Security Argument patterns for Deep Neural Network Development

MARWA ZEROUAL, CEA LIST, IRIT, University of Toulouse
BRAHIM HAMID, IRIT, University of Toulouse
MORAYO ADEDJOUMA, CEA List
JASON JASKOLKA, Carleton University

The use of Machine Learning-based systems, particularly Deep Learning, is overgrowing and finding uses in many industries. Most of these industries have critical safety, security, and dependability requirements. The verification of deep neural networks (DNN) is gaining more interest among the research community, given the use cases demonstrating that they can be attacked and fooled in several ways. Assuring that security requirements have been met and detecting flaws in the early phases of the DNN development is less expensive than changes after deployment on a target system. There is a want for a security assurance case for DNN development similar to that used in classical information security development. A security assurance case is a credible argument supported by evidence, demonstrating that the system satisfies its security requirements and objectives. In this paper, we derive security argument patterns for DNN development, emphasizing verification of the fulfillment of security requirements. We present the argument patterns using the GSN pattern notation. We apply the patterns to build security cases of a DNN used in an autonomous drone case study system.

Categories and Subject Descriptors: D.3.3 [**Language Constructs and Features**] Patterns; K.6.5 [**Security and Protection**]

General Terms: Documentation; Management; Security

Additional Key Words and Phrases: Security cases, deep neural networks, argument pattern, assurance, adversarial examples

ACM Reference Format:

Zeroual, M., Hamid, B., Adedjouma, M. and Jaskolka, J. 2024. A Secure Development Security Argument patterns for Deep Neural Network Development HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 1 (October 2023), 18 pages.

1. INTRODUCTION

DNN have recently witnessed widespread adoption in different industries. Among the safety and security-critical domains, we find them in automotive systems [Kang and Kang(2016)], health care [Miotto et al.(2018)], and air traffic control systems [Lin et al.(2021)]. Despite the fact that DNN guarantee satisfactory results, they introduce new potential security risks. For instance, DNN were used to build a pedestrian detector for a self-driving car. This self-driving car killed a pedestrian who was not near the crosswalk [Harris(2019)]. Thus, an attacker might exploit the lack of the learning phase or tamper with the entries of the DNN to force a self-driving car to behave in dangerous ways and possibly cause an accident. Moreover, DNN are known to be vulnerable to a

Author's address: M. ZEROUAL, and M. Adedjouma, Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France; email: {marwa.zeroual,morayo.dedjouma}@cea.fr; B. Hamid, IRIT, University of Toulouse, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France; email: brahim.hamid@irit.fr; J. Jaskolka, Department of Systems and Computer Engineering, 1125 Colonel By Drive, Ottawa ON K1S 5B6, Canada; email: jason.jaskolka@carleton.ca;

wide range of security threats, including adversarial examples, overfitting, data poisoning, model extraction, and Trojan attacks[Hanif et al.(2018), Carlini and Wagner(2017)]. For such critical domains, we need to question the trustworthiness of the DNN-based systems. Also, we need to establish justified confidence that the system that incorporates DNN components was designed and would behave securely and safely. One possible way to justify this confidence is by using assurance cases. Assurance cases provide an established approach for justifying the safety of systems by providing an explicit argument supported by compelling evidence [Weinstock et al.(2007)]. When assurance cases aim to demonstrate the security of a system, they are known as security cases.
The security assurance of DNN-based systems differs from classical systems because DNN are data-driven and also have characteristics quite different from classical systems. The key open challenge is the need for security cases for data-driven applications and the development of learning algorithms with provable security guarantees. In this paper, we present two security assurance argument patterns. The first one is called the DNN secure development argument pattern. It forms the primary building blocks of a structured security assurance case. It shows the different stages (requirements engineering, data-oriented, model-oriented) that must be ensured when developing DNN components. The second one is called the security requirement satisfaction argument pattern. It focuses on one claim among the leaf claims presented in the first pattern. It proposes two strategies to verify the fulfillment of security requirements by the learned model, either by formal verification or testing.
We use the pattern template provided by [Kelly(1999)] to describe our argument patterns. The template is adapted from the Gang of Four template [Gamma et al.(1994)]. It uses the Goal Structuring Notation (GSN) [GSN Working Group(2021)] to describe the structural details of the pattern graphically, and rather than sample source code, it includes sample text for the eventual assurance case. The proposed patterns are meant to be used by different system development stakeholders. The security engineers can rely on our patterns to determine the DNN-specific security considerations and evaluate the security of the DNN based systems. Indeed, DNN developers can derive and satisfy the security requirements allocated to DNN components using our patterns. Moreover, the patterns are useful to other stakeholders(researchers and students) who require assurance that the security considerations have been explicitly and systematically considered. Other types of machine learning, rather than deep learning, may also benefit from this guidance, particularly regarding security requirements and data management. The underpinning argument patterns are domain-independent. However, in this paper, we only show how to apply them for a DNN from the autonomous driving airborne.

The remainder of this paper is organized as follows. Section 2 provides definitions that will be useful in understanding our work. Section 3 details the DNN lifecycle stages. Section 4 presents the security argument patterns. Section 5 presents a case study and exemplifies the application of the patterns via a collision avoidance drone case study. Section 6 reviews related works and positions our contribution. Finally, Section 7 concludes with perspective work directions.

2. BACKGROUND

In this section, we present a set of concepts and definitions that will be useful for understanding our work.

2.1 Security assurance cases

Assurance cases represent a reasoned and compelling argument comprising combinations of structured claims and sub-claims. They support evidence related to the system design, implementation, and development process. They demonstrate that the system will achieve properties for a given application in a given environment [Kelly(1999)]. When assurance cases claim system security properties, we call them security cases. Security is an open problem because it combines human, technical, and organizational problems. The evolving nature of the threats indicates that there is no one-time solution because new environmental information may arise. Consequently, we need to carefully review our reasoning about the safety or security of the system. Security cases record our reasoning so that other stakeholders can learn from it and trust our claims about having a secure

system. The other stakeholders can also use the assurance cases to assess the impact of a possible change in the system.

2.2 Security assurance argument patterns

Argumentation patterns are general reusable solutions to commonly occurring problems in the design of argumentation frameworks, such as the relation between claim and data in the Toulmin scheme [Villata et al.(2011)]. They provide reusable templates of the types of claims, evidence, justification, and contextual information that must be covered in a compelling argument [Picardi et al.(2020)]. Security argument patterns help to construct a security assurance argument that enables to maintain the structure of the argument by generalizing specific details so they can be instantiated in different situations [Alexander et al.(2011)]. This helps to facilitate the reuse and incremental refinement of the assurance argument, especially considering that different systems may be subject to different standards and requirements in different application contexts.

2.3 Goal structuring notation

There are several notations and existing tools for developing and documenting assurance cases, and the most popular of these is *Goal Structuring Notation (GSN)* [Group et al.(2018)]. GSN is a graphical notation that can visualize arguments that assure critical properties: safety, security, and resilience of systems, services, or organizations. This paper adopts the GSN pattern notation (an extension of core GSN [Habli and Kelly(2010)]) to visualize and present the argument structure. A summary of the graphical elements of the GSN Pattern Notation is provided in Fig. 1.

