The Journal of Systems and Software 222 (2025) 112353

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Automatic instantiation of assurance cases from patterns using large language models☆

Oluwafemi Odu [a], Alvine B. Belle [a,*], Song Wang [a], Segla Kpodjedo [b], Timothy C. Lethbridge [c], Hadi Hemmati [a]

[a] Lassonde School Of Engineering, York University, Toronto, Canada
[b] Department of Software Engineering and Information Technology, École de technologie supérieure, Montreal, Canada
[c] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada

ARTICLE INFO

Keywords:
Requirement engineering
Assurance cases
Assurance case patterns
Pattern formalization
Generative artificial intelligence
Large language models
GPT

ABSTRACT

An assurance case is a structured set of arguments supported by evidence, demonstrating that a system's non-functional requirements (e.g., safety, security, reliability) 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 using 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. This study aims to investigate the potential of Large Language Models (LLMs) in automating the generation of assurance cases that comply with specific patterns. Specifically, we formalize assurance case patterns using predicate-based rules and then utilize LLMs, i.e., GPT-4o and GPT-4 Turbo, to automatically instantiate assurance cases from these formalized 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 practical at this time.

1. Introduction

Complex critical systems such as cyber–physical systems, are increasingly designed to be interoperable and interconnected. The growing complexity of their configurations and operations in dynamic environments underscores the importance of system assurance. Ensuring the correct implementation of non-functional requirements, such as safety and security, is vital to prevent these systems' failure that could result in severe consequences, including fatalities and financial losses (Napolano et al., 2015; Sivakumar et al., 2023).

Assurance cases (ACs) are structured arguments with a supporting body of evidence that allow demonstrating that the non-functional requirements of a system have been correctly implemented (Bloomfield and Bishop, 2009). Assurance cases are utilized across various domains (e.g., medicine (Bagheri et al., 2022; King et al., 2015; Picardi et al., 2019), automotive (Burton et al., 2019; Robert and Ibrahim, 2010; Wagner et al., 2010) to support the certification and compliance of critical systems with industry standards (e.g., ISO 26262 (Palin et al., 2011), DO-178C (Holloway, 2013)). Manually creating assurance cases can be time-consuming, especially for large, complex, and interconnected systems (Maksimov et al., 2019; Sivakumar et al., 2024b). For instance, Maksimov et al. (2019) 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 (Nguyen and Ellis, 2011). 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) (Goal Structuring Notation Standard Working Group, 2023) and the Claims-Arguments-Evidence (CAE) notation (Bloomfield and Bishop, 2009).

☆ Editor: Prof W. Eric Wong.

* Corresponding author.
E-mail address: alvine.belle@lassonde.yorku.ca (A.B. Belle).

https://doi.org/10.1016/j.jss.2025.112353
Received 29 September 2024; Received in revised form 20 December 2024; Accepted 16 January 2025
Available online 24 January 2025

Despite the use of assurance case patterns to create assurance cases, the instantiation process remains tedious, error-prone, and time-consuming. This is primarily due to the heterogeneous nature of system artifacts and the complexity of mission-critical systems (Hartsell et al., 2021). Instantiating these patterns with system-specific information still requires domain expertise to efficiently extract the necessary system artifacts. Experts must then manually replace the abstract elements within these patterns with concrete values from the extracted artifacts to create an assurance case that complies with the given pattern(s).

There is a wealth of literature on the automatic instantiation of assurance case patterns. However, it is worth noting that in this literature, the expressions *'instantiation of assurance case patterns to generate assurance cases'* and *'instantiation of assurance cases from patterns'* are frequently used interchangeably. Most of the existing approaches for instantiating assurance cases from patterns strongly depend on the model-based engineering approach that supports extraction of information from system models (e.g., model-based design (Hartsell et al., 2021; Ayoub et al., 2012; Lin and Shen, 2015; Lin et al., 2016; Sivakumar et al., 2024a). However, a strong dependence on a 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 (Wu et al., 2023). 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). Our experiment results reveal LLMs can effectively generate assurance cases. Thus, our contributions are fourfold:

- We propose a novel method for formalizing ACPs into predicate-based rules complying with GSN. This allows for capturing the internal structure of ACPs more generically and uniformly.
- We explore the use of LLMs to automatically generate assurance cases complying with formalized ACPs.
- We experiment with two popular and very recent LLMs (i.e. GPT-4 Turbo and GPT-4o) to explore their ability to automatically generate ACs from patterns.
- We release the dataset and source code of our experiments to help other researchers replicate and extend our study[1].

The remainder of this paper is organized as follows: Section 2 presents some background concepts. Section 3 discusses related work. Section 4 presents our methodology. Section 5 describes the experimental setup. Section 6 reports the results of our study. In Section 7, we discuss our results. Section 8 identifies the threats to validity associated with our study. We conclude and outline future work in Section 9.

2. Background

2.1. Assurance case

An assurance case is a well-established, structured, reasoned, and auditable set of arguments designed to support a specific goal (Belle et al., 2019). These arguments are often supported by evidence 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 (Bagheri et al., 2022; Lin et al., 2017), security cases (Finnegan and McCaffery, 2014; Xu et al., 2017), and reliability cases (Zhu et al., 2018).

Assurance cases are utilized to prevent system failure in various domains, including medicine (Bagheri et al., 2022; King et al., 2015; Picardi et al., 2019) and automotive (Burton et al., 2019; Robert and Ibrahim, 2010; Wagner et al., 2010). 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 (Finnegan and McCaffery, 2014).

An assurance case comprises three primary components (OMG, 2021; Mansourov and Campara, 2010): (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 (Graydon et al., 2007).

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 (Carlan and Gallina, 2020). These assurance case patterns contain placeholders filled with generic information that can be replaced with system-specific information during their instantiation (Hartsell et al., 2021; Carlan and Gallina, 2020). The use 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 (Viger et al., 2023; Cârlan et al., 2016; Shahandashti et al., 2024a). Assurance deficits refer to *"any knowledge gap that prohibits perfect confidence"* in an assurance case (Hawkins et al., 2011).

2.3. Representation of assurance cases and assurance case patterns

2.3.1. Assurance case representation

Several notations allow for representing assurance cases and can be broadly categorized into textual and graphical notations. Holloway (2008) 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 (Selviandro et al., 2020; Kelly and Weaver, 2004).

Graphical notations also address the limitations of unstructured representations by improving the clarity and structure of assurance cases. Graphical notations include GSN (Goal structuring Notation) (Goal Structuring Notation Standard Working Group, 2023), CAE (Claim-Argument-Evidence) (Bloomfield and Bishop, 2009), and Eliminative Argumentation (EA) (Goodenough et al., 2015). The Object Management Group (OMG) recently introduced SACM (Structured Assurance Case Metamodel) (OMG, 2021) to promote interoperability and standardization (Wei et al., 2019). SACM aligns with existing assurance case notations (e.g., GSN, CAE). Still, GSN is the most popular notation (Wei et al., 2019). 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 (Goal Structuring Notation Standard Working Group, 2023) 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 Fig. 2.
- **A Strategy** is depicted as a parallelogram and describes the inference between a goal and its sub-goals. See S1 in Fig. 2.

[1] https://doi.org/10.6084/m9.figshare.27103225.v2

G1: Collision Avoidance Algorithm (CAA) provides correct instructions for avoiding collisions between two UAVS

S1: Strategize over the capabilities of the CAA to give accurate directives to individuals UAVs

G1.1: GPS is accurate within 5 cms

C1: RTK ground station is provided in flying area

S2: Strategize over the GPS accuracy claims

G1.1.1: RTK here + provides accuracy within 5 cms

Sn1: RTK here + manufacturer's guarantees

G1.1.2: RTK functions correctly on America hexcopters with S-900

Sn2: Field test cases passed

G1.2: CAA provides avoidance directives that prevent violation of minimum time-to-impact (Undeveloped)

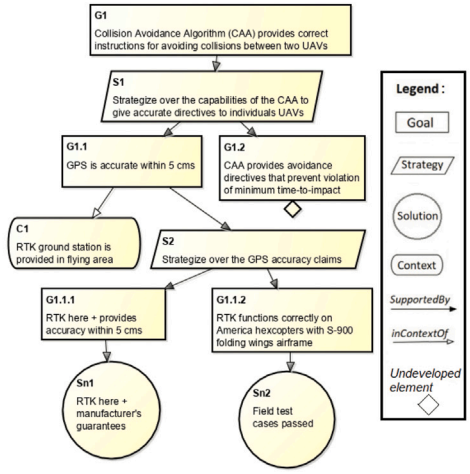

Fig. 1. On the right, an example of a partial safety case (GSN diagram) adapted from Vierhauser et al. (2019); on the left, the equivalent of the safety case in structured prose.

- **A Solution** is depicted as a circle and represents the evidence supporting an argument or goal. See Sn1 and Sn2 in Fig. 1.
- **A Context** is rendered as a rounded rectangle, presenting a contextual artifact. This can be a reference to contextual information, or a statement. See C1 in Fig. 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 (Mansourov and Campara, 2010). It is possible to decorate GSN elements using the *Undeveloped* decorator. The latter allows indicating a GSN element has not been developed yet (Goal Structuring Notation Standard Working Group, 2023). It is depicted as a hollow diamond applied to the bottom center of an element (Goal Structuring Notation Standard Working Group, 2023). Furthermore, the GSN standard (Goal Structuring Notation Standard Working Group, 2023) 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 (Goal Structuring Notation Standard Working Group, 2023) proposes guidelines to convert an assurance case represented in the textual format (e.g., structured prose) into a GSN diagram. Fig. 1 shows an excerpt of an assurance case adapted from Vierhauser et al. (2019). This excerpt is represented in the GSN and the structured prose.

2.3.2. Assurance case pattern representation

GSN and some reference literature on GSN patterns (e.g., Matsuno (2014), Matsuno and Taguchi (2011), Matsuno (2011)) also propose the following additional decorators to help represent assurance case patterns:

- **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 (Goal Structuring Notation Standard Working Group, 2023).
- **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 (Matsuno, 2014; Matsuno and Taguchi, 2011; Matsuno, 2011).
- **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 (Goal Structuring Notation Standard Working Group, 2023).
- **Optionality** — It represents optional and alternative relationships between GSN elements which generalizes n-of-m choices between GSN elements (Goal Structuring Notation Standard Working Group, 2023).
- **Choice**. This decorator is depicted as a solid diamond. The choice decorator denotes possible alternatives in satisfying a relationship (Goal Structuring Notation Standard Working Group, 2023).

