Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Assurance Cases for Learning Enabled Components

Vishnu Vardhan Dadi

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Nico Hochgeschwender
M.sc. Deebul Nair

January 2022

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Vishnu Vardhan Dadi

Abstract

Trustworthy autonomy is the research domain where one of its focuses is addressing the problems of safety-critical systems. Safety-critical systems are the systems whose failure results in injuries, death, and property damage. Perception, localization, and navigation are some of the high-level functions of an autonomous system. Learning Enabled Components (LECs) are software components that leverage knowledge acquisition to operate and are unit components of these system-level functions. LECs are non-deterministic models, with some inherited randomness which makes them highly non-reliable. Using such non-reliable components in safety-critical systems result in hazards. The real-world data contain a lot of noise, introducing more uncertainty in these components, so verifying LECs during and after the development phase is insufficient. Researches in this field are considering runtime verification/monitoring as an approach that can satisfy the safety and reliability requirements of these autonomous systems.

This research provides a literature study on the state-of-the-art runtime verification techniques and LEC safety and reliability requirements. The primary goal of this research work is to prove that using runtime verification techniques will increase the reliability of these LECs. As a proof of concept, we ran some experiments with Convolutional Neural Networks (CNNs) using three different runtime monitors for an image classification task. The results show that using runtime monitors increased the reliability of these CNN models.

Since validating these models that exhibit black-box nature is challenging, in this R&D, we took an approach to assure by argumentation by writing an assurance case. Goal Structuring Notation (GSN) is an argumentation metamodel used to write an assurance case to prove our increase in reliability claim. The findings and inferences of this research work are discussed in detail by hypothesizing our claims.

Acknowledgements

Foremost, I would like to convey my wholehearted gratitude to my supervisors Prof. Dr. Nico Hochgeschwender and M.Sc. Deebul Nair, for providing me an opportunity to work on this Research and Development project. I thank them for helping me with their continuous support, patience, enthusiasm, knowledge, and motivation. Their guidance helped me all the time during this project.

I thank my fellow batchmates and senior students at Hochschule Bonn-Rhein-Sieg: Santosh, Aditya, Prabhudev, Krishnateja, Manoj, and Sathwik for their continuous support in discussions. I am blessed to have the moral support my parents and friends provided, which helped me stay motivated.

Contents

List of Figures

List of Tables

1

Introduction

"Autonomy" refers to a system's ability to operate independently with little or no human assistance. In the past two decades, the increase in the availability of more computation power, supported by the portability and compactness of micro-controllers, helped in significant advancements of autonomous systems. These autonomous systems are deployed to work with humans and assist them in everyday tasks. In some cases, these systems are used to complete tasks that humans find dangerous and monotonous. Over the period, these systems have been widely accepted in many fields of science and technology for their impressive performance. In today's world, Artificial Intelligence (AI) techniques have many promising applications, and autonomous systems are one domain that emerged incredibly with the advancement of AI. Since these systems act as companions and work in the real world, safety and security become major concerns. This led to the development of a new research domain called trustworthy autonomy. This R&D work is related to assurance cases, and Learning Enabled Components (LECs) that are part of this trustworthy autonomy field. This chapter describes the motivation behind this R&D, followed by limitations and the problem statement.

1.1 Motivation

AI has many good applications in today's world, and autonomous systems are among them. Mobile robots are autonomous systems that use Learning Enabled Systems (LESs) for perception, navigation, and localization. These are system-level functions, and these functionalities are achieved using LECs. LECs are software components that are necessary to achieve autonomy in any autonomous system. Non-learning systems function on explicitly programmed rules, whereas, in LESs, these rules emerge with the training to the extent of information the dataset covers. Since in LESs the rules are emerging, there is uncertainty involved in decision making. For instance, the LEC of an LES is trained to detect traffic signs and is deployed in a real-world scenario; there are high chances that the LES is unreliable because it hasn't been trained on all the traffic signs during learning. This is just one cause for uncertainty, but several other reasons like design errors, bugs, hardware problems exist [1]. Another good example is an autonomous vehicle that should navigate from point A to B. The LECs are trained with the information of the environment, and the vehicle is expected to drive in the real world. The Figure 1.1 explains a conceptual state space for this autonomous driving problem. In a non-LES, since the rules are programmed, the vehicle knows the safe and unsafe regions. Whereas in LES, the vehicle is uncertain about some regions

in the state space, as these regions are unexplored or not learned during the training. This kind of uncertainty is known as knowledge uncertainty or epistemic uncertainty. Section 3.1.4 explains more details on this uncertainty concept. When integrated with safety-critical systems like autonomous vehicles, these unreliable software components lead to severe hazards. Uber crash 2018 [2] is a good example of this which raised several safety-related questions among the researchers, developers, and consumers.

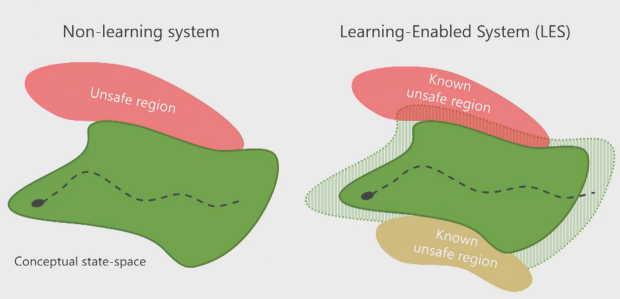


Figure 1.1: Conceptual state space explaining LES and non-LES. Image credits: [3]

LECs are non-deterministic models, and their predictions are stochastic. For example, let us consider a Convolutional Neural Network (CNN), an LEC, trained to classify the images of an MNIST dataset. The final layer of CNN models is the softmax layer. This means the CNN model will always predict a label, even for an out-of-sample image. The Figure 1.2 shows that for a small degree change in rotation of the digit, the model predicts accurately, but in the case where the degree of rotation is more, it wrongly classifies. We can notice that the softmax confidence is still very high even for a wrong classification. This shows how the model is confident in its predictions for an out-of-sample image which explains the non-reliable nature of these LECs.

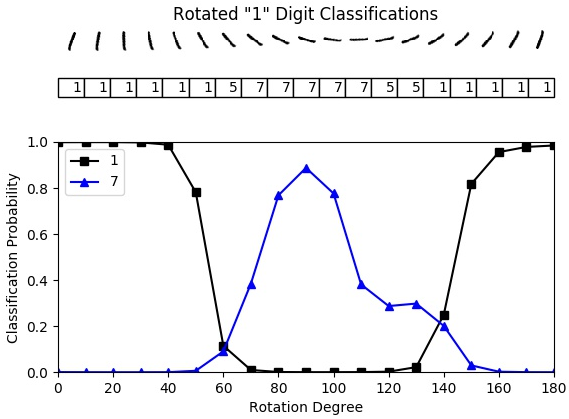


Figure 1.2: Standard CNN network behaviour on MNIST digit image. Image credits: [4]

Though there are many formal verification and validation techniques available to evaluate the per-

formance of LECs, they cannot justify their reliability in several cases, for example, dataset shift or adversarial attacks. There is some inherent randomness present in LECs, leading to aleatoric uncertainty. So only verifying LEC predictions during runtime can arguably increase the model's safety and reliability. The Figure 1.3 shows the proposed architecture for a trustworthy autonomous system. Runtime monitors are part of this architecture.

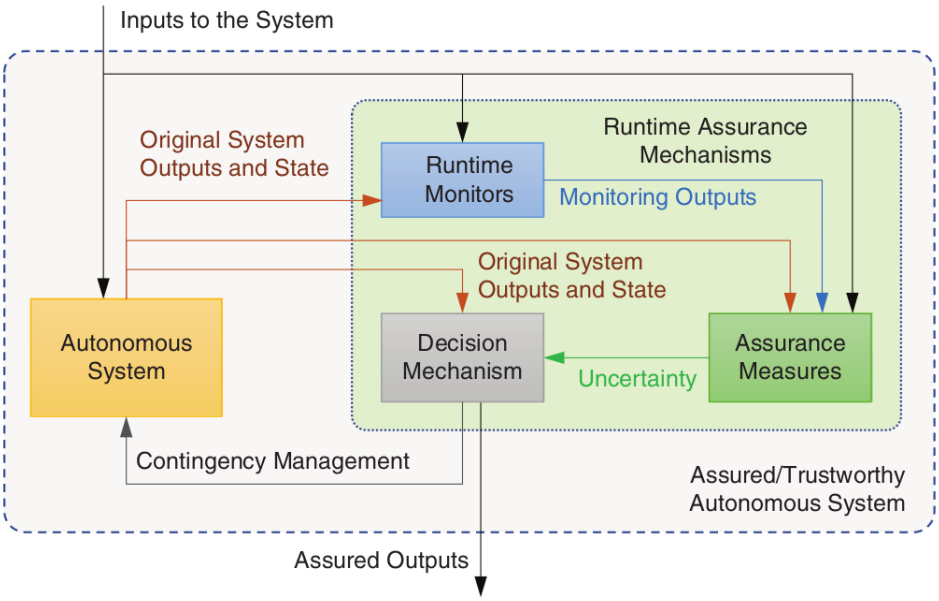


Figure 1.3: Trustworthy autonomous system architecture. Image credits: [5]

1.2 Challenges and Difficulties

The following are the challenges that were faced during this R&D work:

- To choose the suitable assurance case notation. Goal Structuring Notation (GSN) is selected because of its popularity and the maintainability of the standard. There is not much prior work available that compared different assurance cases notations and provided advantages. Apart from this, all the assurance notations have almost the same central idea.
- To build a proper argumentation for proving reliability. A lot of prior work is available on proving the safety of LECs, but there is none on the concept of reliability. The primary goal of this R&D is to prove that using a runtime monitor increases the reliability of LECs. So building a proper argument to satisfy this claim is very challenging.
- Models fail to converge because of higher initial learning rates. This means the model learns only a few classes but not all the classes in the dataset. The model learned the classes with similar features as a single class in the Robocup@work dataset. Hyperparameter tuning is necessary to find the ideal starting learning rate, which solves the problem.

- Runtime monitors developed in this R&D zooms the image multiple times with different zoom levels. This operation is performed on each image, delaying the prediction time, which is bad from the software engineering perspective.

1.3 Problem Statement

This R&D project will provide a literature study on State-Of-The-Art (SOTA) assurance case notations and runtime monitoring/verification techniques. Reasons for choosing the GSN are explained, and the usage of this notation is described with examples. Literature on software reliability testing methods and metrics are provided and explained why LECs are highly non-reliable. The reasons for uncertainty in LECs and their types were explained. Since formal verification and validation methods are insufficient for assuring LECs, using runtime monitors should arguably increase the reliability. The major focus of this R&D is to prove that by using runtime monitors, the reliability of LECs increases. For proof of concept, CNNs are used as LECs for solving an image classification problem. The following Research Questions (RQs) were addressed in this R&D.

RQ1 What are the SOTA methods available to certify/assure LECs?

RQ2 What factors affect the reliability and safety of LECs?

RQ3 What are the SOTA runtime monitoring/verification techniques?

RQ4 What are the necessary design steps to be taken to develop a reliable LEC?

RQ5 How do these design steps increase the model's performance and reliability?

RQ6 How is the performance of models after using Zoom monitor, max monitor, and average probability monitor on Resnet, Mobilenet, and Squeezenet on MNIST and Robocup@work datasets?

RQ7 Will using runtime monitors increase the reliability of LECs?

2

Background

This section explains the preliminaries required to understand the concept of assurance cases, their types, and the step-by-step development of a sample assurance case. A brief discussion is made on the dependability and software reliability theories, explaining the importance of software reliability testing and its methods.

2.1 Assurance Cases

In the context of science and technology, the term '**assurance**' can be understood as a justifiable measure of confidence that a system will perform its functions without fail in a defined environment of use. In general, the word '**case**' means a set of conditions or circumstances. An assurance case can be interpreted as a series of arguments that justify the claims made on a system's specific properties (in general dependability properties). According to [6], an assurance case is formally defined as

"Reasoned, auditable artifact created that supports the contention that its top-level claim (or set of claims) is satisfied, including systematic argumentation and its underlying evidence and explicit assumptions that support the claim(s)."

In general, an assurance case consists of the following components:

- **Claim** - the statement that is to be asserted as true or false.
- **Argument** - logical statements that link claims with evidence and assumptions.
- **Evidence** - artifacts provided in support of claims.
- **Assumption** - accepting statements that support arguments without proof.
- **Justification** - defending an argument with facts and reasons.
- **Conclusion** - final proposition of argumentation.

2.1.1 Usage of an Assurance Case

Assurance cases are primarily used to communicate with the stakeholders about the assurance arguments and evidence that support the claims made on the assurance properties of any system or an entity of

interest. In general, a '**stakeholder**' is a person or an organization who gets affected when a wrong decision is made. System developers, assessors, operators, and auditors can also be stakeholders in some cases. For example, an engineering team should convince the stakeholders that a newly built system is safe to integrate with an existing one using an assurance case. Here the safe operation of a system upon integration is a claim, and the stakeholder takes the decision which can be good or bad for the organization.

An assurance case can address more than one property and, in general, are risk-based properties. The risk-based properties are reliability, availability, maintainability, safety, and security. These are also known as dependability attributes. An '**entity**' is the subject of the assurance case and is interchangeable according to the scope of responsibilities. A service, product, system, activity, facility, process are some entities of an assurance case. Across the scientific community involved in maintaining the assurance cases, we usually find that the term '**system**' is used often in the place of 'entity.'

Using an assurance case typically, one should be able to communicate the following to a stakeholder:

- Should be able to explain the boundaries of the system.
- Should be able to justify the confidence in the performance of the system.
- Should also justify all the properties of interest subject to an entity and the associated confidence.
- Should provide comparative results between performance and confidence satisfying the acceptability criteria of the system.
- Should communicate known deficiencies in the performance of the system.
- Should explain all the risks that are associated with the system properties.
- In the context of a system's operation, all the assumptions and statements should be mentioned.
- Should communicate all the design decisions taken during the lifecycle and the trade-offs that may affect the system. This means the assurance case should be used throughout the system's lifecycle. For instance, some of the design decisions can be:
  - Changes in safety strategies.
  - Changes in action prioritization.
  - Changes in deployment decisions.
- An assurance case should also inform about the critical decisions taken during the lifecycle. This can include:
  - Acceptance of safety requirements.
  - Customer's acceptance.
  - Regulator's acceptance.
- Should communicate the limitations of the validation process while building the assurance case.

2.1.2 Good Practices for Developing Assurance Cases

No one assurance case format is universally accepted. The structure of an assurance case can change, and some sector-specific requirements or standards might be necessary for building a case in specific domains. "**Assurance Case Working Group (ACWG)**" provided some good practices to follow while building an assurance case. ACWG is established to provide guidance to construct, review and maintain the assurance cases. Following the practices discussed in this section, an assurance case can achieve a defensible outcome.

An assurance case should be updated timely, and one should use it throughout the system's lifecycle. The essential points to be noted while updating the assurance case during the lifecycle according to ACWG are:

- The decisions taken during a lifecycle are usually dynamic and updating the design decisions and why we do this will help build confidence.
- Updating during the lifecycle includes different stages starting from design concept to retirement.
- It is also advisable to divide the system requirements into elements and build separate assurance cases with different ownership elements.
- When the assurance cases were built for separate system elements, one should ensure that the child assurance cases are in cohesion with the parent assurance case.
- While building separate assurance cases, one may come across particular situations where the child cases might act like self-fulfilling when tried to find a relationship with the parent case under such circumstances, and enough critical review should be done to avoid biases. It's always better to find some counter-evidence while proving the relationship between the cases.

'**Artefacts**' as a collection acts as a basis for any assurance case. A document or a record that represents information is called an artefact. Depending on the quality management standards of the company [7], these artefacts are controlled, and some of the essential points to be noted about these artifacts are:

- An artefact can be maintained as a paper or an electronic document, or a model within a tool.
- An artefact can be in different forms, but one should ensure that this exists until the end of the lifecycle and should be maintainable to update and publish.
- Maintaining multiple versions of the artefacts over the system's life cycle is a great way to differentiate between upgrades and modifications quickly.
- The readability of an artefact should be good, and care should be taken while considering the font sizes and clarity of graphics.

- The language used in an artefact should not be ambiguous, and things like notations, abbreviations, terms should be defined appropriately.
- Artefacts should be appropriately referenced to quickly identify the sources, which n turn can be auditable.
- Since the assurance cases are significant in size, it is appropriate to provide a concise summary as an artefact rather than the complete documentation. But this summary should provide a link to the complete documentation.

'**Evidence**' asserts the truth of an argument made to support a claim. Artefacts are used as evidence to prove the arguments. The evidence should have the following properties:

- With relevance to the scope of a case, the evidence must justify the context of the claim.
- It is important to have trustworthy evidence, which authorized individuals and organizations accept.
- Evidence is subject to quality standards and should be verifiable.
- In certain situations, the evidence can not justify the arguments in certain conditions, leading to counter-evidences to a claim. In such cases, the uncertainty should be accepted, and the claim can be modified.

The described scope of an assurance case should be valid. An argument is said to be '**valid**' when it formally satisfies the argument's logic which can be both true or false. In some circumstances, invalid cases can also support a claim, and valid cases may not support the claim. In such instances, we say that the assurance case is not '**sound**.' We call an assurance case is sound when arguments are valid and formally satisfies the argument logic. The following points should be considered to state a valid claim:

- The boundaries of the case should be well defined without ambiguity in the context.
- Valid and coherent arguments should be made.
- One or more claims should be made about the scope of the case.
- A logical and structured decomposition of arguments is necessary till proper evidence is provided.
- When reviewed, the arguments should resist fair criticism.
- Using self-evident arguments is possible, but that should be acceptable by the experts in the domain.

Assurance cases can be represented using different structures like text-based, tabular, and graphical notations. It is cheaper to maintain a text-based assurance case and quickly publish it to vast readers, but it has weaknesses. Creating text-based notations with precise language is not possible for everyone, and different readers can infer a different meaning from the same argumentation. Too many cross-references that some assurance cases may need will disrupt the flow of the argumentation and spoil the soundness of the case. These weaknesses can be overcome by using graphical notations, the preferred method of writing assurance cases.

2.1.3 Types of Assurance Cases

Assurance cases can be mainly generalized into three types. They are textual, tabular, and graphical. Depending on the work culture and standards that an organization follows, they have their preference to choose the type of notation. Here the preference may depend on the stakeholders' knowledge, cost, time, storage, and maintenance of the assurance cases.

Textual Notations:

Textual assurance cases are the simplest type of notation that is easy to write initially but are very bad at maintaining clarity while achieving bigger goals. Like any other assurance case, this notation has the same components like claims, arguments, evidence, assumptions, and justifications. These are written as a series of texts hierarchically. For example, a typical assurance case built using textual notation looks as shown below:

The following example is taken from the Holloway's paper titled "Safety case notations: alternatives for the non-graphical inclined?" [8]

"***Claim 1: Control system is acceptably safe.***"
"*Context 1: Definition of acceptably safe.*"
—-"***Claim 1.1: All identified hazards have been eliminated and sufficiently mitigated.***"
—-"*Context 1.1.a: Tolerability targets for hazards. (ref doc.x)*"
—-"*Context 1.1.b: hazards identified from functional hazard analyisis.(ref doc.y)*"
—-"*Strategy 1.1: Argument over all identified hazards (H1,H2,H3).*"
———"***Claim 1.1.1: H1 has been eliminated.***"
———"*Evidence 1.1.1: Formal verification.*"
———"***Claim 1.1.2: Probability of H2 occurring ¡ 0.0001 per annum.***"
———"*Justification 1.1.2: 0.0001 per annum limit for catastrophic hazards.*"
———"*Evidence 1.1.2: Fault Tree analysis.*"
———"***Claim 1.1.3: Probability of H3 occurring ¡ 0.001 per annum.***"
———"*Justification 1.1.3: 0.001 per annum limit for major hazards.*"
———"*Evidence 1.1.3: Fault tree analysis.*"
—-"***Claim 1.2: The software developed to the IL appropriate to the hazard involved.***"
—-"*Context 1.2.a: (Same as Context 1.1.b)*"
—-"*Context 1.2.b: Integrity level (IL) process guidelines defined by reference X.*"
———"***Claim 1.2.2: Primary protection system developed to IL 4.***"
———"*Evidence 1.2.2: Process evidence of IL 4.*"
———"***Claim 1.2.2: Secondary protection system developed to IL 2.***"
———"*Evidence 1.2.2: Process evidence of IL 2.*"

Some of the other known methods for text-based argumentations are NASA's Friendly Argumentation Notation (FAN) [9] and two approaches for assurance argumentation [10]. Some drawbacks of using a textual notation are the language used, maintainability, clarity, and storage. The problem with the language is, not everyone can write statements that hold only one meaning and perspective. It is often a problem that the stakeholders understand the claims differently. Maintaining and storing the assurance cases documents will be very costly and less effective in the long term for more significant projects.