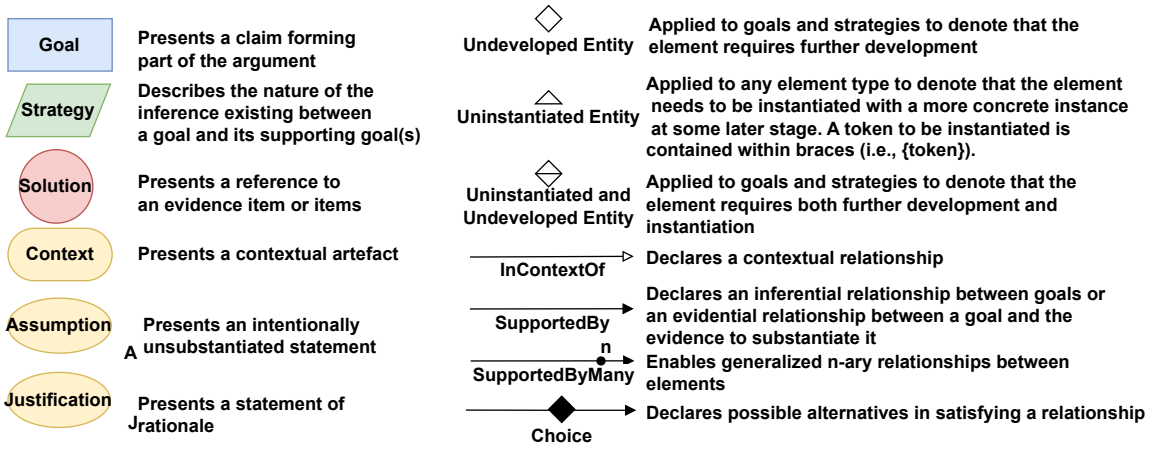

Fig. 1. Principal elements of the GSN Pattern Notation

2.4 Overview of DNN

DNN are a type of artificial neural network. DNN can be used to extract some mapping relations from training data. This relation will later enable matching, as best as possible, new data inputs to the correct output, namely for classification and regression tasks. The processing node in DNN is called a neuron. A neuron processes some received data and forwards it to neurons in the next layer. DNN comprise one input layer, one output layer, and one or more hidden layers. In this work, we will focus only on DNN built through supervised learning and used for the classification of tabular inputs. Tabular inputs comprise a finite discrete set of features (attributes) [Zeroual et al.(2022)]. *An input neuron* is a neuron of data ingestion; it corresponds to one input feature.

Output neuron is a neuron of data outlet; each neuron corresponds to one class label. *A bias neuron* is a neuron of data injection; each neuron adds a constant value to neurons in the next layer. The value is called bias and is used with the weights products sum and neuron inputs to produce an output. *A hidden neuron* is a neuron of data processing; each neuron accepts input data, performs calculations, and produces output. *An edge* is a weighted link to connect neurons in successive layers. Weights increase or decrease the impact of the neuron's inputs on its output. *An activation* function transforms the neuron's output before passing it to the next layer.

2.5 Adversarial examples against DNN

Adversarial examples Adversarial examples are well-crafted inputs fed to the DNN to deceive it, i.e., assign a given input to a class it does not belong [Zeroual et al.(2022)]. They are obtained by slightly modifying a sane input (i.e., initially well-classified). Formally, given DNN that assure the function f, the attackers find a minimal perturbation to add to a sane input x to obtain a new input x' so that $f(x) \neq f(x')$.

2.5.0.1 *A motivating example.* The example is based on an airborne collision avoidance system that uses DNN for decision-making in the presence of an adversary intruder [Wang et al.(2018)]. DNN are becoming increasingly popular in such systems due to better accuracy and less performance overhead than traditional rule-based systems [Julian et al.(2016)]. As shown in Fig. 2, usually, when the distance (one feature of the DNN) between the victim ship (ownship) and the intruder is large, the victim ship advisory system will advise the left to avoid the collision and then advise right to get back to the original track. However, if the incorporated DNN are not verified, there may be a specific situation where the advisory system, for certain approaching angles of the attacker ship, advises the ship incorrectly to take a right turn instead of a left, leading to a fatal collision. If an attacker knows about the presence of such an adversarial case, he can specifically approach the ship at the adversarial angle to cause a collision. In Fig. 2, the neural network in the victim aircraft (ownship) should predict a left turn (right figure) but unexpectedly advises to turn right and collide with the intruder (left figure) due to the presence of adversarial inputs (e.g., if the attacker approaches at certain angles).

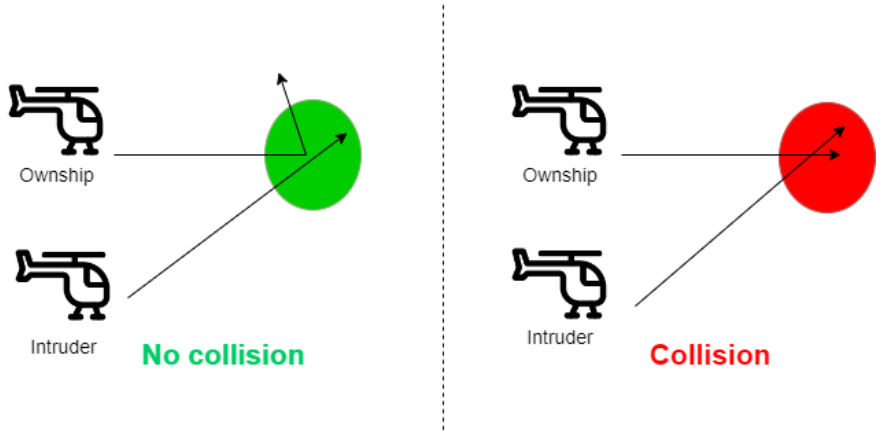

Fig. 2. Collision caused by an adversarial example. Revised from [Wang et al.(2018)]

2.6 DNN verification using interval analysis

As discussed above, adversarial examples attempt to modify the inputs in specific intervals to lead to output range violation. Thus, many formal verification techniques are used to verify safety and security properties of DNN. Among these techniques, we have interval analysis. Interval analysis studies the arithmetic operations on

intervals rather than concrete values. The DNN computations only include additions and multiplications (linear transformations) and an activation function (simple nonlinear operations). Consequently, by setting input features as intervals, we could follow the exact arithmetic performed in the DNN to compute the output intervals. Based on the output interval, we can verify if the input perturbations will finally lead to violations or not [Wang et al.(2018)].

3. DNN LIFECYCLE

Deep learning as a sub-field of machine learning is also data-driven, i.e., the behavior of a DNN-based system is learned from data. Consequently, DNN require new development activities such as data-oriented activities [Delseny et al.(2021)]. Assuring the use of DNN in critical systems requires a deep understanding of the DNN lifecycle and the artifacts generated during it. Thus, we studied existing machine learning lifecycle stages from several works [Ashmore et al.(2021), Schlegel and Sattler(2023), Picardi et al.(2020)]. We compared them to works focusing on deep learning lifecycle [Pina et al.(2023), Miao et al.(2017)]. In our work, we revise the work of [Ashmore et al.(2021)] and adopt it by defining the DNN lifecycle. Fig. 3 shows the critical lifecycle stages of DNN. Several activities are associated with each stage.

