the assessment criteria the two raters used for the review.

Hence, the value of the Kendall's Tau is **0.44**. This indicates a moderate agreement between the two raters. Besides, the averages of their ratings are respectively 1.33 and 1.22. This results in a combined overall average of **1.28**, which is close to 1 i.e. Very High. Thus, the rating results indicate that the manually created safety case meets most of the assessment criteria. To account for their moderate agreement, the two raters then discussed their ratings and reached a consensus. They then used their so-finalized comments to refine the safety case of the trajectory prediction component of Baidu Apollo. Figure E.2 depicts this refined safety case. It notably consists of a total of 38 GSN elements, including 15 goals, 6 strategies, and 5 solutions.

[7] `https://github.com/Oluwafemi17/LLM-Based-Safety-Case-Generation-for-Baidu-Apollo-Are-We-There-Yet-/blob/main/Safety_Case_Assessment.zip`

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

In summary, for RQ3, our HARA process for the trajectory prediction component identified five system functions and five potential malfunctions. We also identified seven operational scenarios and defined four safety goals, which served as the basis for formulating the system safety requirements for the autonomous driving system.
The implementation of stages 1, 2, and 3 of the AMLAS framework [74] resulted in the creation of 16 artefacts, such as system safety requirements and ML performance and robustness requirements. These artefacts facilitated the instantiation of three argument patterns, leading to a complete safety case for the trajectory prediction component of the Baidu Apollo autonomous driving system.

5.4 Results of RQ4

5.4.1 Case Study: Automatic Creation of the Assurance Case of an ML-enabled Autonomous Driving System

Metric Results

Table 5.10: Exact Match Results for Experiments

Exp	Runs					Mean	Median
	Run1	Run2	Run3	Run4	Run5		
Exp 1	0.34	0.37	0.28	0.38	0.56	0.386	0.37
Exp 2	0.03	0.01	0.01	0.01	0.01	0.014	0.01

Table 5.11: BLEU Score Results for Experiments

Exp	Runs					Mean	Median
	Run1	Run2	Run3	Run4	Run5		
Exp 1	0.43	0.52	0.43	0.48	0.49	0.47	0.48
Exp 2	0.01	0.01	0.02	0	0.01	0.01	0.01

Table 5.10 indicates that the mean and median Exact Match results for Experiment 1 are quite low but significantly better than those of Experiment 2, which are notably extremely low. This indicates a weak match between the LLM-generated safety case and the manually created ground-truth safety case under the Exact match metric for both experiments, particularly for Experiment 2.

Table 5.12: Semantic Similarity Results for Experiments

Exp	Runs					Mean	Median
	Run1	Run2	Run3	Run4	Run5		
Exp 1	0.81	0.83	0.81	0.81	0.84	0.82	0.81
Exp 2	0.22	0.22	0.31	0.15	0.16	0.212	0.22

As shown in Table 5.11, the BLEU scores for Experiment 1 are moderate, with a mean score of 0.47 and a median score of 0.48. In contrast, Experiment 2 scored significantly lower, with both mean and median scores of 0.01.

Table 5.12 shows high semantic similarity results for Experiment 1, with mean and median scores of 0.82 and 0.81 respectively, demonstrating that the LLM-generated safety case for the trajectory prediction component using Experiment 1 is semantically close to the manually created ground-truth safety case. However, Experiment 2 shows a much lower semantic similarity, with mean and median scores of 0.21 and 0.22. This indicates a huge difference between the generated safety cases in both Experiments.

Experiment 1 consistently outperformed Experiment 2 across all three similarity metrics. Hence Experiment 1 yields an LLM-generated safety case that is significantly closer to the ground-truth (i.e. manually generated safety case for the trajectory prediction component). One probable reason for this outcome could be the integration of comprehensive SE knowledge, particularly context information, predicate-based rules, one-shot example, and domain information containing system artefacts generated during our manual safety case creation in section 5.3.3.

In summary, the results for RQ4 demonstrate that Experiment 1 consistently outperformed Experiment 2 across all metrics in terms of both mean and median values. While the Exact Match and BLEU scores yielded low and moderate results respectively for Experiment 1, they were remarkably better than those for Experiment 2, which were extremely low. Most notably, the semantic similarity measure for Experiment 1 was high, demonstrating a close alignment in meaning between the LLM-generated safety case and the manually created safety case for the trajectory prediction component.

Chapter 6

Discussion

6.1 Reflections on the Manual Creation of Assurance Cases

Our approach to manually creating an assurance case began with a thorough hazard analysis and risk assessment. This foundational phase was critical, as it allowed us to identify the Baidu Apollo autonomous driving system's risks and hazards, providing a complete understanding of the system and its components for which the assurance case was being developed.

Manually creating assurance cases from scratch can be tedious and time-consuming. To facilitate this process, we incorporated the AMLAS stages in Phase 2 of our methodology. AMLAS offers structured argument patterns and identifies the necessary artefacts and activities needed to instantiate these patterns, thereby making the manual creation of the assurance case more manageable. Despite the use of an AMLAS-based approach to support that task, we found that gathering the necessary artefacts and performing the required activities required significant domain expertise across multiple disciplines. This expertise encompasses understanding domain-specific jargon, international standards, safety and compliance processes within that domain, and a deep knowledge of the Baidu Apollo ADS, its architecture, and its operating environment. Familiarizing ourselves with all these details and completing the safety case took approximately two months. This timeline highlights the complex and time-intensive nature of developing a robust safety case for the trajectory prediction component of Baidu Apollo.

In addition, we relied on several guidelines from the literature to review and refine the manually created safety case. This review process was very useful and informed

the improvement of that safety case. Thus, we think it would greatly enhance AMLAS to incorporate an explicit phase for reviewing generated assurance cases. This phase could focus on criteria relevant to ML-enabled components in Autonomous Systems, thereby further improving that methodology's robustness.

6.2 Reflections on the Use of an LLM-based Approach to Automatically Create Assurance Cases

Our case study suggests that the use of an LLM can provide significant speed and efficiency as it can quickly process large volumes of data and generate content, allowing for rapid generation of safety cases compared to traditional manual methods. This is advantageous for complex systems like Baidu Apollo's ADS, where the manual creation of an assurance case can be a lengthy process, often taking several months.

Our case study also shows that an LLM may produce inaccurate or inconsistent outputs if not provided with sufficient or relevant information. The generation of a reliable assurance case requires detailed domain-specific knowledge that LLMs may not inherently possess. This point is highlighted in our experiments: Experiment 1, which included key SE inputs outperformed Experiment 2, which lacked them.

To enhance the effectiveness of LLMs in generating high-quality assurance cases, we can consider several approaches. These include incorporating additional categories of relevant knowledge such as results of HARA, details on international standards and compliance with these standards, ethics, and operational context relevant to the system being assessed.

While it may appear advantageous to directly "*teach*" AMLAS-based methodologies to an LLM, doing so would necessitate extensive training datasets specifically tailored to these methodologies. Unfortunately, such datasets are often unavailable due to proprietary and sensitive data concerns, posing a significant challenge to this automation approach.

6.3 Semi-Automatic Creation of Assurance Cases: Human ft. Machine

Our case study suggests that combining human expertise with ML capabilities may offer a promising approach for the semi-automatic creation of assurance cases, leveraging the strengths of both manual and automated approaches. The manual approach which starts with a detailed HARA, followed by identifying necessary artefacts, ac-

tivities, and information, ensures thorough extraction of domain information. This can be effectively integrated with LLMs, which provide speed and efficiency, enabling swift generation of reliable assurance cases.

Still, in safety-critical domains where frequent changes occur and static assurance cases may quickly become obsolete, it could be beneficial to explore the use of a semi-automatic approach to facilitate the creation of dynamic assurance cases and support change impact analysis.

Chapter 7

Threats to Validity

7.1 Threats for RQ1

We adopt the classification of Zhou et al. [140] and Wohlin et al. [141] to discuss the threats to the validity of our work.

7.1.1 Internal Validity

In most bibliometric analyses, hundreds of thousands of primary studies are typically surveyed and analyzed, often due to the less stringent selection strategy employed. While this approach ensures a high volume of primary studies for analysis, it also increases the likelihood of introducing noise in the selected studies.

In our study, we have followed the guidelines proposed in [92, 93] to ensure a stringent selection strategy consisting of six phases to eliminate any occurrence of noisy data in our chosen studies. In addition, we have also employed the snowballing technique to identify additional relevant primary studies not found during the database-driven search across five scientific databases. Hence, our bibliometric analysis ensures completeness in the data search and selection process with minimal occurrence of noisy data.

7.1.2 Conclusion Validity

In our study, the search for primary studies has been done from January 2003 to early October 2023. It is crucial to point out that the search for primary studies ended in early October 2023 which means that our search might miss a few studies published between October 2023 and December 2023.

In addition, we utilized a tool called *Connected Papers* to perform snowballing. However, Connected Papers "*could not find enough papers to create a graph* " for four papers comprised in the start set we used for snowballing. This was possibly due to the recency of these four papers. However, to mitigate this, we have followed all the systematic review and bibliometric analysis guidelines in [92, 93, 94] and ensured the transparency and reproducibility of our work.

7.2 Threats for RQ2

7.2.1 Internal Validity

The dataset used in our experiment consists of six assurance case patterns and five partial assurance cases complying with these patterns. We experienced difficulties in obtaining full assurance cases complying with a given pattern due to the large size of these documents and the sensitive and confidential nature of the information contained in them [142]. This limits the availability of full assurance case patterns and derived assurance cases from these patterns, as they are not readily published or available. To mitigate this, we selected patterns and assurance cases spanning various application domains. We also contacted some of these assurance case developers, which was usually fruitless.

The threat identification pattern that Figure B.1 depicts, and that is part of our dataset, is divided into two parts. The first part is highlighted in red while the second part is highlighted in blue. One potential threat to the validity of our work may emerge from the ground-truth assurance case derived from this pattern. Zeroual et al. [3] in their study, provided only an instantiation of the part highlighted in blue due to brevity's sake. They stated that the placeholder *"System"* in the uninstantiated nodes (C0, G0, G1, G2) in the part highlighted in red would be replaced by "ACAS Xu". To obtain a complete assurance case, we refined their assurance case. Hence, we manually instantiated the elements in the initial part (highlighted in red) of the threat identification pattern by simply replacing the placeholder *"System"* with "ACAS Xu". This manual instantiation of the initial part (elements C0, G0, G1, G2) may pose a validity threat, as it may introduce a potential human error. The latter could impact the consistency and correctness of the so-obtained ground-truth assurance case. This underscores the need for methods that facilitate the automatic instantiation of assurance cases from patterns.

To mitigate the aforementioned threats in future work, we aim to partner with the industry to gain access to full assurance cases derived from a given pattern.

7.2.2 Construct Validity

We extracted the domain information used in our experiments from the following cited references which describe our dataset: [3, 4, 5, 6]. Thus, the assurance cases generated by the two LLMs at hand relied solely on the information available in the cited references. Consequently, key details such as arguments or artefacts related to real-life data or scenarios might have been omitted when deriving domain information. It is also possible that subtle details of the way we formalized the patterns based on the GSN standard might have influenced the results. Future work could try repeating the experiments with small permutations on the way the patterns are expressed.

Also, one potential threat to the validity of our results is the number of runs (K = 5) used in our experiments. By performing each experiment five times, we aimed to capture a sufficient amount of variability and ensure the reliability of our findings. We picked that number following the literature focusing on the use of LLMs to automate software modeling tasks (e.g., [88, 22]). However, this number of runs may not fully reflect the variability in the non-deterministic results generated by our LLMs. To mitigate this threat and enhance the robustness of our findings, we utilized various test systems across different domains and included varied experimental conditions, categorizing different types of SE knowledge. However, in future work, we aim to increase the number of runs to ensure greater confidence in our findings.

7.2.3 Conclusion Validity

The knowledge cut-off date for our two LLMs is 2023. The dataset utilized in our work was published before this date, suggesting that our models' training data might overlap with our dataset, potentially affecting the generalizability of our results. To address this, following Shahandashti et al. [73], we formalized our assurance case patterns in the predicated-based format to obtain representations that both LLMs have never seen before. Still, in future work, we plan to validate the effectiveness of our approach using more recent datasets coming from the industry which might not be publicly available.

7.3 Threats for RQ3

7.3.1 Conclusion validity

In our work, we focused on a single case study: Baidu Apollo. This hinders the generalization of our results to the automotive domain. Still, Baidu Apollo is a

globally recognized commercial ADS and a popular open-source platform. Its features are common to most ML-enabled ADSs and it is widely utilized in both academia and industry.

7.4 Threats for RQ4

7.4.1 Construct validity

To gauge the ability of an LLM to generate a correct safety case, we relied on a safety case that we created ourselves. Even though we thoroughly reviewed our safety case, it has not yet been reviewed by the experts who developed Baidu Apollo. This may hinder the verification of the completeness and usefulness of that safety case. Still, we have used the adaptation of a well-established design methodology to manually create that safety case and followed a very stringent process to review and refine that safety case. Still, in the future, we plan to work closely with Baidu Apollo experts to further review our safety case.

Also, one potential threat to the validity of our results is the number of runs (K = 5) used in our experiments. By performing each experiment five times, we aimed to capture a sufficient amount of variability and ensure the reliability of our findings. We picked that number following the literature focusing on the use of LLMs to automate software modeling tasks (e.g., [88, 22]). However, this number of runs may not fully reflect the variability in the non-deterministic results generated by our LLMs.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In conclusion, this thesis reports a bibliometric analysis of assurance case patterns. This allows for providing a comprehensive overview of the literature on assurance case patterns. Our work therefore aims to aid researchers and practitioners in understanding evolutionary trends, and research structure, and making informed decisions about future directions in assurance case pattern research.

This thesis also reports a study that leverages large language models (LLMs) to support the automatic instantiation of assurance cases, complying with formalized assurance case patterns. In this regard, we conducted extensive experiments across different systems to assess the impact of various categories of software engineering knowledge on the LLMs' performance in generating assurance cases. Our experiment results show large language models can automatically generate relatively good assurance cases when leveraging software engineering knowledge, including knowledge represented in the form of patterns. Still, our experiments also show that we still need human expertise to refine the LLM-generated assurance cases to make them suitable for the certification of mission-critical systems.