Fig. 2 shows a sample safety case pattern adapted from Alexander et al. (2007) and represented using GSN.

2.4. Large language models (LLMs)

LLMs are advanced artificial intelligence systems with computational ability to generate human language, with at least the appearance of understanding it (Naveed et al., 2023; Hou et al., 2023). These models are complex neural network structures with massive parameter sizes and trained on vast amounts of data from diverse sources (Hou et al., 2023). LLMs can generate new content, answer questions, and capture inherent rules of a domain (Naveed et al., 2023; Yang et al., 2023; Brown et al., 2020). 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 (Wu et al., 2023), BERT (Devlin et al., 2018), T5 (Raffel et al., 2020), and ERNIE (Sun et al., 2019).

Prompt engineering refers to the different techniques used to provide instructions and guidelines to an LLM to ensure a desired generated response (White et al., 2023). It improves the efficiency and quality of LLMs responses. The most popular prompting techniques include the Chain-of-Thought (CoT) prompting technique (Wei et al., 2022), Zero-shot prompting, (Romera-Paredes and Torr, 2015), and Few-shot prompting (Snell et al., 2017). CoT utilizes a series of intermediate reasoning steps to significantly improve the ability of LLMs to

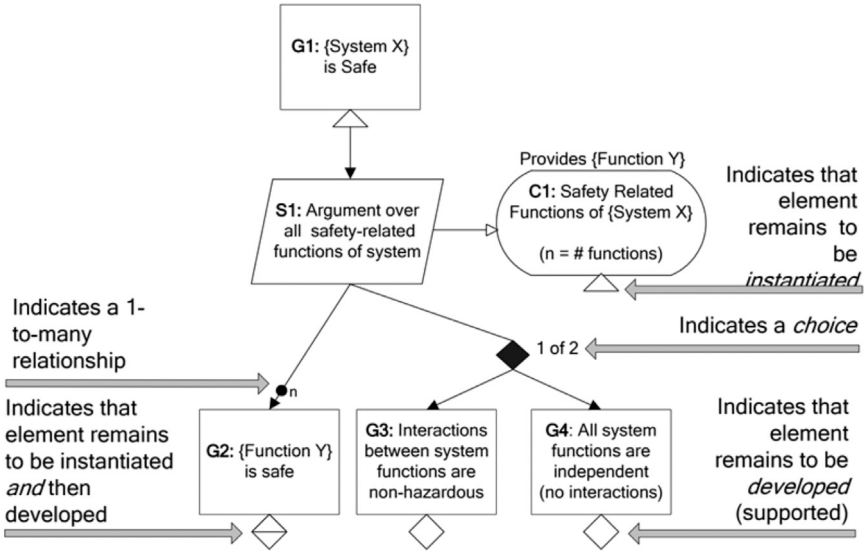

Fig. 2. A sample assurance case pattern adapted from Alexander et al. (2007).

perform complex reasoning tasks (Wei et al., 2022). Zero-shot prompting (Romera-Paredes and Torr, 2015) — a technique whereby we prompt an LLM without any examples, attempting to take advantage of the reasoning patterns it has extracted. Few-shot prompting (Snell et al., 2017) 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 (Yang et al., 2023; Wang et al., 2024b) 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 (Yang et al., 2023; Wang et al., 2024b).

3. Related work

3.1. Formalization of assurance cases and assurance case patterns

Denney and Pai (2013) 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 (2014) 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 (2021) developed *'CyberGSN'*, integrating informal GSN elements with formal Cyberlogic to facilitate safety case creation and maintenance. To address semantic correctness and logical consistency in assurance cases, Murugesan et al. (2023) 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 (2015) surveyed twenty studies focusing on proposed formal assurance arguments. Their results revealed that the 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. (2024b), in which the authors extracted predicates from the structural rules embedded in 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 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"* Shahandashti et al. (2024b). 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.

3.2. Automatic instantiation of assurance case patterns

Several approaches support the automatic instantiation of assurance case patterns (e.g., Hawkins et al., 2015b,a; Wardziński and Jarzębowicz, 2016; Wardziński and Jones, 2017). For instance, some approaches (e.g., Hawkins et al., 2015b,a) 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 artifacts from system models and mapping these artifacts to placeholders in the ACP to generate an assurance case. Other approaches (e.g., Wardziński and Jarzębowicz, 2016; Wardziński and Jones, 2017) utilized reference tables to keep track of system artifacts, requirements, and the mapping of these artifacts 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 (Hartsell et al., 2021; Ayoub et al., 2012; Lin and Shen, 2015; Lin et al., 2016; Sivakumar et al., 2024a)). Additionally, the use and maintenance of reference tables can be complex and challenging, especially for large systems with numerous requirements and artifacts as evidence. This complexity can lead to the omission of important artifacts 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.

3.3. Rule-based learning in LLMs

LLMs sometimes produce inaccurate results or *'hallucinate'* (Ji et al., 2023). To address this issue, recent research (Yang et al., 2023; Wang et al., 2024b) recommended integrating rule-based knowledge into LLMs. This allows LLMs to rely on structured rules when there are insufficient example-based learning resources, thereby enhancing their accuracy and reliability. Yang et al. (2023) presented a novel learning paradigm allowing LLMs to assimilate knowledge from rules in a manner akin to human learning processes. Their approach leverages the LLMs' in-context capabilities to first extract knowledge from textual rules and then encode this rule knowledge explicitly by training the model using in-context signals. Wang et al. (2024a) introduced a novel 'grammar prompting' technique to improve the ability of LLMs to generate strings from structured languages using a domain-specific grammar in the Backus–Naur Form (BNF). This method augments each example with a specialized grammar sufficient for the output, and the LLM predicts a BNF grammar from the input to generate the output accordingly. Their experiments show that this approach enables LLMs to effectively handle a variety of domain-specific language generation tasks, including semantic parsing and molecule generation.

3.4. LLMs for software modeling

LLMs are currently being utilized for a variety of downstream SE tasks, including software defect prediction (Gomes et al., 2023), static code analysis (Mohajer et al., 2024), automated program repair (Xia et al., 2022), code generation (Ahmad et al., 2021), and software modeling (Chen et al., 2023b,a). In the field of software modeling, Chen et al. (2023b) 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. (2023a) 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. (2023) 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. (2024b) 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 to produce safety cases that are structurally, semantically, and reasonably accurate. Khakzad Shahandashti et al. (2024) 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 our approach, both Sivakumar et al. (2024b) and Khakzad Shahandashti et al. (2024) 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.

4. Approach

Fig. 3 shows a high-level overview of our approach. It consists of four phases that allow LLMs to instantiate assurance cases from assurance case patterns. In phase I (Formalization), based on predicate-based logic, we propose a novel method that formalizes assurance case patterns in compliance with GSN. In phase II (Data collection), we collect the data required to generate assurance cases and validate the generation process. The data consists of assurance cases and assurance case patterns used to manually instantiate these assurance cases. In phase III (Data pre-processing), based on the predicates specified in Phase I, we formalize and represent the assurance case patterns that the LLM would utilize for creating an assurance case. In phase IV (LLM-based assurance case generation), for each assurance case to be generated, we feed a formalized assurance case pattern(s) to an LLM(s) and prompt that LLM to derive an assurance case from it. Note that, one can also adopt other prompt strategies like few-shot learning to get more accurate results by helping the LLM recognize the generation task it needs to perform.

The novelty of our approach 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. patterns) to guide the generation of assurance cases. Below, we describe each step of the approach in more detail.

4.1. Phase I: Formalization of assurance case patterns into predicates

Inspired by the foundational work of Denney and Pai (2013) 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 (Murugesan et al., 2023).

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 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 (Goal Structuring Notation Standard Working Group, 2023) 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.

4.1.1. 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.

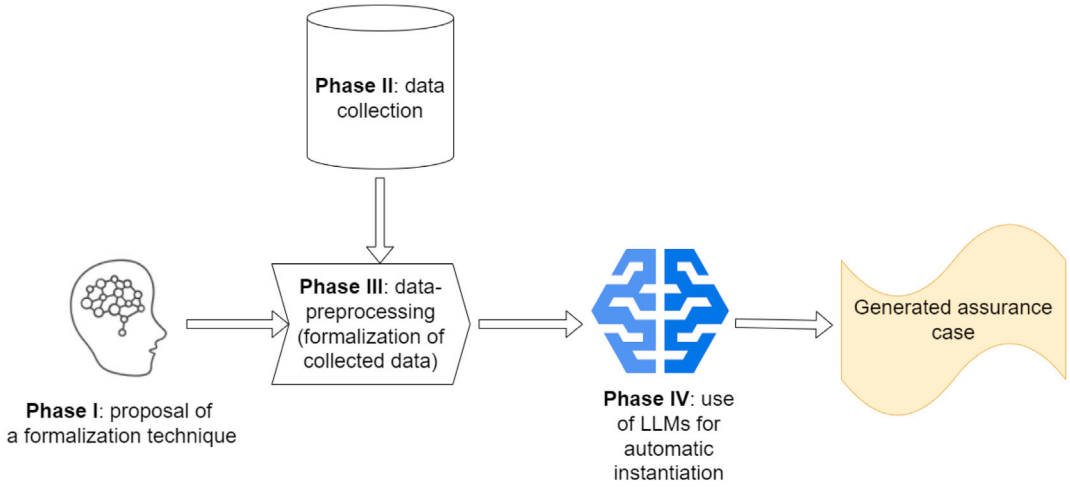

Fig. 3. High-level overview of our approach.

- **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.

4.1.2. 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).

4.1.3. 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. Phase II: Data collection

In our previous work (Odu et al., 2024), we conducted a bibliometric analysis on assurance case patterns. This allowed us to collect 92 primary studies published within the past two decades that 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. We divided the dataset into two categories: one to be used as a one-shot example, and the other for evaluating our approach's performance.

4.3. Phase III: Data pre-processing

Based on the defined predicates for GSN elements, relationships, and decorators we proposed in Phase I, we convert 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). Fig. 4 illustrates a predicate-based representation of an assurance case pattern in our dataset.

```
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)
```

Fig. 4. A simple predicate-based representation of the assurance case pattern depicted in Fig. 9 (see appendix).

4.4. Phase IV: Using LLM to automatically generate assurance cases

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 (Goal Structuring Notation Standard Working Group, 2023) to turn 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 5.5.