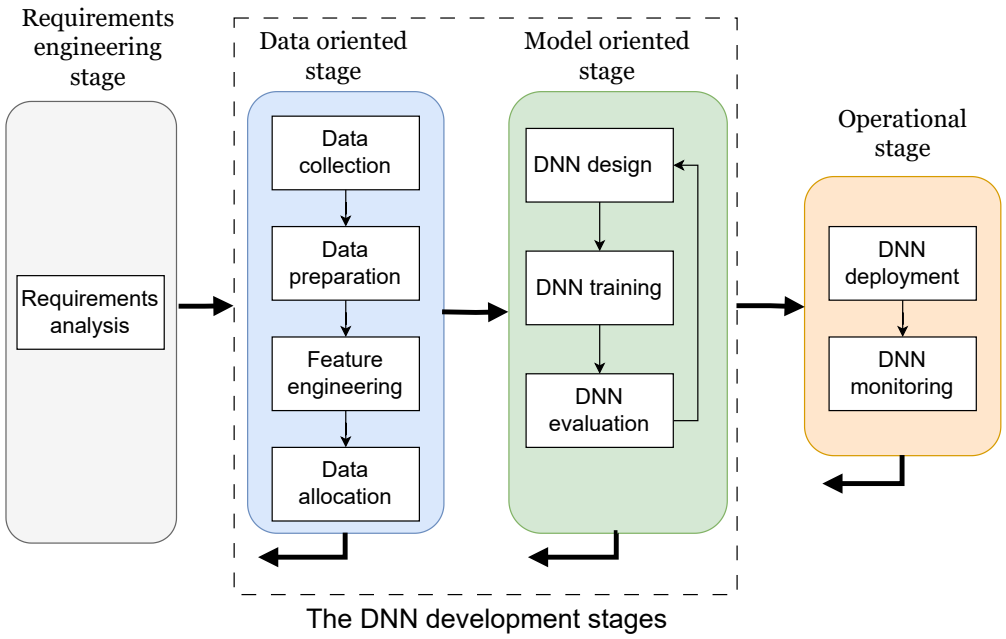

Fig. 3. Deep Neural Network Life cycle revised from [Ashmore et al.(2021)]

In the rest of this section, we describe each stage in the lifecycle, and then we discuss the development activities associated with it.

3.1 Requirements engineering stage

This stage aims to define the DNN requirements. The requirements elicitation technique must allow the identification of all the requirements. A part of the DNN requirements is generally allocated from the system requirements (which functionality and interfaces to realize). The rest of the requirements should constrain the best data and model choice to realize the allocated requirements.

3.2 Data-oriented stage

This stage focuses on obtaining the data-sets necessary to develop the DNN. It includes the following activities:

Data collection. This activity concerns collecting data from different sources (web scraping, manual data entry, data acquisition from sensors or devices), or re-using pre-existing data.

Data preparation. The preparation of the collected data involves cleaning the data of any errors and inconsistencies. After data cleaning, the data engineer proceeds to the labeling: assigning relevant tags to data instances.

Feature engineering. This activity involves transforming the prepared data into a suitable representation or set of features that DNN can effectively utilize. It involves extracting, selecting, and creating informative features from the available data.

Data allocation. This activity splits the data into training, validation, and test data. The training data is used to optimize the model's parameters, and the validation set helps fine-tune the model and select hyperparameters. Moreover, the test data can be used for the DNN evaluation activity in the following stage.

3.3 Model oriented process

When the data becomes available, a deep learning developer uses the labeled dataset to learn and optimize the DNN parameters, enabling the DNN to make accurate predictions or perform a specific task.

DNN design. The success of DNN depends heavily on design choices, such as finding the most efficient architecture for a given task [Miikkulainen et al.(2019)]. Much of the recent work in deep learning has indeed focused on proposing different architectures for different learning tasks(e.g., Recurrent neural network (RNN), Convolutional neural network (CNN), Feedforward neural network (FNN)) [Hong et al.(2018)]. The DNN architecture involves the distribution of neurons through different layers and different connections linking them. It also involves the activation function that will be used.

DNN training. The training activity aims to optimize the performance of the DNN model concerning an objective function that reflects the requirements for the model. The training data is used to find internal model parameters (e.g., the weights of a neural network) that minimize an error metric for the given dataset. The validation data are then used to assess the DNN model's generalization ability. These two steps are typically iterated many times, with the training hyperparameters (parameters initialization and activation function) tuned between iterations to further improve the DNN model's performance.

DNN evaluation. This activity is crucial to ensure that the trained DNN model meets the desired criteria and performs effectively on new, unseen data. It helps identify and address any issues, bugs, or inconsistencies early in the development cycle, minimizing potential risks and ensuring the DNN operates as intended before moving to the deployment phase.

3.4 DNN operational process

At this stage, the DNN is deployed in the target system and monitored for metrics and errors during execution. This stage will be out of the assurance scope of this work.

4. SECURITY ARGUMENT PATTERNS FOR DNN DEVELOPMENT

In this section, we introduce the security argument patterns that we have proposed. We comprehensively explain the process of deriving these patterns and delve into the reasoning behind their construction. Furthermore, we elaborate on the practical application of these patterns in developing DNN to establish robust security assurance cases.

4.1 DNN Secure development argument pattern

In this section, we present a security assurance argument pattern called *DNN Secure development argument pattern*. This pattern provides claim decompositions to demonstrate that the DNN adequately satisfies its required security requirements during its development stages.

4.1.1 *Pattern derivation.* Using DNN in security-critical systems requires ensuring that DNN development meets the security requirements. Due to the complexity of DNN applications, it is essential to integrate security in the early design phase of the DNN lifecycle [Zhang and Jaskolka(2022)]. Existing development assurance standards used for certification (e.g., DO-178C/ED-12C, ISO26262, EN50128) are not data-driven and do not consider the DNN-specific development activities (the data and model-oriented stages) [Delseny et al.(2021)]. As with any engineering artifact, assurance cases can only be provided by understanding the complex, iterative process employed to produce and use DNN components, i.e., each stage from the deep learning lifecycle [Ashmore et al.(2021)].

DNN developers must provide sufficient evidence that the DNN under assurance have been developed adequately, considering security concerns at all stages of the development lifecycle. There is a need to decompose the processes to ensure that each has been adequately performed to support the claim about the security of the DNN. Poor decomposition can lead to missed processes or procedures that can leave the DNN development inadequate in terms of its security considerations and compliance requirements [Jaskolka et al.(2021)].

The argument pattern shows the earlier activities of the DNN development where the security requirements must be satisfied. The argument pattern is in accordance with the DNN lifecycle presented in the previous section 3. This pattern is for a single DNN model, the requirements of which have been identified by considering the broader system context, including defined security requirements derived from the system security process.

4.1.2 *Pattern description.*

4.1.2.1 *Pattern name.* DNN Secure development argument pattern

4.1.2.2 *Context.* You are developing a DNN model that realizes a task and will be deployed in a security-critical system. In addition, to prove that your DNN model performs well, you need to guarantee that as a developer, your DNN satisfies its security requirements.

4.1.2.3 *Intent.* The pattern provides a claim decomposition to argue that the development of a DNN satisfies its security requirements if the security requirements are satisfied in the different activities of the DNN development process. The goal is to provide a convincing argument that the DNN under development complies with the necessary processes and activities to support claims that the DNN is adequately secure. The aim is to help designers and developers construct such an argument while reducing the effort required to do so.

4.1.2.4 *Motivation.* Multiple activities are undertaken while developing DNN. Each activity generates some artifacts whose security should be ensured. Consequently, we will have many security assurance fragments. The DNN secure development argument pattern is a preliminary brick that indicates how the DNN-related patterns should be assembled.

4.1.2.5 *Applicability.* The pattern should be applied when compliance is required in the secure development processes for the DNN under assurance. The pattern assumes that the relevant development methodology provides sufficient context for its instantiation.

4.1.2.6 *Structure.* The argumentation strategy is captured by the argument pattern shown in Fig. 4. It's represented using GSN pattern notation.

4.1.2.7 *Participants.* The pattern participants are as follows:

Goal: G0. The overall objective of the argument; to support the claim that the DNN satisfies its security requirements in the operating environment.

Context: C0. The description of the the DNN for which the security assurance case is being developed.

Context: C1. The description of the system in which the DNN will be deployed.

Assumption: A0. The DNN must satisfy all the elicited security requirements. This elicitation must be made as part of the system requirements engineering process. If an argument to support this assumption exists, then that argument can be linked here in place of the assumption.

Strategy: S0. The strategy requires splitting the goal into two sub-goals regarding the DNN development stages.

Goal: G1. This goal states that the data oriented stage must satisfy security requirements. This claim remains undeveloped in this work.