A significant part of this thesis also involved manually constructing a safety case for the ML-enabled trajectory prediction component of an open-source autonomous driving system and then employing our LLM-based approach to automatically attempt to generate the same safety case. The results indicated that integrating human expertise with machine capabilities offers a promising method for semi-automatic assurance case creation, effectively combining the strengths of both approaches.

8.2 Future work

For future work, we plan to further validate our approach by expanding our experiments to include the instantiation of a broader range of assurance case patterns across various domains. This will involve utilizing a larger set of LLMs, including both current state-of-the-art models and next-generation models as they become available. By doing so, we aim to assess the adaptability of our approach across different application domains, such as healthcare, cybersecurity, and aerospace industries. This extensive experimentation will provide deeper insights into the strengths and limitations of our approach when applied to distinct types of assurance case patterns and domains.

Also, we intend to conduct a comprehensive user study to evaluate the reliability, well-formedness, and usability of the LLM-generated assurance cases. This will involve collaborating with domain experts and safety engineers to gather qualitative feedback on the quality and relevance of the generated assurance cases. Their insights will help us refine our approach, ensuring they align with the practical needs and expectations of industry professionals who are tasked with creating and maintaining assurance cases.

In addition, we intend to develop an approach for detecting assurance case patterns. This approach will leverage machine learning models to automate the validation and identification of assurance case patterns within complex assurance cases. This will ensure the structural integrity and compliance of these assurance cases with established best practices and support automated consistency checks between a given pattern and an assurance case.

In safety-critical domains (e.g., automotive, aerospace) where frequent changes occur and static assurance cases quickly become obsolete, there is a pressing need for adaptive methodologies. We aim to explore the development and application of a semi-automatic approach that combines the efficiency of LLMs with the expertise of human engineers. This approach will facilitate the creation of dynamic assurance cases that can evolve in response to system updates, regulatory changes, and emerging risks. This dynamic capability is particularly crucial for supporting change impact analysis.

Appendix A

List of Primary Studies and Publication Venue

A.1 List of Primary Studies

Table A.1: List of Primary Studies

No.	Study Title	Author(s) & Publication Year	Venue	Search Method
1	A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models	Wagner et al. (2010)	ISSRE	Database-Driven
2	A design and implementation of an assurance case language	Matsuno Yutaka (2014)	DSN	Database-Driven
3	A formal basis for safety case patterns	Denney & Pai (2013)	SAFECOMP	Database-Driven
4	A framework to support generation and maintenance of an assurance case	Chung-Ling et al. (2016)	ISSREW	Database-Driven
5	A Generic Goal-Based Certification Argument for the Justification of Formal Analysis	Habli & Kelly (2009)	ENTCS	Snowballing
6	A Layered Argument Strategy for Software Security Case Development	Biao et al. (2017)	ISSREW	Database-Driven
7	A method to generate reusable safety case argument-fragments from compositional safety analysis	Šljivo et al. (2017)	JSS	Snowballing
8	A New Approach to Creating Clear Safety Arguments	Hawkins et al. (2011)	SCSC	Snowballing

Continued on next page

Table A.1 – continued from previous page

No.	Study Title	Author(s) & Publication Year	Venue	Search Method
9	A Pattern for Arguing the Assurance of Machine Learning in Medical Diagnosis Systems	Picardi et al. (2019)	SAFECOMP	Database-Driven
10	A Pattern to Argue the Compliance of System Safety Requirements Decomposition	Oliveira et al. (2014)	SugarLoaf-PLoP	Snowballing
11	A Pattern-Based Approach towards the Guided Reuse of Safety Mechanisms in the Automotive Domain	Khalil et al. (2014	IMBSA	Database-Driven
12	A Pragmatic Approach to Reasoning about the Assurance of Safety Arguments	Weaver et al. (2003)	SCS	Snowballing
13	A principles-based ethics assurance argument pattern for AI and autonomous systems	Porter et al. (2023)	Springer AI and Ethics	Database-Driven
14	A safety case pattern for model-based development approach	Ayoub et al. (2012)	NFM	Database-Driven
15	A Safety Case Pattern for Systems with Machine Learning Components	Wozniak et al. (2020)	SAFECOMP	Database-Driven
16	A safety-case approach to the ethics of autonomous vehicles	Menon & Alexander (2020)	Safety and Reliability	Snowballing
17	A Security Argument Pattern for Medical Device Assurance Cases	Finnegan & McCaffery (2014)	ISSREW	Database-Driven
18	A Security Property Decomposition Argument Pattern for Structured Assurance Case Models	Jaskolka et al. (2021)	EuroPLoP	Database-Driven
19	A Systematic Approach for Developing Software Safety Arguments	Hawkins & Kelly (2010)	ISSC	Snowballing
20	A Systematic Approach to Justifying Sufficient Confidence in Software Safety Arguments	Ayoub et al. (2012)	SAFECOMP	Snowballing
21	An Approach to Assure Dependability Through ArchiMate	Yamamoto Shuichiro (2015)	SAFECOMP	Database-Driven
22	An Assurance Case Pattern for the Interpretability of Machine Learning in Safety-Critical Systems	Ward & Habli (2020)	SAFECOMP	Database-Driven
23	An evaluation of argument patterns based on data flow	Yamamoto Shuichiro (2014)	ICT-EURASIA	Database-Driven
24	An evaluation of argument patterns to reduce pitfalls of applying assurance case	Yamamoto & Matsuno (2013)	ASSURE	Database-Driven
25	Applying Safety Case Pattern to Generate Assurance Cases for Safety-Critical Systems	Chung-Ling & Wuwei (2015)	HASE	Database-Driven

Continued on next page

Table A.1 – continued from previous page

No.	Study Title	Author(s) & Publication Year	Venue	Search Method
26	Applying the Goal Structuring Notation (GSN) to Argue Compliance of Equipment with the European EMC Directive	Ghatge et al. (2023)	IEEE Letters on Electromagnetic Compatibility Practice and Applications	Database-Driven
27	Arguing from Hazard Analysis in Safety Cases: A Modular Argument Pattern	Gleirscher & Carlan (2017)	HASE	Database-Driven
28	Arguing on Software-Level Verification Techniques Appropriateness	Carlan et al. (2017)	SAFECOMP	Snowballing
29	Argument patterns for multi-concern assurance of connected automated driving systems	Warg & Skoglund (2019)	CERTS	Database-Driven
30	Argumentation pattern: An approach to issuing software reliability case	Boxuan & Minyan (2014)	EITRT	Database-Driven
31	Assurance argument patterns and processes for machine learning in safety-related systems	Picardi et al. (2020)	SafeAI	Database-Driven
32	Assurance Case Pattern using SACM Notation	Selviandro Nungki (2021)	ICoICT	Database-Driven
33	Assurance Case Patterns for Cyber-Physical Systems with Deep Neural Networks	Kaur et al. (2020)	SAFECOMP	Database-Driven
34	Assurance Cases for Block-Configurable Software	Hawkins et al. (2014)	SAFECOMP	Snowballing
35	Assurance Cases for Proofs as Evidence	Chaki et al. (2009)	PCC	Snowballing
36	Assurance of Automotive Safety - A Safety Case Approach	Palin & Habli (2010)	SAFECOMP	Database-Driven
37	Assuring Safety for Component-Based Software Engineering	Conmy & Bate (2014)	HASE	Database-Driven
38	Automated Method for Assurance Case Construction from System Design Models	Hartsell et al. (2021)	ICSRS	Database-Driven
39	Automating Pattern Selection for Assurance Case Development forCyber-Physical Systems	Ramakrishna et al. (2022)	SAFECOMP	Database-Driven
40	Automotive safety case pattern	Macher et al. (2014)	EuroPLoP	Database-Driven
41	Combining GSN and STPA for Safety Arguments	Hirata & Nadjm-Tehrani (2019)	SAFECOMP	Snowballing
42	Composition of Safety Argument Patterns	Denney & Pai (2016)	SAFECOMP	Database-Driven

Continued on next page

Table A.1 – continued from previous page

No.	Study Title	Author(s) & Publication Year	Venue	Search Method
43	Computer-Aided Generation of Assurance Cases	Wang et al. (2023)	SAFECOMP	Database-Driven
44	Confidence Arguments for Evidence of Performance in Machine Learning for Highly Automated Driving Functions	Burton et al. (2019)	SAFECOMP	Database-Driven
45	Constructing Security Cases Based on Formal Verification of Security Requirements in Alloy	Zeroual et al. (2023)	SAFECOMP	Database-Driven
46	CyberGSN: A Semi-formal Language for Specifying Safety Cases	Beyene & Carlan (2021)	DSN-W	Database-Driven
47	Deriving Safety Case Fragments for Assessing MBASafe's Compliance with EN 50128	Gallina et al. (2016)	SPICE	Snowballing
48	Design and implementation of GSN patterns: A step toward assurance case language	Matsuno Yutaka (2014)	IPSJ	Database-Driven
49	Developing Assurance Cases for D-MILS Systems	Hawkins et al. (2015)	HiPEAC	Snowballing
50	Enabling Cross-Domain Reuse of Tool Qualification Certification Artefacts	Gallina et al. (2014)	SAFECOMP	Snowballing
51	Enhancing state-of-the-art safety case patterns to support change impact analysis	Carlan & Gallina (2020)	ESREL	Database-Driven
52	Enhancing the Cyber Resilience of Critical Infrastructures through an Evaluation Methodology Based on Assurance Cases	Koelemeijer Dorien (2018)	Procedia Computer Science	Database-Driven
53	Ethics in conversation: Building an ethics assurance case for autonomous AI-enabled voice agents in healthcare	Kaas et al. (2023)	TAS	Database-Driven
54	Evidence arguments for using formal methods in software certification	Denney & Pai (2013)	ISSREW	Database-Driven
55	Evolution of Formal Model-Based Assurance Cases for Autonomous Robots	Gleirscher et al. (2019)	SEFM	Database-Driven
56	Experiences with Assurance Cases for Spacecraft Safing	Nguyen & Ellis (2011)	ISSRE	Database-Driven
57	ExplicitCase: Tool-Support for Creating and Maintaining Assurance Arguments Integrated with System Models	Carlan et al. (2019)	ISSREW	Database-Driven
58	Facilitating construction of safety cases from formal models in Event-B	Prokhorova et al. (2015)	Information and Software Technology	Database-Driven
59	General Development Framework and Its Application Method for Software Safety Case	Fuping et al. (2013)	Journal of Software	Database-Driven

Continued on next page

Table A.1 – continued from previous page

No.	Study Title	Author(s) & Publication Year	Venue	Search Method
60	Generation of Safety Case Argument-Fragments from Safety Contracts	Šljivo et al. (2014)	SAFECOMP	Snowballing
61	Hazard Contribution Modes of Machine Learning Components	Colin et al. (2020)	AAAI	Snowballing
62	Identifying and implementing security patterns for a dependable security case - From security patterns to D-case	Patu & Yamamoto (2013)	CSE	Database-Driven
63	Integrated Formal Methods for Constructing Assurance Cases	Carlan et al. (2016)	ISSREW	Snowballing
64	Justifying the transition from trustworthiness to resiliency via generation of safety cases	Chung-Ling et al. ()	SNPD	Database-Driven
65	Justifying the validity of safety assessment models with safety case patterns	Sun et al. (2011)	ISSC	Database-Driven
66	Model Based System Assurance Using the Structured Assurance Case Metamodel	Wei et al. (1999)	JSS	Snowballing
67	Model-based Generation of Hazard-driven Arguments and Formal Verification Evidence for Assurance Cases	Yan et al. (2022)	MODELSWARD	Snowballing
68	Model-Connected Safety Cases	Retouniotis et al. (2017)	IMBSA	Snowballing
69	On using results of code-level bounded model checking in assurance cases	Carlan et al. ()	SAFECOMP	Database-Driven
70	Parameterised Argument Structure for GSN Patterns	Matsuno & Taguchi (2011)	ICQS	Database-Driven
71	Patterns for Integrating NIST 800-53 Controls into Security Assurance Cases	Viger et al. (2023)	SAFECOMP	Database-Driven
72	Preventing recurrence of industrial control system accident using assurance case	Napolano et al. (2015)	ISSREW	Database-Driven
73	Product-line assurance cases from contract-based design	Nei et al. (2021)	JSS	Database-Driven
74	SACS - A pattern language for safe adaptive control software	Hauge et al. (2011)	PLoP	Database-Driven
75	Safe & sec case patterns	Taguchi et al. (2015)	SAFECOMP	Database-Driven
76	Safety argument pattern language of safety-critical software	Wang et al. (2020)	DSA	Database-Driven
77	Software Reliability Case Development Method Based on the 4+1 Principles	Shihao et al. (2018)	ICRMS	Database-Driven
78	Software safety assurance - what is sufficient?	Hawkins & Kelly (2009)	ISSC	Snowballing

Continued on next page

Table A.1 – continued from previous page

No.	Study Title	Author(s) & Publication Year	Venue	Search Method
79	Software Safety Certification Framework Based on Safety Case	Zeng et al. (2012)	CSSS	Database-Driven
80	Structural analysis of safety case arguments in a model-based development environment	Zechner & Huhn (2009)	MBEES	Database-Driven
81	Support for safety case generation via model transformation	Chung-Ling et al. (2017)	ACM SIGBED Review	Database-Driven
82	The Need for a Weaving Model in Assurance Case Automation	Hawkins et al. (2015)	AUJ	Snowballing
83	Tool support for assurance case development	Denney & Pai (2018)	ASE	Database-Driven
84	Tool-Supported Safety-Relevant Component Reuse: From Specification to Argumentation	Šljivo et al. (2018)	AEiC	Snowballing
85	Towards a case-based reasoning approach for safety assurance reuse	Ruiz et al. (2012)	SAFECOMP	Database-Driven
86	Towards assurance for plug & Play medical systems	King et al. (2015)	SAFECOMP	Database-Driven
87	Towards Developing Safety Assurance Cases for Learning-Enabled Medical Cyber-Physical Systems	Bagheri et al. (2022)	SafeAI	Snowballing
88	Towards Goal-Based Software Safety Certification Based on Prescriptive Standards	Stensrud et al. (2011)	WoSoCER	Database-Driven
89	Towards safety case integration with hazard analysis for medical devices	Wardziski & Jarzbowicz (2016)	SAFECOMP	Database-Driven
90	Uniform model interface for assurance case integration with system models	Wardziski & Jones (2017)	SAFECOMP	Database-Driven
91	Using Process Models in System Assurance	Hawkins et al. (2016)	SAFECOMP	Snowballing
92	Weaving an Assurance Case from Design: A Model-Based Approach	Hawkins et al. (2015)	HASE	Snowballing