5. Experimental setup

5.1. Research questions

Our goal is to explore the potential of generative AI, particularly LLMs, in facilitating the automatic generation of assurance cases complying with specific assurance case patterns. To achieve this, we investigate three research questions (RQs):

(RQ1): Are LLMs capable of creating well-formed and semantically valid assurance cases when they do not have SE (software engineering) knowledge specified in the prompts? In this RQ, we assess the effectiveness of LLMs in creating assurance cases for a given system without providing SE knowledge to the LLMs.

(RQ2): Are LLMs capable of automatically instantiating assurance cases from assurance case patterns when SE knowledge is specified in the prompts? In this RQ, we investigate the impact of providing various types of SE knowledge to our LLMs through prompt engineering. We assess the performance of these LLMs in generating assurance cases based on a given assurance case pattern. The categories of software engineering (SE) knowledge are: (1) examples; (2) domain information; (3) contextual information; (4) predicate-based rules; and (5) a combination of the aforementioned four categories of knowledge. Thus, to facilitate the reasoning about this research question, we further break it into four sub-research questions:

(RQ2.1): Which impact does an example have on the performance of LLMs when it comes to generating assurance cases?

(RQ2.2): Which impact does the domain information have on the performance of LLMs when it comes to generating assurance cases?

(RQ2.3): Which impact does contextual information have on the performance of LLMs when it comes to generating assurance cases?

(RQ2.4): Which impact do predicate-based rules have on the performance of LLMs when it comes to generating assurance cases?

(RQ3): Which of the evaluated LLMs performs best when it comes to automatically instantiating assurance cases from assurance case patterns? In this RQ, we compare the performance of the analyzed LLMs (i.e. GPT-4 Turbo, and GPT-4o) in the automatic instantiation of assurance case from patterns.

5.2. Description of the dataset used in the experiments

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 1 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 our replication package[2].

[2] https://doi.org/10.6084/m9.figshare.27103225.v2

5.2.1. 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 (Zeroual et al., 2023). 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 (Zeroual et al., 2023). 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 (Zeroual et al., 2023). To ensure that ACAS Xu is acceptably secure against security threats, Zeroual et al. (2023) 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.

5.2.2. BlueROV2 and its assurance framework

The BlueROV2 system is an advanced Unmanned Underwater Vehicle (UUV) or underwater Remotely Operated Vehicle (ROV) (Hartsell et al., 2021). Its main objective is to autonomously track pipelines on the seafloor while avoiding static obstacles such as plants and rocks (Hartsell et al., 2021). 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 (Hartsell et al., 2021). Hartsell et al. (2021), 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. Fig. 12 (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.

5.2.3. 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 (Lin et al., 2017). To demonstrate the application of assurance cases for certifying the safety of critical systems, several studies (Lin and Shen, 2015; Lin et al., 2017) have utilized the GPCA system as a case study.

Lin et al. (2017) 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.

5.2.4. 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 (Xu et al., 2017). 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) (Xu et al., 2017). The EM 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 (Xu et al., 2017). Ensuring that the IM server software is acceptably secure requires a demonstration to show that critical assets

Table 1
Overview of our dataset.

System	Domain	Assurance Case Patterns (ACPs)			Assurance Cases (ACs)	
		Decorators	Placeholders	Elements	Elements	Relationships
ACAS XU	Aviation	11	10	22	24	23
BLUEROV2	Automotive	17	8	18	24	21
GPCA	Medical	6	21	23	27	26
IM SOFTWARE	Computing	1	9	15	24	23
DEEPMIND	Medical	16	26	17	23	23

such as user account information, and authentication information are well-protected (Xu et al., 2017). Thus, Xu et al. (2017) 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.

5.2.5. 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 (Ward and Habli, 2020).

Ward and Habli (2020) 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 ML component of the DeepMind system as a one-shot example in our experiments. Our choice of that example is random.

5.3. Large language models setups

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. Our selection of these ChatGPT models was guided by existing literature (Chen et al., 2023b,a; Sivakumar et al., 2024b; Shahandashti et al., 2024b; Viger et al., 2024; Gohar et al., 2024) which highlights it as the most common and effective model for software modeling tasks. Also, the use of GPT-4o and GPT-4 Turbo facilitates the possibility of obtaining reproducible results (Khakzad Shahandashti et al., 2024; Shahandashti et al., 2024b), an essential feature for ensuring consistency in software modeling tasks. 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 (OpenAI, 2023b). 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 text (OpenAI, 2023a). 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", the maximum output token limit available at the time of the experiments, to accommodate longer text across all experiments.

5.4. Description of the experiments and the supporting information

In our experiments, we applied the Chain-of-Thought (CoT) (Wei et al., 2022) 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.

5.4.1. Description of the supporting information

As stated previously, we can specify several categories of SE knowledge in our prompts. Like in the literature (e.g., Chen et al., 2023a; Sivakumar et al., 2024b; Shahandashti et al., 2024b), 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 Appendix A.1).
- **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 artifacts (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 Refs. Hartsell et al. (2021), Lin et al. (2017), Xu et al. (2017), Zeroual et al. (2023) that describe our dataset.

5.4.2. Description of the experiments

Table 2 provides a descriptive summary of the various experiments in our study. Each row, labeled from Experiment 1 to Experiment 9, describes the individual configuration of each experiment, while each column represents one of four distinct categories of software engineering (SE) knowledge: Example, Context Information, Domain Information, and 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, Experiment 2 includes all four, while Experiment 9 only includes Predicate Rules.

Table 2
Overview of our experiments.

	Example	Context information	Domain information	Predicate rules
Experiment 1				
Experiment 2	x	x	x	x
Experiment 3		x	x	x
Experiment 4	x	x		x
Experiment 5		x		x
Experiment 6	x		x	x
Experiment 7			x	x
Experiment 8	x			x
Experiment 9				x

> 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.

Fig. 5. A sample system prompt for Experiment 1.

5.4.2.1. Experiment without software engineering knowledge specified in the prompts. In this experiment (i.e. Experiment 1), to answer RQ1, 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 context information, domain information, examples, and predicate rules to both LLMs.

5.4.2.2. Experiments with SE knowledge specified in the prompts. Building on previous work from literature (e.g., Shahandashti et al., 2024b; Chen et al., 2023a; Sivakumar et al., 2024b) 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.

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.

5.5. Description of the structure of the prompts used in the experiments

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.

5.5.1. 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. Fig. 5 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.
- **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. Fig. 6 shows a sample of our user prompt for Experiment 1.

5.5.2. Output generated by the LLM

The **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.

5.5.3. Description of system prompts with s.e knowledge

Fig. 7 depicts the template (i.e. the generic structure) of the system prompts we used in our experiments with SE knowledge.

5.5.3.1. 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 Fig. 7.

5.5.3.2. 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 Fig. 7.

5.5.4. 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. Fig. 8 shows an excerpt of a sample user prompt in our experiments with SE knowledge.

5.6. Evaluation metrics

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.

5.6.1. Exact match

As in Chang et al. (2024), we employ this metric to gauge the accuracy with which our LLMs 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* (Seatgeek, 2024).

5.6.2. BLEU score

As in Hou et al. (2023), to assess the similarity between the generated text and the ground-truth, we utilize the BLEU score. 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 (Papineni et al., 2002). It ranges between 0 and 1, where 0 indicates 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* (SacreBLEU, 2024) for its assessment.

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.

Fig. 6. A sample user prompt for Experiment 1.

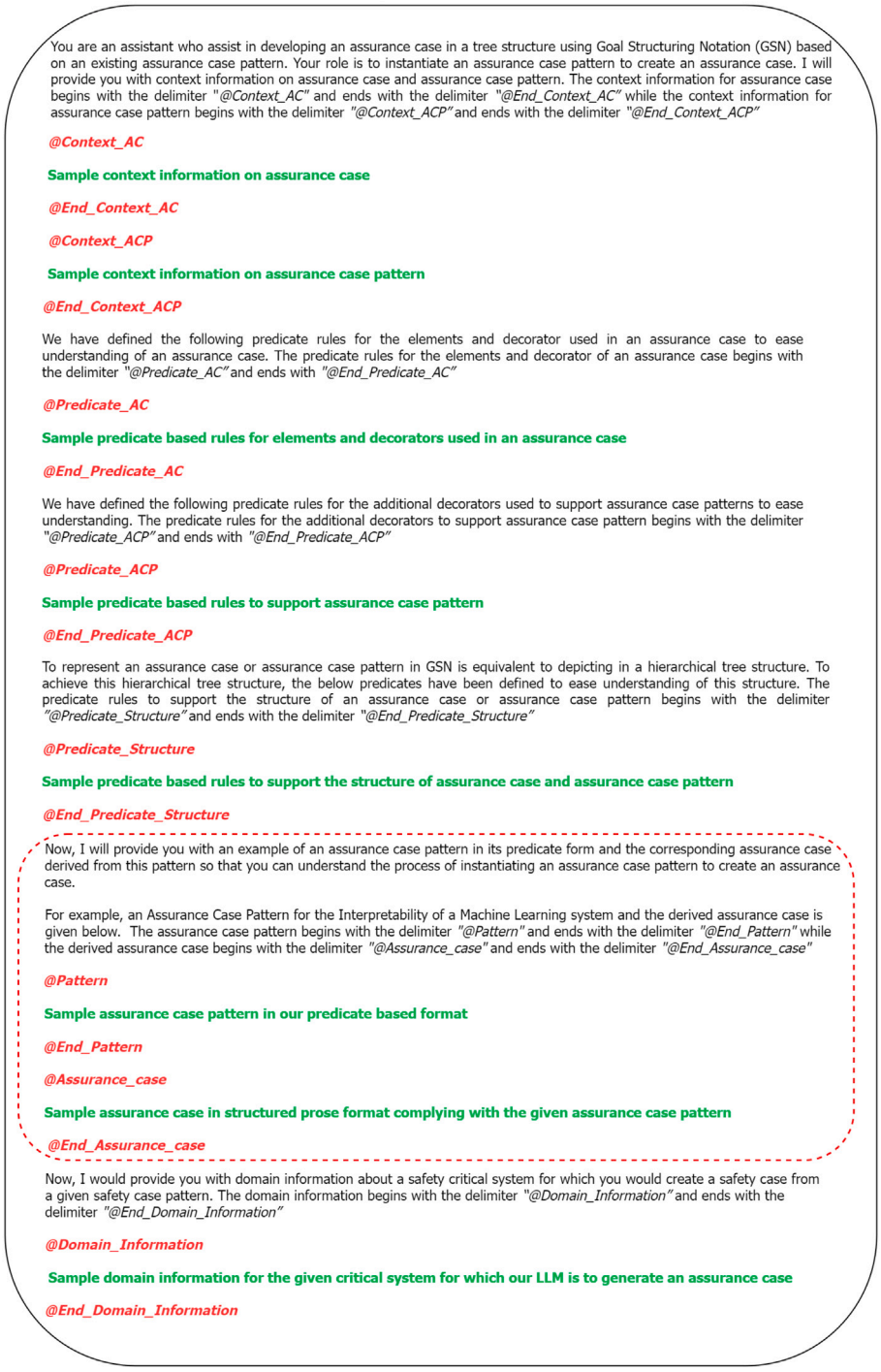

Fig. 7. Generic structure of our system prompts for experiments with SE knowledge.