Goal: G2. This goal states that the model oriented stage satisfies the security requirements.

Context: C2. This context node refers to the list of security requirements.

Strategy: S1. The argumentation strategy involves going through the different model oriented activities and ensuring that that the security requirements are met in each activity.

Context: C3. This context node refers to the description of the DNN model oriented activities.

Goal: G3. The first sub-goal states that the selected DNN architecture is appropriate for the given set of DNN security requirements.

Goal: G4. The second sub-goal states that the DNN training is appropriate for the given set of DNN security requirements.

Goal: G5. The last sub-goal states that the trained DNN satisfies the security requirements.

Strategy: S2. The decomposition strategy argues through verifying each security requirement.

Goal: G6.X. This goal states that the DNN satisfies one security requirement. {X} is used to enumerate the security requirements.

4.1.2.8 *Consequences.* The pattern enables a separation of concerns in the argumentation about the fulfillment of DNN security requirements. Decomposition of the security requirements allows us to focus on security requirements related to the model-oriented stages, and more specifically to each activity in this stage, one at a time. However, after instantiating the pattern, some undeveloped goals will remain. Specifically, each goal of the model-oriented activities will require further decomposition and support. One of the key challenges in constructing assurance arguments is determining the granularity by which each goal is decomposed so that sufficient evidence can be generated to fully support the goal [Jaskolka(2018)]. Because there are many different ways to decompose goals, it is not immediately evident when to stop the decomposition or what sufficient granularity would be to ensure that the required evidence could be provided. In some cases, these goals can be decomposed by adopting other argument patterns recast in the context of security.

4.2 Security requirement satisfaction argument pattern

4.2.1 *Pattern derivation.* When the DNN model completes training with its decision logic determined, it systematically evaluates its generality and quality through testing or verification. From a security assurance point of view, we should ensure that the evaluation activity generates evidence elements that will demonstrate the fulfillment of security requirements. The security argument should demonstrate the relationship between the verification evidence and the DNN security requirements. We distinguish two processes to generate the evidence elements:

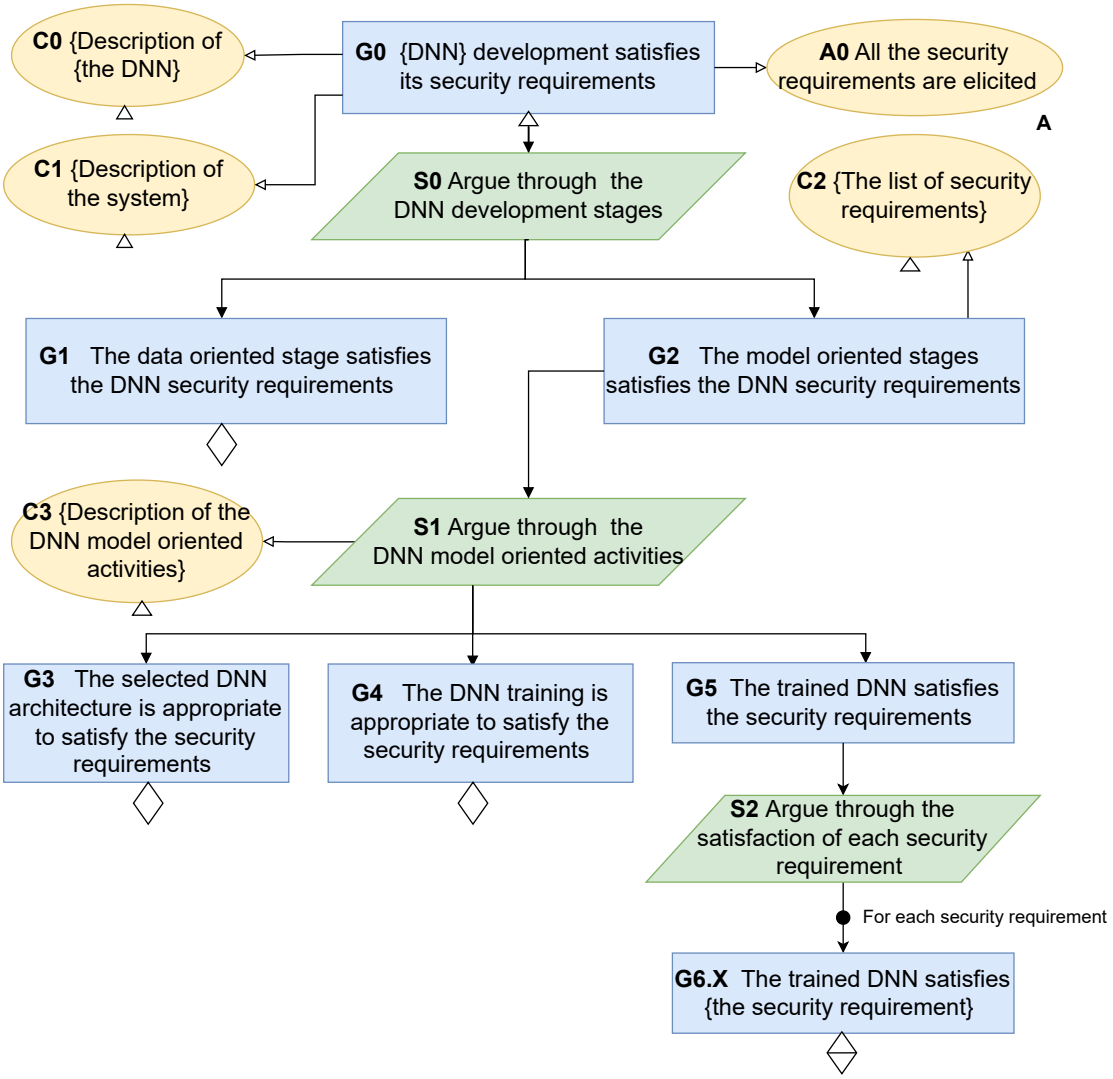

Fig. 4. DNN Secure development argument pattern

4.2.1.1 *Test-based evaluation.* it utilises the test data to demonstrate that the model generalizes to cases not present in the model learning stage. This shall involve an independent evaluation of the requirements considered during the model learning stage. Specifically, those security requirements associated with ensuring the robustness of models are evaluated on the independent verification data set i.e. that the performance is maintained in the presence of adverse conditions or signal perturbations. The test team should examine those cases which lie on boundaries or which are known to be problematic within the context to which the DNN model is to be deployed [Hawkins et al.(2021)]. Beyond using test data, several works developed tools to test the DNN. Work in [Pei et al.(2017)] generates test inputs for a DNN. It identified incorrect corner case behaviors in autonomous systems(e.g., self-driving cars crashing into guard rails and malware masquerading as benign software). Evidence is then the inputs used for testing and the results of testing.

4.2.1.2 *Formal verification.* formal verification methods like deductive methods, abstract interpretation, or model checking aim at verifying that a model of a system satisfies some properties on a mathematical basis [Mamalet et al.(2021)]. In the current context, the objective is to demonstrate the compliance of a DNN to its security requirements in all possible situations without explicitly testing the behavior of the DNN in each of them. Authors from [Urban and Miné(2021)] studied the state of the art of the methods proposed for verifying properties of neural networks such as Satisfiability Modulo Theory (SMT), Mixed Integer Linear Programming (MILP), and Symbolic intervals. To verify the local robustness of DNN, work from [Huang et al.(2017)] verified that for a given DNN input, there is a region around the input where the DNN output remains unmodified. In another work from [Gopinath et al.(2018)], an additional binary classifier is trained to distinguish between the adversarial and original samples that can be regarded as the detector model. This method can be used to build robust DNN to adversarial perturbations. The formally-specified properties shall be a sufficient representation of the DNN security requirements in the context of the defined operating environment. An explicit justification shall be documented for the sufficiency of the translation to formal properties [Hawkins et al.(2021)]. Evidence is then the result of the formal verification of the security requirements.