A.2 Publication Venue and Acronym

Table A.2: Publication Venue Names and Acronyms

Acronym	Publication Venue Name

Continued on next page

Table A.2 – *Continued from previous page*

Acronym	**Publication Venue Name**
AAAI	Association for the Advancement of Artificial Intelligence Conference
AEiC	Ada-Europe International Conference on Reliable Software Technologies
ASE	Automated Software Engineering
ASSURE	International Workshop on Assurance Cases for Software-Intensive Systems
AUJ	Ada User Journal
CERTS	International Workshop on Security and Dependability of Critical Embedded Real-Time Systems
CSE	International Conference on Computational Science and Engineering
CSSS	International Conference on Computer Science and Service System
DSA	International Conference on Dependable Systems and Their Applications
DSN	International Conference on Dependable Systems and Networks
DSN-W	International Conference on Dependable Systems and Networks Workshops
EITRT	International Conference on Electrical and Information Technologies for Rail Transportation
ENTCS	Electronic Notes in Theoretical Computer Science
ESREL	European Conference on Safety and Reliability
EuroPLoP	European Conference on Pattern Languages of Programs
HASE	International Symposium on High-Assurance Systems Engineering
HiPEAC	International Conference on High-Performance Embedded Architectures and Compilers
ICoICT	International Conference on Information and Communication Technology
ICQS	International Conference on Quality Software
ICRMS	International Conference on Reliability, Maintainability, and Safety
ICSRS	International Conference on System Reliability and Safety
ICT-EURASIA	EurAsian Conference on Information and Communication Technology
IMBSA	International Symposium on Model-Based Safety Assessment
IPSJ	Journal of Information Processing
ISSC	International Conference on System Safety
ISSRE	International Symposium on Software Reliability Engineering
ISSREW	International Symposium on Software Reliability Engineering Workshops
JSS	Journal of Systems and Software
MBEES	Model-based development of Embedded Systems Workshop
MODELSWARD	International Conference on Model-Driven Engineering and Software Development
NFM	International Symposium on NASA Formal Methods
PCC	Proof Carrying Code Workshop
PLoP	Conference on Pattern Languages of Programs
SafeAI	The AAAI's Workshop on Artificial Intelligence Safety
SAFECOMP	International Conference on Computer Safety, Reliability, and Security
SCS	Workshop on Safety critical systems and software
SSS	Safety Critical System Symposium
SEFM	International Conference on Software Engineering and Formal Methods
SNPD	International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing
SPICE	International Conference on Software Process Improvement and Capability Determination
SugarLoafPLoP	The Latin American Conference on Pattern Languages of Programs
TAS	International Symposium on Trustworthy Autonomous Systems
WoSoCER	International Workshop on Software Certification

Appendix B

RQ2 Dataset

B.1 ACAS XU System

B.2 BlueROV2 System

B.3 GPCA System

B.4 IM Software System

B.5 DeepMind System

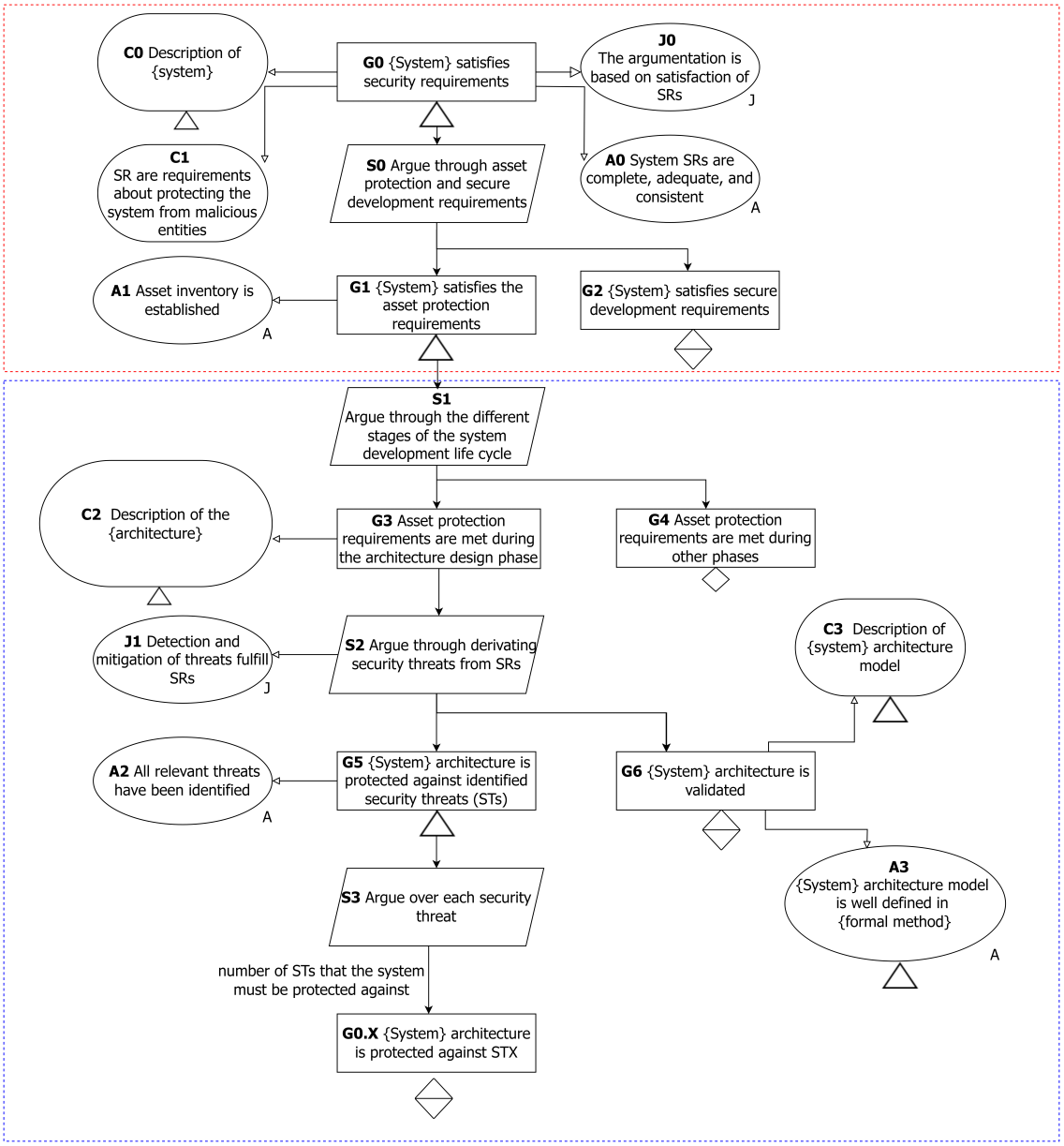

Figure B.1: Assurance Case Pattern for Threat Identification - Adapted from [3]

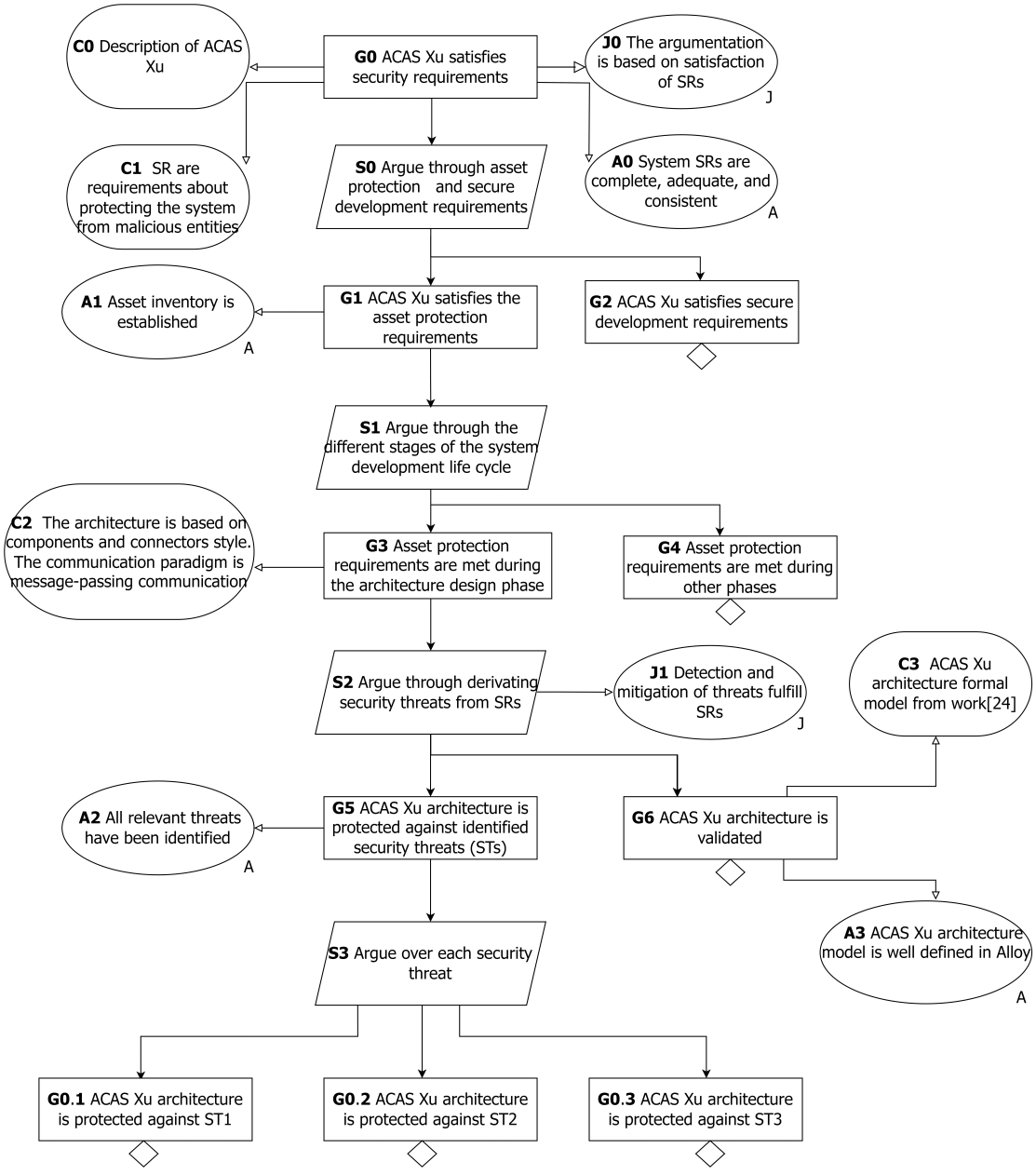

Figure B.2: Assurance Case for Threat Identification - Adapted from [3]

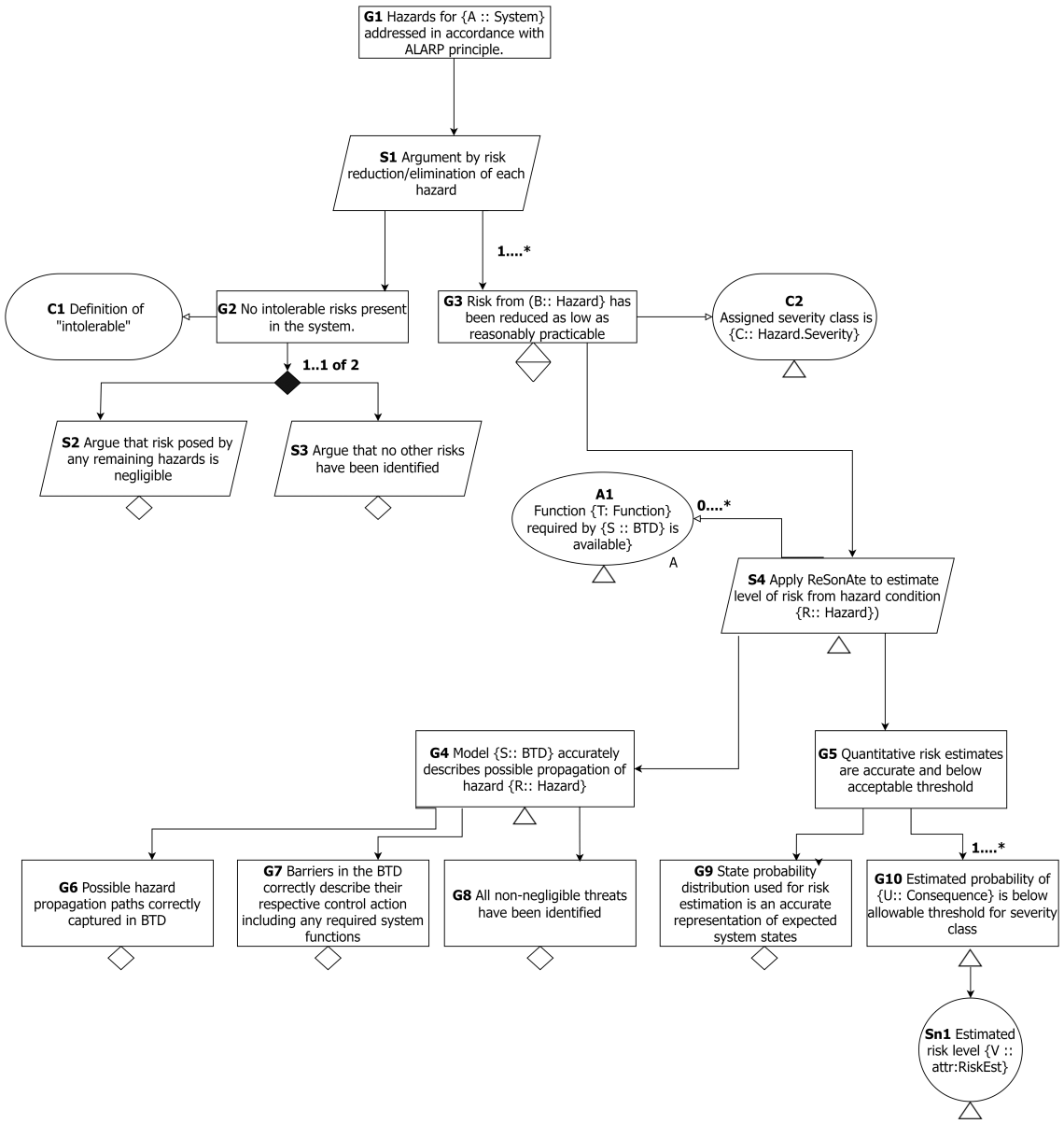

Figure B.3: BlueROV2 Assurance Case Pattern Containing the ALARP Pattern Sequentially Composed with the ReSonAte Pattern - Adapted from [4]