Tabular Notations:

Tabular notations also have similar assurance case components and are simpler to write when compared to graphical notations. Unlike textual notations, tabular notations maintain better clarity as the assurance case components are grouped and tabulated as per the level of claim. An example for a tabular assurance case is shown in Figure 2.1.

Claims		Context and assumption	Strategy and argument	Evidence and reference
Top claim	ABC medical device is safe for its intended use	Refer to intended use. "Safe" and "mitigated" mean residual risk is acceptable per 21 CFR 860.7(d)(1).	Argue that all applicable hazards are identified and mitigated. Confidence argument on why hazards are identified correctly, completely, and appropriately.	Intended use, risk acceptance policy, other evidence as needed.
Top subclaims	Sources of **harm** (top hazards) are mitigated	Explain the potential harm and its severity. Describe context and assumption as applicable.	Argue that hazardous situations are identified and mitigated. Confidence argument on why hazardous situations are identified correctly, completely, and appropriately.	Evidence to support strategy or argument, as applicable.
Subclaims	Risk of **hazardous situations** is mitigated	Explain the hazardous situations. Describe context and assumption as applicable.	Argue that causes are identified and mitigated. Confidence argument on why causes are identified correctly, completely, and appropriately.	Evidence to support strategy or argument, as applicable.
Subclaims	Risks of **causes** are mitigated	Causes include faults, conditions, interactions, and contributing factors. Describe context and assumption, if any.	Argue that subcauses are identified and mitigated. Confidence argument on why subcauses are identified correctly, completely, and appropriately.	Evidence to support strategy or argument, as applicable.
Subclaims	Risks of **subcauses** are mitigated	Describe context and assumption information, as applicable.	Argue that controls are established. Confidence argument on why controls are collectively sufficient to reduce the risk (severity or probability) to be at acceptable level.	Evidence to support strategy or argument, as applicable.
Subclaims	**Risk control** is established	Describe context and assumption information, as applicable.	Argument on why control implementation is correct, complete ,and appropriate.	Requirements, design, testing, labeling, standard operating procedures, etc.

Figure 2.1: Tabular assurance case example of a medical device. Image credit: [11]

Similar to the textual assurance cases, the language used will be a major drawback. Storage and maintenance are costly.

Graphical Notations:

Graphical notations are currently the widely used tools for writing assurance cases. These notations are comprehensive and are good at maintaining clarity, and there is not much problem with the requirement for the usage of extensive language. "Structured Assurance Case Metamodel (SACM)" [12] is a standard first developed by the "Object Management Group (OMG)". OMG is a popular community that developed

many tools and notations for software development and maintenance. This community is popularly known for its contibution towards creating Unified Modeling Language (UML) booch2005unified and Systems Modeling Language (SysML) friedenthal2014practical diagrams. OMG's SACM standard is the foundational metamodel for many graphical notations.

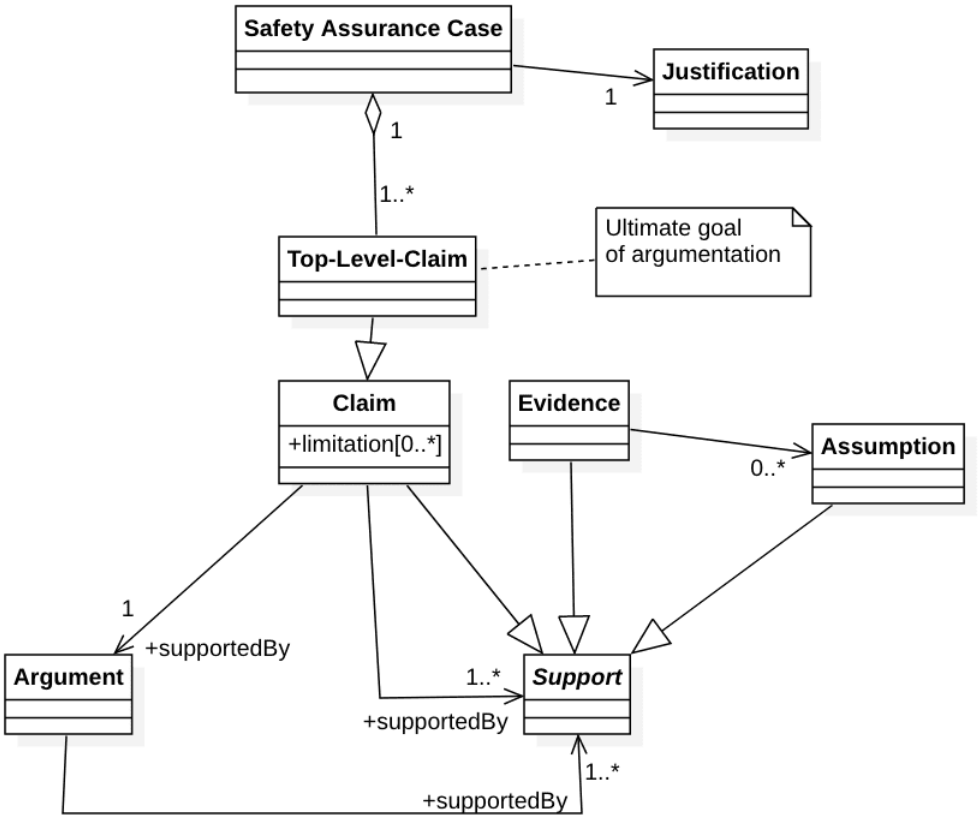


Figure 2.2: SACM block diagram. Image credit: [13]

Popular graphical notations for writing assurance cases are Claims Arguments Evidence (CAE) and Goal Structuring Notation (GSN). The principal components and their purpose in any of these notations are almost identical, but the structures have some differences. The earliest graphical notation is CAE and was developed in the city university of London, UK. The foundation for this notation is mainly associated with the work of Adelard LLP [14]. CAE contains the general blocks for claims, sub-claims, justification, and apart from that, and it has side-warrant, backing, and system information block. CAE is one of those notations used extensively for a certain period. It is still a preferred notation in many of the software domains. Figure 2.3 shows structure of CAE notation. The popular tool for writing structured arguments

using CAE notation is Assurance and Safety Case Environment (ASCE), a commercial product of Adelard.

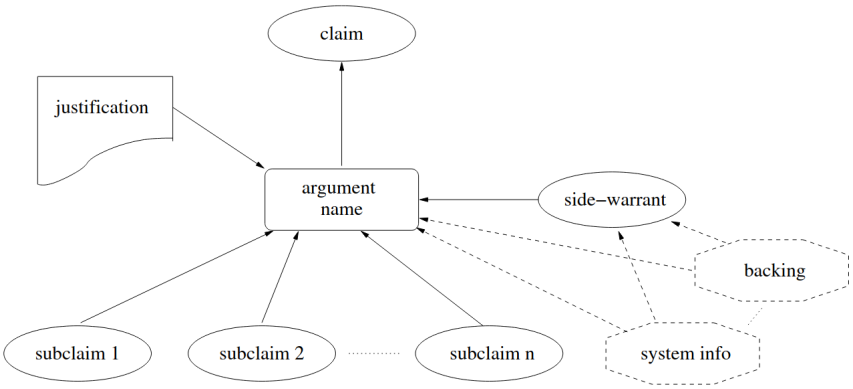


Figure 2.3: CAE block diagram. Image credit: [15]

GSN is the current state-of-the-art graphical notation widely used in many domains. This notation was developed in the 1990s, where the University of York is the major contributor. This notation is closely associated with Tim kelly's doctoral thesis and finds most of the foundational ideas from his work. Similar to CAE, GSN also evolved over the years and at present maintained by ACWG. An example of GSN notation is shown in the Figure 2.4.

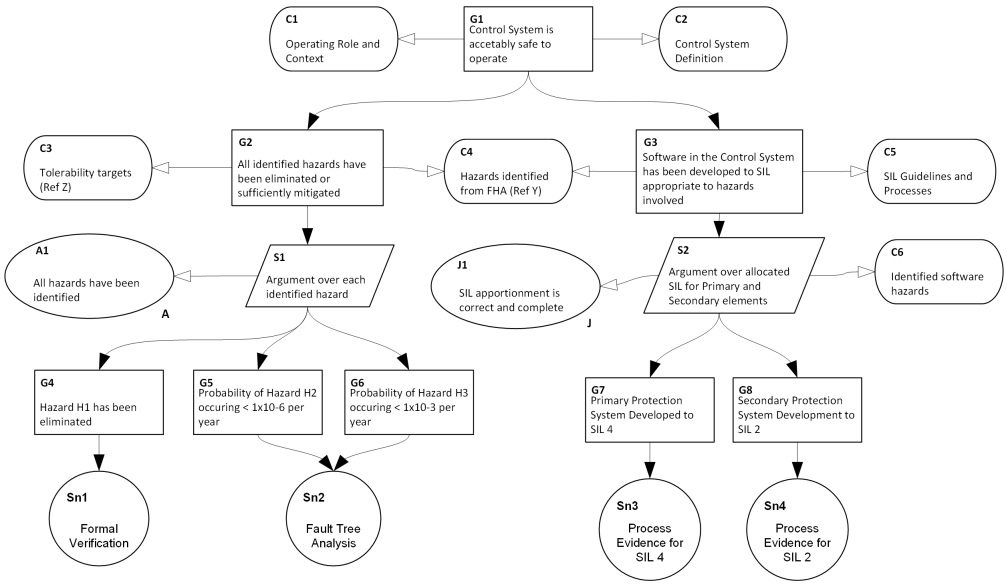


Figure 2.4: GSN use-case example. Image credit: [16]

2.1.4 Limitations of Assurance Cases

Assurance cases are used as a tool to assure some claims with arguments, which means here, we are indirectly using it primarily to support our goals. We call it an unbalanced argument if we don't argue with positive and negative statements. Assurance cases are unbalanced arguments, which means, like an argument in a court of law, we don't use both positive and negative arguments while building an argumentation. This will lead to raising biased arguments that help in achieving the goal. This is the primary criticism that the assurance cases face [17]. The following are some of the biases and effects authors face while writing a GSN.

Confirmation bias is the tendency to relate or conclude by inferring from some previous information. For example, providing reports from past experiments that we believe are related to our claims as evidence.

Cognitive bias is the tendency to move towards rational thinking by providing illogical statements and, in other words, overlooking the facts by prioritizing our perspectives and prior beliefs.

Disconfirmation bias is not being able to open for different kinds of information. This is, in a way, related to confirmation bias. We should not always scrutinize evidence based on our prior beliefs.

Conservatism bias is the tendency not to favor new emerging information. For example, the belief we built by performing many experiments in the past should not affect our judgment about new data.

The stakeholders sometimes experience the ***ostrich effect*** by not listening to the new evidence and firmly believe that the argumentation is not valid. It is referred to as an ostrich putting its head into the sand.

Observer-expectancy effect is experienced by the observer of an experiment, who, in his mind, picture the result based on some beliefs, which makes him unconsciously misinterpret the new data.

As per the ACWG, one should not look at these biases as limitations but instead use them to avoid bias while making arguments in an assurance case. Levsson [18] stated that:

"*To avoid confirmation bias and compliance-only exercises, certification should focus not on showing that the system is safe but in attempting to show that it is unsafe.*"

From the previous sections, we can understand that we use an assurance case, not active our top-level claims instead, we use it as a tool to build an argumentation. Understanding this argumentation will assure the stakeholders that the claim is acceptable. In short, we do not **achieve** safety or security goals we instead **satisfy** them.

2.2 Goal Structuring Notation

GSN is a graphical assurance case notation that is currently well maintained for writing structured argumentation. Compared to tabular or text-based argumentations, graphical notations are more expressive and help keep clarity. GSN was developed in the 1990s at the University of York, York, England. Initially, it was created as a part of the ASAM-II project [19] with heavy influence on the argumentation processes and requirements engineering. The current version has gone through many refinements and is now maintained by ACWG. GSN has the same pivotal components as claims, arguments, evidence, justifications, and solutions as any other assurance case structure. The keywords used might be slightly different, but the general meaning holds the same. For example, in GSN, the keyword goal is used instead of claim, strategy instead of argument, solution in the place of evidence.

The core GSN is built with the following elements and they can seen in the Figure 2.5:

Goal	claim of the argument that is represented in a rectangle.
Context	statement the describes the scope of a goal, which is represented as shown in the figure
Strategy	the argument infers the relationship between goal and subgoals, represented in a parallelogram.
Assumption	statement that is not proved but assumed, represented in an oval with the letter 'A' on the right top or bottom.
Justification	a statement that justifies the argument, represented in an oval with the letter 'J' on the right top or bottom.
Solution	evidence that is represented in a circle

Table 2.1: Components of GSN

The goal structure represented using GSN is a directed acyclic graph, which means that one element can have multiple parents and children, and the graph structure will not allow loops. So the relations between the elements are represented by two other components:

SupportedBy	used as a connection between goal-strategy, strategy-goal, goal-goal, and goal-solution, represented using an arrow with a solid head.
InContextOf	used as connection between goal:context,assumption,justification and strategy:context, assumption, justification. Represented by an arrow with a hollow head.

Table 2.2: GSN relationship identifiers

More formally, the relations that are represented using the GSN are:

- The relation between the parent goals and subgoals as premise and conclusion.
- Supporting evidence as a solution for the goals and
- Relation explaining the boundaries or scope of the argument stating the context.

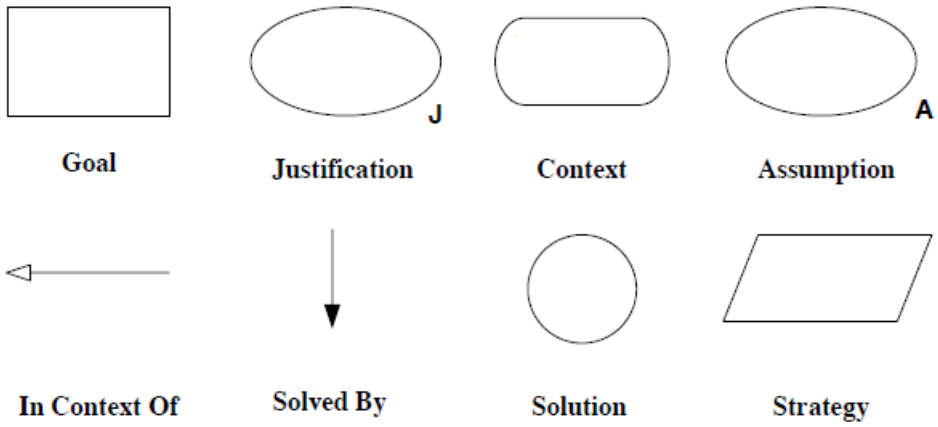


Figure 2.5: Components of GSN. Image credit: [20]

2.2.1 Good Practices for the Development of GSN

To write a sound assurance case using GSN, the author should meet the following goals for the document:

- **Clarity**: Provide explicit goals/claims and references, and arguments should present good cohesion.
- **Comprehensibility**: the goal here is to maintain consistency in the information understood between the author and reader by writing good statements relevant to the context of the claim.
- **Exactness**: GSN should contemplate the actual state of the evidence.

The primary purpose of using the graphical argumentation methods is to help the stakeholders or the readers to build a logical judgment about the goal. GSN or any other graphical notations support the authors in achieving this. There can be some advantages in using GSN because of its structure, which helps maintain clarity. For example, the claims made at the higher level of the GSN structure hold information at an abstract level, whereas we can observe much more logical statements when going down through the structure. This structure will extend to one point, holding meaningful statements supporting the higher claims, thus representing a directed acyclic graph. GSN elements that explain the context, strategies between the goals and sub-goals followed by assumptions, justifications for the claims made makes the argumentation sound. So these elements have to use as much as necessary to maintain the soundness of the argument. I one has to make the two most relevant claims, it is suggested to go back to the top-level claims and start refining the subgoals such that the two similar claims can be represented as a single claim. One can use the elements of the GSN to explain further the claim made in these situations. The relationship between the parent goals and sub-goals should always be represented using strategies using the 'InContextOf' element. The assurance case can be as long as necessary, which means the GSN can go to the next page in many cases. In such cases, a connector element is available in GSN that

explains the connection across multiple pages.

Six-Step method to develop GSN:

T.Kelly kelly1999arguing from York University explained a systematic approach that helps build a sound assurance case using GSN. This approach consists of six steps, and it is applied to both top-down and bottom-up development of GSN with slight modifications.

Top-Down development:

1. Identification of goals for argumentation.
2. Defining the scope of the goal.
3. Identification of strategies that support our goals.
4. Defining the scope/basis for choosing the strategy.
5. Identification of subgoals by elaborating the strategy or providing evidence for the goal.
6. Identification of the solution for the final subgoal.

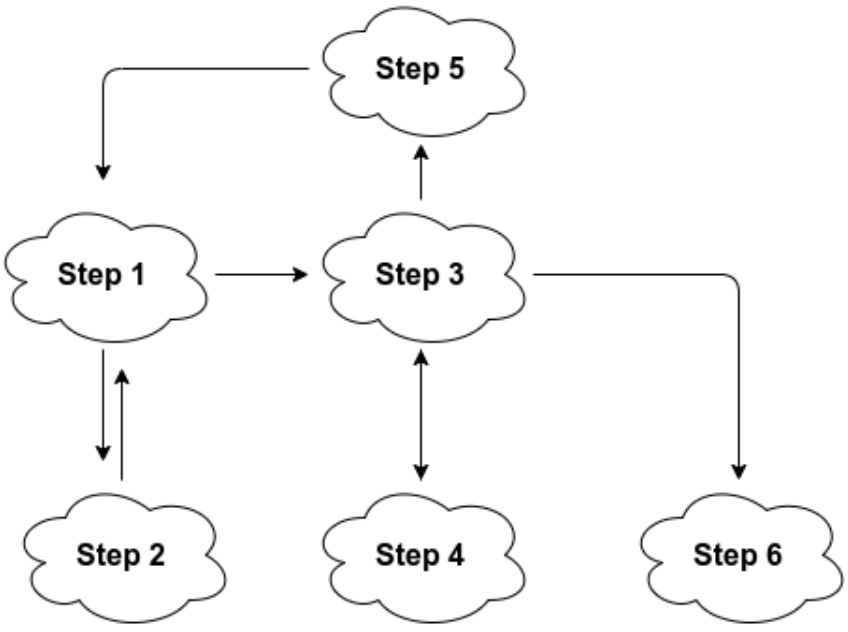
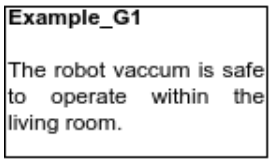


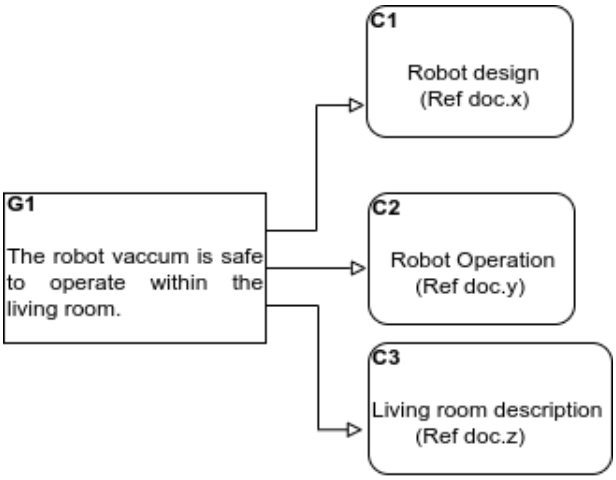
Figure 2.6: Top-down approach of GSN structure

A detailed step-by-step example for creating a top-down GSN structure is shown for a simple autonomous mobile robot use case using this six-step method.