4.2.2 *Pattern description.*

4.2.2.1 *Pattern name.* Security requirement satisfaction argument pattern

4.2.2.2 *Context.* You have developed a DNN that realizes a task and you will deploy it in a security-critical system. In addition, to prove that your DNN performs well, you need to evaluate it regarding the security requirements. You can either use test methods or formal verification to prove the fulfillment of the security requirement.

4.2.2.3 *Intent.* The goal of this pattern is to provide a convincing argument about the verification of the security requirements before deploying the DNN. The pattern indicates what are the evidence elements necessary to support the claims about the satisfaction of security requirements. The pattern shows also that we can use tests and/or formal verification to generate the evidence elements.

4.2.2.4 *Motivation.* The motivation behind verifying the fulfillment of security requirements specifically for DNN model stems from the potential risks associated with these complex machine learning models. DNN are increasingly utilized in critical applications such as autonomous vehicles, healthcare systems, financial services, and cybersecurity, where the consequences of security breaches can be severe. Consequently, the assurance of the security requirements fulfillment is of utmost importance.

4.2.2.5 *Applicability.* The pattern should be applied when compliance is required to verify the fulfillment of security requirements. The pattern assumes that the relevant standard or development methodology provides sufficient context for its instantiation.

4.2.2.6 *Structure.* The argumentation strategy is captured in the argument pattern shown in Fig. 5. It's represented using GSN pattern notation.

4.2.2.7 *Participants.* The pattern participants are as follows:

Goal: G6.X. This goal states that the DNN satisfies one security requirement.

Context: C4.X. This context node refers to the security requirement to satisfy.

Strategy: S3. The argumentation strategy provides a choice over how the claim can be supported. The choice in the argument should be interpreted as "at-least-1".

Justification: J0. The justification node states that the choice of the verification approach to use should be justified.

Goal: G7.X. This goal states that the test data must demonstrate that the DNN security requirement is satisfied.

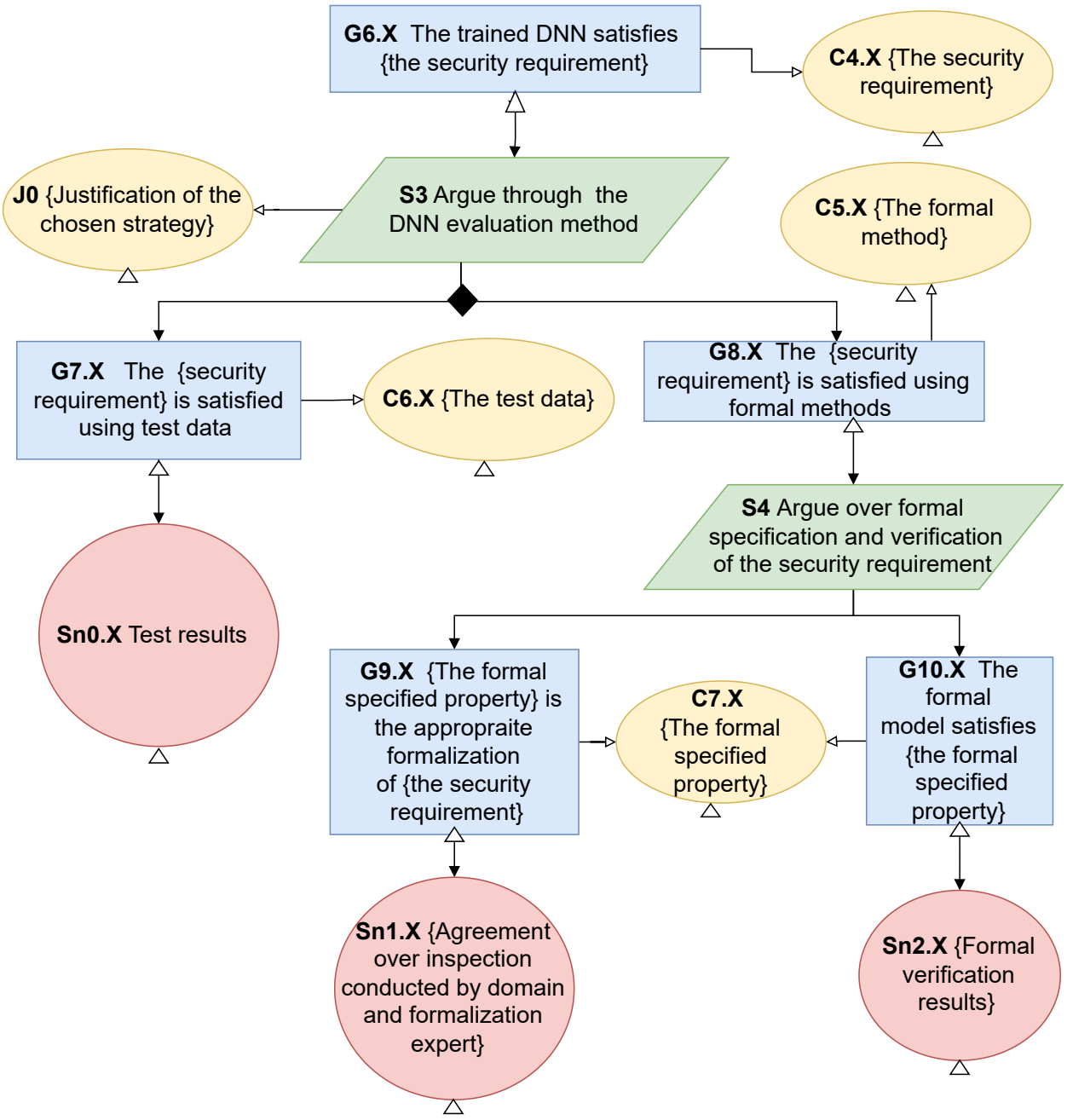

Fig. 5. Security requirement satisfaction argument pattern

Context: C6.X. This context node refers to the test data: a sufficient range of inputs representing the operating environment, that are not included in the data used in the DNN learning stage.

Solution: Sn0.X. This node refers to the evidence from the test results.

Goal: G8.X. This goal states that formal verification must demonstrate that the DNN security requirement is satisfied.

Context: C5.X. This context node refers to the formal verification method that is used.

Strategy: S4. The argumentation strategy involves arguing through the formal specification and verification of the security requirement.

Goal: G9.X. This goal states that formal specification of the security requirement is the appropriate one.

Context: C7.X. This context node refers to the formal specified property proposed as the formalization of the security requirement.

Solution: Sn1.X. The solution node refers to the expertise required to justify the correctness of the security requirement formal specification.

Goal: G10.X. This goal states that formal model satisfies the specified property.

Solution: Sn2.X. This node refers to the evidence from the formal verification results

4.2.2.8 *Consequences.* The pattern demonstrates that the DNN model will meet its security requirements when exposed to inputs not present during the development of the model. It also shows the relationship between the verification evidence and the security requirements. However, after instantiating the pattern, some goals will remain undeveloped. Specifically, goals related to DNN testing will require further decomposition and support. The further decomposition can be based on the specific requirements that must be satisfied in support of the instantiated security requirement.

5. CASE STUDY

In this section, we illustrate how to apply the proposed patterns to build security cases that argue for a secure development of DNN used in collision avoidance drones. First, we present the use case. Afterward, we instantiate the argument patterns.

5.1 Use case description

We illustrate our contributions through a use case scenario from ACAS Xu [Manfredi and Jestin(2016)] used for collision avoidance in Unmanned aircraft systems (drones).
We consider encounters between just two drones [Katz et al.(2017)]. We take the perspective of one of them and call it the ownship, and the second drone is called the intruder (Fig. 6).