5.6.3. 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 (Salton and Buckley, 1988; Rahutomo et al., 2012). Cosine similarity values range from −1 to 1, where −1 indicates no similarity or completely dissimilar texts, and 1 indicates identical texts. We rely on a Python library called *scikit-learn* (Pedregosa et al.,

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)

Fig. 8. A sample user prompt for experiments with SE knowledge.

2011) to automatically compute the values of that metric.

6. Results

6.1. RQ1: Are LLMs capable of creating well-formed and semantically valid assurance cases when they do not have SE knowledge specified in their prompts?

6.1.1. Metric results

Table 3 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 3 reports are extremely low and therefore mediocre. More specifically, the exact match values in Table 3 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 3 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 3 shows, the semantic similarity results are slightly better compared to both the Exact match and BLEU score values but still remain below average (i.e. 0.5). Note that the ACAS XU system under GPT-4o and

Table 3
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

DeepMind system under GPT-4o yields the highest semantic similarity median value of 0.23 across all runs in Experiment 1.

6.1.2. 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 Goal Structuring Notation Standard Working Group (2023), 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.
  Furthermore, 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 complicate the understanding of how the evidence (through solutions) supports the arguments within an assurance case.

> In summary, our analysis of Experiment 1 for **RQ1** 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.

6.2. RQ2: Are LLMs capable of automatically instantiating assurance cases from assurance case patterns with SE knowledge specified in the prompts?

6.2.1. Metric results

Similar to the methodology used by Chen et al. (2023a), Sivakumar et al. (2024c), we performed each of our experiments five times (K = 5) for each of the four analyzed systems. Therefore, across the 8 experiments involving SE knowledge, both GPT-4o and GPT-4 Turbo generated a total of 320 assurance cases for our test systems. Tables 4, 5, and 6 respectively report the median of the Exact match results, BLEU scores, and Semantic similarity results we obtained across our 8 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 4 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, 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 6 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 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 considers

Table 4
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
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

Table 6
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

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.

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 ACP, 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 ACP, a single abstract parameter can be instantiated with multiple concrete values, depending on the multiplicity within the pattern structure (Hartsell et al., 2021). 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.

Figs. 10 and 11 (see Appendix) both illustrate the effects of cardinality ambiguity in multiplicity relationships. Fig. 11 illustrates a graphical notation (i.e., a GSN representation) of the assurance case that GPT-4 Turbo generated for the BlueROV2 system. Fig. 10 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

[3] GPT-4 Turbo and GPT-4o generated both Figs. 10 and 11 in structured prose format. Thus, in accordance with the GSN standard guidelines (Goal Structuring Notation Standard Working Group, 2023), we have converted both Figures into a graphical notation (GSN).

the discussion and reasoning about these two figures, we compare both figures with the ground-truth assurance case of the BlueROV2 system that Fig. 13 depicts.

The differences between the assurance cases generated by GPT-4o and GPT-4 Turbo for the BLUEROV2 system are significant. Figs. 10 and 11 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 Fig. 13, 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 Fig. 12 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 G3, three instances of A1, and one instance of G10.

However, as illustrated in Fig. 11, 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 the cardinality of the multiplicity relationships in the pattern illustrated in Fig. 12. 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.

In the remainder of this section, we further discuss the experiment results in light of the RQ2 four sub-research questions.

6.2.2. RQ2.1: Comparative analysis of one-shot vs zero-shot experiments

The results reported in Tables 4 to 6 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.

6.2.3. RQ2.2: Impact of domain information

The results presented in Tables 4 to 6 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.

6.2.4. RQ2.3: Impact of contextual information

Experiments leveraging contextual information are Experiments 2, 3, 4, and 5. The results reported in Tables 4 to 6 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 in accordance with 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. Also, 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.

6.2.5. RQ2.4: 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 4 to 6 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.

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.

6.3. RQ3: which of the evaluated LLMs performs best when it comes to automatically instantiating assurance cases from assurance case patterns?

Tables 4 to 6 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. This suggests that GPT-4o seems to understand subtle ACP-related nuances like cardinality ambiguity in multiplicity relationships better than GPT-4 Turbo.

In summary, for **RQ3**, our analysis reveals that GPT-4o outperforms GPT-4 Turbo in 77% of the 8 experiments incorporating SE knowledge, across various metrics and systems. Overall, GPT-4o shows an enhanced grasp of SE knowledge, including nuanced pattern details like cardinality ambiguity in multiplicity relationships, enabling it to produce assurance cases that are closer to the ground truth compared to GPT-4 Turbo.

7. Discussion

7.1. Analyzing varying one-shot example

As stated in Section 5.2.5, to conduct our four one-shot experiments with SE knowledge, we randomly picked DeepMind's AC together with its ACP 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.

7.1.1. Methodology

To assess the impact of varying the example on Experiment 2 results, we adapted the popular Leave One Out Cross-Validation (LOOCV) (Stone, 1974; Vehtari et al., 2017) 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.

7.1.2. Results

Tables 7, 8, and 9 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 in bold indicate the highest median value for a given test system across the different one-shot examples.

The results in Tables 7 to 9 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 7 to 9 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.

Our analyses of the results we obtained when varying different systems' ACs and ACPs as one-shot examples highlight that the selection of a specific system's AC and ACP as an example is crucial, as it influences the performance of LLMs across different metrics. Thus, choosing the most suitable one-shot example can significantly improve the quality of one-shot experiments.

Table 7
Exact match comparison of varying one-shot example.

Test system	Model	One-shot example				
		ACAS XU	BlueROV2	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

Table 8
BLEU score comparison of varying one-shot example.

Test system	Model	One-shot example				
		ACAS XU	BlueROV2	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

Table 9
Semantic similarity comparison of varying one-shot example.

Test system	Model	One-shot example				
		ACAS XU	BlueROV2	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

7.2. 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.

7.2.1. Methodology used for the manual assessment

As we stated in Section 6.2.4, Experiment 6 yields the best results for both LLMs. We therefore decided to focus on the manual assessment of that experiment's results. To manually assess these results, we relied on a metric called ***reasonability*** (Chen et al., 2023b; Sivakumar et al., 2024b; Shahandashti et al., 2024b; Sivakumar et al., 2024c). A reasonable GSN element is a GSN element that "*could reasonably be in the ground-truth but is not*" (Sivakumar et al., 2024b,c). 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 i.e. GSN elements which the human experts have not thought of but that could have enriched the assurance case.

Two researchers (i.e. raters) – and more specifically two authors 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 (Kendall, 1938) as in the literature (e.g., Khakzad Shahandashti et al., 2024; Sivakumar et al., 2024b; Chen et al., 2023b). 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 Khakzad Shahandashti et al. (2024), Sivakumar et al. (2024b), we relied on an online tool called ***GIGAcalculator***[4] to automatically compute the value of that coefficient with a 95% confidence interval.

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

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

7.2.2. 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 10 reports for each rater, for each LLM, and for 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 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, especially in generating initial drafts of assurance cases — a task that can be laborious, error-prone, and time-consuming when performed manually (Hartsell et al., 2021; Menghi et al., 2023).

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. (2023b) 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. (2024c) 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.

[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.

Table 10
Average reasonability rating of assurance cases 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

8. Threats to validity

8.1. 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 (Mohamad et al., 2021). 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 Fig. 9 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. (2023) 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 and have access to high-quality and large-scale examples, we aim to partner with the industry to gain access to full assurance cases derived from a given pattern.

We utilized the Exact Match, BLEU Score, and Semantic Similarity metrics to assess the quality of the LLM-generated assurance cases against the ground truth assurance cases. These three metrics are among the most widely used ground-truth similarity measures in the literature (e.g., Hou et al., 2023; Sivakumar et al., 2024b; Shahandashti et al., 2024b; Chang et al., 2024) for evaluating both lexical and semantic similarity between LLM-generated content and the ground truth. Finally, as in the literature (e.g., Chen et al., 2023a; Shahandashti et al., 2024b; Sivakumar et al., 2024b), we incorporated a qualitative analysis through a reasonability measure to evaluate the assurance cases generated from Experiment 6. This manual assessment provides a validation approach to ensure that the generated assurance cases are reasonable and logically coherent. Still, there is a slight possibility that the evaluation metrics we adopted are not comprehensive enough for the experiments we performed. Hence, in future work, we plan to explore the use of additional metrics (e.g., ROUGE (Chang et al., 2024), Edit Similarity (Hou et al., 2023)) to further assess our experiment results.

8.2. Construct validity

We extracted the domain information used in our experiments from the following cited references which describe our dataset: (Zeroual et al., 2023; Hartsell et al., 2021; Lin et al., 2017; Xu et al., 2017). Thus, the ACs generated by the two LLMs at hand relied solely on the information available in the cited references. Consequently, key details such as arguments or artifacts related to real-life data or scenarios might have been omitted when deriving domain information.

It is also possible that subtle details of how 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 how the patterns are expressed. We intend to enhance our comprehensive predicate-based rules by integrating multiplicity-related considerations into the predicate formulation. This refinement would aim to address the cardinality ambiguity issue in multiplicity relationships within assurance case patterns, as identified in our experimental analysis. By addressing these issues, we aim to improve the robustness and expand the applicability of our approach.

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 in accordance with the literature focusing on the use of LLMs to automate software modeling tasks (e.g., Chen et al. (2023b), Sivakumar et al. (2024b)). 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.

8.3. 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 in accordance with Shahandashti et al. (2024b), 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 by using more recent datasets coming from the industry, especially the ones that are not publicly available.