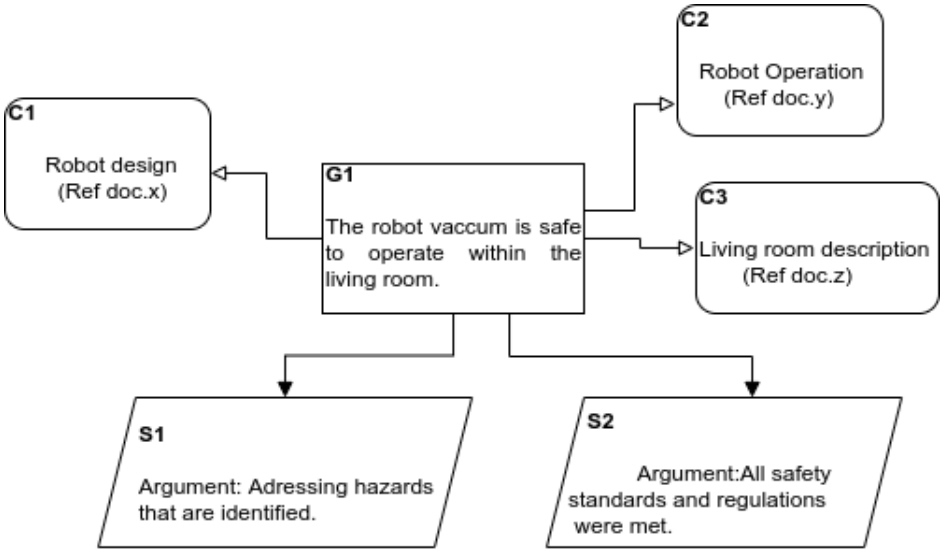
***Step-1*:** The top-level claim is identified and stated as the goal in this step. This goal should be more abstract and should not express tiny details. Not providing many details in top-level goals is a suggested practice because we don't want readers to draw different meanings to the top-level goals without a broader understanding of the scope of the goal and flow of the argumentation. A simple top-level goal is shown below.



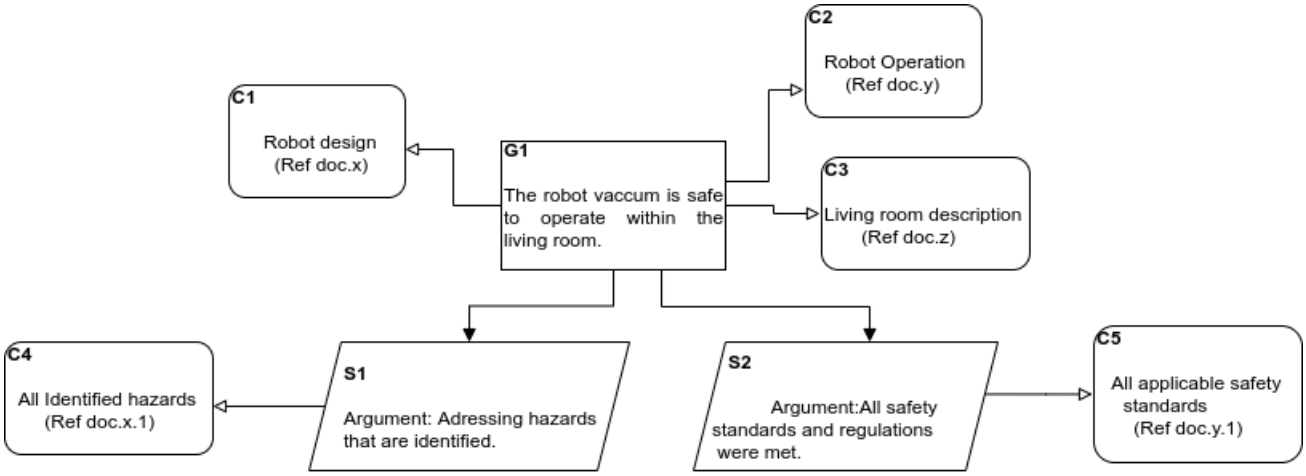
***Step-2*:** It is crucial to state on what basis one has claimed a goal while building an argumentation. The assurance case should be detailed enough that provide proof for validating the claims. GSN has a 'context' element to provide evidence and explain that claim's boundaries. It is good to treat our claims as statements with no universal validity and provide enough evidence to state the same. This can be achieved by identifying the information about the system, the context, and the argument made. From the figure we can see that for the claim safe operation, we provided the information about the robot design and operation, which shows that the robot is working at an acceptable level of risk. Description of the living room will explain the boundaries of the claim.



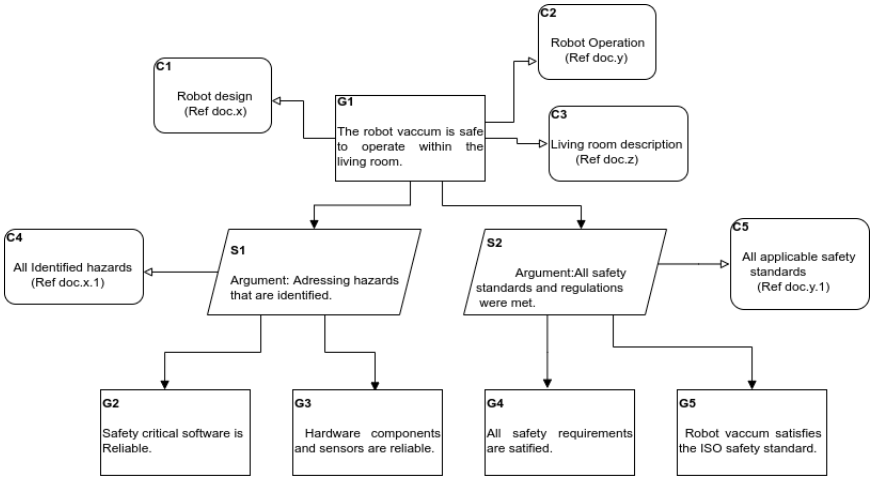
***Step-3*:** This high-level goal is decomposed into low-level goals by identifying the strategies that help build the argumentation. Here, the author has to think from the author's and reader's perspectives and then argue. The author's perspective can be "to say that the goal is true what arguments I have to make." Similarly, the reader's perspective can be "to convince that the goal is true what arguments I have to make."



***Step-4*:** The argument made in the strategy only holds true when we can explain why we made that argument. Similar to the contexts built for the goals, we follow the same process to define the scope of the strategy.



***Step-5*:** A new subgoal is created to apply that argument from the strategy. Here we can see the strategy as an essential argument or an argument that maintains clarity for creating the new goal. The strategy maintains the relationship between the parent goal and sub-goal. Generally, strategies are the elements in GSN that drive the argument in a meaningful way. This means one should take care of confirmation bias that naturally every human experiences while building an argument. Bias in strategies affects the soundness of an assurance Ccase.



***Step-6*:** By the time we provide some evidence as a solution for the argument, our goals will be atomic, which means they are at a low level and mainly to the point. We provide the reports as evidence that generally contains the experiment results or certification stating that we satisfied/passed specific standards, validating our overall argument. Figure 2.7 shows the final structure of the assurance case.

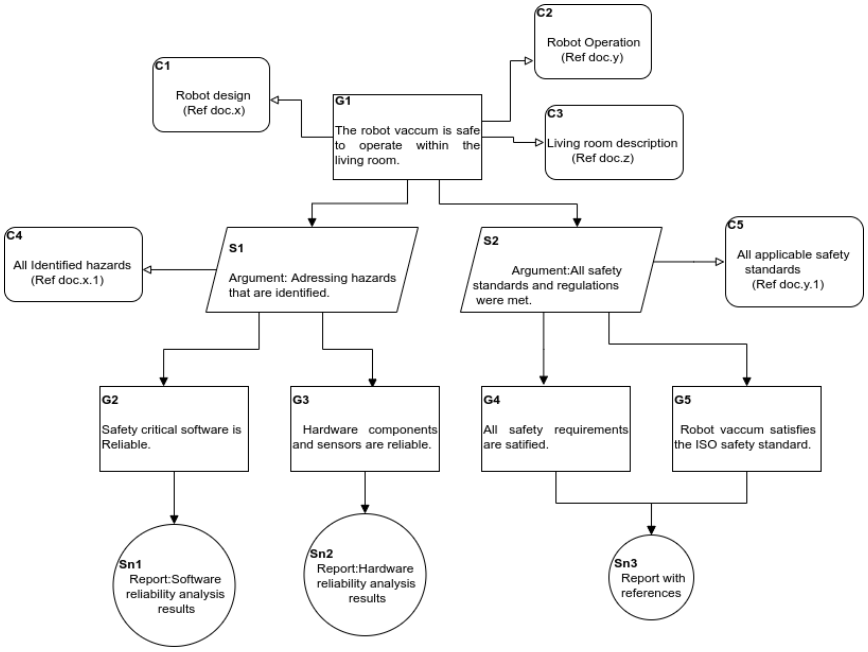


Figure 2.7: Top-down approach example

Sometimes we can encounter a situation where we cannot decompose the goal further and neither provide a solution. The bottom-up approach will be an excellent choice to create an argument in such cases. In

the bottom-up approach, we first take all the evidence and build up the tree to make logical modifications to the claims. Here, there can be a chance that we end up with such a claim where we can satisfy only a part of the desired goal. But, this is good in reality as the evidence we have will only meet the low-level goals but not our desired goal. This means we have to run other experiments to get additional evidence to satisfy our desired goals.

Bottom-Up development:

This approach is adopted from the top-down development and consists of seven steps. Figure 2.8 shows the relation between different stages of bottom-up development.

1. With the evidence, identify the solution for GSN.
2. Use the evidence as an assertion and create goals by inference.
3. Using the subgoals and evidence build back high-level goals.
4. Explain how goals and subgoals are related to one another using strategies.
5. Define the scope for the goals.
6. Check for completeness of the structure.
7. If the current goal structure is for a subgoal, join it to other high-level goals.

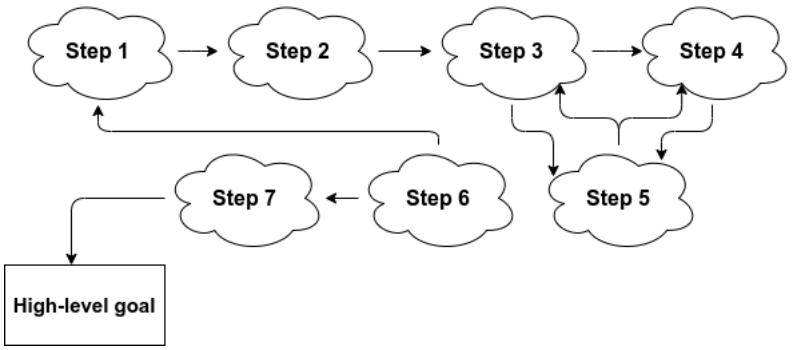
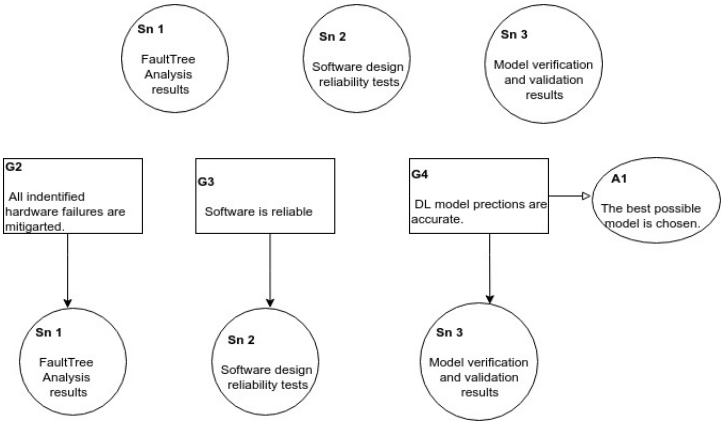


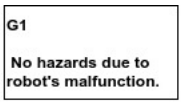
Figure 2.8: Bottom-up approach of GSN structure

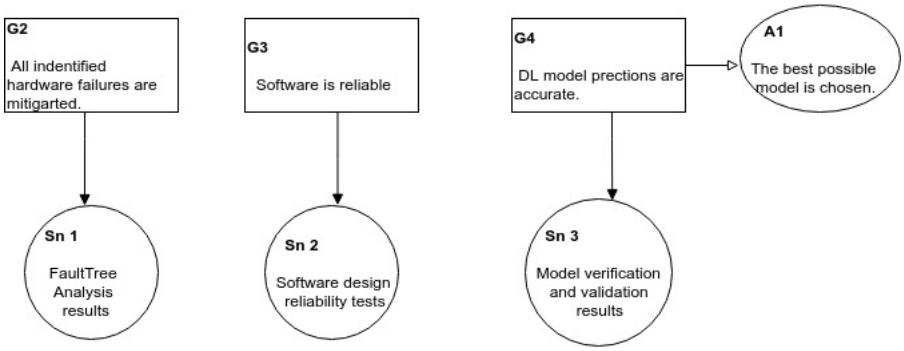
***Step-1*:** For the bottom-up approach, first, we have to identify all the relevant evidence for building an assurance case. When it comes to a safety case, this evidence is usually verification and validation results, fault tree analysis results, hazard analysis reports, and other relevant test results that support the goal.

***Step-2*:** After identifying the relevant evidence, we should consider what claims this evidence can support. These claims are generally assertion statements for the solutions. Since these claims are low-level claims, they can be atomic.

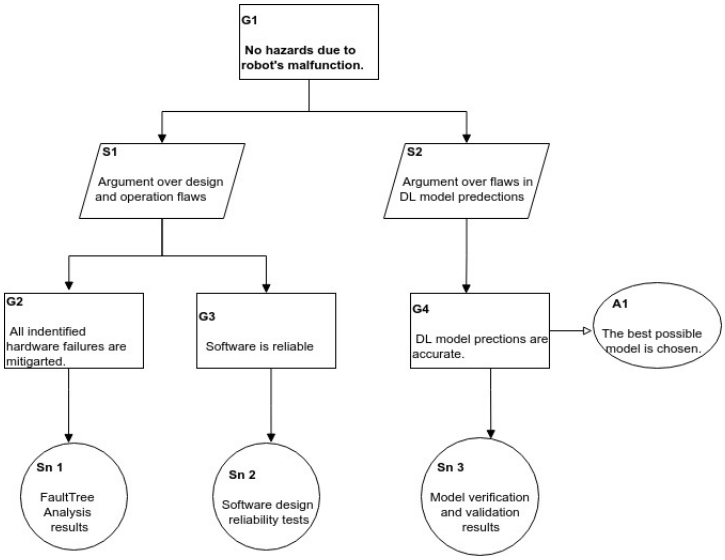


***Step-3*:** Now, as the atomic goals are created at the bottom of the tree structure, we have to develop top-level plans that claim something at the system level. Depending on the type of argumentation and the top-level goals, these intermediate goals can be abstract and relate to safety properties or similar dependability attributes. For instance, these claims can relate to the hazard analysis to support the high-level safety claims.

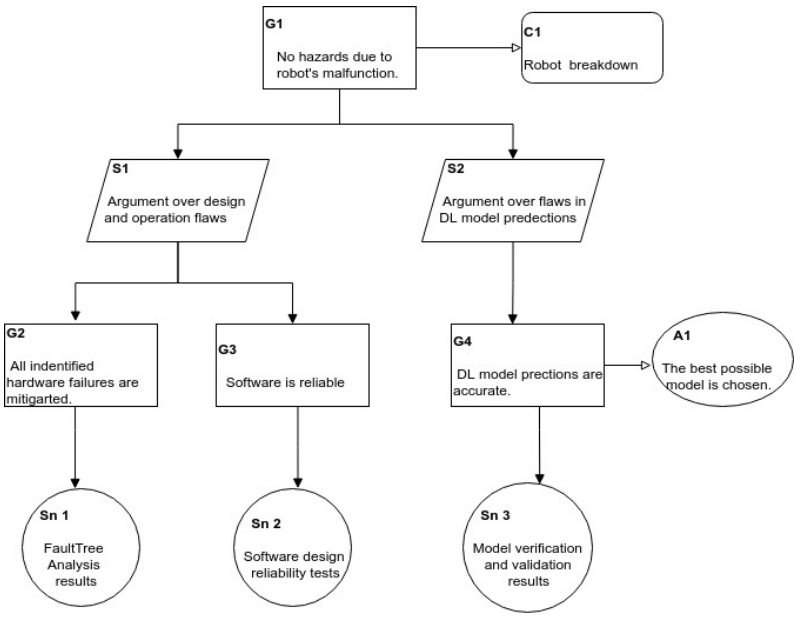




***Step-4*:** A proper strategy is needed that satisfies the relation between these different goals. The strategy is meaningful enough and should help decompose the higher-level goals into lower-level ones.

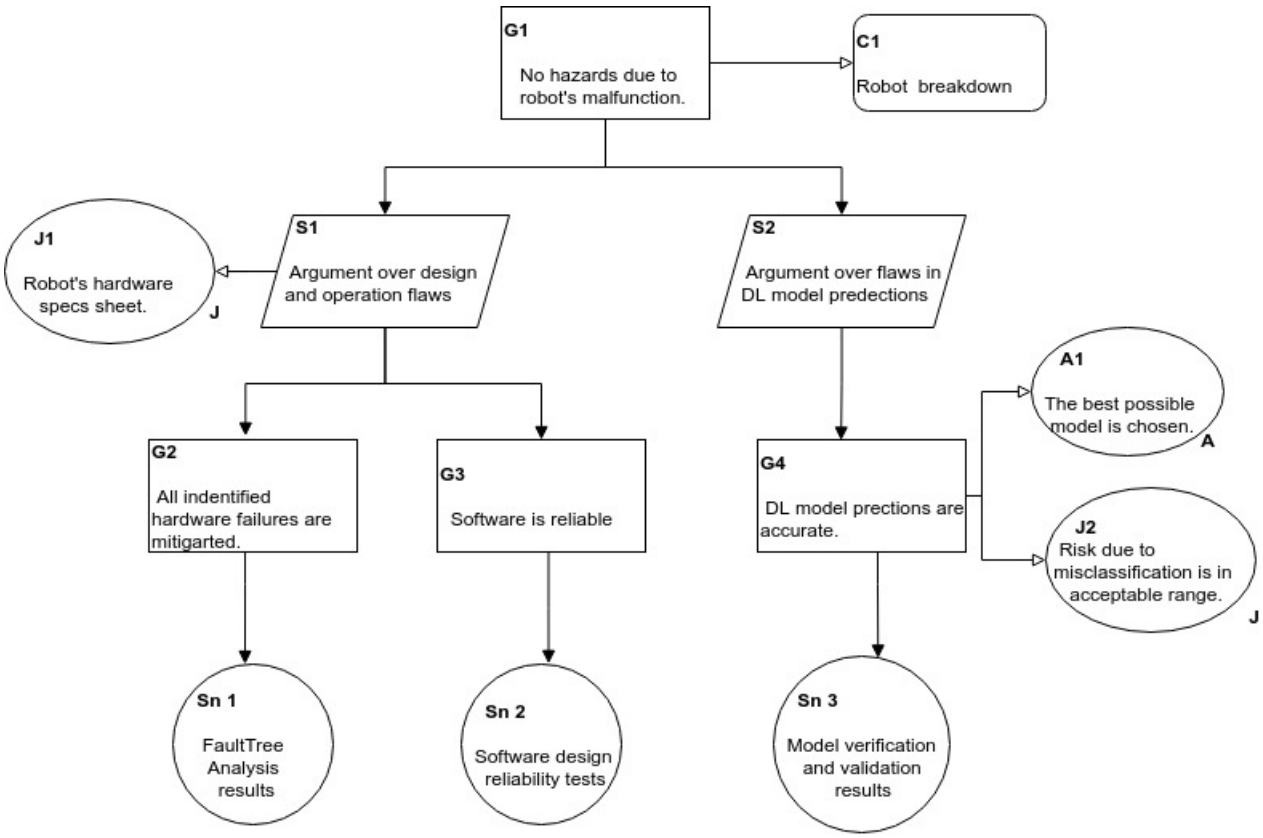


***Step-5*:** In general, concerning the kind of evidence we have, the scope of the claim is limited to certain conditions. So for these goals, we have to explain the scope using the context element of the GSN.