The ownship is equipped with ACAS Xu using the DNN showed in Fig. 7. ACAS Xu uses as inputs various

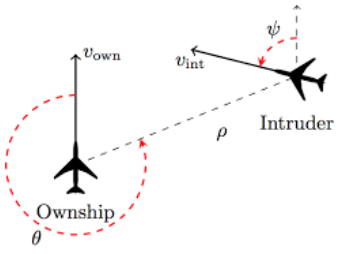

Fig. 6. ACAS Xu system

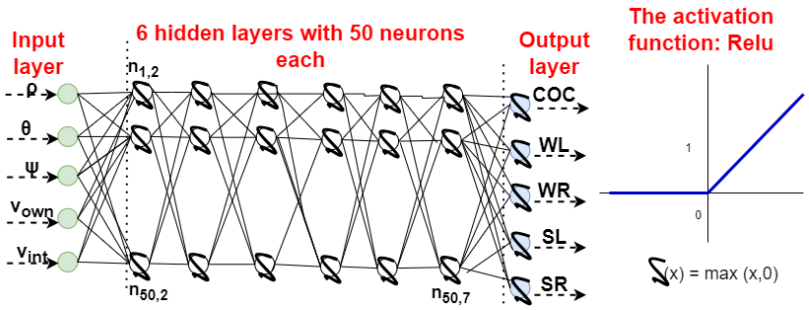

Fig. 7. ACAS Xu deep neural network

parameters related to the state of the ownship and the intruder, such as the distance between them (ρ), their

velocities, respectively (v_{own}, v_{intr}), the angle to the intruder (θ), and the heading of the intruder (ψ). This information is used to predict a set of recommended actions to avoid a collision. The first option is to do nothing, also known as the "Clean of Conflict"(COC) option. However, we also have the option to make a weak adjustment by turning left (WL) or right (WR). Additionally, we can make a strong adjustment by turning left (SL) or right (SR). Each output in the DNN corresponds to the score for this action (minimal for the best). The DNN uses rectified linear unit (ReLU) activations, which are the most popular in practical hardware/software implementations [Paulsen et al.(2020)].

5.2 DNN Secure development argument pattern application

We illustrate the application of the DNN secure development argument pattern on the case study. The field {DNN} on the uninstantiated goal G0 will be replaced by the name of the target DNN: DNN from ACAS Xu. The pattern application focuses mainly on providing different context elements. C0 describes the DNN: the functionality, the DNN inputs, the DNN outputs and the activation function. C1 describes the system in which the DNN will be deployed. It refers to the ACAS Xu system. The node C2 includes all the security requirements of the ACAS Xu. In this paper, we focus on the main requirement: for any malicious input provided by an attacker within a given input range (e.g., for attacker aircraft's speeds between 0 and 500 mph), the DNN returns the decision that avoid collisions. This main requirement is further refined into the requirements presented in table I from the works [Wang et al.(2018), Katz et al.(2017)]. Although this is not an extensive list of requirements, it represents some of the most important requirements for ACAS Xu. We assume that the security requirements are all elicited. The node C3 describes the DNN design activities. For the security requirements, we create an instance of the goal G6.X for each security requirement from the list in Table I as shown in Fig. 8.

Table I. Security requirements of the DNN

SR	Description
SR1	If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold
SR2	If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
SR3	If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal
SR4	If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for COC will not be minimal.

5.3 Security requirement satisfaction argument pattern application

We apply this argument pattern for each security requirement. We showcase the pattern application using only SR1 for brevity, as represented in Fig. 9. We refer to the security requirement SR1 in the nodes C4.1,G6.1,G7.1,J0, G9.1, G10.1. The evidence elements provided by the different secure development activities can guide the argumentation strategy. We choose to use formal methods as an evaluation method. DNN testing tools try to find adversarial examples but do not provide guarantees about their absence [Wang et al.(2018)] as referred to in node J0. Context node C5.1 describes the formal method used to verify SR1. *ReluVal* is a DNN verification tool that uses interval analysis to verify security properties of DNN: It detects the presence of adversarial examples that violate security property by bypassing the intervals [Wang et al.(2018)]. The reasons that justify this choice are presented in section 2.6. The first step in applying the strategy is the formal specification of the security requirement. The requirements are specified using inputs and outputs of the DNN by constraining their ranges of values. We describe SR1 as an example. In SR1, when the intruder is distant and is significantly slower than the ownship, the confidence score of a COC advisory will always be below a certain fixed threshold. SR1 requires bounding the confidence score of the COC (clean of conflict) and never trusting a situation too much. To specify the property, we first determined value ranges based on the training data's minimum and maximum values in the corresponding variables (distance, velocity). We also chose the thresholds based on our knowledge of the