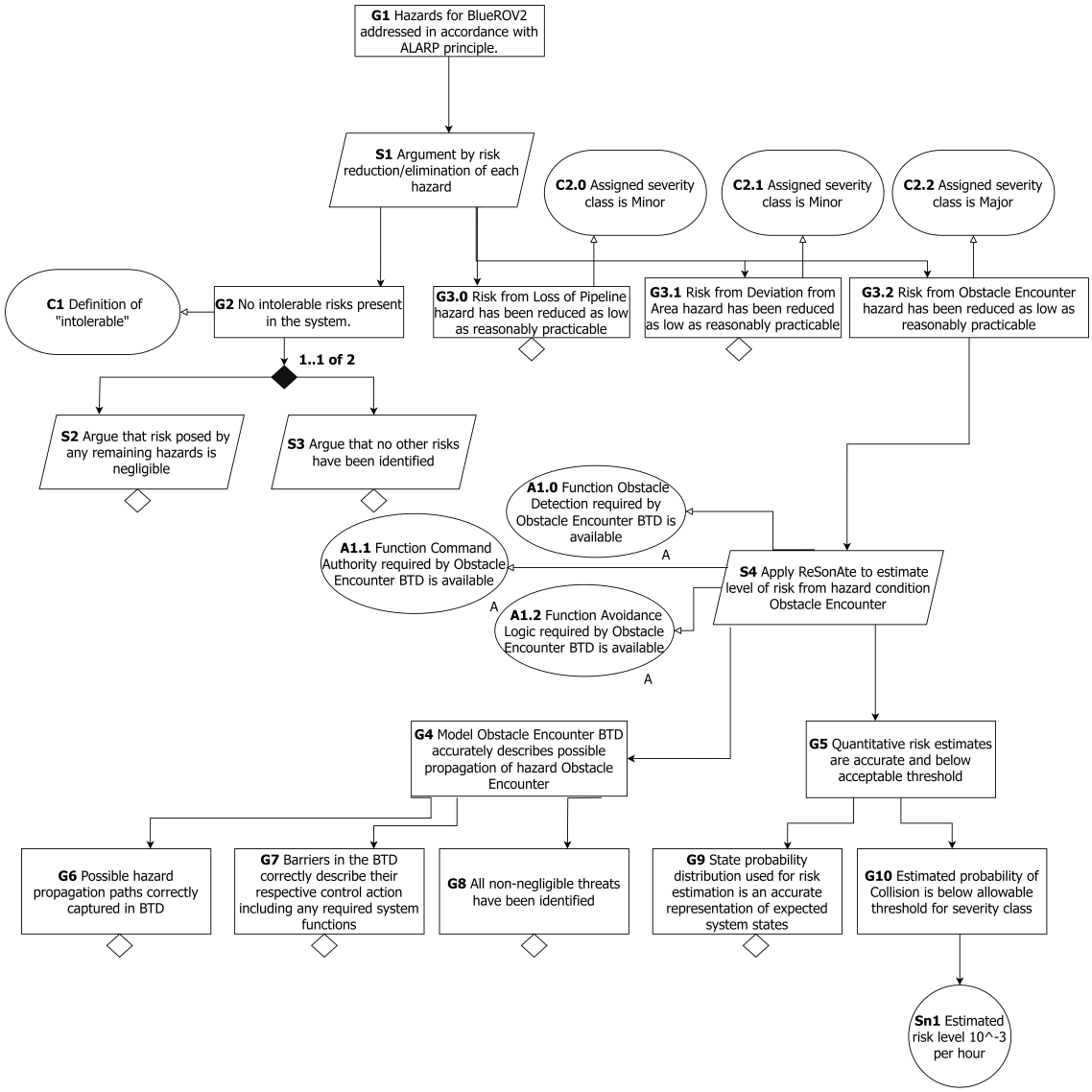

Figure B.4: BlueROV2 Assurance Case - Adapted from [4]

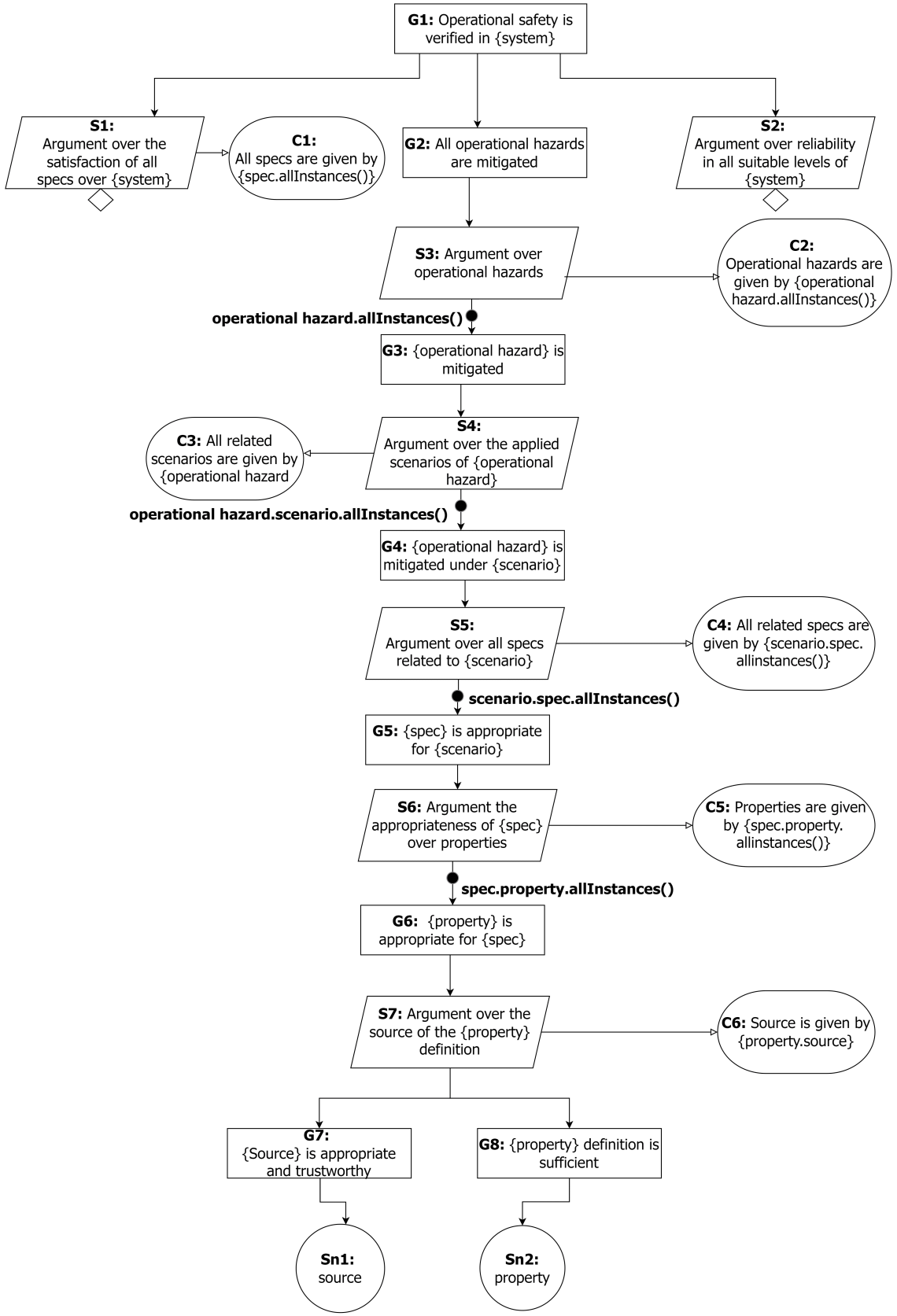

Figure B.5: GPCA Assurance Case Pattern - Adapted from [5]

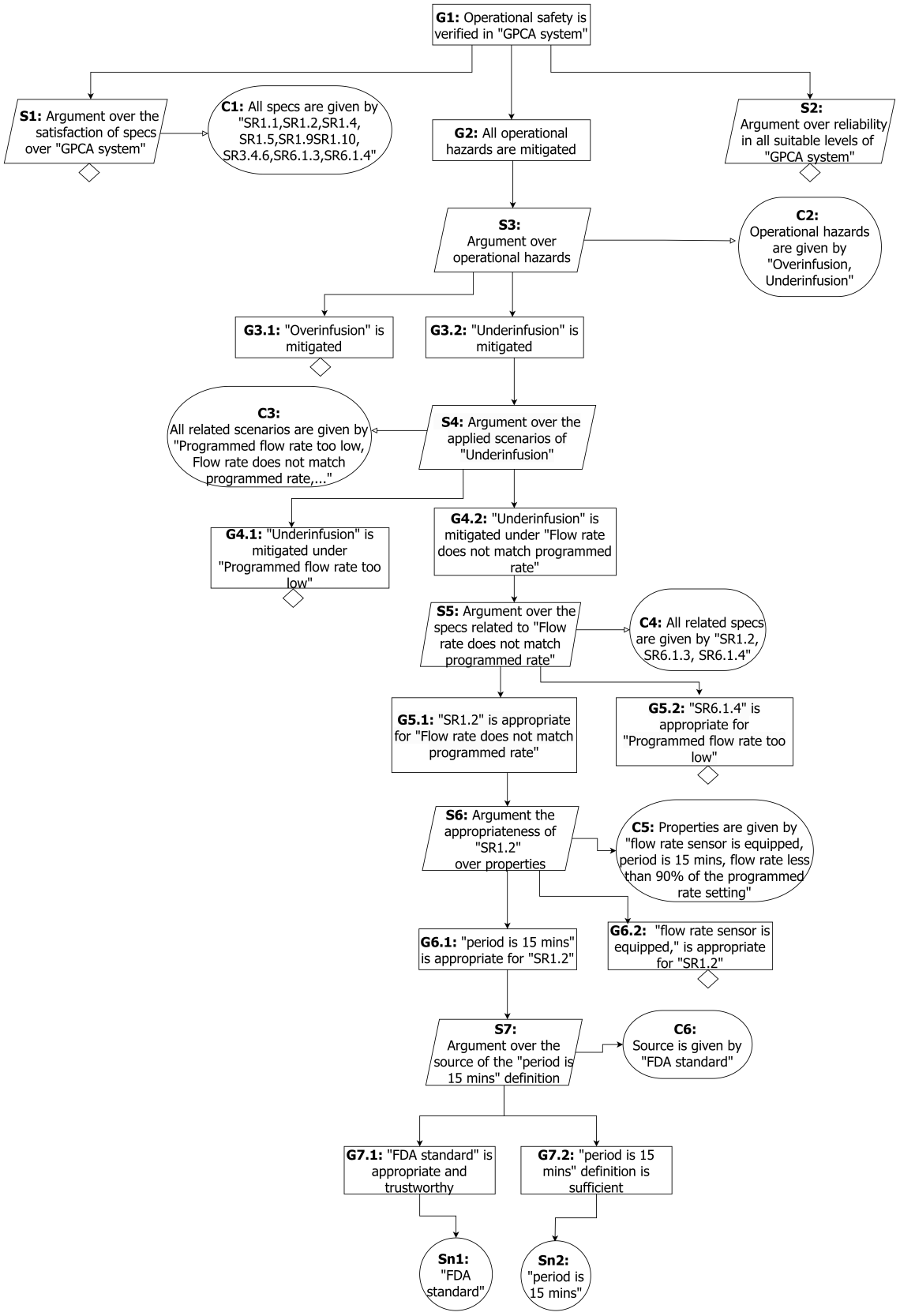

Figure B.6: GPCA Assurance Case - Adapted from [5]

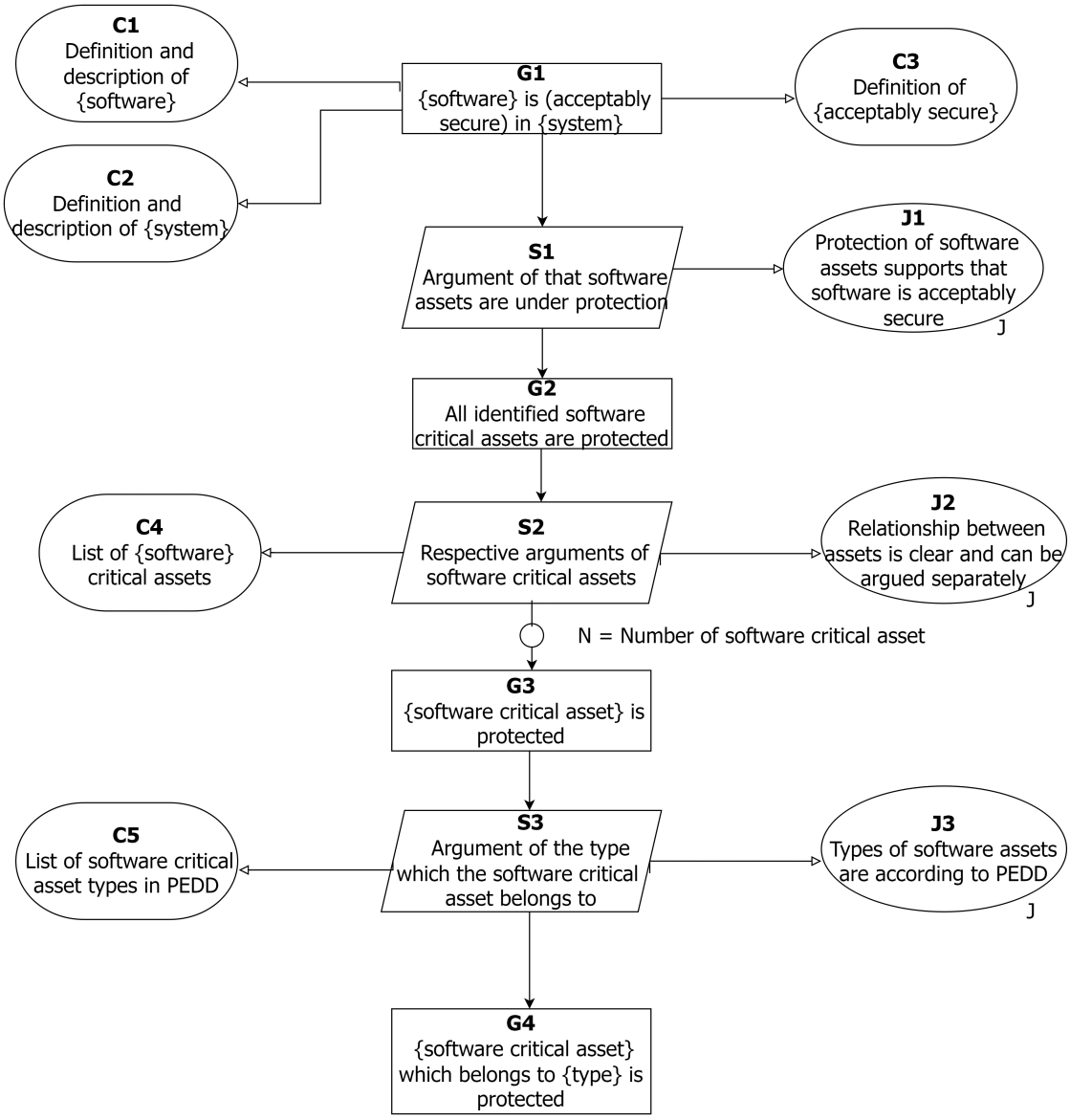

Figure B.7: IM Software Assurance Case Pattern - Adapted from [6]