9. Conclusion and future work

In this study, we relied on large language models (LLMs) to support the automatic instantiation of assurance cases in compliance with assurance case patterns formalized using predicate-based rules. We conducted a variety of experiments across four systems to evaluate the impact of different categories of software engineering knowledge on the performance the LLMs yield when instantiating 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 need human expertise to refine the LLM-generated assurance cases to make them suitable for the certification of mission-critical systems. This is in accordance with other results in the literature (e.g., Chen et al., 2023b; Sivakumar et al., 2024b).

By combining the expertise of humans with the speed and efficiency of LLMs, we can achieve the swift generation of more reliable assurance cases. This can be accomplished by incorporating additional categories of relevant knowledge, such as the results of Hazard Analysis and Risk Assessment (HARA), system requirements, detailed insights into international standards such as ISO 26262, compliance requirements, ethical considerations, and the operational context relevant to the system under assessment. Moreover, integrating Retrieval-Augmented Generation (RAG) can further enhance this process. RAG leverages the strengths of information retrieval and LLMs, enabling the model to access a wider range of domain-specific artifacts, arguments, and evidence curated by human experts.

We have developed *"SmartGSN"*[6] — the first generative AI-powered online tool for managing assurance cases complying with GSN. This tool offers the capability to automatically convert our LLM-generated assurance cases in the textual format into structured GSN diagrams (argument structure). This feature significantly reduces the manual effort traditionally required to translate the LLM-generated assurance cases in natural language into argument structures, thereby enhancing both the usability and practical benefits of the LLM-generated assurance cases.

In future work, we intend to improve our comprehensive predicate-based rules by integrating multiplicity-related considerations into the predicate formulation. This refinement would aim to address the cardinality ambiguity issue in multiplicity relationships within assurance case patterns, as identified in our experimental analysis. By addressing these issues, we aim to improve the robustness and expand the applicability of our approach. Also, we will further validate our approach by conducting more experiments aiming at instantiating additional assurance case patterns using a wider range of LLMs (e.g., Llama-3, Gemini).

Finally, we plan to integrate the automatic conversion of argument patterns directly into predicate forms within SmartGSN. This enhancement will further minimize user effort, maximizing the efficiency and benefits of our approach and tool. Since our approach can be applied to other contexts that may be very different, we also plan to explore the use of our approach to instantiate software artifacts. Examples could include generating software designs from design patterns, or software architecture models from architectural patterns.

CRediT authorship contribution statement

Oluwafemi Odu: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alvine B. Belle:** Writing – original draft, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Song Wang:** Writing – original draft, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Segla Kpodjedo:** Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Timothy C. Lethbridge:** Writing – original draft, Validation, Methodology, Investigation, Conceptualization. **Hadi Hemmati:** Writing – original draft, Validation, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Oluwafemi Odu reports financial support was provided by York University. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[6] https://smartgsn.vercel.app/''

Acknowledgments

We would like to thank the Mitacs Globalink Research Internship program for helping us secure the funding required to develop *SmartGSN*. We would like to thank the two Mitacs interns (i.e. Emiliano Berrones Gutiérrez and Daniel Méndez Beltrán) who are currently developing *SmartGSN*. We would also like to thank two former graduate students (i.e. Mithila Sivakumar and Kimya Khakzad Shahandashti) for their involvement in the early development of a former *SmartGSN* version.

Appendix

A.1. 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

A.2. Domain information for ACAS Xu system

@Domain_Information

ACAS Xu (Airborne Collision Avoidance System Xu) is a collision avoidance system designed for use in unmanned aerial vehicles (UAVs), commonly known as drones. 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.

The scenario involves two drones. One called the "intruder" which is any other drone or object that poses a collision threat to the ownship. and the other called the "ownship" which is the perspective we adopt. The ownship is equipped with ACAS Xu and has a functional space in which it must operate. This space is conceptually partitioned into two operational areas: collision avoidance threshold and collision volume with an elevated risk of collision for the ownship with intruders. When no risk of collision is detected, the ownship follows the current heading to the destination area. Otherwise, if another drone is detected in the collision volume, the ownship will turn right or left to avoid the collision and prevent the intruder from reaching the collision avoidance threshold.

The architecture of ACAS Xu contains the following components.

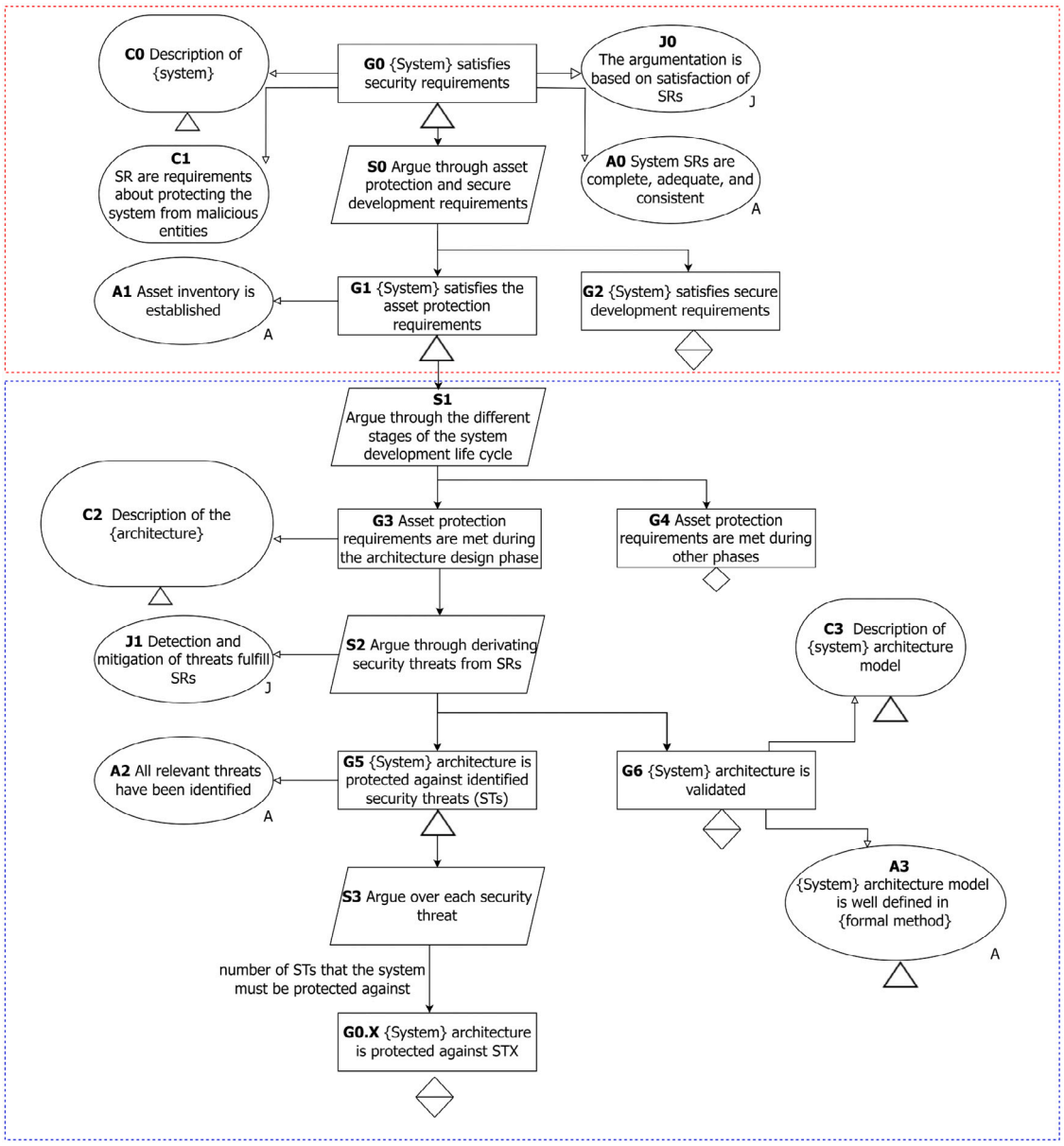

Fig. 9. Assurance case pattern for Threat Identification. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
Source: Adapted from Zeroual et al. (2023).

- Sensors: The ownship's sensors gather data on potential intruders, including their velocity, angle, and distance relative to the ownship.
- Processor: The collected data is processed to compute a suitable avoidance strategy (e.g., turn left, turn right, or do nothing).
- Planner: Based on the processor's decision, a trajectory is planned to navigate the ownship safely while avoiding collisions.
- Actuator: The planned trajectory is executed by the actuator, ensuring the ownship follows the new path.

ACAS Xu's security can be compromised if an attacker alters the messages sent to the processor, leading to incorrect decisions that may result in collisions. Therefore, ensuring the security of ACAS Xu involves: security requirements decomposition that aims to identify security threats, and formalization of the system and the security threats to later verify the absence of threats when developing a secure system. If it can be shown that all the relevant threats have been identified and mitigated, then the system is acceptably secure.

The following security requirements (SRs) below are imposed to design a secure ACAS Xu.

- SR1: The GPS messages are genuine and have not been intentionally altered.
- SR2: The processor must receive data only from valid sensors.
- SR3: The system should employ mechanisms to mitigate unauthorized disclosure of the planning information.
- SR4: ACAS Xu development shall be done considering security risk assessment procedures.

The four SR are decomposed into requirements about the satisfaction of asset protection (SR1–SR3) and secure development process requirements (SR4). The former concerns requirements to protect resources that are worth protecting. The latter concerns the requirements about the development activities that must conform to a relevant secure development methodology and/or security standard.

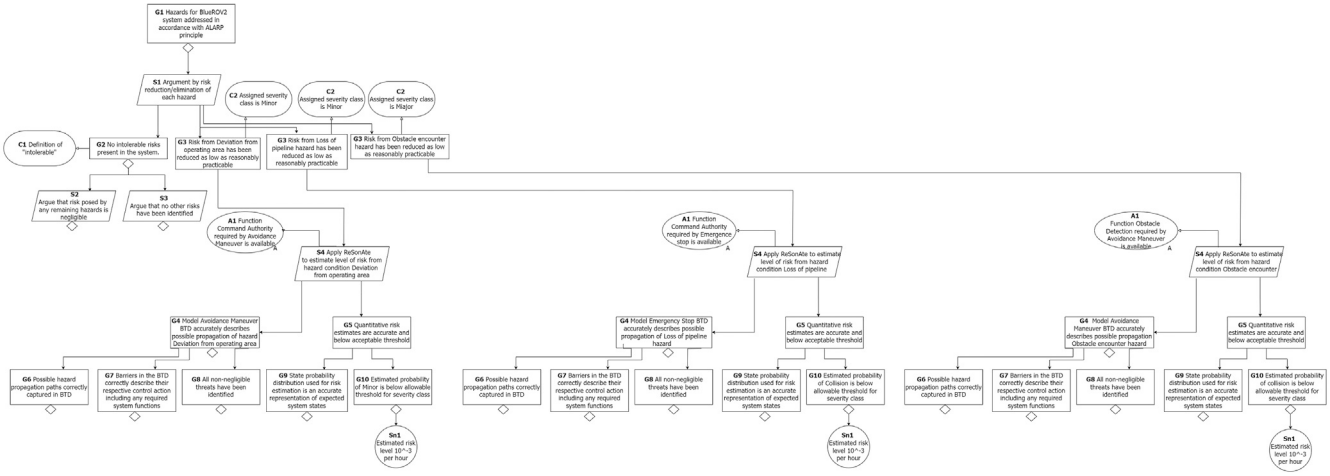

Fig. 10. An Assurance case for the BLUEROV2 System Generated by GPT-4o.