***Step-6*:** By now, we already have a tree structure, so it is good to go through the structure and add assumptions and justifications that increase the assurance case's soundness. This is the stage where

we can refine our claims and check for the clarity of the overall structure.



***Step-7*:** Generally, the bottom-up approach is used to create an argument structure that supports some low-level goals, joining with other high-level goals. This method helps in creating such a smaller structure with less confirmation bias. These structures, when joined together, can build an assurance case that is sound. Figure 2.9 shows the final structure of the assurance case.

2.2.2 Evaluating a GSN

Using GSN, we assure top-level claims by decomposing them into low-level claims. This decomposition should be appropriate enough for stakeholders to understand. As discussed in the previous sections, stakeholders are a group of people who can be auditors, users, developers, and sometimes even business analysts. This diversity in the stakeholders will introduce statements in the GSN that are more generalized,

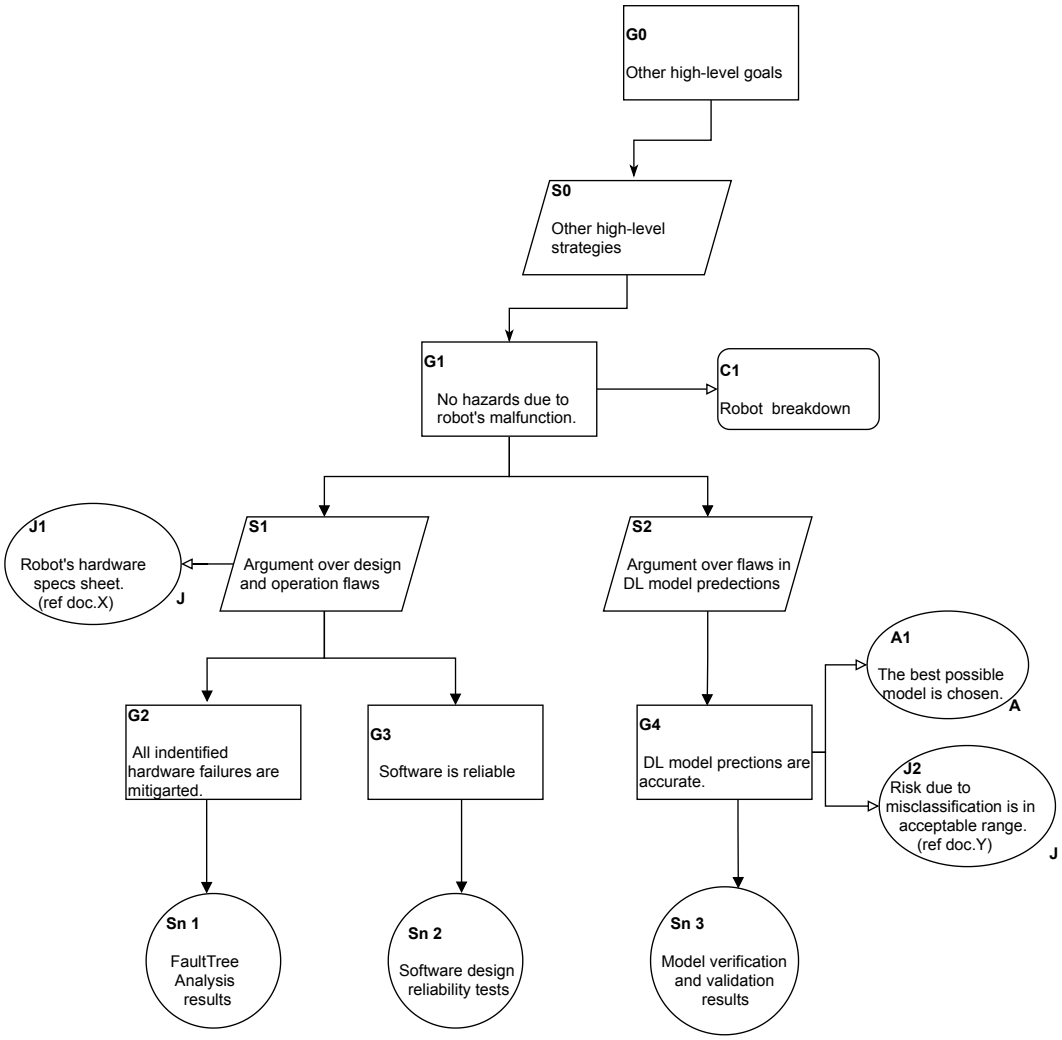


Figure 2.9: Bottom-up approach example

which can affect the soundness of the assurance case. So the author must evaluate the GSN so that the clarity is maintained without compromising soundness.

Assurance cases built during the development lifecycle will help identify the problems during the product development phase. So it is common and better to write an assurance case during the lifecycle. The assurance cases can be refined to make the arguments and statements compelling during the development phase. As we know about the stakeholders before, making generalized statements without missing the true meaning is possible. In this phase, timely self-review and intermediate discussion with the stakeholders make the assurance case sound.

***Stages of the review process*:**

Reviewing GSNs for safety-critical processes usually contains many stages to meet the industrial quality standards. This reviewing process can differ across organizations. But every GSN review process should at least have these four stages. Figure 2.10 explains different stages of GSN review process.

1. Understanding Arguments.
2. Checking GSN structure.
3. Checking expressiveness.
4. A critical review of arguments

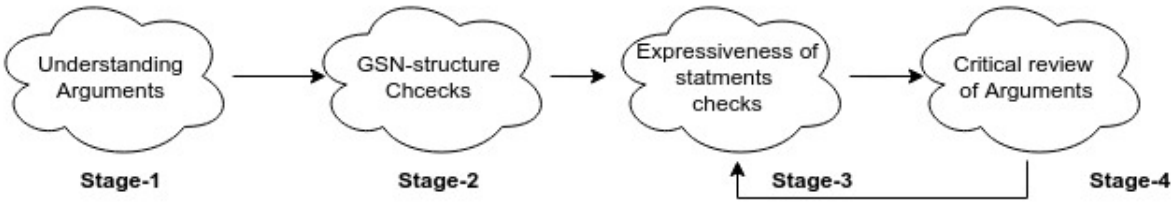


Figure 2.10: Stages of GSN evaluation

***Stage-1*:** In the initial stage, the authors and the reviewers understand the claims, arguments, and assumptions of the assurance case. This is the phase in the review process that takes minimal effort. The argument's structure is evaluated with the GSN notation guidelines.

***Stage-2*:** The GSN checks in this stage are straightforward. We provide supporting arguments or assumptions to the claims that we make and explain the scope of those claims. By the end of this review stage, it is expected to have no unclear arguments in the assurance case.

***Stage-3*:** In this stage, we check whether all the statements made in claims, arguments, assumptions, justifications are clear and expressive. We will add any references to support the basis of our claims

and arguments.

***Stage-4*:** For any inductive statement, the truth will be in the premises, and this truth varies depending on the statement's scope. To check the sufficiency of the statement, the critic should look through the lenses of the attributes like coverage, relevance, dependency, directness, and robustness. The criticism will help find the proper relationship between the arguments.

After this review process, it is suggested to review the evidence for any errors. The hardware component specs sheets and other reference sheets are checked whether the latest versions are being used.

2.3 Dependability

Dependability, usability, cost, functionality, and performance are vital parts that characterize any computing system. Laprie et al. [21] in their report "Fundamental concepts of dependability," they explained dependability as "the ability to provide a justifiably trusted service for any computing system." Here, the word justifiably trusted means preventing the computing system from most frequent failures. This also includes outrages, which means providing incorrect service for the called functionalities of the computing system. If a system fails to provide this trust, it is considered non-dependable. The concept of dependability and its fundamentals explained in this section has been taken [21] for reference. The dependability concept can be broken down into three essential parts.

- **Threats:** Threats cause due to errors, failures, and faults. The failure of the overall system occurs due to some errors in parts of the system. A fault is a reason for an error.
- **Attributes:** The concept of dependability comprises of six attributes. According to Laprie's [21] report,
  - "***Safety*** *- the absence of catastrophic consequences on the user and the environment.*"
  - *"**Reliability** - continuity of correct service."*
  - *"**Availability** - readiness of correct service."*
  - *"**Safety/Confidentiality** - the absence of unauthorized disclosure of information."*
  - *"**Integrity** - the absence of improper system state alterations."*
  - *"**Maintainability** - ability to undergo repairs and modifications."*
- **Means:** While developing a dependable system, the following four techniques are used:
  - ***Fault prevention*:** avoiding the faults.
  - ***Fault tolerance*:** delivering correct service despite faults.
  - ***Fault removal*:** reducing fault's severity.
  - ***Fault forecasting*:** identifying the likelihood of faults.

To develop a dependable system, the developers have to look into and satisfy all the attributes mentioned above. This R&D is focused on satisfying the reliability attribute of a LEC. Figure 2.11 shows different components of dependability.

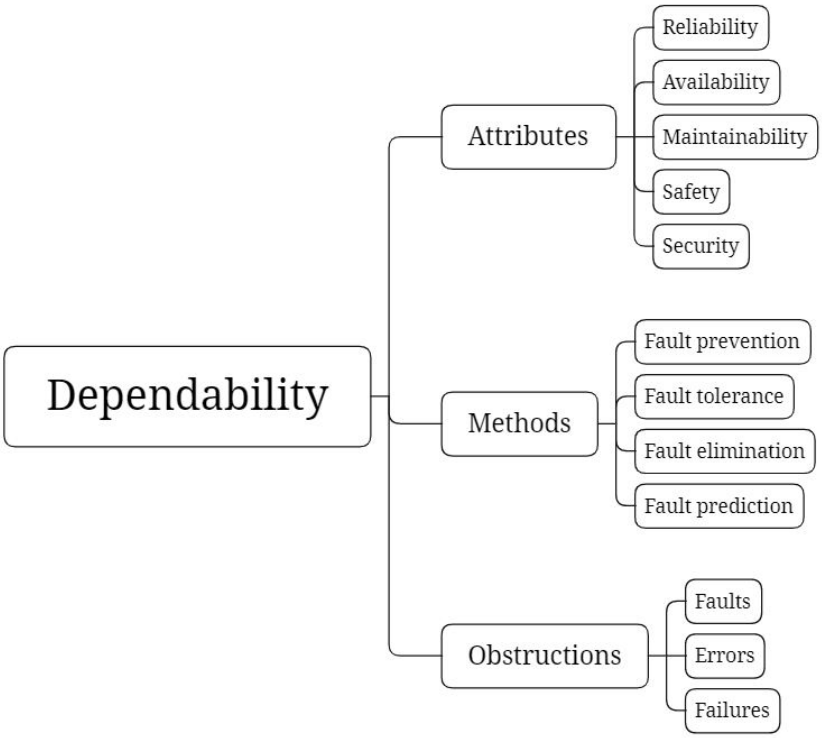


Figure 2.11: Components of dependability

2.4 Reliability

A product is reliable if it provides a service continuously without failure for a specified period. In several cases, these constraints are limited even for a specified environment. In different engineering domains, this term reliability is dependent on multiple factors. Some of these factors are even system-specific. Whatever the domain is, the general metric that tells the reliability of a product is the failure rate. The failure rate is the number of failed operations (F) out of "N" operations over a period of time (T). Equation 2.1 shows the formula for failure rate. The focus of this R&D is on the software domain. This section briefly discusses the notion of software reliability and the methods and metrics for determining software reliability.

$$Failure\ rate(\lambda) = \frac{F}{N * T} \tag{2.1}$$

2.4.1 Software Reliability

Reliability is an important quantitative term of any software product. However, it is constrained by development time and budget in many circumstances, and balancing the two is difficult. Similar to any other domain, software reliability depends on the failure rate. There are many software practices that on following help in achieving better reliability. Unlike any other product involving hardware components, time is an essential parameter in defining the failure probability. This is different in the case of software as there won't be any wear and tear of software over the period. Here software complexity can be an essential parameter; as the software grows, it is hard to achieve reliability. Hardware flaws are mostly physical faults, but software faults are primarily due to design flaws. When accompanied by carelessness, design flaws, insufficient testing, and code incompatibility will increase the failure rate.

2.4.2 Software Reliability Metrics

In any engineering field, to evaluate a product's performance or reliability, a valid metric is needed, but this is not in the case of software. This is because there won't be any software changes unless there are any software upgrades and changes in the environment [22]. Nevertheless, many studies tried to quantify the software reliability with a metric. The available software reliability metrics can be categorized into four groups.

- **Product metrics:**
  - *Lines of Code (LOC)* - determining the software's size, which defines the software's complexity. If software complexity increases, reliability decreases.
  - *Functional point metric* - number of inputs, interfaces, frameworks, outputs which increases the complexity of the software.
  - *Complexity-oriented metrics* - explain the complexity of the software control structure.
- **Project management metrics:** The design decisions during the development process and providing time, budget sufficiently.
- **Process metrics:** Software quality assurance and development process complying with the ISO or similar standards increases the reliability.
- **Fault and failure metrics:** Running the software for many iterations and the failure probability calculating the failure probability is used for calculating different metrics.
  - *Mean time between the failures (MTBF)* = (Total time) / (Number of test runs)
  - *Meantime between repairs* = (Maintenance time) / (Number of repairs)
  - *Failure rate* = 1 / (MTBF)
  - *Failure probability* = (Number of Failures) / (Number of test cases)

2.4.3 Software Reliability Testing

The reliability of software decreases as the complexity of the software increases. It is relatively easy to guarantee reliability in the unit phase, but further upon integrating this software, units increase the software's complexity. By the end of the product phase, quantifying the reliability is challenging. Based on the approach for solving this problem, there are two software reliability testing methods.

***White-box testing*:**
Software developers do White-box testing in any project. In the white-box testing method, the software tester (here developer) knows everything about the software's design patterns, code implementation, and product structure. This includes writing many test cases and passing them. Some of the white box testing methods are:

- *Statement coverage testing* - test cases are written to run every functional code block at least once.
- *Decision coverage testing or conditional testing* - making sure every logical condition in the software is run at least once.
- *Dataflow coverage testing or loop testing* - running test cases such that every functional pair is run at least once.
- *Mutation testing* - creating a mutant condition by modifying the predefined set of rules and testing for failures and, for example, changing the sequence of steps while creating a user account in the case of an online e-commerce website and finding out the consequences of that.

***Black-box testing*:**
Software testers will perform black-box testing. In black-box testing, the software tester does not know anything about the software's design patterns, structure, and implemented code. This is also known as behavioral testing. In some cases, the tester will act as an end-user and find the errors while using the end software product. Some of the black-box testing methods are:

- *Functional testing* - in functional testing software's basic usability, free navigation, main functions operation, accessibility, and the display of error messages when something breaks are tested. In brief, all major functionalities are expected by the client being tested.
- *Non -functional testing* - aspects like performance and software reliability are tested.
- *Regression testing* - Regression testing is performed after fixing a bug or a software update. This test ensures that this new change does not affect other software functionalities.

2.5 Learning Enabled Components

The software components that are influenced by the learning methods like machine learning and deep learning for providing intelligent services or functions are LECs. Such components are integrated with

safety-critical applications like robots, self-driving cars, IoT devices, medical diagnoses, and many other engineering applications for their ability to provide an approximate solution for complex problems. In simple terms, DNNs and Machine Learning (ML) models are good examples for LECs. Though some of these learning techniques have been proposed long back, these techniques recently gained popularity. This is because of the availability of extensive data for training and GPUs that reduced the training time to solve these complex problems.

DNNs are data-driven models, which means they are non-deterministic in nature. In a non-deterministic model, the model will not output the same value every time for a given input. This non-determinism will result in stochastic predictions. The knowledge acquired by DNNs is limited to the data that these models are trained on during the learning process. There are three learning processes, supervised learning, unsupervised learning, and reinforcement learning. The model will learn from training examples and the target labels in supervised learning. In contrast, the model tries to find patterns or clusters in the data and categorize them in unsupervised learning. Reinforcement learning is a technique where the agent performs an action, and there exists a reward system that will help the agent learn.

In this R&D, an image classification problem is considered, and a Convolution Neural Network (CNN) is used as a LEC. CNN is a deep neural network used to solve problems with image data. Image classification is a supervised learning problem.

3

State of the Art

This Section describes the state-of-the-art techniques and practices available to improve the safety and reliability of learning enabled components. The concept of uncertainty in Deep Neural Networks (DNNs) and runtime monitoring techniques were briefly discussed.

3.1 Safety Assurance of Deep Neural Networks

DNNs have a wide range of applications such as autonomous systems [23], control systems [24], image classification [25] and understanding, Natural Language Processing (NLP) [26]. DNNs are mathematical models that map inputs to outputs, and they contain many intermediate layers, and these intermediate layers are mathematical functions. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are the popular DNNs types that have recently gained much attention. Surveys on DNNs show that depending on the type of application and data involved in the learning process, different DNN architectures were proposed by researchers in this field. In this R&D, as proof of concept for Learning Enabled Components (LECs), CNNs were used. In the field of computer vision, CNNs have shown some promising results in object detection and classification tasks. Throughout this report, when we mention LECs, we are mostly talking about the CNNs for the object classification task. The development phase, verification, and validation metrics are relevant to CNNs for the object classification task in later sections of this report.

3.1.1 Development of DNNs

Schwalbe's [27] work on safety assurance for the deep learning model highlighted the development phase's importance in certifying DNNs. To ensure that a LEC is fault-free, some design decisions and best practices are to be followed, which are like building blocks for safety cases. These development practices are categorized and explained in this section.

Experience-based design

The design decision is preferred to be taken by the experts in the field, and all the general steps involved in building a pipeline are adequately reasoned. This reasoning explains why certain design choices were made and should elaborate on these decisions' limitations. Some critical design choices to be made are:

- Training purpose
- Feature selection for training
- Model selection
- Data augmentation techniques
- Micro design decisions for choosing: optimizers, loss functions, activation functions
- Hyperparameter tuning
- Early stopping criteria for training

Similar work using these design decisions for reinforcement learning problems using neural networks done by Amodei [28] is a good example proving these choices' importance.

Uncertainty estimation

DNN models will only output predictions, but not the confidence of those predictions. Confidence is the variance in the predictions. McAllister's [29] work on the safety of autonomous vehicles suggests that propagation of uncertainty of each component of the model and monitoring that accumulated uncertainty will help assure the safety of the model predictions. It is better to know the model's confidence in the predictions so that some mitigating actions can be taken for lesser confidence. Some prior work in this domain that deals with the method of quantifying the uncertainty and using that for confidence are: Bayesian neural networks [30], McAllister [29] work where they find distributions for weights and biases, methods that use specialized loss functions that consider uncertainty, neuron dropout, and neuron weight decaying techniques. These methods prove to increase even adversarial robustness in some cases.