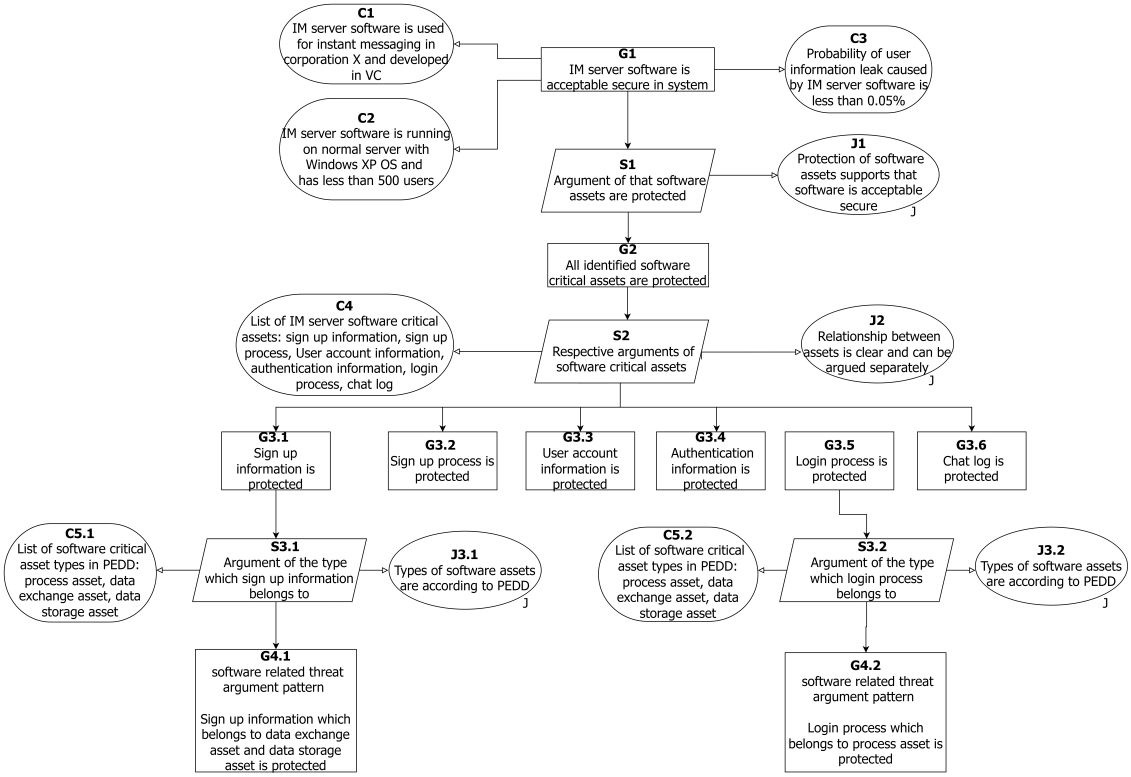

Figure B.8: IM Software Assurance Case - Adapted from [6]

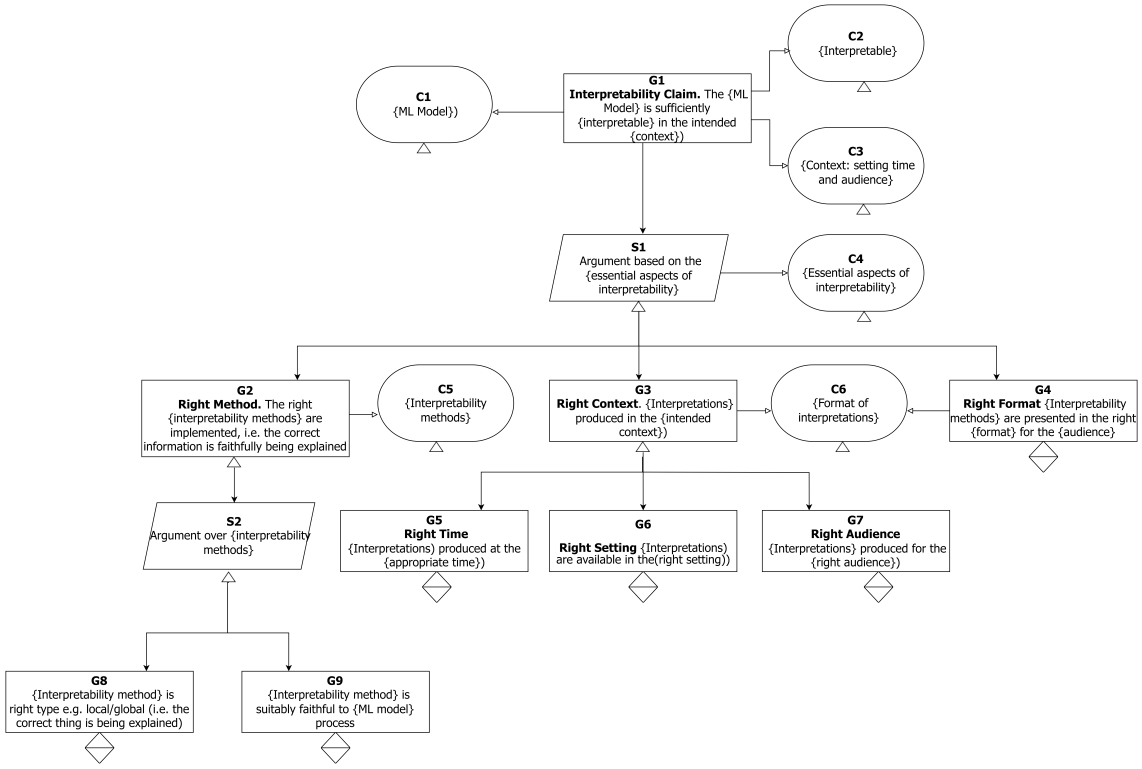

Figure B.9: DeepMind Assurance Case Pattern - Adapted from [7]

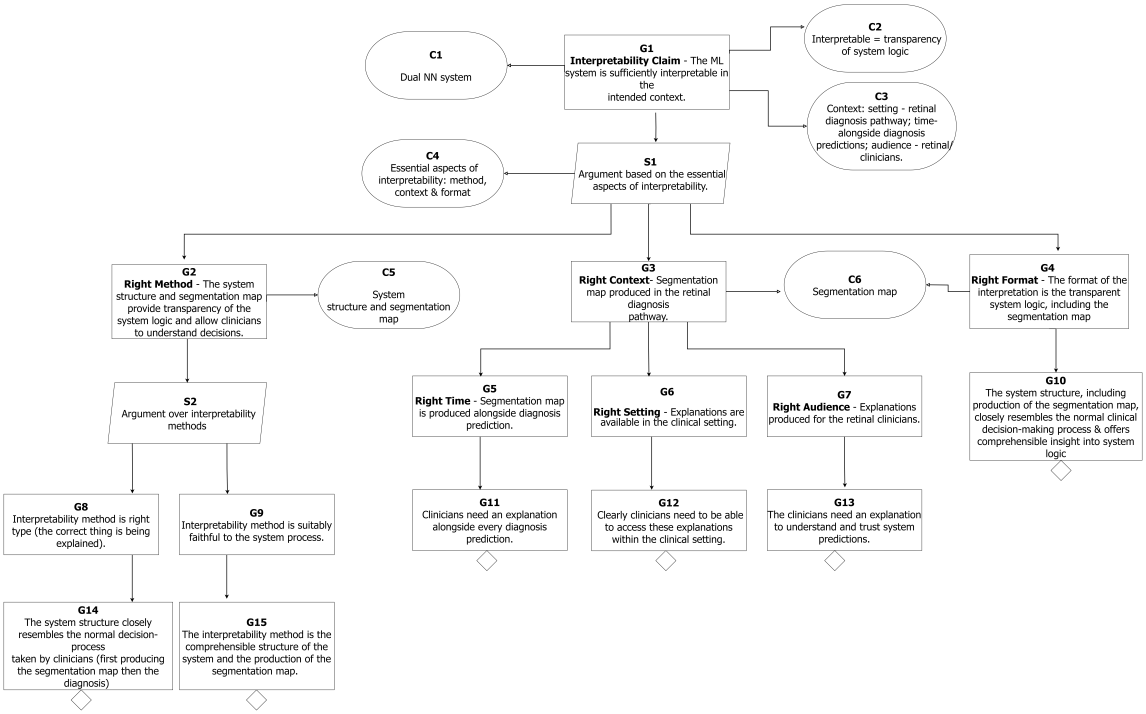

Figure B.10: DeepMind Assurance Case - Adapted from [7]