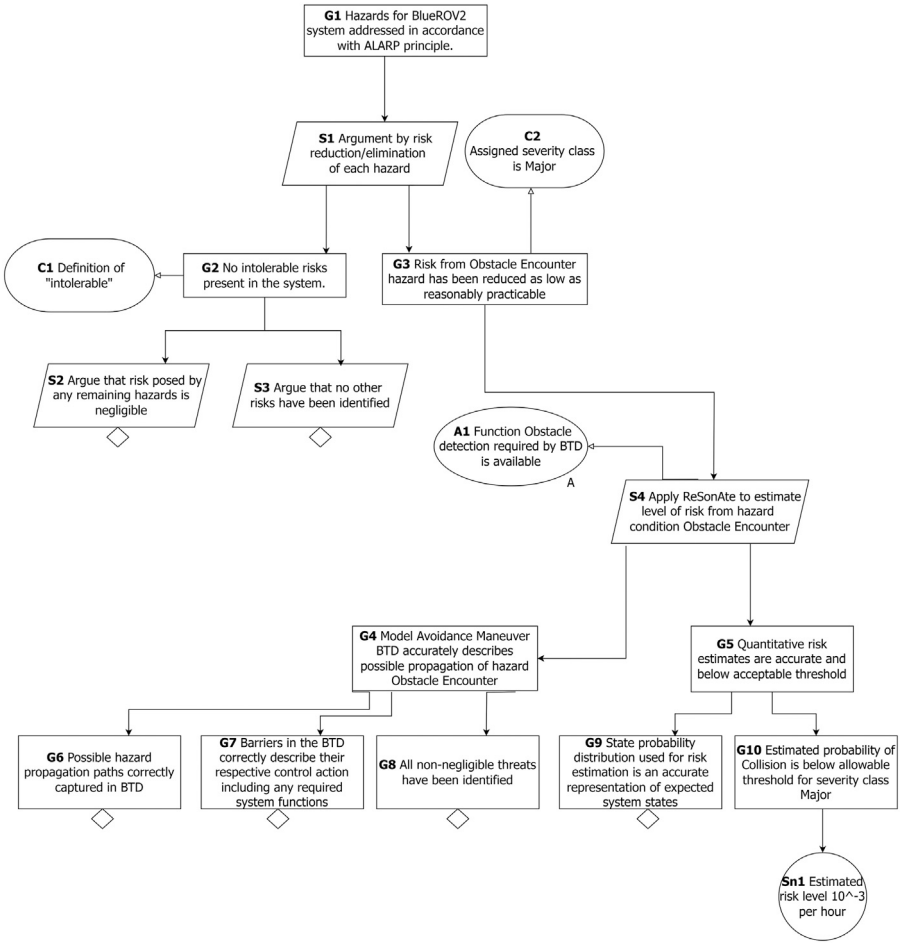

Fig. 11. An assurance case for the BLUEROV2 system generated by GPT-4 Turbo.

In addition, ACAS Xu has low level elements that capture functional architecture in terms of components and connectors, and the behavioral aspects of the architectural elements. These elements include the following.

- Component: a modeling artifact which represents a piece of software architecture.
- MsgPassing: the representation of a message exchanged between two components (sender, receiver).
- Port: the interaction point through which a Component can communicate with its environment.
- ConnectorMPS: a link that enables communication between Ports.
- Payload: the useful data contained in a Message.

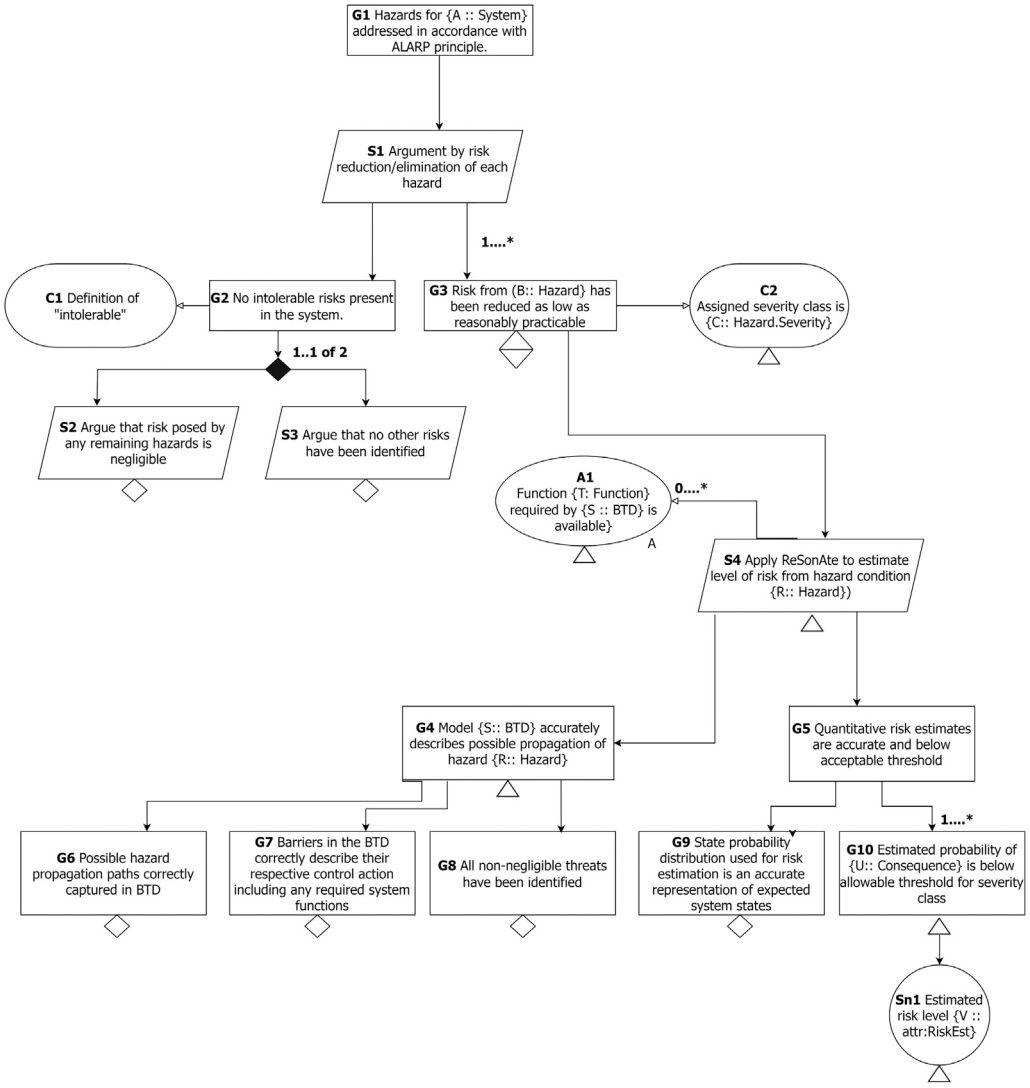

Fig. 12. The ALARP pattern sequentially composed with the ReSonAte pattern.
Source: Adapted from Hartsell et al. (2021).

Based on the Microsoft STRIDE threat analysis technique, the following security threats (STs) against the components and the communication links are identified from the security requirements (SRs).

- ST1: Tampering — This threat is identified from SR1 and involves GPS sensors and processor.
- ST2: Spoofing — SR2 Sensors and processor
- ST3: Elevation of privileges — SR3 Planning system

Finally, to ensure that ACAS Xu is acceptably secure, during the creation of its security case, an instance of the goal (G0.X) is created for each security threat against which the system must be protected, where X denotes the order of the threat.

@End_Domain_Information

A.3. Assurance case pattern for ACAS XU system

See Fig. 9.

A.4. Assurance case for the BLUEROV2 system generated by GPT-4o

See Fig. 10.

A.5. Assurance case for the BLUEROV2 system generated by GPT-4 Turbo

See Fig. 11.

A.6. Assurance case pattern for BlueROV2 system

See Fig. 12.

A.7. Ground truth assurance case for the BlueROV2 system

See Fig. 13.