Enhancing Robustness

In DNNs, the term robustness refers to the change in model output for small input data changes. The real-world data contain a lot of uncertainty; hence robustness is one critical aspect of ensuring safety and reliability. The prior work on DNN's robustness problem shows two common ways to increase the model's robustness. Since DNNs are data-driven models, these methods deal with regularizing the training data.

- Adding adversarial examples to training data [31].
- Removing features that are not robust in training data [32].

3.1.2 Verification

Verification of a DNN model is done during the design and training time. The process of testing whether the model met all of the formal requirements specified before the model development phase is known as verification. This is the stage where we check whether the model satisfies all the rules; it can be done only with search algorithms and solvers. Some of the popular tools according to the literature that are used to verify DNN models are:

- Search algorithms - VERIVIS [33]
- Solvers - Reluplex [34], Satisfiability Modulo Theory (SMT) [35]
- Estimation of output bounds - DeepGo [36], ReluVal [37]

3.1.3 Validation

Validation of DNNs is performed after model training. In general validation, step tells us whether the model achieved our expected goal. In addition, it also helps in identifying the missing rules that may be necessary for achieving our expectations. As validation helps identify the missing requirements, it is a crucial step to be considered to increase the model's safety and reliability. According to Schwalbe's [27] work on safety cases for LECs, the validation methods are categorized as:

Data validation

Both testing and training data should wrap broad information about the input space. The more the model knows about the working environment, the better the reliability of predictions. Methods for data validation as per the literature are:

- Counter example generation [38]
- Test case generation [31], [39]

Qualitative analysis

The quality of a DNN model can be assessed by observing how the model's performance is affected by the change in "attention" features. Attention features are the features that contribute most to the output decision. Some prior work related to this theory:

- Model specification methods - signal back-tracking methods [40]
- Feature visualization techniques - Dream distill [41]
- Feature modification techniques - RISE [42], LIME [43]
- Feature explanation techniques - textual explanations [44]

Quantitative analysis

Due to the black-box nature of these DNNs, it is tough to quantify the insights even though these are mathematical models. This could be the reason for inadequate prior work on this problem. But some indirect approaches support quantitative analysis.

- Extraction of rules - inductive logic programming [45], validity interval analysis [46]
- Performance analysis of sub-tasks [47]
- Network structure analysis - Net2Vec [48], TCAV [49]

3.1.4 Uncertainty in DNNs

In the context of DNNs, uncertainty refers to a lack of confidence in input data or decisions. According to [50], *"uncertainty is a general concept that reflects our lack of sureness about something or someone, ranging from just short of complete sureness to an almost complete lack of conviction about an outcome."* Uncertainty in the DNNs can arise for many reasons, but it can be mainly categorized into aleatoric and epistemic uncertainty as per the literature on uncertainty [51], [52].

Aleatoric uncertainty

This is the kind of uncertainty that cannot be reduced and is the natural property of any physical word or system that is modeled. DNNs are non-deterministic models. Therefore, randomness is an inherent nature of these models. This uncertainty cannot be reduced by providing more information to the model. Aleatoric uncertainty is referred to with many names across many research domains. Some of those names are external uncertainty [53], stochastic uncertainty [54], natural uncertainty, random uncertainty [55], objective uncertainty [56], irreducible uncertainty, real-world uncertainty, primary uncertainty, fundamental uncertainty, inherent uncertainty.

Epistemic uncertainty

This uncertainty occurs because the model lacks the required knowledge about the physical problem it is dealing with. This uncertainty can be reduced by providing more information to the model about the environment. Like aleatoric uncertainty, epistemic uncertainty is also referred to with many names across the research domains. Those names are subjective uncertainty, informative uncertainty,

incompleteness uncertainty, internal uncertainty, knowledge uncertainty, secondary uncertainty, and functional uncertainty [57].

This uncertainty in DNN models makes its predictions non-reliable. When these uncertain LECs are integrated with the autonomous system,s they perform many system-level functions like perception, navigation, and localization. Figure 3.1 shows a good summary of uncertainty types and the theories to handle those uncertainties.

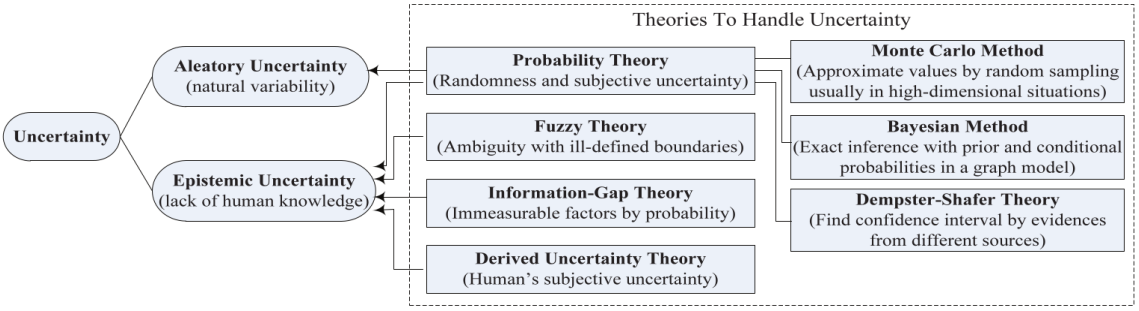


Figure 3.1: Uncertainty Theories. Image credit: [58]

3.2 Safety and Reliability Requirements for DNNs

DNNs are data-driven models, which possibly makes them non-robust and is a reason for their black-box nature. This black-back nature limits the domain knowledge about the models and makes it hard to formulate the requirements. Satisfying the safety requirements increases the overall reliability of the model. G. Schwalbe [27] categorized these requirements depending on the data representation, and the requirements discussed in this section are closely related to their work.

Scenario coverage and safety related performance

During the prediction phase, for every input test sample, there are some metrics that will explain the overall coverage of the features in the image. Chih-Hong Cheng [59] proposed some metrics for the scenario coverage.

- Scenario coverage metric
- Neuron k-activation metric
- Neuron activation pattern metric
- Adversarial confidence loss metric
- Scenario-based performance degradation metric
- Interpretation precision metric

- Occlusion sensitivity covering metric
- Weighted accuracy/confusion metric

Occlusion in the input image data should not affect the model, as this compromises the safety of any autonomous system.

Algorithm Robustness

The trained DNN model should be robust to adversarial attacks [60] and identify out-of-distribution [61] test samples while predicting. Satisfiability Modulo Theories (SMT) solvers [34] are used to satisfy the algorithm's robustness.

Fault tolerance

Runtime monitoring techniques will help increase the fault tolerance, which can be best explained in the case of autonomous road vehicles. Functional safety standard [62] ISO 26262 describes more about these techniques and model redundancy methods.

Plausibility

One of the most critical requirements is the model's quality. The DNN model should not be sensitive to any intermediate steps. If we see it from a software perspective, updating the model to operate in a new environment should not affect the overall robustness of the model. In the case of autonomous vehicles, the requirements should also include the general physical rules like considering the gravity, traction, and translation invariance [63]. Achieving quality also includes knowledge about the algorithm's or model's limitations and faults [64].

3.3 Runtime Monitoring or Verification Techniques for DNNs

Runtime monitors find the abnormal behavior of any software functionality by identifying the deviation between the performed and expected actions. This deviation is calculated with the help of predefined system states or by observing the behavior traces from previous iterations. There are different runtime monitoring and verification techniques depending on the application and software functionality. As mentioned in the earlier sections, studies show that validating the safety or reliability of a LEC is possible only by verification in runtime. Fault Detection Isolation and Recovery [65] is a field of study which is a part of safety-critical systems like autonomous systems and avionics. LECs are the components that are part of autonomous systems that make them non-reliable. The monitors discussed in this section are limited only to the LECs and safety-critical systems. These runtime monitors are categorized based on the type of specification and correctness information and are closely related to the work of Chih-hong [66].

3.3.1 Information Correctness Monitoring

Neural Networks (NN) trained with labeled data will have a direct correctness specification. In a supervised classification problem, the correctness of the LEC is measured with the metrics like accuracy, precision, recall, F1 score. It is hard to measure the model's correctness directly without labeled data, so indirect correctness measurement techniques are used. In indirect correctness measurement techniques, the distance of the input data from the training data is usually measured. This distance helps identify whether this is an out of distribution sample and if so, it is recognized as an unknown class. Finding this dataset shift is used as an indirect correctness measurement technique, and many researchers used this concept to develop new monitoring techniques.

Monitoring the Neural network computations

In this research problem, researchers developed techniques to find the deviation of the network predictions from the target values. Some popular methods are neuron activation patterns monitoring processes [67], where for each training example, the neurons activated during the training are monitored to create binary decision diagrams (BDD). During the prediction stage, if the activation of the input does not match any of the BDDs, the monitor finds it a problem. Zero-shot learning [68] is another method where the features in the image are monitored by grouping them as combinations. For instance, in a traffic signal sign, the number, circles/polygons, the color of the signal are made as a combination, and this group is remembered for each training example. The input image features are extracted and compared with the combinations learned before testing. If a similar feature combination is not present in the image, it is considered a new class.

Class distribution monitoring

Neural networks, by default, have no process to find 'out-of-distribution' data, and it will predict one class even for a random input with higher confidence. One common technique to monitor this problem is the change-point-detection technique [69]. In this technique, k data points are compared with 2k other points in the data set. The key idea is to find the data set shift that will lead to unreliable predictions.

4

Methodology

This Chapter explain about the experimental setup, building a deep learning pipeline, runtime monitors and metrics considered for evaluating the model.

4.1 Experimental Design

To test the objective of RQ5, a CNN model is developed following the design steps discussed in section 3.1.1, and its performance is evaluated. The CNN model developed is trained and tested for an image classification problem, and inferences will be drawn based on the performance metrics. The datasets used for this experiment are discussed in section 4.1.1, and the evaluation metrics used for the image classification problem are discussed in section 4.1.4. The different phases of a DNN pipeline development followed to test the objective of RQ5 are described in section 4.1.2. The objective for RQ6 is tested by using the runtime monitors along with the models trained to address the RQ5. The runtime monitors used to test the objective of RQ6 are described along with pseudocode in section 4.1.3.

4.1.1 Datasets

The datasets used in this R&D are MNIST digit classification dataset [70] and RoboCup@work dataset. Both the datasets contain multiple classes and are used for image classification task.

MNIST dataset

The MNIST dataset is a database of images with handwrittendigits and is a very popular dataset used for testing Machine Learning (ML) and DNN classifiers. This dataset contains ten classes representing the handwritten digits from 0 to 9. Each image is a single-channel grayscale image of size 28x28 pixels. The sample images from the data set are shown in the Figure 4.1. This dataset contains around 70,000 images, among which 60,000 images were used for training the model and 10,000 images were used for testing the model.

Robocup dataset

This dataset contains fifteen classes referring to different tools used in the Robocup competition. Each image in this dataset is a three-channel RGB image resized to a size of 64x64 pixels. This dataset contains

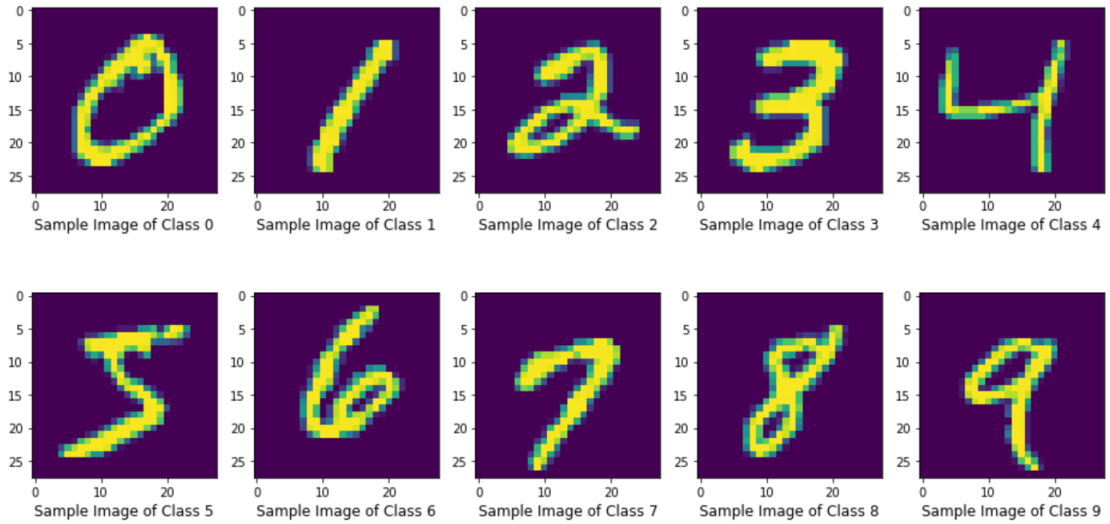


Figure 4.1: MNIST dataset sample images

the following classes: Axis, Bearing, Bearing_box, Container_box_red, Container_box_blue, Distance_tube, F20_20_B, F20_20_G, M20, M20_100, M30, Motor, R20, S40_40_B, S40_40_G.

4.1.2 Pipeline

This section explains the various stages of model development, according to the best practices mentioned in section 3.1.1.

Data preprocessing

In this stage, all the images are reshaped and normalized. By default, all the images in the MNIST dataset are of size 28x28 pixels. The images in the Robotcup@work data set are of different sizes, so all the images are resized on 64X64 pixels. The reason for choosing 64x64 pixels as reshape size for the RoboCup dataset is to reduce the training time, and this size can also retain most of the information about the features in the image. The dataset is split into training, testing, and validation sets in the ratio of 70%, 20%, and 10%, respectively. These experiments were run using PyTorch framework, and to speed up the training process, the datatype used for running these experiments is tensor. These models were trained on GPU, so the tensor variables are converted to GPU using the PyTorch library functions. Both the datasets contain a good number of images, and so there is no class imbalance problem.

Data augmentation

Data augmentation is the technique that involves applying affine transformations, color jitterings, adding noise, blurring to the images. Data augmentation helps in designing a generalized model, which will make the model more robust for noisy images. For the MNIST dataset, the data augmentation technique

used is only zooming of the images at a different level. The other affine transformation, random flipping techniques, are not used because it affects the training performance in the case of the MNIST dataset. For example, the horizontal flipping and vertical flipping of digit 3 and 9 respectively will appear as a completely different number, introducing uncertainty in the model. For the Robocup dataset, different affine transformations, flipping, cropping, blurring and also zooming of images is performed.

Model selection

The LECs used as proof of concept in this R&D are CNNs. After the success of AlexNet [71], many researchers started looking into methods that solve the image classification problem with the DNN approach. The proposed CNN architectures are tested on the ImageNet dataset, containing 1000 classes, with labeled ground truth data. Many computer vision researchers proposed different DNN architectures, which reduced the mean error of this problem over the period. ResNet [72], MobileNet [73], and squeezeNet [74] models are selected for testing the runtime monitors in this R&D work.

***ResNet*:**
The CNN architectures proposed before residual networks have many layers, and because of the vanishing gradients problem, those architectures failed to achieve lower classification errors. Though residual networks have many layers, they follow a skip connection approach that passes the activation values of the current layer to the subsequent layers and also to another layer, skipping some layers in between. This technique helped deal with the vanishing gradient problem, which significantly reduced the classification error for the ImageNet problem. The ResNet architecture is shown in Figure 4.2. In this R&D work for solving the MNIST digit classification problem, the ResNet18 model was used, and for the RoboCup@work tool classification problem RestNet50 model was used.

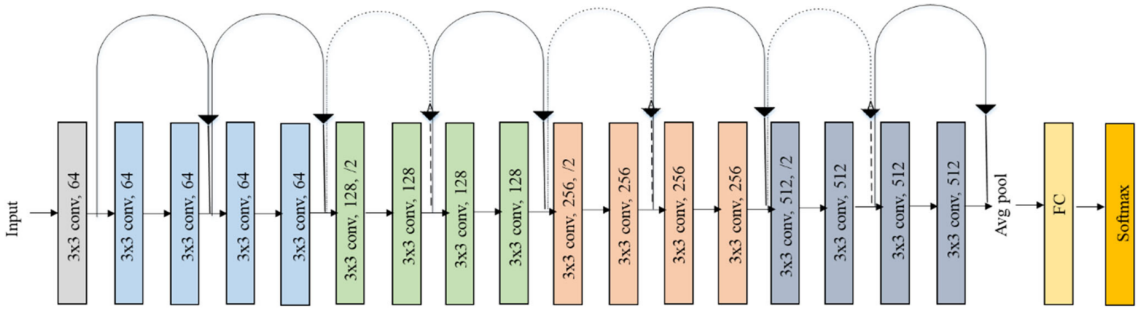


Figure 4.2: ResNet architecture. Image credit: [72]

***MobileNet*:**
MobileNet architecture is lightweight, takes less computation time, and was intentionally designed to work on object detection and classification tasks on mobile devices. Two architectures are proposed over the period. The former is the theMobileNetV1, and the latter is the MobileNetV2. MobileNetV2

takes even lesser time than MobileNetV1. Figure 4.3 shows the MobileNet architectures. In this R&D work for solving the MNIST digit classification problem,and RoboCup@worktool classification problem MobileNetV2 model was used.

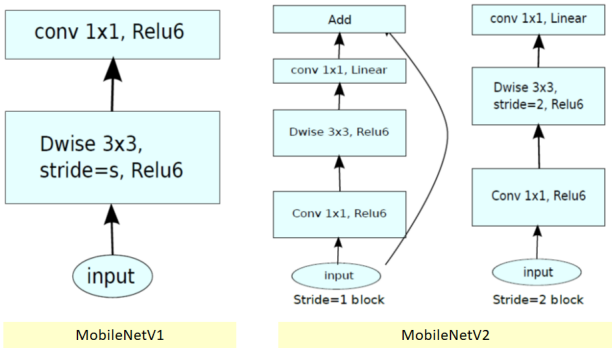


Figure 4.3: MobileNet architecture. Image credit: [73]

***SqueezeNet*:**
SqueezeNet is a smaller CNN model, which is computationally lightweight to deploy on devices with limited memory resources. The size of this architecture makes exporting much easier from the cloud to the device after training the new model. The figure 4.4 shows the architecture for SqueezeNet. In this RD work for solving the MNIST digit classification problem,and RoboCup@worktool classification problem SqueezeNet 1.0 model was used.

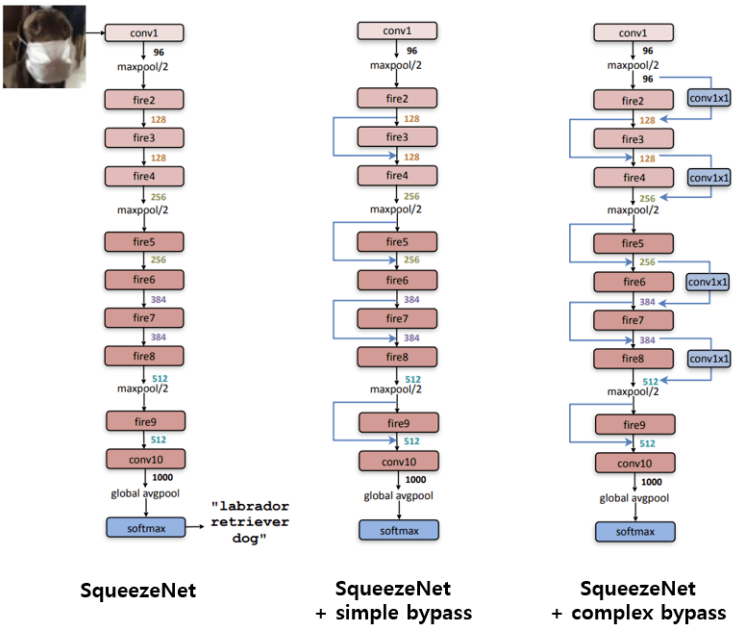


Figure 4.4: SqueezeNet architecture. Image credit: [74]

Hyperparameter tuning

Hyperparameter tuning is performed to determine the best batch size and the learning rate for training the model. Here the right batch size increases the training speed, and the learning rate helps find the best local minimum. The best initial learning rate is found using the cyclic learning rate technique proposed by Leslie [75]. For all the experiments performed in this R&D, Adam optimizer [76] and cross-entropy loss were used as optimizer and loss functions.