Appendix C

LLM-Generated Assurance Cases

C.1 Assurance case for the BLUEROV2 System Generated by GPT-4o

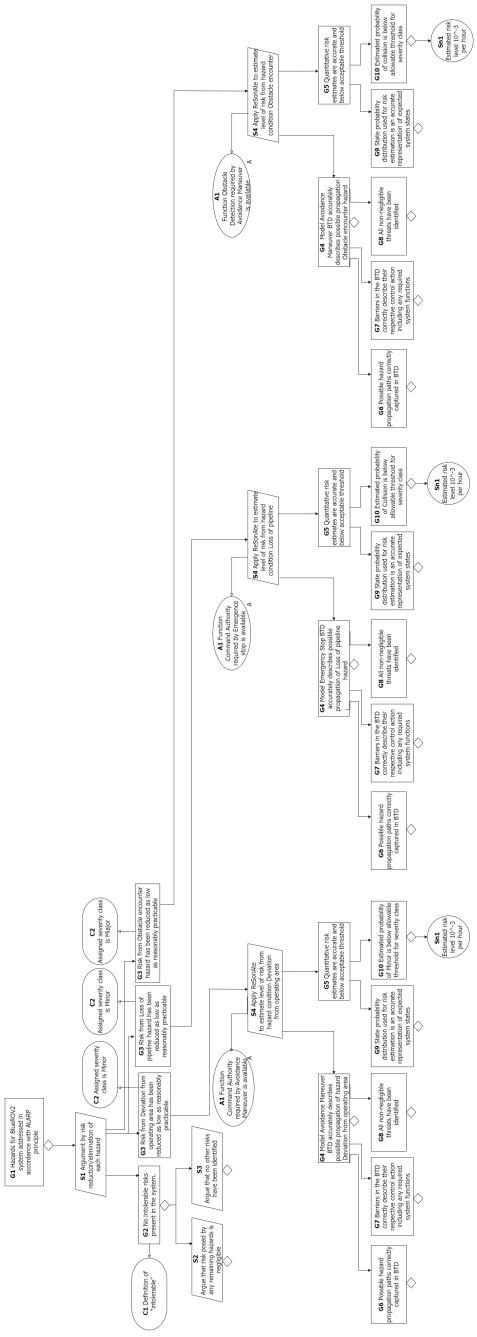

Figure C.1: An Assurance case for the BLUEROV2 System Generated by GPT-4o

C.2 Assurance case for the BLUEROV2 System Generated by GPT-4 Turbo

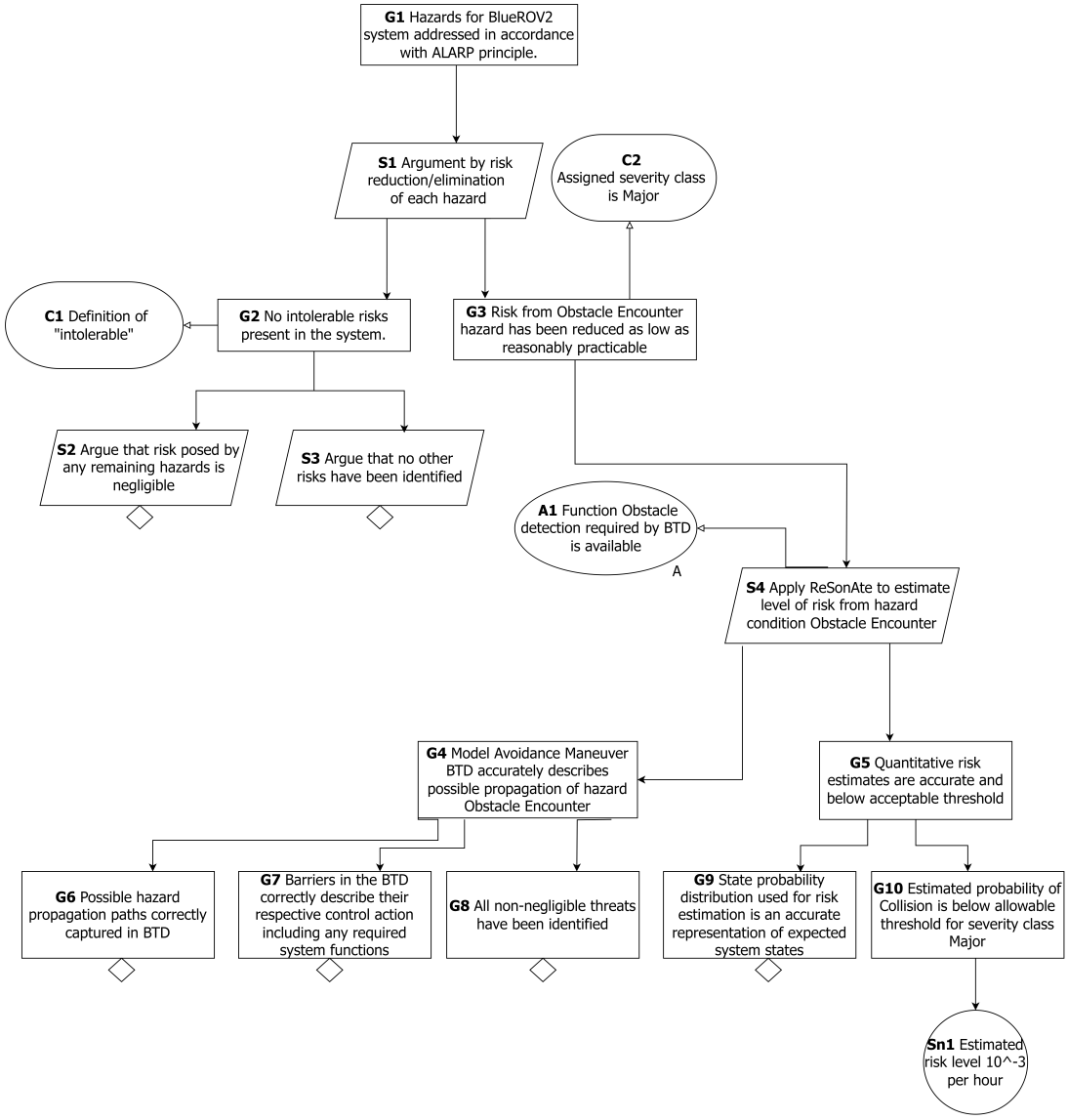

Figure C.2: An Assurance case for the BLUEROV2 System Generated by GPT-4 Turbo

Appendix D

Contextual Information

@Context_AC

An assurance case, such as a safety case or security case, can be represented using Goal Structuring Notation (GSN), a visual representation that presents the elements of an assurance case in a tree structure. The main elements of a GSN assurance case include Goals, Strategies, Solutions (evidence), Contexts, Assumptions, and Justifications.

Additionally, an assurance case in GSN may include an undeveloped element decorator, represented as a hollow diamond placed at the bottom center of a goal or strategy element. This indicates that a particular line of argument for the goal or strategy has not been fully developed and needs to be further developed.

I will explain each element of an assurance case in GSN so you can generate it efficiently.

1. Goal – A goal is represented by a rectangle and denoted as G. It represents the claims made in the argument. Goals should contain only claims. For the top-level claim, it should contain the most fundamental objective of the entire assurance case.

2. Strategy – A strategy is represented by a parallelogram and denoted as S. It describes the reasoning that connects the parent goals and their supporting goals. A Strategy should only summarize the argument approach. The text in a strategy element is usually preceded by phrases such as "Argument by appeal to...", "Argument by ...", "Argument across ..." etc.

3. Solution – A solution is represented by a circle and denoted as Sn. A solution element makes no claims but are simply references to evidence that provides support to a claim.

4. Context (Rounded rectangles) – In GSN, context is represented by a rounded rectangle and denoted as C. The context element provides additional background information for an argument and the scope for a goal or strategy within an assurance case.

5. Assumption – An assumption element is represented by an oval with the letter 'A' at the top- or bottom-right. It presents an intentionally unsubstantiated statement accepted as true within an assurance case. It is denoted by A

6. Justification (Ovals) – A justification element is represented by an oval with the letter 'J' at the top- or bottom-right. It presents a statement of reasoning or rationale within an assurance case. It is denoted by J.

@End_Context_AC

@Context_ACP

Assurance case patterns in GSN (Goal Structuring Notation) are templates that can be re-used to create an assurance case. Assurance case patterns encapsulate common structures of argumentation that have been found effective for addressing recurrent safety, reliability, or security concerns. An assurance case pattern can be instantiated to develop an assurance case by replacing generic information in placeholder decorator with concrete or system specific information.

To represent assurance case patterns in GSN format, additional decorators have been provided to support assurance case patterns. These additional decorators are used together with the elements of an assurance case to represent assurance case pattern. I will explain each additional decorator below to support assurance case pattern in GSN.

1. Uninstantiated - This decorator denotes that a GSN element remains to be instantiated, i.e. at some later stage, the generic information in placeholders within a GSN element needs to be replaced (instantiated) with a more concrete or system specific information. This decorator can be applied to any GSN element.

2. Uninstantiated and Undeveloped – Both decorators of undeveloped and uninstantiated are overlaid to form this decorator. This decorator denotes that a GSN element requires both further development and instantiation.

3. Placeholders – This is represented as curly brackets “” within the description of an element to allow for customization. The placeholder "" should be directly inserted within the description of elements for which the predicate "HasPlaceholder (X)" returns true. The placeholder "" can sometimes be empty or contain generic information that will need to be replaced when an assurance case pattern is instantiated.

4. Choice - A solid diamond is the symbol for Choice. A GSN choice can be used to denote alternatives in satisfying a relationship or represent alternative lines of argument used to support a particular goal.

5. Multiplicity - A solid ball is the symbol for multiple instantiations. It represents generalized n-ary relationships between GSN elements. Multiplicity symbols can be used to describe how many instances of one element-type relate to another element.

6. Optionality - A hollow ball indicates ‘optional’ instantiation. Optionality represents optional and alternative relationships between GSN elements.

The following steps is used to create an assurance case from an Assurance cases pattern.

1. Create the assurance case using only elements and decorators defined for assurance cases.

2. Remove all additional assurance case pattern decorators such as (Uninstantiated, Placeholders, Choice, Multiplicity, Optionality, and the combined Uninstantiated and Undeveloped decorator)

3. Remove the placeholder symbol "" and replace all generic information in placeholders “” with system specific or concrete information.

@End_Context_ACP

Appendix E

RQ3 AMLAS Artefacts

E.1 Safety Case Pattern for the Trajectory Prediction Component

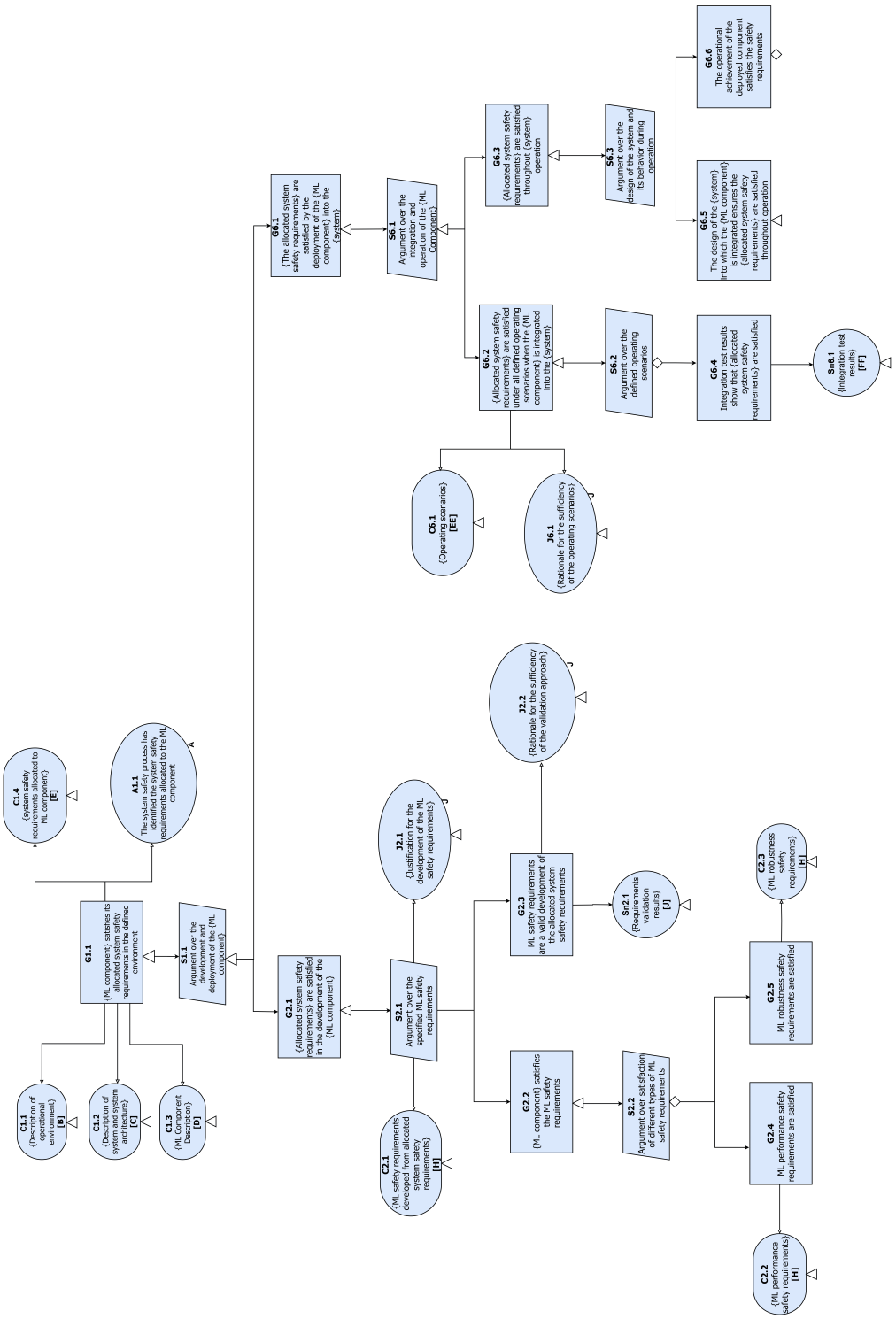

Figure E.1: Combined Safety Case Pattern Adapted from AMLAS [137] Stages 1, 2, and 6

E.2 Safety Case for the Trajectory Prediction Component

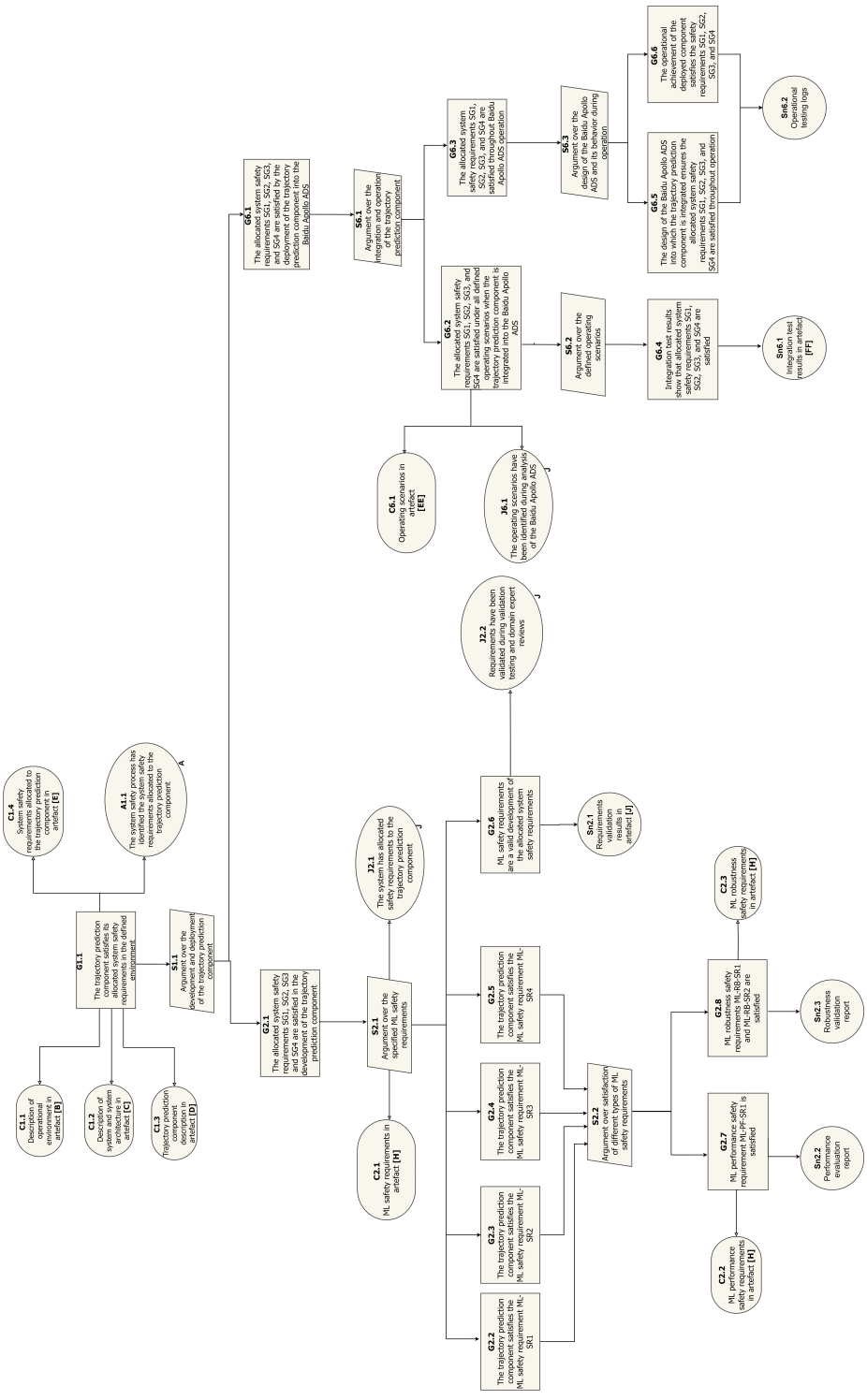

Figure E.2: Safety Case for the Trajectory Prediction Component of Baidu Apollo ADS

Bibliography

[1] 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.