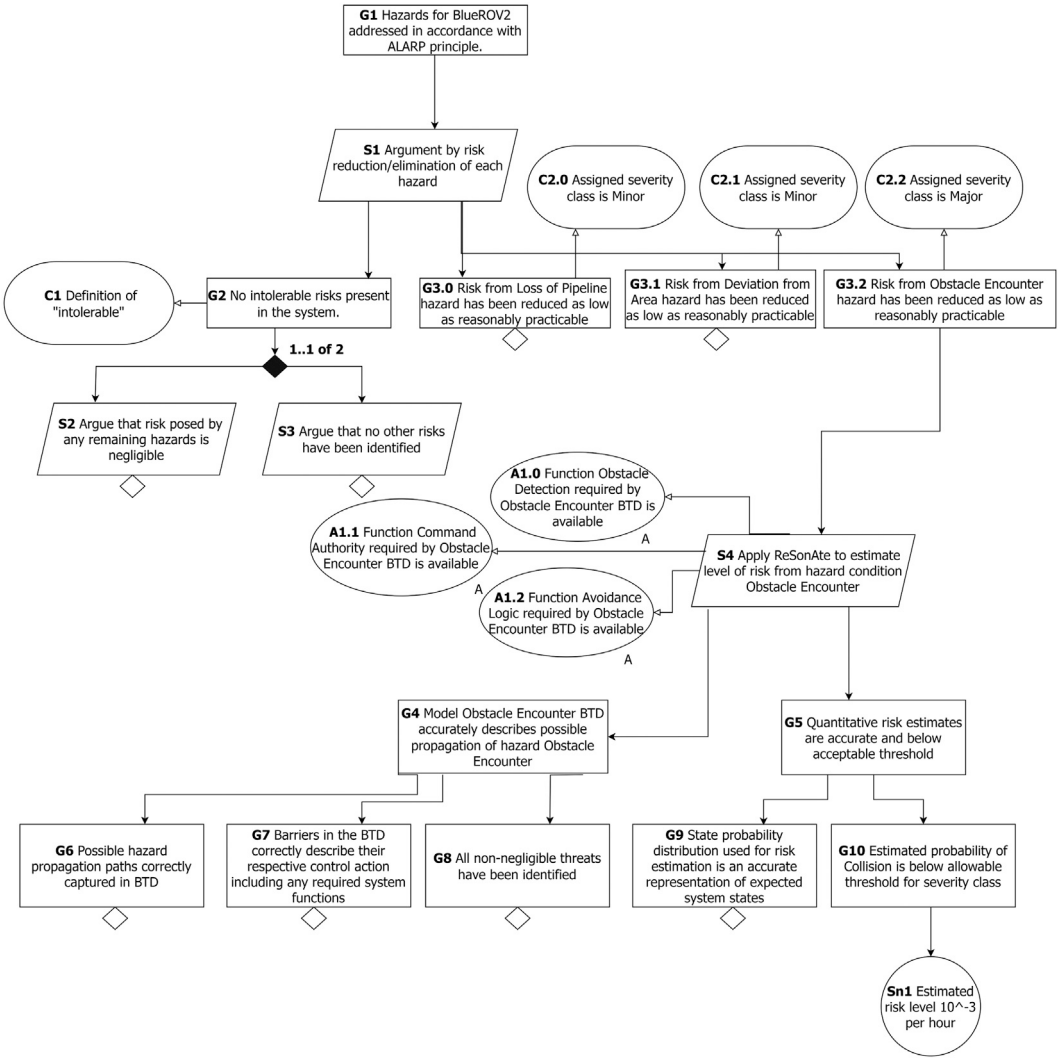

Fig. 13. Ground truth assurance case for the BlueROV2 system.
Source: Adapted from Hartsell et al. (2021).

Data availability

We have shared the link to our data.

References

Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K.W., 2021. Unified pre-training for program understanding and generation. arXiv preprint arXiv:2103.06333.

Alexander, R., Kelly, T., Kurd, Z., McDermid, J., 2007. Safety Cases for Advanced Control Software: Safety Case Patterns. Final Report, NASA Contract FA8655-07-1-3025, Univ. of York.

Ayoub, A., Kim, B., Lee, I., Sokolsky, O., 2012. 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, pp. 141–146.

Bagheri, M., Lamp, J., Zhou, X., Feng, L., Alemzadeh, H., 2022. Towards developing safety assurance cases for learning-enabled medical cyber-physical systems. arXiv preprint arXiv:2211.15413.

Belle, A.B., Lethbridge, T.C., Kpodjedo, S., Adesina, O.O., Garzón, M.A., 2019. A novel approach to measure confidence and uncertainty in assurance cases. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops. REW, IEEE, pp. 24–33.

Beyene, T.A., Carlan, C., 2021. 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, pp. 63–66.

Bloomfield, R., Bishop, P., 2009. 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, pp. 51–67.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020. Language models are few-shot learners. Adv. Neural Inf. Process. Syst. 33, 1877–1901.

Burton, S., Gauerhof, L., Sethy, B.B., Habli, I., Hawkins, R., 2019. 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, pp. 365–377.

Cârlan, C., Beyene, T.A., Ruess, H., 2016. Integrated formal methods for constructing assurance cases. In: 2016 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 221–228.

Carlan, C., Gallina, B., 2020. Enhancing state-of-the-art safety case patterns to support change impact analysis. In: 30th European Safety and Reliability Conference.

Chaaben, M.B., Burgueño, L., Sahraoui, H., 2023. 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, pp. 7–12.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al., 2024. A survey on evaluation of large language models. ACM Trans. Intell. Syst. Technol. 15 (3), 1–45.

Chen, B., Chen, K., Hassani, S., Yang, Y., Amyot, D., Lessard, L., Mussbacher, G., Sabetzadeh, M., Varró, D., 2023a. On the use of GPT-4 for creating goal models: an exploratory study. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops. REW, IEEE, pp. 262–271.

Chen, K., Yang, Y., Chen, B., López, J.A.H., Mussbacher, G., Varró, D., 2023b. 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, pp. 162–172.

Denney, E., Pai, G., 2013. 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, pp. 21–32.

Devlin, J., Chang, M.W., Lee, K., Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Finnegan, A., McCaffery, F., 2014. A security argument pattern for medical device assurance cases. In: 2014 IEEE International Symposium on Software Reliability Engineering Workshops. IEEE, pp. 220–225.

Goal Structuring Notation Standard Working Group, 2023. GSN (version 3). URL: https://scsc.uk/gsn. (Accessed 30 November 2023).

Gohar, U., Hunter, M.C., Lutz, R.R., Cohen, M.B., 2024. Codefeater: Using llms to find defeaters in assurance cases. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. pp. 2262–2267.

Gomes, L., da Silva Torres, R., Côrtes, M.L., 2023. BERT-and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study. Inf. Softw. Technol. 160, 107217.

Goodenough, J.B., Weinstock, C.B., Klein, A.Z., 2015. Eliminative Argumentation: A Basis for Arguing Confidence in System Properties. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005, Citeseer.

Graydon, P.J., 2015. Formal assurance arguments: A solution in search of a problem? In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, pp. 517–528.

Graydon, P.J., Knight, J.C., Strunk, E.A., 2007. Assurance based development of critical systems. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. DSN'07, IEEE, pp. 347–357.

Hartsell, C., Mahadevan, N., Dubey, A., Karsai, G., 2021. Automated method for assurance case construction from system design models. In: 2021 5th International Conference on System Reliability and Safety. ICSRS, IEEE, pp. 230–239.

Hawkins, R.D., Habli, I., Kelly, T., 2015a. The need for a weaving model in assurance case automation. Ada User J. 187–191.

Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T., 2015b. Weaving an assurance case from design: a model-based approach. In: 2015 IEEE 16th International Symposium on High Assurance Systems Engineering. IEEE, pp. 110–117.

Hawkins, R., Kelly, T., Knight, J., Graydon, P., 2011. 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, pp. 3–23.

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

Holloway, C.M., 2013. Making the implicit explicit: Towards an assurance case for DO-178C. In: International System Safety Conference, No. NF1676L-16361.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H., 2023. Large language models for software engineering: a systematic literature review (2023). arXiv preprint arXiv:2308.10620.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P., 2023. Survey of hallucination in natural language generation. ACM Comput. Surv. 55 (12), 1–38.

Kelly, T., Weaver, R., 2004. The goal structuring notation–a safety argument notation. In: Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases, Vol. 6. NJ, Citeseer Princeton.

Kendall, M.G., 1938. A new measure of rank correlation. Biometrika 30 (1–2), 81–93.

Khakzad Shahandashti, K., Sivakumar, M., Mohajer, M.M., Boaye Belle, A., Wang, S., Lethbridge, T., 2024. 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. pp. 52–56.

King, A.L., Feng, L., Procter, S., Chen, S., Sokolsky, O., Hatcliff, J., Lee, I., 2015. 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, pp. 228–242.

Lin, C.L., Shen, W., 2015. Applying safety case pattern to generate assurance cases for safety-critical systems. In: 2015 IEEE 16th International Symposium on High Assurance Systems Engineering. IEEE, pp. 255–262.

Lin, C.L., Shen, W., Drager, S., 2016. A framework to support generation and maintenance of an assurance case. In: 2016 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 21–24.

Lin, C.L., Shen, W., Hawkins, R., 2017. Support for safety case generation via model transformation. ACM SIGBED Rev. 14 (2), 44–52.

Maksimov, M., Kokaly, S., Chechik, M., 2019. A survey of tool-supported assurance case assessment techniques. ACM Comput. Surv. 52 (5), 1–34.

Mansourov, N., Campara, D., 2010. System Assurance: Beyond Detecting Vulnerabilities. Elsevier.

Matsuno, Y., 2011. D-Case Editor: A Typed Assurance Case Editor. University of Tokyo.

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

Matsuno, Y., Taguchi, K., 2011. Parameterised argument structure for GSN patterns. In: 2011 11th International Conference on Quality Software. IEEE, pp. 96–101.

Menghi, C., Viger, T., Di Sandro, A., Rees, C., Joyce, J., Chechik, M., 2023. 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, pp. 135–139.

Mohajer, M.M., Aleithan, R., Harzevili, N.S., Wei, M., Belle, A.B., Pham, H.V., Wang, S., 2024. Effectiveness of ChatGPT for static analysis: How far are we? In: Proceedings of the 1st ACM International Conference on AI-Powered Software. pp. 151–160.

Mohamad, M., Steghöfer, J.P., Scandariato, R., 2021. Security assurance cases—state of the art of an emerging approach. Empir. Softw. Eng. 26 (4), 70.

Murugesan, A., Wong, I.H., Stroud, R., Arias, J., Salazar, E., Gupta, G., Bloomfield, R., Varadarajan, S., Rushby, J., 2023. Semantic analysis of assurance cases using s (CASP). In: Goal Directed Execution of Answer Set Programs (GDE) Workshop in Int'l Conf. on Logic Programming. ICLP workshops

Napolano, M., Machida, F., Pietrantuono, R., Cotroneo, D., 2015. Preventing recurrence of industrial control system accident using assurance case. In: 2015 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 182–189.

Naveed, H., Khan, A.U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Barnes, N., Mian, A., 2023. A comprehensive overview of large language models. arXiv preprint arXiv:2307.06435.

Nguyen, E.A., Ellis, A.G., 2011. Experiences with assurance cases for spacecraft safing. In: 2011 IEEE 22nd International Symposium on Software Reliability Engineering. IEEE, pp. 50–59.

Odu, O., Belle, A.B., Wang, S., Shahandashti, K.K., 2024. A PRISMA-driven bibliometric analysis of the scientific literature on assurance case patterns. arXiv preprint arXiv:2407.04961.

OMG, 2021. Structured assurance case metamodel (version 2.2). URL: https://www.omg.org/spec/SACM/2.2/About-SACM. (Accessed 27 January 2024).