Model training

After completing the preprocessing steps and selecting the hyperparameters for the network, the model is trained for some epochs. PyTorch early stopping function is used as a callback for training. This early stopping function will stop the training when the model starts overfitting. When the validation loss keeps increasing and the training loss keeps decreasing after some training epochs, then it is assumed that the model is overfitting. This call-back function will stop the training in such cases.

Saving and loading the models

Model saving operation is performed using PyTorch save model functions. This function saves all the model state parameters, hyperparameters, and weights. The model is saved after each epoch to use the trained models for prediction or restart training from the same checkpoint. The saved models can loaded again using the PyTorch load model function to continue the training or for prediction.

Model testing

This is the stage where the model is tested, and its performance is evaluated. The trained model is reloaded using the PyTorch load model function, and the model is set to evaluation mode. This feature is specific to PyTorch, wherein features like dropout are turned off in the evaluation mode. This is necessary because we do not want the model to drop any neuron connections between the layers. The performance of the model is evaluated using the evaluation metrics described in the section 4.1.5.

Data logging

The hyperparameters after the tuning, model evaluation metrics during the training and after testing are logged for further analysis. Data logging at various stages of the model development and testing phase helps explain the model's bad and good performance. In this R&D, "Neptune.ai" data logger is used, which has good features for model comparison.

4.1.3 Runtime Monitors

This section describes the functioning of three runtime monitors developed to increase the reliability of the LECs considered for proving the objective of RQ6.

Evidential zoom monitor

The core implementation of this monitor works on the evidential uncertainty method described in the research paper titled "Evidential Deep Learning to Quantify Classification Uncertainty" [4]. The concept of evidential uncertainty helps identify the model's predictions' confidence. This means that the predictions' standard deviation will indirectly tell the confidence in the predictions. The higher the standard deviation, the lower is the prediction confidence. A pseudo-code of the evidential zoom monitor working is explained below in Algorithm 1. First, the input image is zoomed into different zoom levels, where these zoom levels or image sizes are sampled from a Dirichlet distribution. These images are passed through the model for predictions. The mean uncertainty for all the predictions of the images at different zoom levels is used as confidence in the model's prediction. If the confidence value is greater or less than the threshold values, respective contingency action is taken.

Algorithm 1 Evidential zoom monitor

image sizes randomly sampled from Dirichlet distribution and to compute different zoom levels
for *image in zoomed_imgs* **do**
    $logits \leftarrow \ model \leftarrow image$
    $evidence \leftarrow \ relu\_evidence \leftarrow logits$
    $alpha = \ evidence \ + \ 1$
    $uncertainty = \frac{num\_classes}{\sum alpha}$
    $prediction = max \leftarrow logits$
    $probability = \frac{alpha}{\sum alpha}$
    $mean\_uncertainty = \frac{uncertainty}{\sum uncertainty}$
    if $mean\_uncertainty > Threshold$ **then**
        $contingency\ action\ for\ uncertain\ predictions$
    end if
    monitor_prediction = $out\ of\ distribution\ image$
end for

Max monitor

This monitor will zoom the input image 'X_i' into different zoom levels, say X_n (where n is the number of zoomed images), and pass those zoomed images one by one into the pre-trained models for predictions. This will result in obtaining y_n predictions, out of which the maximum number of predictions with the same predicted labels is the monitor output. A pseudo-code of the max monitor working is explained below in Algorithm 2.

Algorithm 2 Max monitor

image sizes randomly sampled from Dirichlet distribution and to compute different zoom levels
for X_i *in* X_n **do**
    $logits \leftarrow \ model \leftarrow X_i$
    $y_n = max \leftarrow logits$
    $model\_predictions = append\ y_n$
end for
$monitor\_prediction = \max \leftarrow model\_predictions$

Average probability monitor

This monitor will also zoom the input image 'X_i' into different zoom levels, say X_n (where n is the number of zoomed images), and pass those zoomed images one by one into the pre-trained model. Here, the mean of final layer activation values for 'n' zoomed images are taken, and the maximum value of arguments is returned as the monitor prediction. A pseudo-code of the average probability monitor working is explained below in Algorithm 3.

Algorithm 3 Average probability monitor

image sizes randomly sampled from Dirichlet distribution and to compute different zoom levels
for X_i *in* X_n **do**
    $logits_i \leftarrow \ model \leftarrow X_i$
    $model\_logits = append\ logits_i$
end for
$monitor\_prediction = max \leftarrow \frac{model\_logits}{\sum model\_logits}$

4.1.4 Evaluation Metrics

As mentioned in the section, the experiments conducted in this R&D are on a classification problem. MNIST data set is used for this experimentation, a multi-class data set. According to [77], the following classification metrics evaluate the model performance. The following metrics were calculated using the Sklearn metrics module in the experiments.

Classification accuracy

Classification accuracy is the ratio of correctly predicted labels with the ground truth labels. The classification accuracy of a multi-class classification problem is determined using the following equation. Here N is the number of training examples, $\hat{y}_i$ is the predicted label, and y_i is the ground truth. The formula for calculating classification accuracy is shown in the Equation 4.1

$$Accuracy(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} 1(\hat{y}_i = y_i) \tag{4.1}$$

Precision

The precision is the ratio of true positives with the sum of true positives (tp) and false positives (fp). Precision tells us the ability of the model not to label negative samples as positive samples. The precision values leis between 0 and 1, where precision value close to 1 represent the model is best, and the precision value close to 0 represent the model is worst. The formula for calculating precision is shown in the Equation 4.2

$$Precision = \frac{tp}{tp + fp} \tag{4.2}$$

Recall

The recall is the ratio of true positives with the sum of true positives (tp) and false negatives (fn). Recall tells us the ability of the model to predict all the positive samples. The recall values lie in the range of 0 and 1, where the value close to 0 represents the worst, and the values close to 1 represent the best. The formula for calculating recall is shown in the Equation 4.3

$$Recall = \frac{tp}{tp + fn} \tag{4.3}$$

F1 score

F1-score is the harmonic mean of precision and recall. F1-score tells us how well the model classifies the positive samples. Since it is a harmonic mean, both metrics will be given equal weightage in determining the F1-score. The F1-score values lie between 0 and 1, and the best value is close to 1, and the worst is close to 0. The formula for calculating F1 score is shown in the Equation 4.4

$$F1\ score = \frac{precision * recall}{precision + recall} \tag{4.4}$$

5

Results

In this section, the hypothesis for the research questions mentioned in section 3.1.1 will be addressed by discussing the performance evaluation of Learning Enabled Components (LECs).

5.1 Reliability Improvement by Design Choices

In this section, the hypothesis for RQ5 is discussed. As proof of concept, a Convolutional Neural Network (CNN) model is developed by following all the design choices mentioned in section 3.1.1. The model performance is tested with the evaluation metrics, and the inference is discussed to support the hypothesis. The experimentation setup is as follows:

- Datasets used: MNIST dataset, RoboCup@work dataset
- Network architectures: ResNet18, ResNet50, MobileNetV2, SqueezeNet
- Evaluation metrics: Classification accuracy, precision, recall, F1 score

5.1.1 Objective

This experiment aims to assess whether the model's performance and reliability increase after developing the model using the best design choices supported by the literature.

5.1.2 Hypothesis

The model's performance and reliability will increase when developed following the best design choices.

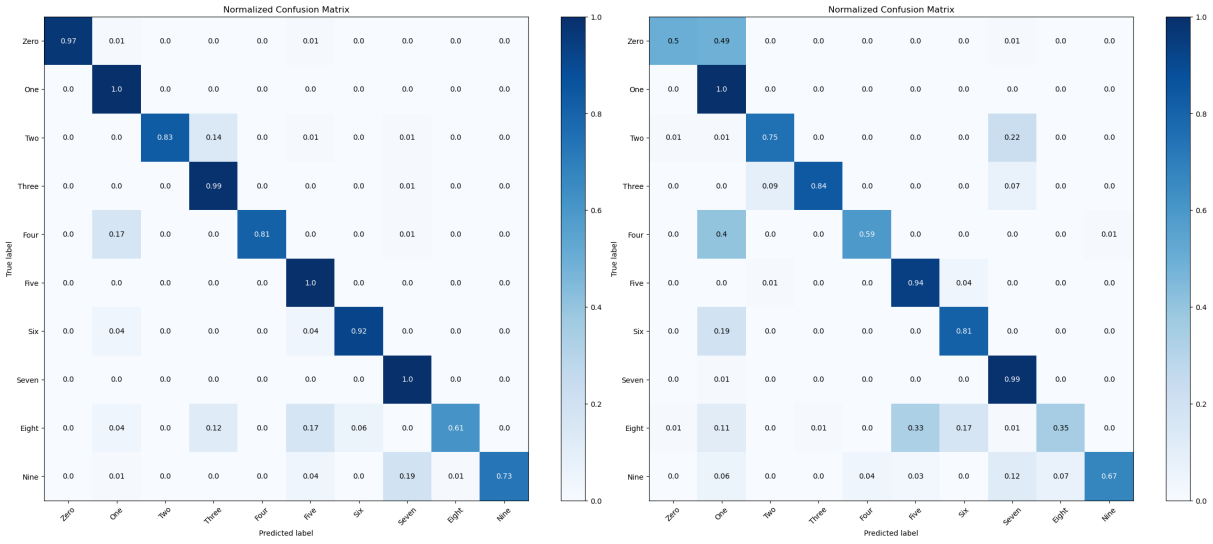
5.1.3 Observation

The experiment results for the MNIST dataset, with three models RestNet18, MobileNetV2 and SqueezeNet are shown in the Table 5.1. The confusion matrix for the ResNet18 and MobileNetV2 models developed before following design choices are shown in the Figures 5.1a and 5.1b. The results show that following the design choices, especially performing hyperparameter tuning, increased the model's

performance. In the case of ResNet18 and MobileNetV2, the early stopping function helped stop the training before the model started overfitting. SuqeezeNet model did not converge initially because of using a higher initial learning rate. Performing hyperparameter tuning helped in this case to find the best initial learning rate.

Models	**Followed design choices**	**Accuracy**	**Precision**	**Recall**	**F1 score**
ResNet18	No	90.49	0.91	0.89	0.89
MobileNetV2	No	79.43	0.81	0.75	0.74
SqueezeNet	No	Not converged	Not converged	Not converged	Not converged
ResNet18	Yes	97.42	0.97	0.97	0.97
MobileNetV2	Yes	97.16	0.97	0.97	0.97
SqueezeNet	Yes	92.72	0.93	0.92	0.92

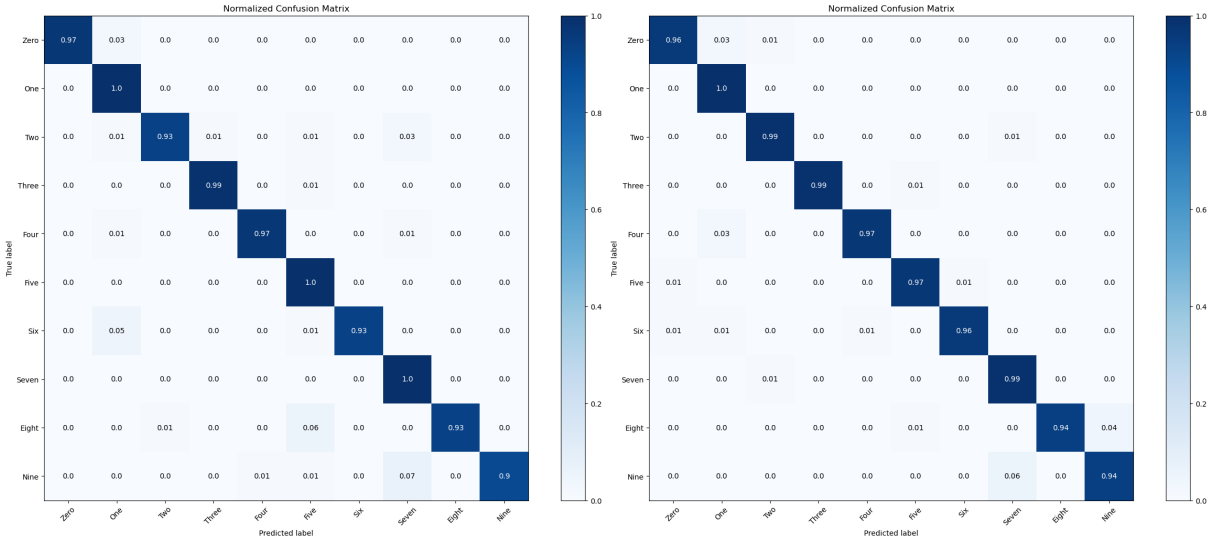
Table 5.1: MNIST dataset performance before and after following best design choices



(a) Confusion matrix for ResNet18 model

(b) Confusion matrix for MobileNetV2

Figure 5.1: Confusion matrix before following design choices



(a) Confusion matrix for ResNet18 model (b) Confusion matrix for MobileNetV2 model

Figure 5.2: Confusion matrix after following design choices

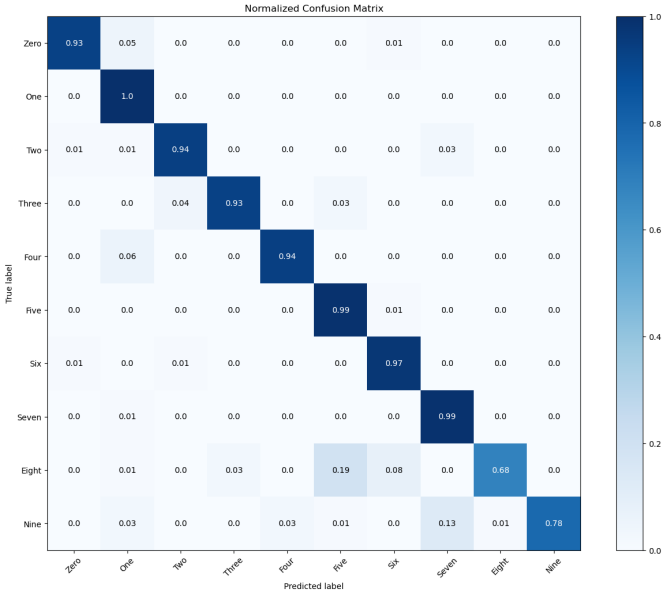


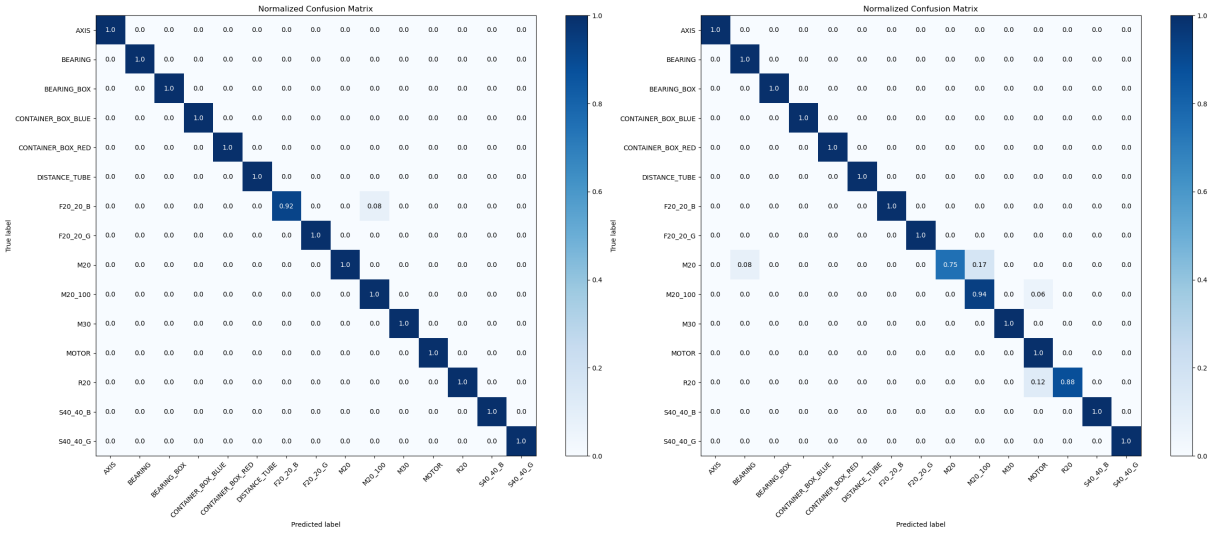
Figure 5.3: Confusion matrix for MNIST-squeezeNet model after following design choices

The experiment results for the RoboCup@work dataset, with three models RestNet50, MobileNetV2 and SqueezeNet are shown in the Table 5.2. The confusion matrix for the ResNet50, MobileNetV2, SqueezeNet models developed considering the design choices are shown in the Figures 5.4a, 5.4b, and 5.5 respectively. Here all three models failed to converge because the initial learning rates are higher, and the

hyperparameter tuning helped in this case.

Models	Followed design choices	Accuracy	Precision	Recall	F1 score
ResNet50	No	Not converged	Not converged	Not converged	Not converged
MobileNetV2	No	Not converged	Not converged	Not converged	Not converged
SqueezeNet	No	Not converged	Not converged	Not converged	Not converged
ResNet50	Yes	99.32	1.0	1.0	1.0
MobileNetV2	Yes	95.31	0.97	0.97	0.97
SqueezeNet	Yes	99.62	0.98	0.98	0.98

Table 5.2: RoboCup@work dataset performance before and after following best design choices



(a) Confusion matrix for ResNet50 model

(b) Confusion matrix for MobileNetV2 model

Figure 5.4: Confusion matrix after following design choices

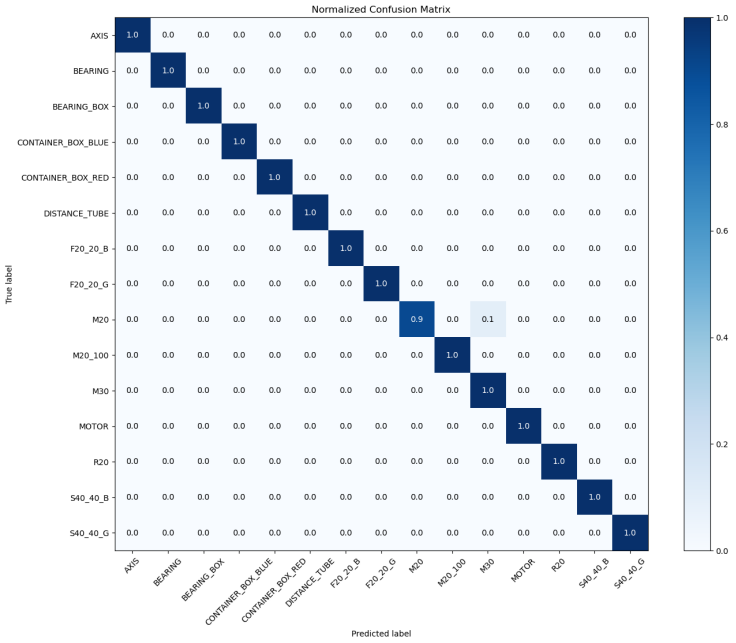


Figure 5.5: Confusion matrix for RoboCup@work-squeezeNet model after following design choices

To test the model's reliability in the case of predicting an out-of-distribution sample, we added two MNIST dataset images as shown in Figure 5.6. We used softmax activation, which is commonly used as the final activation layer before the prediction stage in CNNs, to find the model's confidence in its prediction. Table 5.3 shows different models and their prediction confidence using softmax.