[2] Robert Alexander et al. “Safety cases for advanced control software: Safety case patterns”. In: *Final Report, NASA Contract FA8655-07-1-3025, Univ. of York (October 2007)* (2007).

[3] Marwa Zeroual et al. “Formal model-based argument patterns for security cases”. In: *Proceedings of the 28th European Conference on Pattern Languages of Programs*. 2023, pp. 1–12.

[4] 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.

[5] Chung-Ling Lin, Wuwei Shen, and Richard Hawkins. “Support for safety case generation via model transformation”. In: *ACM SIGBED Review* 14.2 (2017), pp. 44–52.

[6] Biao Xu, Minyan Lu, and Dajian Zhang. “A layered argument strategy for software security case development”. In: *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2017, pp. 331–338.

[7] Francis Rhys Ward and Ibrahim Habli. “An assurance case pattern for the interpretability of machine learning in safety-critical systems”. In: *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39*. Springer. 2020, pp. 395–407.

[8] Carmen Carlan and Barbara Gallina. “Enhancing state-of-the-art safety case patterns to support change impact analysis”. In: *30th European Safety and Reliability Conference*. 2020.

[9] 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.

[10] Mario Trapp, Daniel Schneider, and Peter Liggesmeyer. "A safety roadmap to cyber-physical systems". In: *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach*. Springer, 2013, pp. 81–94.

[11] Simon Burton et al. "Confidence arguments for evidence of performance in machine learning for highly automated driving functions". 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. 365–377.

[12] Robin Bloomfield and Peter Bishop. "Safety and assurance cases: Past, present and possible future–an Adelard perspective". In: *Making Systems Safer: Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, 9-11th February 2010*. Springer. 2009, pp. 51–67.

[13] Andrew L King et al. "Towards assurance for plug & play medical systems". In: *Computer Safety, Reliability, and Security: 34th International Conference, SAFECOMP 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings 34*. Springer. 2015, pp. 228–242.

[14] Chiara Picardi et al. "A pattern for arguing the assurance of machine learning in medical diagnosis systems". In: *Computer Safety, Reliability, and Security: 38th International Conference, SAFECOMP 2019, Turku, Finland, September 11–13, 2019, Proceedings 38*. Springer. 2019, pp. 165–179.

[15] Maryam Bagheri et al. "Towards Developing Safety Assurance Cases for Learning-Enabled Medical Cyber-Physical Systems". In: *arXiv preprint arXiv:2211.15413* (2022).

[16] P Robert and H Ibrahim. "Assurance of Automotive Safety–A Safety Case Approach". In: *Proc. 29th International Conference, SAFECOMP*. Vol. 2010. 2010, pp. 82–96.

[17] Stefan Wagner et al. "A case study on safety cases in the automotive domain: Modules, patterns, and models". In: *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE. 2010, pp. 269–278.

[18] Rob Palin et al. "ISO 26262 safety cases: Compliance and assurance". In: (2011).

[19] C Michael Holloway. "Making the implicit explicit: Towards an assurance case for DO-178C". In: *International System Safety Conference*. NF1676L-16361. 2013.

[20] Omar Jaradat, Iain Bate, and Sasikumar Punnekkat. "Facilitating the maintenance of safety cases". In: *Current Trends in Reliability, Availability, Maintainability and Safety: An Industry Perspective*. Springer. 2016, pp. 349–371.

[21] 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.

[22] Mithila Sivakumar et al. "Exploring the capabilities of large language models for the generation of safety cases: the case of GPT-4". In: *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*. IEEE. 2024, pp. 35–45.

[23] Elisabeth A Nguyen and Alex G Ellis. "Experiences with assurance cases for spacecraft safing". In: *2011 IEEE 22nd International Symposium on Software Reliability Engineering*. IEEE. 2011, pp. 50–59.

[24] Goal Structuring Notation Standard Working Group. *GSN (Version 3)*. Accessed on November 30, 2023. 2023. URL: `https://scsc.uk/gsn`.

[25] Anaheed Ayoub et al. "A safety case pattern for model-based development approach". In: *NASA Formal Methods: 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings 4*. Springer. 2012, pp. 141–146.

[26] Chung-Ling Lin and Wuwei Shen. "Applying safety case pattern to generate assurance cases for safety-critical systems". In: *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE. 2015, pp. 255–262.

[27] Chung-Ling Lin, Wuwei Shen, and Steven Drager. "A framework to support generation and maintenance of an assurance case". In: *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2016, pp. 21–24.

[28] Mithila Sivakumar et al. "I came, I saw, I certified: some perspectives on the safety assurance of cyber-physical systems". In: *arXiv preprint arXiv:2401.16633* (2024).

[29] Tianyu Wu et al. "A brief overview of ChatGPT: The history, status quo and potential future development". In: *IEEE/CAA Journal of Automatica Sinica* 10.5 (2023), pp. 1122–1136.

[30] Mithila Sivakumar et al. "Design of the safety case of the reinforcement learning-enabled component of a quanser autonomous vehicle". In: *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*. IEEE. 2024, pp. 57–67.

[31] Richard Hawkins, Thomas Richardson, and Tim Kelly. "Using process models in system assurance". In: *Computer Safety, Reliability, and Security: 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings 35*. Springer. 2016, pp. 27–38.

[32] 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.

[33] Yongcheng Wang et al. "DevCase: design and implementation of a novel web-based graphical editor for safety cases complying with the GSN". In: ().

[34] Alvine Boaye Belle et al. "A novel approach to measure confidence and uncertainty in assurance cases". In: *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE. 2019, pp. 24–33.

[35] Anita Finnegan and Fergal McCaffery. "A security argument pattern for medical device assurance cases". In: *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE. 2014, pp. 220–225.

[36] Shihao Zhu, Minyan Lu, and Biao Xu. "Software Reliability Case Development Method Based on the 4+ 1 Principles". In: *2018 12th International Conference on Reliability, Maintainability, and Safety (ICRMS)*. IEEE. 2018, pp. 197–202.

[37] OMG. *Structured Assurance Case Metamodel (version 2.2)*. 2021. URL: `https://www.omg.org/spec/SACM/2.2/About-SACM`.

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

[39] Patrick J Graydon, John C Knight, and Elisabeth A Strunk. "Assurance based development of critical systems". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE. 2007, pp. 347–357.

[40] C Michael Holloway. "Safety case notations: Alternatives for the non-graphically inclined?" In: *2008 3rd IET International Conference on System Safety*. IET. 2008, pp. 1–6.

[41] 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.

[42] 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 Princeton, NJ. 2004.

[43] Tor Stålhane and Thor Myklebust. "The agile safety case". In: *Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings 35*. Springer. 2016, pp. 5–16.

[44] 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).

[45] Torin Viger, Simon Diemert, and Olivia Foster. "Patterns for Integrating NIST 800-53 Controls into Security Assurance Cases". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2023, pp. 165–175.

[46] 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.

[47] Kimya Khakzad Shahandashti et al. "A PRISMA-driven systematic mapping study on system assurance weakeners". In: *Information and Software Technology* (2024), p. 107526.

[48] 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.

[49] Richard David Hawkins and Tim Kelly. "A systematic approach for developing software safety arguments". In: *27th International System Safety Conference*. 2010.

[50] Yutaka Matsuno. "A design and implementation of an assurance case language". In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2014, pp. 630–641.

[51] Yutaka Matsuno and Kenji Taguchi. "Parameterised argument structure for GSN patterns". In: *2011 11th International Conference on Quality Software*. IEEE. 2011, pp. 96–101.

[52] Yutaka Matsuno. "D-case editor: A typed assurance case editor". In: *University of Tokyo* (2011).

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

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

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

[56] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

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

[58] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[59] Yu Sun et al. "Ernie: Enhanced representation through knowledge integration". In: *arXiv preprint arXiv:1904.09223* (2019).

[60] Jules White et al. "A prompt pattern catalog to enhance prompt engineering with chatgpt". In: *arXiv preprint arXiv:2302.11382* (2023).

[61] 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.

[62] Bernardino Romera-Paredes and Philip Torr. "An embarrassingly simple approach to zero-shot learning". In: *International conference on machine learning*. PMLR. 2015, pp. 2152–2161.

[63] Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning". In: *Advances in neural information processing systems* 30 (2017).

[64] Siyuan Wang et al. "Can LLMs Reason with Rules? Logic Scaffolding for Stress-Testing and Improving LLMs". In: *arXiv preprint arXiv:2402.11442* (2024).

[65] Monika Szczygielska and Aleksander Jarzębowicz. "Assurance case patterns on-line catalogue". In: *Advances in Dependability Engineering of Complex Systems: Proceedings of the Twelfth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, July 2-6, 2017, Brunów, Poland*. Springer. 2018, pp. 407–417.

[66] Christopher Preschern et al. "Pattern-based safety development methods: overview and comparison". In: *Proceedings of the 19th European Conference on Pattern Languages of Programs*. 2014, pp. 1–20.

[67] Mario Gleirscher and Stefan Kugele. "Assurance of system safety: A survey of design and argument patterns". In: *arXiv preprint arXiv:1902.05537* (2019).

[68] Fang Yan, Simon Foster, and Ibrahim Habli. "Safety case generation by model-based engineering: state of the art and a proposal". In: *The Eleventh International Conference on Performance, Safety and Robustness in Complex Systems and Applications, proceedings*. International Academy, Research, and Industry Association. 2021, pp. 4–7.

[69] Ewen Denney and Ganesh Pai. "A formal basis for safety case patterns". In: *Computer Safety, Reliability, and Security: 32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings 32*. Springer. 2013, pp. 21–32.

[70] Tewodros A Beyene and Carmen Carlan. "CyberGSN: a semi-formal language for specifying safety cases". In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2021, pp. 63–66.

[71] Anitha Murugesan et al. "Semantic Analysis of Assurance Cases using s (CASP)". In: *To appear at Goal Directed Execution of Answer Set Programs (GDE) Workshop in Int'l Conf. on Logic Programming (ICLP)*. 2023.

[72] Patrick John Graydon. "Formal assurance arguments: A solution in search of a problem?" In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2015, pp. 517–528.

[73] Kimya Khakzad Shahandashti et al. "Using GPT-4 Turbo to Automatically Identify Defeaters in Assurance Cases". In: *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*. IEEE. 2024, pp. 46–56.

[74] Richard Hawkins et al. "Guidance on the assurance of machine learning in autonomous systems (AMLAS)". In: *arXiv preprint arXiv:2102.01564* (2021).

[75] Markus Borg et al. "Ergo, SMIRK is safe: a safety case for a machine learning component in a pedestrian automatic emergency brake system". In: *Software quality journal* 31.2 (2023), pp. 335–403.

[76] Quanser. *Qcar*. URL: `https://www.quanser.com/products/qcar/`.

[77] 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.

[78] Richard David Hawkins, Ibrahim Habli, and Tim Kelly. "The need for a weaving model in assurance case automation". In: *Ada User Journal* (2015), pp. 187–191.

[79] Andrzej Wardziński and Aleksander Jarzębowicz. "Towards safety case integration with hazard analysis for medical devices". In: *Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings 35*. Springer. 2016, pp. 87–98.

[80] Andrzej Wardziński and Paul Jones. "Uniform model interface for assurance case integration with system models". In: *Computer Safety, Reliability, and Security: SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS, Trento, Italy, September 12, 2017, Proceedings 36*. Springer. 2017, pp. 39–51.

[81] Ioannis Nearchou et al. "An Assurance Case Driven Development Paradigm for Autonomous Vehicles: An F1TENTH Racing Car Case Study". In: *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE. 2023, pp. 156–161.

[82] Gereon Weiss et al. "Approach for Argumenting Safety on Basis of an Operational Design Domain". In: *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 2024, pp. 184–193.

[83] Simon Burton et al. "Safety assurance of machine learning for chassis control functions". In: *Computer Safety, Reliability, and Security: 40th International Conference, SAFECOMP 2021, York, UK, September 8–10, 2021, Proceedings 40*. Springer. 2021, pp. 149–162.

[84] Luiz Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. "BERT-and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study". In: *Information and Software Technology* 160 (2023), p. 107217.

[85] Mohammad Mahdi Mohajer et al. "Effectiveness of ChatGPT for Static Analysis: How Far Are We?" In: *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 2024, pp. 151–160.

[86] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. "Practical program repair in the era of large pre-trained language models". In: *arXiv preprint arXiv:2210.14179* (2022).

[87] Wasi Uddin Ahmad et al. "Unified pre-training for program understanding and generation". In: *arXiv preprint arXiv:2103.06333* (2021).

[88] 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.

[89] 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.

[90] 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.

[91] Kimya Khakzad Shahandashti et al. "Assessing the Impact of GPT-4 Turbo in Generating Defeaters for Assurance Cases". In: *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*. 2024, pp. 52–56.

[92] 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.

[93] Matthew J Page et al. "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews". In: *Bmj* 372 (2021).

[94] Naveen Donthu et al. "How to conduct a bibliometric analysis: An overview and guidelines". In: *Journal of business research* 133 (2021), pp. 285–296.

[95] IEEE. *IEEE Xplore*. 2023. URL: `https://ieeexplore.ieee.org/Xplore/home.jsp`.

[96] Elsevier. *SCOPUS*. 2023. URL: `https://www.scopus.com/home.uri`.

[97] Association of Computing Machinery. *ACM Digital Library*. 2023. URL: `https://dl.acm.org/`.

[98] Elsevier. *Engineering Village (EV)*. 2023. URL: https://www.engineeringvillage.com/home.url.

[99] Google. *Google Scholar*. 2023. URL: https://scholar.google.com/.

[100] A.W. Harzing. *Publish or Perish Software*. 2006. URL: https://harzing.com/resources/publish-or-perish.

[101] 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.

[102] *Connected Papers*. URL: https://www.connectedpapers.com/.

[103] Terri Gotschall. "EndNote 20 desktop version". In: *Journal of the Medical Library Association: JMLA* 109.3 (2021), p. 520.

[104] Nees Van Eck and Ludo Waltman. "Software survey: VOSviewer, a computer program for bibliometric mapping". In: *scientometrics* 84.2 (2010), pp. 523–538.

[105] Google Charts. *Sankey Diagram*. 2023. URL: https://developers.google.com/chart/interactive/docs/gallery/sankey.

[106] Steven Batt et al. "Learning Tableau: A data visualization tool". In: *The Journal of Economic Education* 51.3-4 (2020), pp. 317–328.