OpenAI, 2023a. How should I set the temperature parameter. URL: https://platform.openai.com/docs/guides/text-generation/how-should-i-set-the-temperature-parameter. (Accessed 15 May 2024).

OpenAI, 2023b. Openai API. URL: https://openai.com/api/.

Palin, R., Ward, D., Habli, I., Rivett, R., 2011. ISO 26262 Safety Cases: Compliance and Assurance. IET.

Papineni, K., Roukos, S., Ward, T., Zhu, W.J., 2002. Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. pp. 311–318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Picardi, C., Hawkins, R., Paterson, C., Habli, I., 2019. 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, pp. 165–179.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. 21 (140), 1–67.

Rahutomo, F., Kitasuka, T., Aritsugi, M., et al., 2012. Semantic cosine similarity. In: The 7th International Student Conference on Advanced Science and Technology, Vol. 4, No. 1. ICAST, University of Seoul South Korea, p. 1.

Robert, P., Ibrahim, H., 2010. Assurance of automotive safety–a safety case approach. In: Proc. 29th International Conference, Vol. 2010. SAFECOMP, pp. 82–96.

Romera-Paredes, B., Torr, P., 2015. An embarrassingly simple approach to zero-shot learning. In: International Conference on Machine Learning. PMLR, pp. 2152–2161.

SacreBLEU, 2024. SacreBLEU. URL: https://github.com/mjpost/sacreBLEU. (Accessed 25 June 2024).

Salton, G., Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. Inf. Process. Manage. 24 (5), 513–523.

Seatgeek, 2024. Seatgeek/fuzzywuzzy: Fuzzy string matching in Python. URL: https://github.com/seatgeek/fuzzywuzzy?tab=readme-ov-file. (Accessed 25 June 2024).

Selviandro, N., Hawkins, R., Habli, I., 2020. 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, pp. 3–18.

Shahandashti, K.K., Belle, A.B., Lethbridge, T.C., Odu, O., Sivakumar, M., 2024a. A PRISMA-driven systematic mapping study on system assurance weakeners. Inf. Softw. Technol. 107526.

Shahandashti, K.K., Belle, A.B., Mohajer, M.M., Odu, O., Lethbridge, T.C., Hemmati, H., Wang, S., 2024b. Using GPT-4 turbo to automatically identify defeaters in assurance cases. In: 2024 IEEE 32nd International Requirements Engineering Conference Workshops. REW, IEEE, pp. 46–56.

Sivakumar, M., Belle, A.B., Shahandashti, K.K., Odu, O., Hemmati, H., Kpodjedo, S., Wang, S., Adesina, O.O., 2024a. I came, I saw, I certified: some perspectives on the safety assurance of cyber-physical systems. arXiv preprint arXiv:2401.16633.

Sivakumar, M., Belle, A.B., Shan, J., Adesina, O., Wang, S., Chechik, M., Fokaefs, M., Shahandashti, K.K., Odu, O., 2023. The last decade in review: Tracing the evolution of safety assurance cases through a comprehensive bibliometric analysis. arXiv preprint arXiv:2311.07495.

Sivakumar, M., Belle, A.B., Shan, J., Shahandashti, K.K., 2024b. 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, pp. 35–45.

Sivakumar, M., Belle, A.B., Shan, J., Shahandashti, K.K., 2024c. Prompting GPT–4 to support automatic safety case generation. Expert Syst. Appl. 255, 124653.

Snell, J., Swersky, K., Zemel, R., 2017. Prototypical networks for few-shot learning. Adv. Neural Inf. Process. Syst. 30.

Stone, M., 1974. Cross-validatory choice and assessment of statistical predictions. J. R. Stat. Soc. Ser. B Stat. Methodol. 36 (2), 111–133.

Sun, Y., Wang, S., Li, Y., Feng, S., Chen, X., Zhang, H., Tian, X., Zhu, D., Tian, H., Wu, H., 2019. Ernie: Enhanced representation through knowledge integration. arXiv preprint arXiv:1904.09223.

Vehtari, A., Gelman, A., Gabry, J., 2017. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Stat. Comput. 27, 1413–1432.

Vierhauser, M., Bayley, S., Wyngaard, J., Xiong, W., Cheng, J., Huseman, J., Lutz, R., Cleland-Huang, J., 2019. Interlocking safety cases for unmanned autonomous systems in shared airspaces. IEEE Trans. Softw. Eng. 47 (5), 899–918.

Viger, T., Diemert, S., Foster, O., 2023. Patterns for integrating NIST 800-53 controls into security assurance cases. In: International Conference on Computer Safety, Reliability, and Security. Springer, pp. 165–175.

Viger, T., Murphy, L., Diemert, S., Menghi, C., Joyce, J., Di Sandro, A., Chechik, M., 2024. AI-supported eliminative argumentation: Practical experience generating defeaters to increase confidence in assurance cases. In: 2024 IEEE 35th International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 284–294.

Wagner, S., Schätz, B., Puchner, S., Kock, P., 2010. 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, pp. 269–278.

Wang, B., Wang, Z., Wang, X., Cao, Y., A Saurous, R., Kim, Y., 2024a. Grammar prompting for domain-specific language generation with large language models. Adv. Neural Inf. Process. Syst. 36.

Wang, S., Wei, Z., Choi, Y., Ren, X., 2024b. Can LLMs reason with rules? Logic scaffolding for stress-testing and improving LLMs. arXiv preprint arXiv:2402.11442.

Ward, F.R., Habli, I., 2020. 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, pp. 395–407.

Wardziński, A., Jarzębowicz, A., 2016. 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, pp. 87–98.

Wardziński, A., Jones, P., 2017. 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, pp. 39–51.

Wei, R., Kelly, T.P., Dai, X., Zhao, S., Hawkins, R., 2019. Model based system assurance using the structured assurance case metamodel. J. Syst. Softw. 154, 211–233.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al., 2022. Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural Inf. Process. Syst. 35, 24824–24837.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D.C., 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. arXiv preprint arXiv:2302.11382.

Wu, T., He, S., Liu, J., Sun, S., Liu, K., Han, Q.L., Tang, Y., 2023. A brief overview of ChatGPT: The history, status quo and potential future development. IEEE/CAA J. Autom. Sin. 10 (5), 1122–1136.

Xia, C.S., Wei, Y., Zhang, L., 2022. Practical program repair in the era of large pre-trained language models. arXiv preprint arXiv:2210.14179.

Xu, B., Lu, M., Zhang, D., 2017. A layered argument strategy for software security case development. In: 2017 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 331–338.

Yang, W., Lin, Y., Zhou, J., Wen, J., 2023. Enabling large language models to learn from rules. arXiv preprint arXiv:2311.08883.

Zeroual, M., Hamid, B., Adedjouma, M., Jaskolka, J., 2023. Formal model-based argument patterns for security cases. In: Proceedings of the 28th European Conference on Pattern Languages of Programs. pp. 1–12.

Zhu, S., Lu, M., Xu, B., 2018. Software reliability case development method based on the 4+ 1 principles. In: 2018 12th International Conference on Reliability, Maintainability, and Safety. ICRMS, IEEE, pp. 197–202.

Oluwafemi Odu is a research assistant at the Department of Electrical Engineering and Computer Science at York University, Toronto, Canada. His research interests include the safety assurance of cyber–physical systems, and machine learning. He is currently a master's student at the Department of Electrical Engineering and Computer Science at York University, Toronto, Canada. He just defended his M.Sc. thesis. The latter has been recommended for the best thesis award.

Alvine B. Belle is an Assistant Professor at the Department of Electrical Engineering and Computer Science at York University, Toronto, Canada. Her work focuses on generative AI, safety assurance, autonomous systems and EDI in computing. She holds a Ph.D. in software engineering from the University of Quebec (Ecole de Technologie Supérieure), Montreal, Canada. She worked for two years as a postdoctoral researcher at the University of Ottawa, Ottawa, Canada. She has recently completed a graduate diploma in public administration and governance at McGill University, Montreal, Canada. She has authored several papers in top-tier venues such as Information and Software Technology, ACM Computing Surveys, ICSA (formerly ECSA), and ACM Transactions on Computing Education.

Song Wang received the dual B.E. degrees from Sichuan University, the master's degree from the Institute of Software Chinese Academy of Sciences, and the Ph.D. degree from the University of Waterloo. He is an Associate Professor with the York University, Canada. He worked at the intersection of software engineering and artificial intelligence. He has more than 60 high-quality publications including IEEE Transactions on software engineering, ACM Computing Surveys, ICSE, TOSEM, FSE, ASE. He is the recipient of four Distinguished/Best Paper Awards. He is currently serving as an Associate Editor of ACM Transactions on Software Engineering.

Segla Kpodjedo received a Ph.D. degree in software engineering from Ecole Polytechnique de Montreal, Canada. After getting his PhD degree, he worked as a postdoctoral fellow at Ecole Polytechnique de Montreal, Montreal, Canada. He currently works as a Professor at the University of Quebec (Ecole de Technologie Superieure), Montreal, Canada. His current research interests include machine learning, software evolution, empirical studies in software engineering, graph matching, combinatorial optimization, and meta-heuristics. He is the author of dozens of papers published in top-tier venues including MODELS, ICPC, ICSA (formerly ECSA), GECCO, ISSRE, IEEE Transactions on software engineering, Information and Software Technology and Empirical Software Engineering.

Timothy C. Lethbridge is a Professor at the University of Ottawa, Canada. His research currently focuses on software modeling tools, particularly the user experience of such tools, their educational use, code generation, and the Umple technology. He is a licensed Professional Engineer, a senior member of both ACM and IEEE and a fellow of the Canadian Information Processing Society (CIPS). He received the IEEE Computer Society TCSE Outstanding Educator Award in 2016. He is co-general chair of the International Conference on Software Engineering (ICSE) 2025 in Ottawa.

Hadi Hemmati is an Associate Professor at the electrical engineering and computer science department, at York University. Previously he was an Associate Professor at the electrical and software engineering department at the University of Calgary, AB, Canada. In the past, he was also an Assistant Professor at the University of Manitoba, and a postdoctoral fellow at the University of Waterloo, and Queen's University. He received his Ph.D. from the University of Oslo, Norway. His main research interests are automated software engineering (with a focus on software testing, debugging, and repair), and trustworthy AI (with a focus on robustness and explainability). His research has a strong focus on pragmatic software/ML solutions for large scale systems and empirically investigating them in practice. He has been a PI on multiple industry research projects in different domains such as IT, aviation, insurance, urban development, fintech, and beyond.