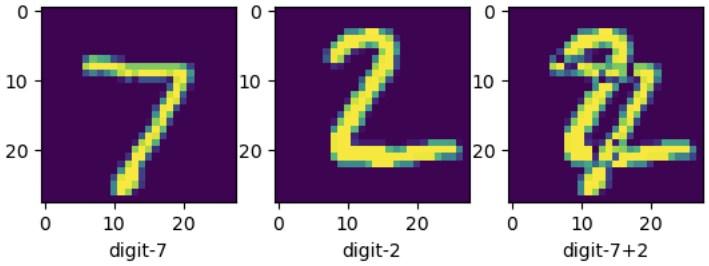


Figure 5.6: Out of distribution example

Model	Prediction	Softmax Confidence
ResNet18	9	1.0
MobileNetV2	4	0.99
SqueezeNet	2	1.0

Table 5.3: Softmax confidence for out-of-distribution sample

5.1.4 Verdict

In this scenario, we can say that the **hypothesis can neither be rejected nor accepted**. We can see that the design practices helped increase the model's prediction accuracy in all three cases, from which we can say reliability improved. But, from the out-of-distribution example, we can understand that it is still highly on-reliable.

5.2 Reliability Improvement using Runtime Monitors

In this section, the hypothesis for RQ6 and RQ7 is discussed. As proof of concept, the CNN models developed by following design choices to prove the previous hypothesis are used, and runtime monitors are added to check the model's reliability. The experimentation setup is as follows:

- Datasets used: MNIST dataset, RoboCup@work dataset
- Network architectures: ResNet18, ResNet50, MobileNetV2, SqueezeNet
- Runtime monitors: Evidential zoom monitor, Max monitor, Average probability monitor
- Evaluation metrics: Classification accuracy, precision, recall, F1 score

5.2.1 Objective

This experiment assesses whether the model's performance and reliability increase by using runtime monitors.

5.2.2 Hypothesis

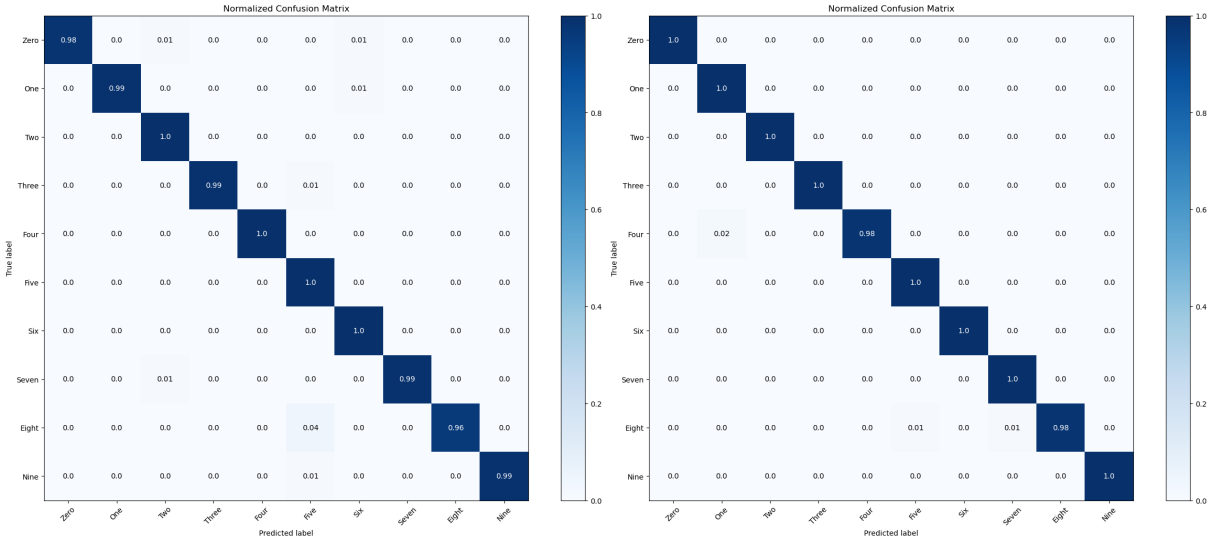
Using runtime monitors will increase the model's performance and reliability.

5.2.3 Observation

The experiment results for the MNIST dataset, with RestNet18 model using evidential-zoom monitor, average-probability monitor, and max monitor, is shown in the Table 5.4. The confusion matrix after using the three monitors is shown in Figures 5.7a, 5.7b, 5.8 respectively.

Monitors	**Accuracy**	**Precision**	**Recall**	**F1-Score**
No-monitor	97.42	0.97	0.97	0.97
Evidential-monitor	98.81	0.97	0.98	0.98
Average-probability monitor	98.80	0.97	0.98	0.98
Max-monitor	98.03	0.98	0.98	0.98

Table 5.4: ResNet18 model predictions with monitors for MNIST dataset



(a) Confusion matrix using evidential zoom monitor (b) Confusion matrix using average probability monitor

Figure 5.7: Confusion matrix for Resnet18 on MNIST dataset

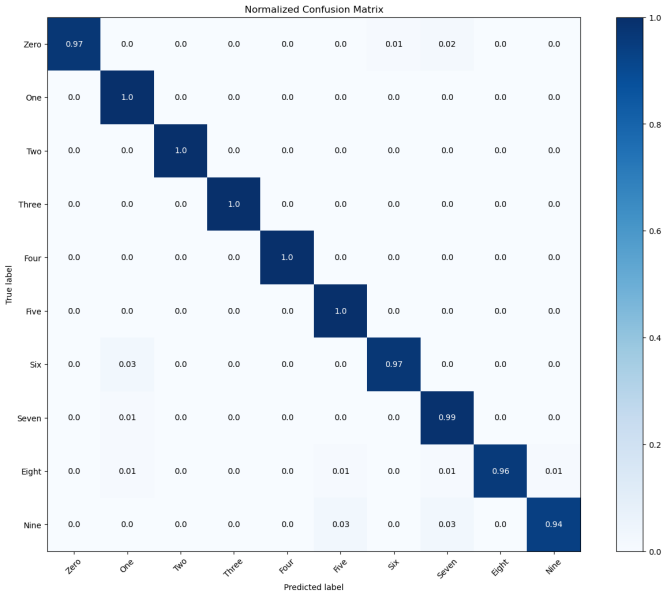
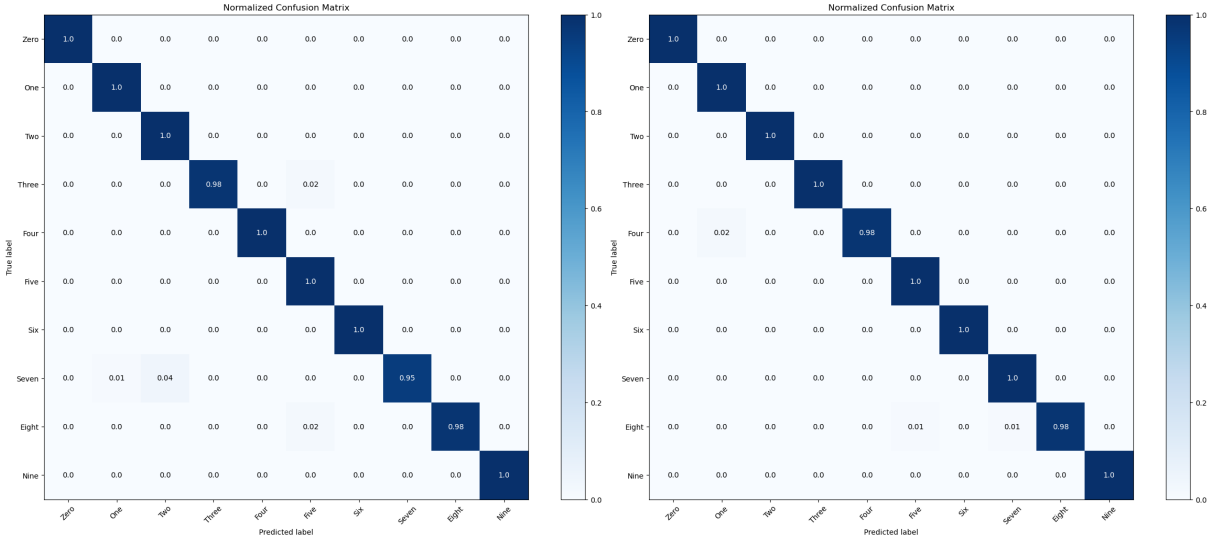


Figure 5.8: Confusion matrix for MNIST-ResNet model using max monitor

The experiment results for the MNIST dataset, with MobileNetV2 model using evidential-zoom monitor, average-probability monitor, and max monitor, is shown in the Table 5.5. The confusion matrix after using the three monitors is shown in Figures 5.9a, 5.9b, 5.10 respectively.

Monitors	Accuracy	Precision	Recall	F1-Score
No-monitor	97.16	0.97	0.97	0.97
Evidential-monitor	98.35	0.98	0.98	0.98
Average-probability monitor	98.55	0.98	0.98	0.98
Max-monitor	98.14	0.98	0.98	0.98

Table 5.5: MobileNetV2 model predictions with monitors for MNIST dataset



(a) Confusion matrix using evidential zoom monitor (b) Confusion matrix using average probability monitor

Figure 5.9: Confusion matrix for MobileNetV2 on MNIST dataset

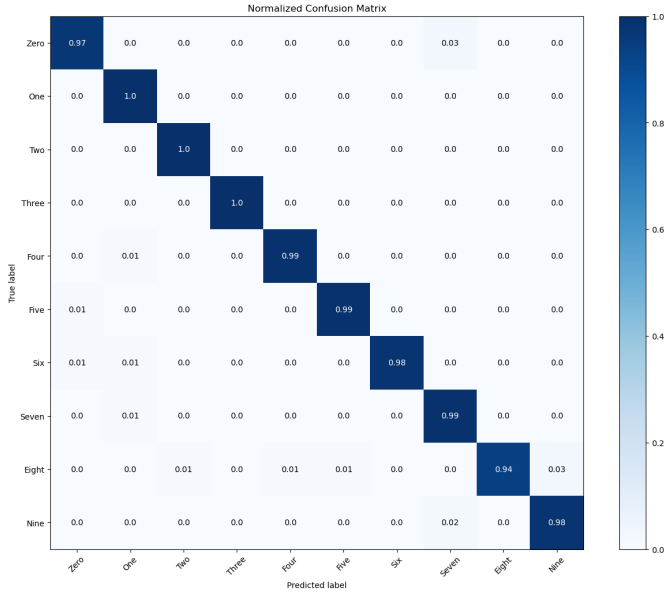
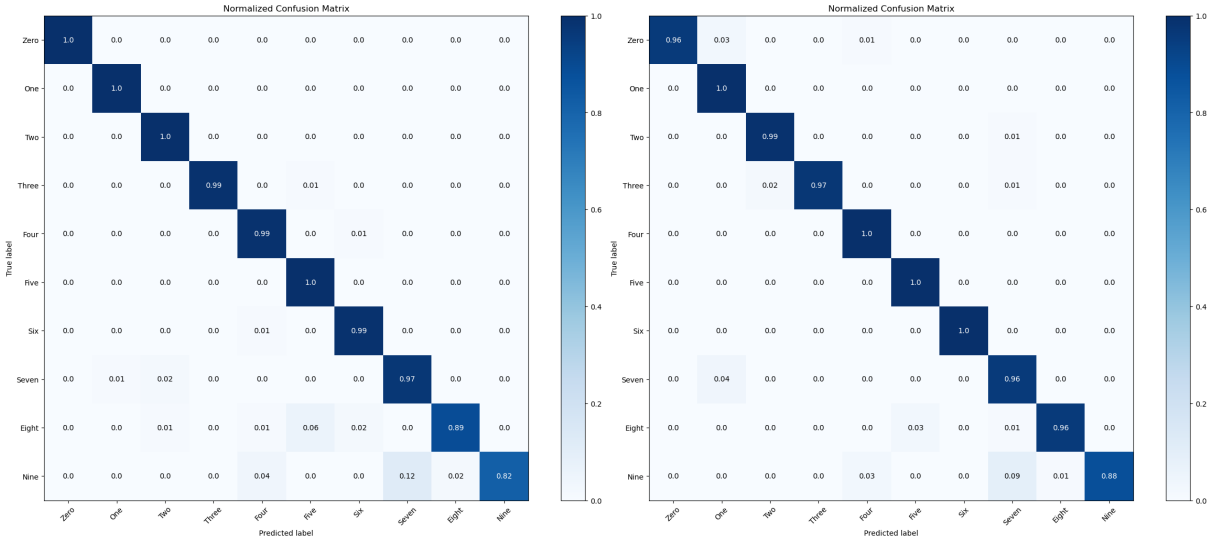


Figure 5.10: Confusion matrix for MNIST-MobileNetV2 model using max monitor

The experiment results for the MNIST dataset, with SqueezeNet model using evidential-zoom monitor, average-probability monitor, and max monitor, is shown in the Table 5.6. The confusion matrix after using the three monitors is shown in Figures 5.11a, 5.11b, 5.12 respectively.

Monitors	**Accuracy**	**Precision**	**Recall**	**F1-Score**
No-monitor	92.72	0.93	0.92	0.92
Evidential-monitor	97.18	0.97	0.96	0.97
Average-probability monitor	95.72	0.95	0.95	0.95
Max-monitor	95.34	0.94	0.95	0.95

Table 5.6: SqueezeNet model predictions with monitors for MNIST dataset



(a) Confusion matrix using evidential zoom monitor (b) Confusion matrix using average probability monitor

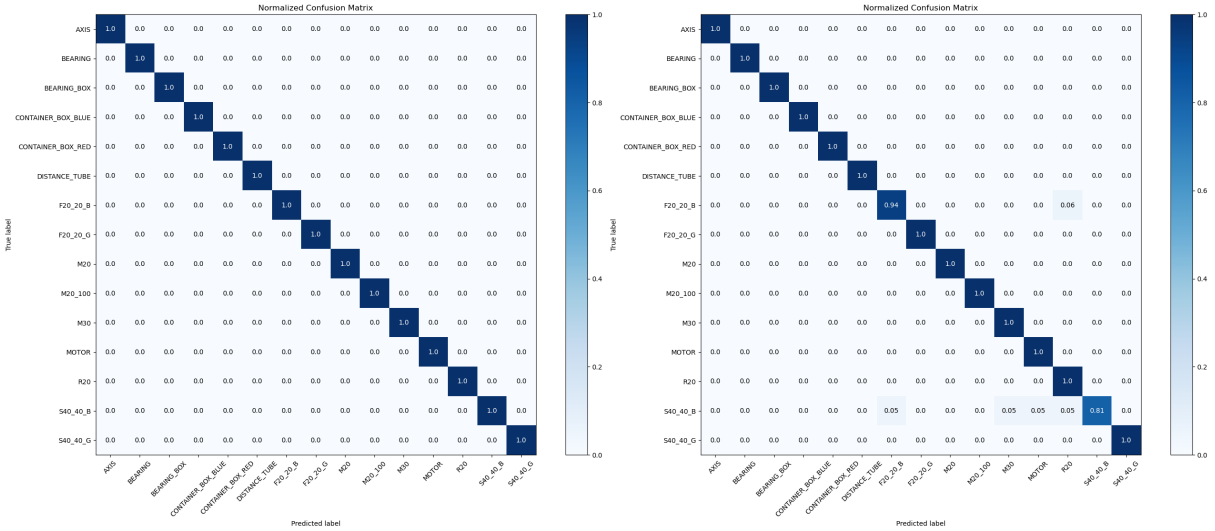
Figure 5.11: Confusion matrix for SqueezeNet on MNIST dataset

Figure 5.12: Confusion matrix for MNIST-SqueezeNet model using max monitor

The experiment results for the RoboCup@Work dataset, with ResNet50 model using evidential-zoom monitor, average-probability monitor, and max monitor, is shown in the Table 5.7. The confusion matrix after using the three monitors is shown in Figures 5.13a, 5.13b, 5.14 respectively.

Monitors	**Accuracy**	**Precision**	**Recall**	**F1-Score**
No-monitor	99.32	1.0	1.0	1.0
Evidential-monitor	1.0	1.0	1.0	1.0
Average-probability monitor	99.94	1.0	1.0	1.0
Max-monitor	99.74	0.99	1.0	1.0

Table 5.7: ResNet50 model predictions with monitors for RoboCup@Work dataset



(a) Confusion matrix using evidential zoom monitor (b) Confusion matrix using average probability monitor

Figure 5.13: Confusion matrix for ResNet50 on RoboCup@Work dataset

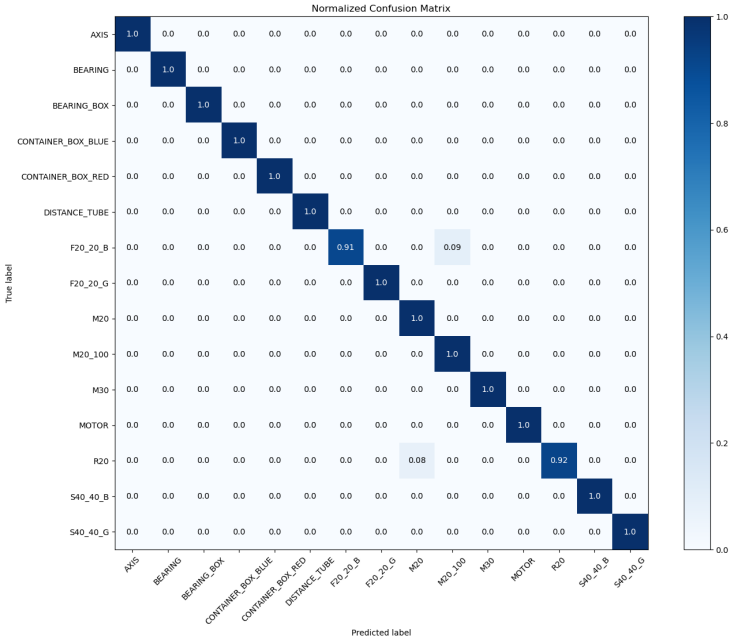
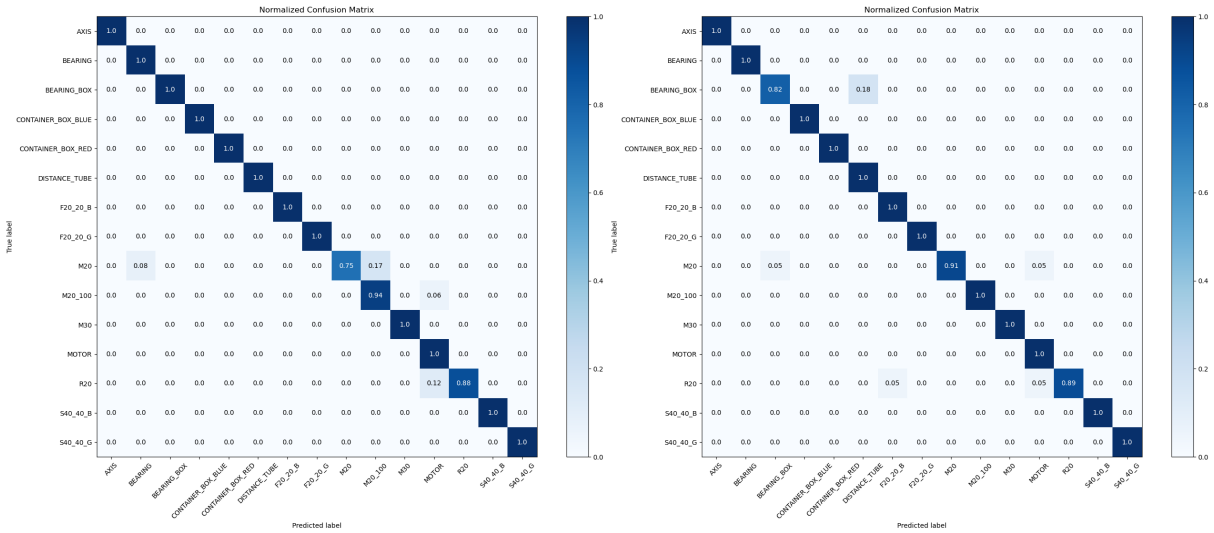


Figure 5.14: Confusion matrix for RoboCup@work-ResNet50 model using max monitor

The experiment results for the RoboCup@Work dataset, with MobileNetV2 model using evidential-zoom monitor, average-probability monitor, and max monitor, is shown in the Table 5.8. The confusion matrix after using the three monitors is shown in Figures 5.15a, 5.15b, 5.16 respectively.