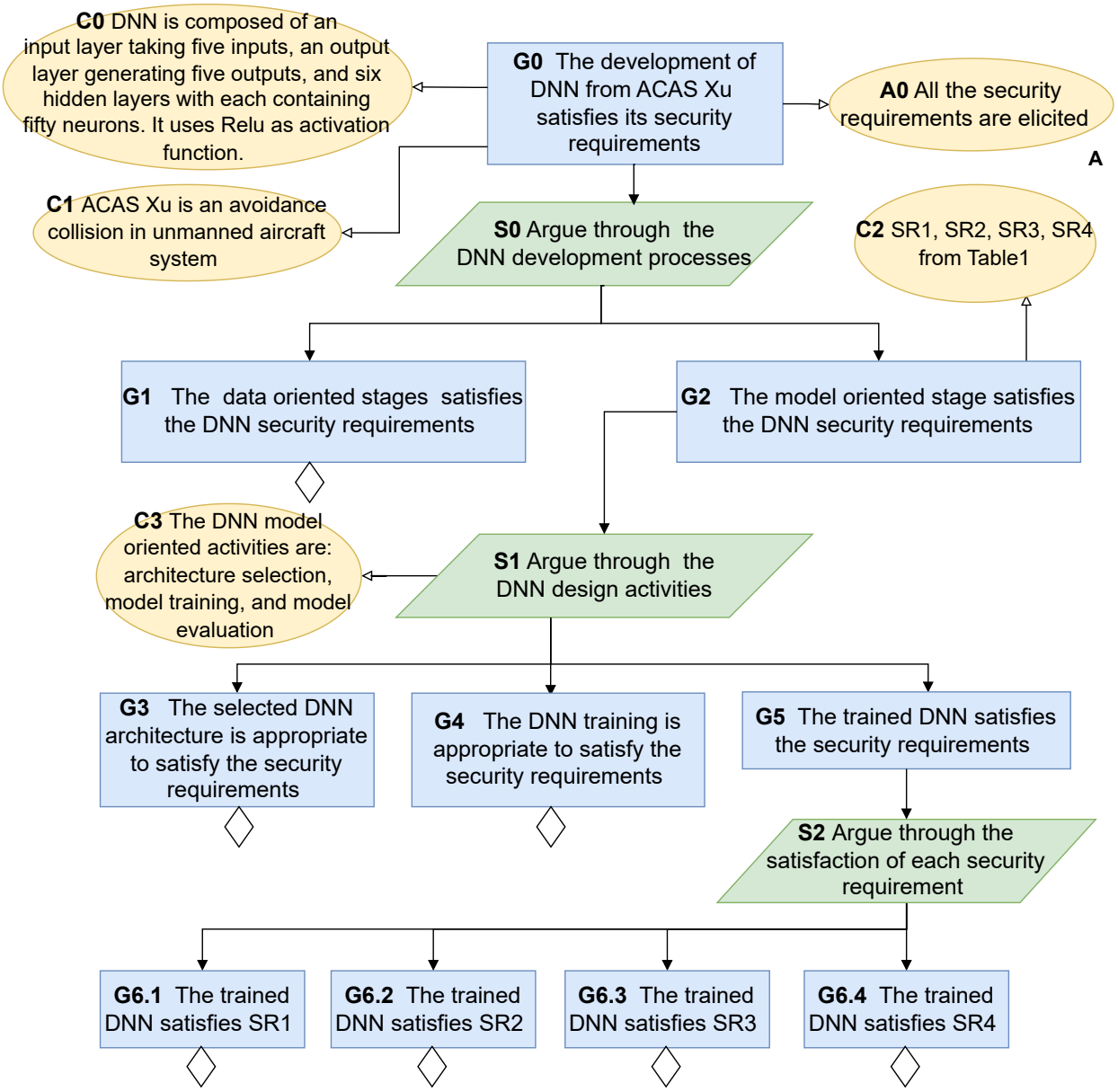

Fig. 8. Application of DNN secure development argument pattern

literature. We provide a report of this expertise as an evidence in the node Sn1.1 to support the claim G9.1. As a result, the formal specified property that corresponds to SR1 is the following [Katz et al.(2017)]:

$$Prop(1) = \rho \geq 55947.691 \wedge v_{own} \geq 1145 \wedge v_{int} \leq 60 \Rightarrow score(COC) \leq 1500 \quad (1)$$

We refer to the property Prop(1) as {formal specified property} while instantiating nodes G9.1, G10.1, C5.1. We use ReluVal to check the property Prop(1). This tool allows verification of the property, so the satisfaction of

the security requirement. The formal verification results will be used as evidence elements in the node Sn2.1 to support the claim G10.1.

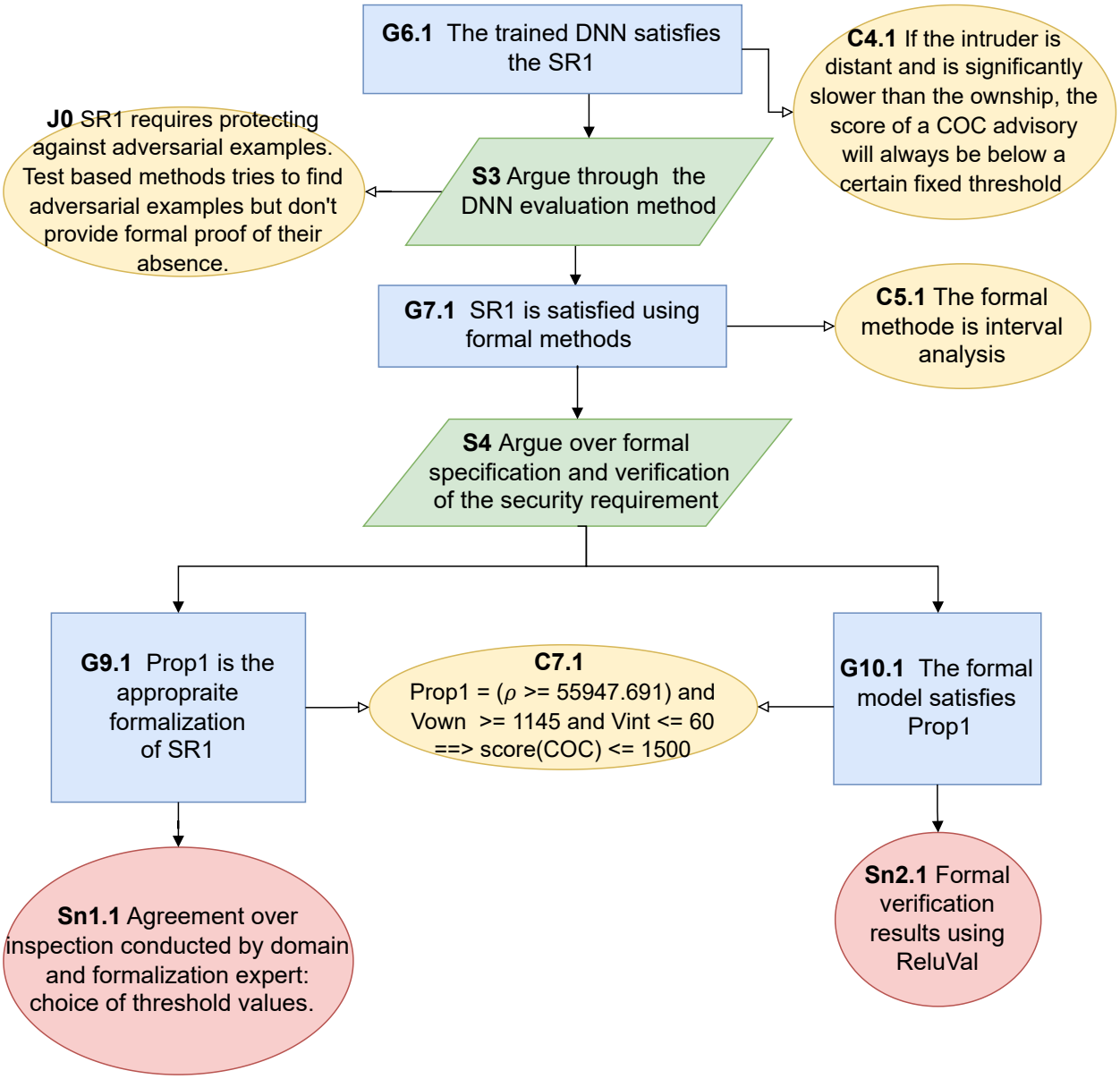

Fig. 9. Application of security requirement satisfaction argument pattern

6. RELATED WORK

To the best of our knowledge, our work is the first to propose security argument patterns for the DNN development. Prior research has explored various assurance aspects like reliability, interpretability, and safety. Arguing about the safety of DNN, or more generally machine learning models, garners the most significant attention, with numerous studies emphasizing its paramount importance [Schwalbe and Schels(2020)]. Starting with the work in [Picardi et al.(2020)] where authors proposed a safety argument pattern for a machine learning model and an assurance process for the instantiation of the argument pattern. This process is built upon existing best practices. It highlights the required activities that should be undertaken and the artifacts that should be generated to support the assurance claims, namely, the operating environment description, the defined machine learning safety requirements, the developed machine learning Model, and the data used for the development and testing of the machine learning model. The initial application of this safety argument pattern focused on clinical diagnosis [Picardi et al.(2019)] and autonomous driving [Burton et al.(2019)].
The safety assurance argument pattern is extended by identifying the interpretability property as a key consideration in [Ward and Habli(2020)]. A machine learning model is interpretable if we understand how it genuinely works and why it makes the decisions that it does. The argumentation strategy involves that some interpretability methods have been used. However, the argument pattern does not show how evidence elements that support the sufficiency of the argument based on these methods must be gathered.
Moreover, the authors in [Zhao et al.(2020)] extended the safety argument pattern with a reliability pattern. The argument pattern is tailored for deep learning models such as DNN. The argumentation strategy involves reducing errors in each lifecycle activity. The evidence elements supporting this pattern are generated using formal verification and coverage-guided testing.
Coming back to our security argument patterns, the preliminary argument pattern outlines the essential elements required to effectively argue about the security development of DNNs. Furthermore, the second argument pattern incorporates various evidence elements that can be employed to substantiate the claims made in these arguments. Moreover, we provide a detailed case study that showcases their utilization in a real-world scenario. The patterns application gives us confidence that what we have is a pattern, not a one-shot solution to a problem.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented two argument patterns for the security development of DNN components. The argument patterns describe the arguing strategies and the types of evidence that are necessary to generate a compelling and credible assurance case. We showed the application of the patterns in the ACAS Xu case study. Several areas for future work exist, three of which are as follows. Firstly, the requirements engineering stage in which the DNN security requirements are determined remains an open challenge. In our pattern arguments, we assumed that the security requirements were all identified in a previous stage (see node A0 in the first pattern). However, we can investigate deeply this stage and define a pattern argument that ensures the identification of all the DNN security requirements. Identifying the DNN security requirements dictates the classical and deep learning-specific security requirements. Secondly, our paper focuses on developing and verifying the DNN model, which is a data-driven model. Therefore, many security threats can appear in data-oriented stages, which also require assurance for the DNN security in this stage. We aim to investigate the data-oriented stage in our future works and provide an argument pattern for the data verification. Thirdly, the patterns we built are domain-independent. We can apply them in critical domains(healthcare, automotive, and manufacturing). Thus, we aim to propose domain-oriented argument templates by choosing the argumentation strategies that sound with each domain.