[107] OpenAI. *OpenAI API*. 2023. URL: https://openai.com/api/.

[108] OpenAI. *How should i set the temperature parameter*. Accessed on May 15, 2024. URL: https://platform.openai.com/docs/guides/text-generation/how-should-i-set-the-temperature-parameter.

[109] Yupeng Chang et al. "A survey on evaluation of large language models". In: *ACM Transactions on Intelligent Systems and Technology* 15.3 (2024), pp. 1–45.

[110] Seatgeek. *Seatgeek/fuzzywuzzy: Fuzzy String Matching in Python*. 2024. URL: https://github.com/seatgeek/fuzzywuzzy?tab=readme-ov-file.

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

[112] 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.

[113] SacreBLEU. *SacreBLEU*. 2024. URL: https://github.com/mjpost/sacreBLEU

[114] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5 (1988), pp. 513–523.

[115] Faisal Rahutomo, Teruaki Kitasuka, Masayoshi Aritsugi, et al. "Semantic cosine similarity". In: *The 7th international student conference on advanced science and technology ICAST*. Vol. 4. 1. University of Seoul South Korea. 2012, p. 1.

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

[117] Fang Yan et al. "Model-Based Generation of Hazard-Driven Arguments and Formal Verification Evidence for Assurance Cases". In: *10th International Conference on Model-Driven Engineering and Software Development*. SciTePress. 2022, pp. 252–263.

[118] Ewen Denney and Ganesh Pai. "Tool support for assurance case development". In: *Automated Software Engineering* 25.3 (2018), pp. 435–499.

[119] 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.

[120] Rob Weaver, Jane Fenn, and Tim Kelly. "A pragmatic approach to reasoning about the assurance of safety arguments". In: *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*. 2003, pp. 57–67.

[121] Anaheed Ayoub et al. "A systematic approach to justifying sufficient confidence in software safety arguments". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2012, pp. 305–316.

[122] Irfan Sljivo et al. "Generation of safety case argument-fragments from safety contracts". In: *Computer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings 33*. Springer. 2014, pp. 170–185.

[123] Nungki Selviandro. "Assurance Case Pattern using SACM Notation". In: *2021 9th International Conference on Information and Communication Technology (ICoICT)*. IEEE. 2021, pp. 494–499.

[124] Fuping Zeng, Minyan Lu, and Deming Zhong. "General Development Framework and Its Application Method for Software Safety Case." In: *J. Softw.* 8.12 (2013), pp. 3262–3268.

[125] Marten HL Kaas et al. "Ethics in conversation: Building an ethics assurance case for autonomous AI-enabled voice agents in healthcare". In: *Proceedings of the First International Symposium on Trustworthy Autonomous Systems*. 2023, pp. 1–13.

[126] Zoe Porter et al. "A principles-based ethics assurance argument pattern for AI and autonomous systems". In: *AI and Ethics* (2023), pp. 1–24.

[127] Mithila Sivakumar et al. "Prompting GPT–4 to support automatic safety case generation". In: *Expert Systems with Applications* 255 (2024), p. 124653.

[128] Mervyn Stone. "Cross-validatory choice and assessment of statistical predictions". In: *Journal of the royal statistical society: Series B (Methodological)* 36.2 (1974), pp. 111–133.

[129] Aki Vehtari, Andrew Gelman, and Jonah Gabry. "Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC". In: *Statistics and computing* 27 (2017), pp. 1413–1432.

[130] Maurice G Kendall. "A new measure of rank correlation". In: *Biometrika* 30.1-2 (1938), pp. 81–93.

[131] 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.

[132] Baidu. *Apollo*. URL: `https://github.com/ApolloAuto/apollo`.

[133] Zi Peng et al. "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on apollo". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 1240–1250.

[134] Kecheng Xu et al. "Data driven prediction architecture for autonomous driving and its application on apollo platform". In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 175–181.

[135] Haoyang Fan et al. "Baidu apollo em motion planner". In: *arXiv preprint arXiv:1807.08048* (2018).

[136] Yanjun Huang et al. "A survey on trajectory-prediction methods for autonomous driving". In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 652–674.

[137] Catherine Menon and Rob Alexander. "A safety-case approach to the ethics of autonomous vehicles". In: *Safety and reliability*. Vol. 39. 1. Taylor & Francis. 2020, pp. 33–58.

[138] Tim Kelly. "Reviewing assurance arguments-a step-by-step approach". In: *Workshop on assurance cases for security-the metrics challenge, dependable systems and networks (DSN)*. 2007.

[139] Shuichiro Yamamoto and Shuji Morisaki. "A system theoretic assurance case review". In: *2016 11th International Conference on Computer Science & Education (ICCSE)*. IEEE. 2016, pp. 992–996.

[140] 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.

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

[142] 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.

[143] Ibrahim Habli and Tim Kelly. "A generic goal-based certification argument for the justification of formal analysis". In: *Electronic Notes in Theoretical Computer Science* 238.4 (2009), pp. 27–39.

[144] Irfan Šljivo et al. "A method to generate reusable safety case argument-fragments from compositional safety analysis". In: *Journal of Systems and Software* 131 (2017), pp. 570–590.

[145] AL De Oliveira et al. "A pattern to argue the compliance of system safety requirements decomposition". In: *Proceedings of the 10th Conference on Pattern Languages of Programs (SugarLoafPLoP), Sagadi Manor, Estonia*. 2014, pp. 9–12.

[146] Maged Khalil, Alejandro Prieto, and Florian Hölzl. "A pattern-based approach towards the guided reuse of safety mechanisms in the automotive domain". In: *Model-Based Safety and Assessment: 4th International Symposium, IMBSA 2014, Munich, Germany, October 27-29, 2014. Proceedings 4*. Springer. 2014, pp. 137–151.

[147] Ernest Wozniak et al. "A safety case pattern for systems with machine learning components". In: *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39*. Springer. 2020, pp. 370–382.

[148] Jason Jaskolka et al. "A security property decomposition argument pattern for structured assurance case models". In: *26th European Conference on Pattern Languages of Programs*. 2021, pp. 1–10.

[149] Shuichiro Yamamoto. "An approach to assure Dependability through ArchiMate". 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. 50–61.

[150] Shuichiro Yamamoto. "An evaluation of argument patterns based on data flow". In: *Information and Communication Technology: Second IFIP TC5/8 International Conference, ICT-EurAsia 2014, Bali, Indonesia, April 14-17, 2014. Proceedings 2*. Springer. 2014, pp. 432–437.

[151] Vikas Ghatge et al. "Applying the Goal Structuring Notation (GSN) to Argue Compliance of Equipment with the European EMC Directive". In: *IEEE Letters on Electromagnetic Compatibility Practice and Applications* (2023).

[152] Mario Gleirscher and Carmen Carlan. "Arguing from hazard analysis in safety cases: a modular argument pattern". In: *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE. 2017, pp. 53–60.

[153] Carmen Cârlan et al. "Arguing on software-level verification techniques appropriateness". In: *Computer Safety, Reliability, and Security: 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings 36*. Springer. 2017, pp. 39–54.

[154] Fredrik Warg and Martin Skoglund. "Argument patterns for multi-concern assurance of connected automated driving systems". In: *4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019.

[155] Boxuan Wang and Minyan Lu. "Argumentation Pattern: An Approach to Issuing Software Reliability Case". In: *Proceedings of the 2013 International Conference on Electrical and Information Technologies for Rail Transportation (EITRT2013)-Volume II*. Springer. 2014, pp. 373–381.

[156] Chiara Picardi et al. "Assurance argument patterns and processes for machine learning in safety-related systems". In: *Proceedings of the Workshop on Artificial Intelligence Safety (SafeAI 2020)*. CEUR Workshop Proceedings. 2020, pp. 23–30.

[157] Ramneet Kaur et al. "Assurance case patterns for cyber-physical systems with deep neural networks". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2020, pp. 82–97.

[158] Richard Hawkins et al. "Assurance cases for block-configurable software". In: *Computer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings 33*. Springer. 2014, pp. 155–169.

[159] Sagar Chaki et al. "Assurance cases for proofs as evidence". In: *Proceedings of the Third International Workshop on Proof-Carrying Code and Software Certification*. 2009.

[160] Philippa Conmy and Iain Bate. "Assuring safety for component based software engineering". In: *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*. IEEE. 2014, pp. 121–128.

[161] 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.

[162] Georg Macher, Harald Sporer, and Christian Kreiner. "Automotive safety case pattern". In: *Proceedings of the 19th European Conference on Pattern Languages of Programs*. 2014, pp. 1–18.

[163] Celso Hirata and Simin Nadjm-Tehrani. "Combining GSN and STPA for safety arguments". 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. 5–15.

[164] Ewen Denney and Ganesh Pai. "Composition of safety argument patterns". In: *Computer Safety, Reliability, and Security: 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings 35*. Springer. 2016, pp. 51–63.

[165] Timothy E Wang et al. "Computer-aided generation of assurance cases". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2023, pp. 135–148.

[166] 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.

[167] Barbara Gallina, Elena Gómez-Martínez, and Clara Benac Earle. "Deriving safety case fragments for assessing mbasafe's compliance with en 50128". In: *Software Process Improvement and Capability Determination: 16th International Conference, SPICE 2016, Dublin, Ireland, June 9-10, 2016, Proceedings 16*. Springer. 2016, pp. 3–16.

[168] Yutaka Matsuno. "Design and implementation of GSN patterns: A step toward assurance case language". In: *IPSJ Online Transactions* 7 (2014), pp. 59–68.

[169] Richard Hawkins, Tim Kelly, and Ibrahim Habli. "Developing Assurance Cases for D-MILS Systems." In: *MILS@ HiPEAC*. 2015.

[170] Barbara Gallina et al. "Enabling cross-domain reuse of tool qualification certification artefacts". 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. 255–266.

[171] Dorien Koelemeijer. "Enhancing the cyber resilience of critical infrastructures through an evaluation methodology based on assurance cases". In: *Procedia Computer Science* 126 (2018), pp. 1779–1791.

[172] 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.

[173] Mario Gleirscher, Simon Foster, and Yakoub Nemouchi. "Evolution of formal model-based assurance cases for autonomous robots". In: *Software Engineering and Formal Methods: 17th International Conference, SEFM 2019, Oslo, Norway, September 18–20, 2019, Proceedings 17*. Springer. 2019, pp. 87–104.

[174] Carmen Cârlan et al. "Explicitcase: tool-support for creating and maintaining assurance arguments integrated with system models". In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, pp. 330–337.

[175] Yuliya Prokhorova, Linas Laibinis, and Elena Troubitsyna. "Facilitating construction of safety cases from formal models in Event-B". In: *Information and Software Technology* 60 (2015), pp. 51–76.

[176] Colin Smith, Ewen Denney, and Ganesh Pai. *Hazard contribution modes of machine learning components*. Tech. rep. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2020.

[177] Vaise Patu and Shuichiro Yamamoto. "Identifying and Implementing Security Patterns for a Dependable Security Case–From Security Patterns to D-Case". In: *2013 IEEE 16th International Conference on Computational Science and Engineering*. IEEE. 2013, pp. 138–142.

[178] OpenAI. *Reproducible outputs*. Accessed on May 15, 2024. URL: `https://platform.openai.com/docs/guides/text-generation/reproducible-outputs`.

[179] L Sun, O Lisagor, and T Kelly. "Justifying the validity of safety assessment models with safety case patterns". In: *6th IET International Conference on System Safety 2011*. IET. 2011, pp. 1–6.

[180] Timothy Patrick Kelly et al. "Arguing safety: a systematic approach to managing safety cases". PhD thesis. Citeseer, 1999.

[181] Athanasios Retouniotis et al. "Model-connected safety cases". In: *Model-Based Safety and Assessment: 5th International Symposium, IMBSA 2017, Trento, Italy, September 11–13, 2017, Proceedings 5*. Springer. 2017, pp. 50–63.

[182] Mirko Napolano et al. "Preventing recurrence of industrial control system accident using assurance case". In: *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2015, pp. 182–189.

[183] Damir Nešić, Mattias Nyberg, and Barbara Gallina. "Product-line assurance cases from contract-based design". In: *Journal of Systems and Software* 176 (2021), p. 110922.

[184] André Alexandersen Hauge and Ketil Stølen. "SACS: A pattern language for safe adaptive control software". In: *Proceedings of the 18th Conference on Pattern Languages of Programs*. 2011, pp. 1–22.

[185] Kenji Taguchi, Daisuke Souma, and Hideaki Nishihara. "Safe & sec case patterns". 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. 27–37.

[186] Shuanqi Wang et al. "Safety Argument Pattern Language of Safety-Critical Software". In: *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE. 2020, pp. 459–467.

[187] R.D. Hawkins and T.P. Kelly. "Software safety assurance - what is sufficient?" In: *4th IET International Conference on Systems Safety 2009. Incorporating the SaRS Annual Conference*. 2009, pp. 1–6. DOI: 10.1049/cp.2009.1542.

[188] Fuping Zeng, Minyan Lu, and Deming Zhong. "Software safety certification framework based on safety case". In: *2012 International Conference on Computer Science and Service System*. IEEE. 2012, pp. 566–569.

[189] Axel Zechner and Michaela Huhn. "Structural Analysis of Safety Case Arguments in a Model-based Development Environment." In: *MBEES*. 2009, pp. 115–127.

[190] Irfan Sljivo et al. "Tool-supported safety-relevant component reuse: From specification to argumentation". In: *Reliable Software Technologies–Ada-Europe 2018: 23rd Ada-Europe International Conference on Reliable Software Technologies, Lisbon, Portugal, June 18-22, 2018, Proceedings 23*. Springer. 2018, pp. 19–33.

[191] Alejandra Ruiz, Ibrahim Habli, and Huáscar Espinoza. "Towards a case-based reasoning approach for safety assurance reuse". In: *Computer Safety, Reliability, and Security: SAFECOMP 2012 Workshops: Sassur, ASCoMS, DESEC4LCCI, ERCIM/EWICS, IWDE, Magdeburg, Germany, September 25-28, 2012. Proceedings 31*. Springer. 2012, pp. 22–35.

[192] Erik Stensrud et al. "Towards goal-based software safety certification based on prescriptive standards". In: *2011 First International Workshop on Software Certification*. IEEE. 2011, pp. 13–18.