Acknowledgments

Many thanks to the shepherd of the paper, Prof. Dr.WASHIZAKI Hironori, for his feedback during the shepherding process.

REFERENCES

Rob Alexander, Richard Hawkins, and Tim Kelly. 2011. *Security Assurance Cases: Motivation and the State of the Art*. Technical Report CESG/TR/2011/1. University of York, York, UK.

Rob Ashmore, Radu Calinescu, and Colin Paterson. 2021. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–39.

Simon Burton, Lydia Gauerhof, Bibhuti Bhusan Sethy, Ibrahim Habli, and Richard Hawkins. 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, 365–377.

Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE symposium on security and privacy*. IEEE, 39–57.

Hervé Delseny, Christophe Gabreau, Adrien Gauffriau, Bernard Beaudouin, Ludovic Ponsolle, Lucian Alecu, Hugues Bonnin, Brice Beltran, Didier Duchel, Jean-Brice Ginestet, et al. 2021. White paper machine learning in certified systems. *arXiv preprint arXiv:2103.10529* (2021).

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Divya Gopinath, Guy Katz, Corina S Păsăreanu, and Clark Barrett. 2018. Deepsafe: A data-driven approach for assessing robustness of neural networks. In *Automated Technology for Verification and Analysis: 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings 16*. Springer, 3–19.

Assurance Case Working Group et al. 2018. Goal structuring notation community standard (version 2).

GSN Working Group. 2021. GSN Community Standard Version 3. Available: https://scsc.uk/r141C:1?t=1.

Ibrahim Habli and Tim Kelly. 2010. A safety case approach to assuring configurable architectures of safety-critical product lines. In *International Symposium on Architecting Critical Systems*. Springer, 142–160.

Muhammad Abdullah Hanif, Faiq Khalid, Rachmad Vidya Wicaksana Putra, Semeen Rehman, and Muhammad Shafique. 2018. Robust machine learning systems: Reliability and security for deep neural networks. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design*. IEEE, 257–260.

Mark Harris. 2019. NTSB Investigation Into Deadly Uber Self-Driving Car Crash Reveals Lax Attitude Toward Safety. *IEEE Spectrum* (November 2019).

Richard Hawkins, Colin Paterson, Chiara Picardi, Yan Jia, Radu Calinescu, and Ibrahim Habli. 2021. Guidance on the assurance of machine learning in autonomous systems (AMLAS). *arXiv preprint arXiv:2102.01564* (2021).

Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. 2018. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. *arXiv preprint arXiv:1810.03487* (2018).

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 3–29.

Jason Jaskolka. 2018. Challenges in Assuring Security and Resilience of Advanced Metering Infrastructure. In *Proceedings of the 18th Annual IEEE Canada Electrical Power and Energy Conference (EPEC 2018)*. IEEE, Toronto, ON, Canada, 1–6.

Jason Jaskolka, Brahim Hamid, Alvi Jawad, and Joe Samuel. 2021. Secure Development Decomposition – An Argument Pattern for Structured Assurance Case Models. In *Proceedings of the 28th Conference on Pattern Languages of Programs (PLoP 2021)*. Online, 1–12.

Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10.

Min-Joo Kang and Je-Won Kang. 2016. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one* 11, 6 (2016), e0155781.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 97–117.

Tim P Kelly. 1999. Arguing safety-a systematic approach to safety case management. *DPhil Thesis York University, Department of Computer Science Report YCST* (1999).

Yi Lin, YuanKai Wu, Dongyue Guo, Pan Zhang, Changyu Yin, Bo Yang, and Jianwei Zhang. 2021. A deep learning framework of autonomous pilot agent for air traffic controller training. *IEEE Transactions on Human-Machine Systems* 51, 5 (2021), 442–450.

Franck Mamalet, Eric Jenn, Gregory Flandin, Hervé Delseny, Christophe Gabreau, Adrien Gauffriau, Bernard Beaudouin, Ludovic Ponsolle, Lucian Alecu, Hugues Bonnin, et al. 2021. *White paper machine learning in certified systems*. Ph. D. Dissertation. IRT Saint Exupéry; ANITI.

Guido Manfredi and Yannick Jestin. 2016. An introduction to ACAS Xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–9.

Hui Miao, Ang Li, Larry S Davis, and Amol Deshpande. 2017. Modelhub: Deep learning lifecycle management. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1393–1394.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 293–312.

Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19, 6 (2018), 1236–1246.

Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020. Reludiff: Differential verification of deep neural networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 714–726.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

Chiara Picardi, Richard Hawkins, Colin Paterson, and Ibrahim Habli. 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, 165–179.

Chiara Picardi, Colin Paterson, Richard David Hawkins, Radu Calinescu, and Ibrahim Habli. 2020. Assurance argument patterns and processes for machine learning in safety-related systems. In *Proceedings of the Workshop on Artificial Intelligence Safety (SafeAI 2020)*. CEUR Workshop Proceedings, 23–30.

Débora Pina, Adriane Chapman, Daniel De Oliveira, and Marta Mattoso. 2023. Deep Learning Provenance Data Integration: a Practical Approach. In *Companion Proceedings of the ACM Web Conference 2023*. 1542–1550.

Marius Schlegel and Kai-Uwe Sattler. 2023. Management of machine learning lifecycle artifacts: A survey. *ACM SIGMOD Record* 51, 4 (2023), 18–35.

Gesina Schwalbe and Martin Schels. 2020. A survey on methods for the safety assurance of machine learning based systems. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*.

Caterina Urban and Antoine Miné. 2021. A review of formal methods applied to machine learning. *arXiv preprint arXiv:2104.02466* (2021).

Serena Villata, Guido Boella, and Leendert van der Torre. 2011. Argumentation patterns. In *Proceedings of the 8th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2011)*. 133–150.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*. 1599–1614.

Francis Rhys Ward and Ibrahim Habli. 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, 395–407.

Charles B Weinstock, Howard F Lipson, and John Goodenough. 2007. *Arguing security-creating security assurance cases*. Technical Report. Carnegie Mellon University.

Marwa Zeroual, Brahim Hamid, Morayo Adedjouma, and Jason Jaskolka. 2022. Towards logical specification of adversarial examples in machine learning.

Xinrui Zhang and Jason Jaskolka. 2022. Security patterns for machine learning: The data-oriented stages. In *Proceedings of the 27th European Conference on Pattern Languages of Programs*. 1–12.

Xingyu Zhao, Alec Banks, James Sharp, Valentin Robu, David Flynn, Michael Fisher, and Xiaowei Huang. 2020. A safety framework for critical systems utilising deep neural networks. In *Computer Safety, Reliability, and Security: 39th International Conference, SAFECOMP 2020, Lisbon, Portugal, September 16–18, 2020, Proceedings 39*. Springer, 244–259.

Received June 2023; revised October 2023; accepted February 2024