Monitors	Accuracy	Precision	Recall	F1-Score
No-monitor	95.31	0.97	0.97	0.97
Evidential-monitor	99.16	1.0	1.0	1.0
Average-probability monitor	98.81	0.98	0.99	0.99
Max-monitor	98.74	0.99	1.0	1.0

Table 5.8: MobileNetV2 model predictions with monitors for RoboCup@Work dataset



(a) Confusion matrix using evidential zoom monitor (b) Confusion matrix using average probability monitor

Figure 5.15: Confusion matrix for MobileNetV2 on RoboCup@Work dataset

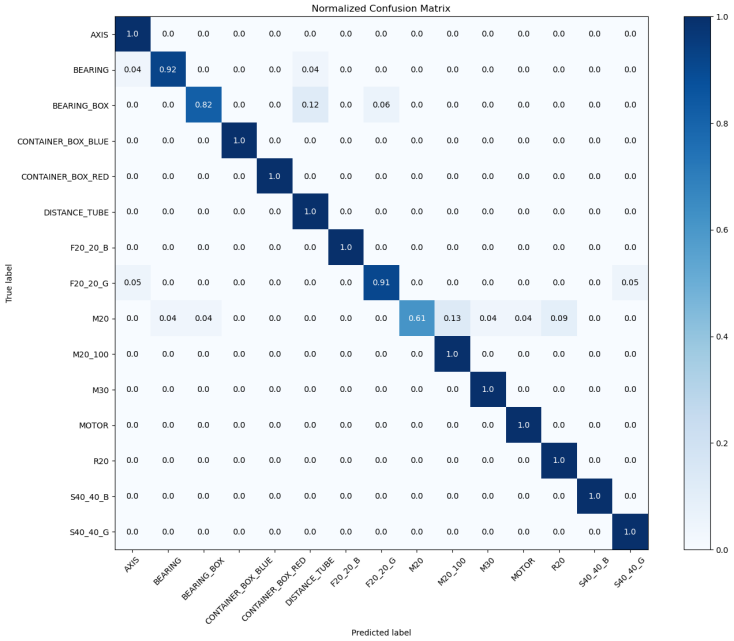
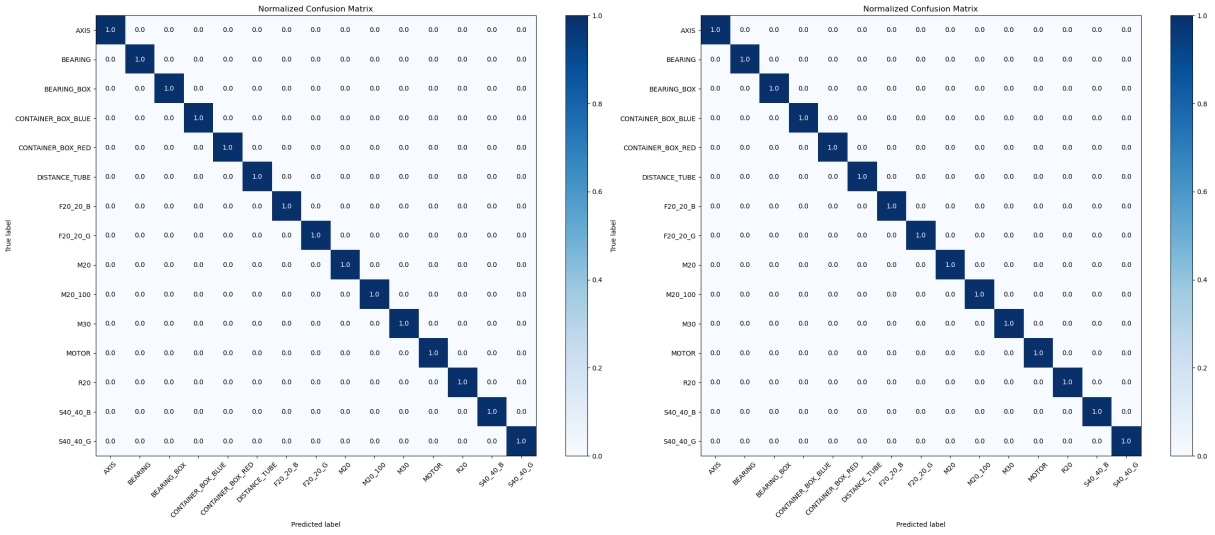


Figure 5.16: Confusion matrix for RoboCup@work-MobileNetV2 model using max monitor

The experiment results for the RoboCup@Work dataset, with SqueezeNet model using evidential-zoom monitor, average-probability monitor, and max monitor, is shown in the Table **??**. The confusion matrix after using the three monitors is shown in Figures 5.17a, 5.17b, 5.18 respectively.

Monitors	**Accuracy**	**Precision**	**Recall**	**F1-Score**
No-monitor	99.62	0.98	0.98	0.98
Evidential-monitor	1.0	0.99	1.0	1.0
Average-probability monitor	1.0	0.99	0.99	0.99
Max-monitor	99.83	0.99	1.0	1.0

Table 5.9: SqueezeNet model predictions with monitors for RoboCup@Work dataset



(a) Confusion matrix using evidential zoom monitor (b) Confusion matrix using average probability monitor

Figure 5.17: Confusion matrix for SqueezeNet on RoboCup@Work dataset

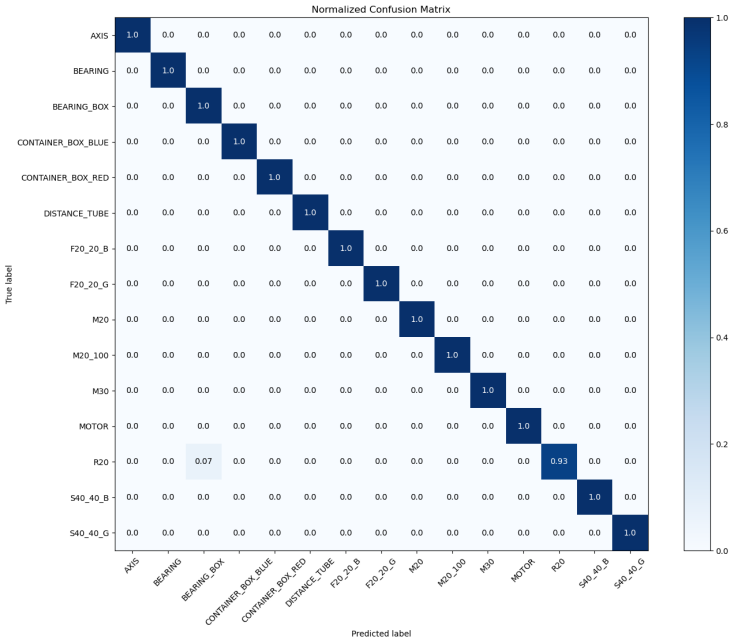


Figure 5.18: Confusion matrix for RoboCup@work-SqueezeNet model using max monitor

Similar to the previous hypothesis testing, we use an out-of-distribution image to test the monitor's reliability. Out of the three monitors we used in this R&D work, only the evidential-zoom monitor can quantify the uncertainty in the predictions. Figure 5.19 shows the out-of-distribution image example. The prediction results are shown in Table 5.10. From these results, we can see the uncertainty value for wrong predictions is very high that is 1.0. This means the monitor identified that the prediction was wrong. We can use this uncertainty value as a threshold to take other mitigating actions.

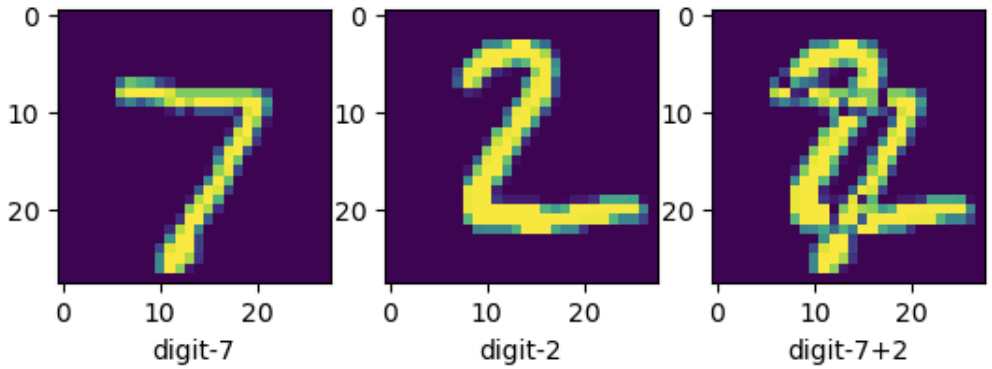


Figure 5.19: Out of distribution example

Model	Prediction	Evidential Monitor Uncertainty
ResNet18	8	1.0
MobileNetV2	4	1.0
SqueezeNet	9	1.0

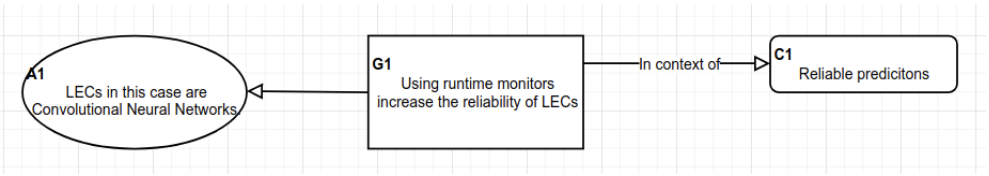
Table 5.10: Evidential-zoom monitor found Uncertainty in predictions

5.2.4 Verdict

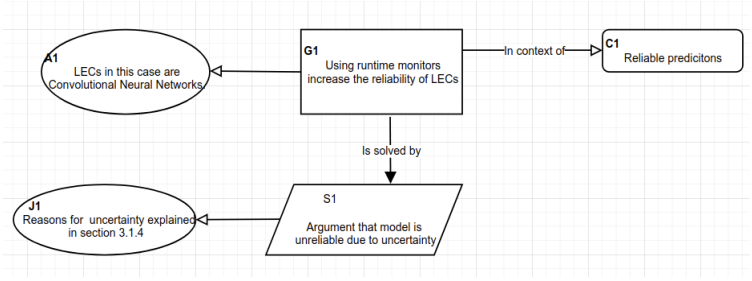
From the results described in section 5.2.3, we can say that **the hypothesis can be accepted**. Here, our runtime monitor, i.e., evidential-zoom monitor, can distinguish between the valid input and the out-of-distribution sample. This means the model's reliability by following best development practices and using runtime monitors increases the reliability compared with the model without any of these.

5.3 Final Argumentation on LECs

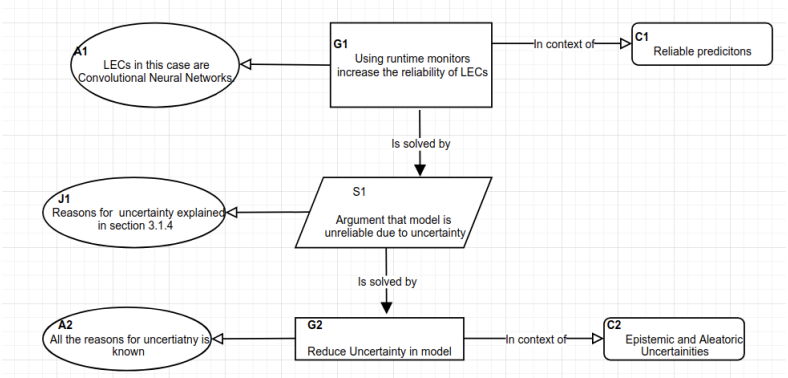
In this section, the step-by-step development of an assurance case for proving that using runtime monitors increases the reliability of LECs is explained. In the first step, the goal of our assurance case is mentioned, and here we are talking in the context of reliable predictions. The assumption is made that we are only talking in the case of CNNs.



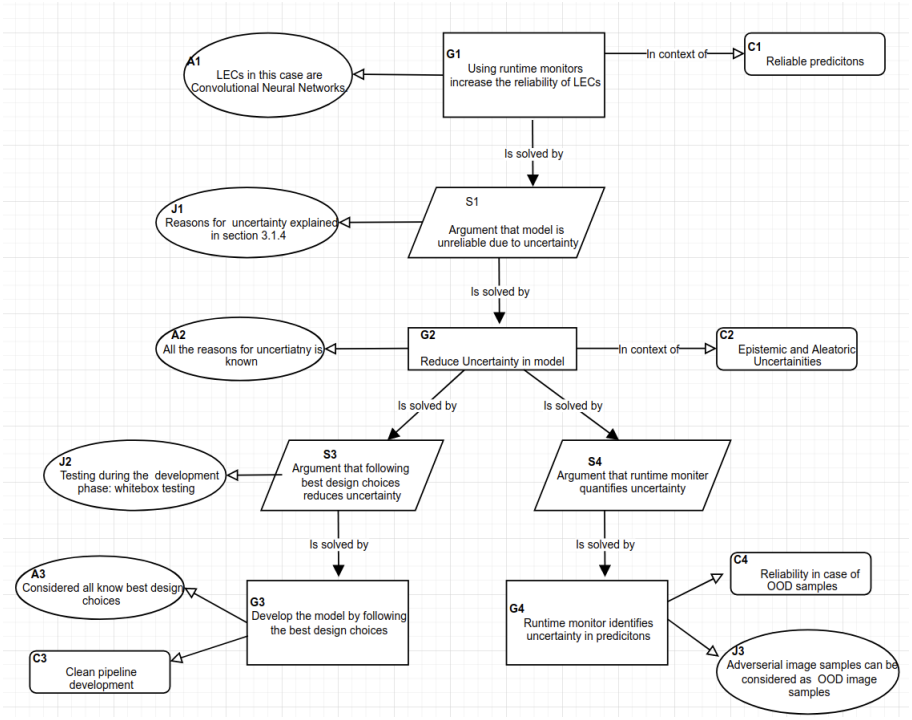
This goal is further decomposed into a smaller goal, with the strategy stating that LECs are unreliable due to the uncertainties. The uncertainties in CNNs are briefly explained in section 3.1.4.



With the help of the strategy, we decomposed the high-level goal as a subgoal. Here our subgoal is now reducing uncertainty increases reliability. We are talking in the context of epistemic and aleatoric uncertainties since we are using CNNs. These two uncertainties are well known and cover most of the reasons for uncertainty in CNNs.



To reduce the uncertainty, the strategy we are following here is considering the best design decisions and using the runtime monitors that can quantify the uncertainty. Good design decisions will help develop the best fit model for the training data, increasing the model's reliability. For this, we have developed a clean pipeline and used an evidential-zoom uncertainty monitor. The runtime monitor can identify the out-of-distribution images as the prediction uncertainty for such images are high.



In this case, the artefacts, which are the experiment results, are provided as a solution to the final subgoals. The final assurance case argumentation for proving our primary goal, runtime monitors increase the reliability of monitors is shown in Figure 5.20

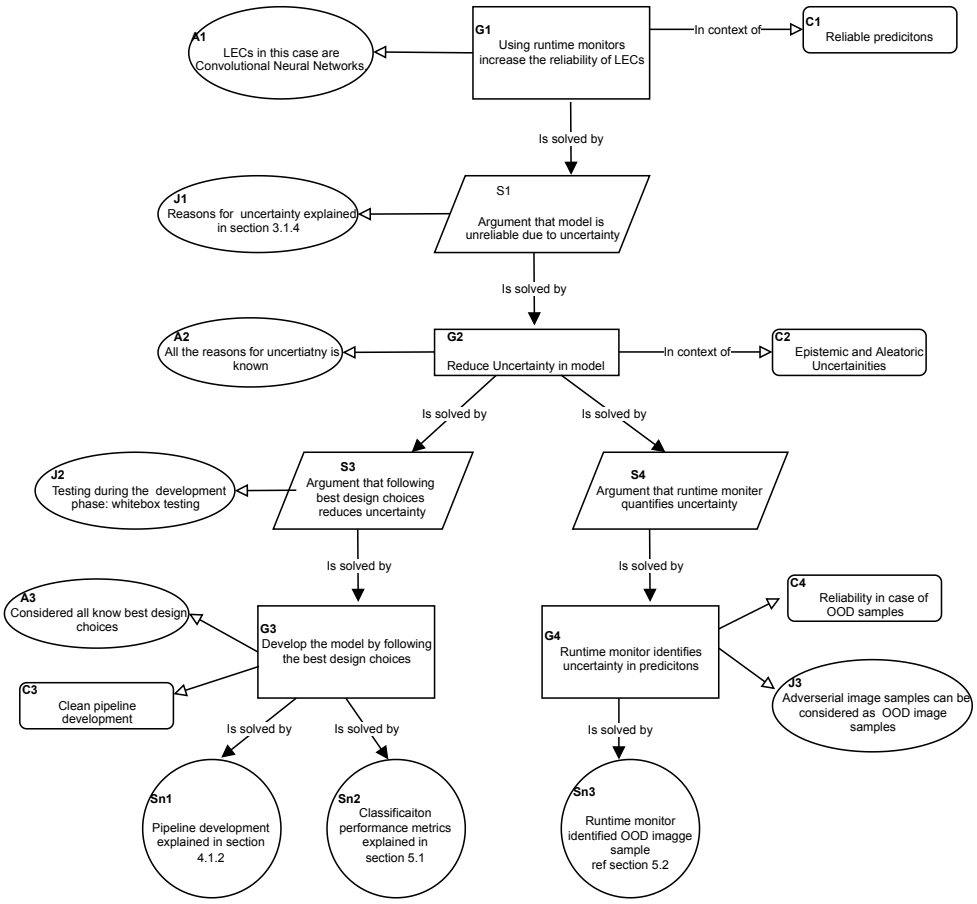


Figure 5.20: Assurance case argument for proving the reliability of LECs

5.4 Summary of All Experiments

All the experiments conducted in this R&D towards proof of concept is shown in Table 5.11

Model	Dataset	Monitors	Accuracy	Precision	Recall	F1-Score
ResNet 18	MNIST	No-monitor	97.42	0.97	0.97	0.97
ResNet 18	MNIST	Evidential-zoom monitor	98.81	0.97	0.98	0.98
ResNet 18	MNIST	Average-probability monitor	98.80	0.97	0.98	0.98
ResNet 18	MNIST	Max monitor	98.03	0.98	0.98	0.98
MobileNet V2	MNIST	No-monitor	97.16	0.97	0.97	0.97
MobileNet V2	MNIST	Evidential-zoom monitor	98.35	0.98	0.98	0.98
MobileNet V2	MNIST	Average-probability monitor	98.55	0.98	0.98	0.98
MobileNet V2	MNIST	Max monitor	98.14	0.98	0.98	0.98
SqueezeNet	MNIST	No-monitor	92.72	0.93	0.92	0.92
SqueezeNet	MNIST	Evidential-zoom monitor	97.18	0.97	0.96	0.97
SqueezeNet	MNIST	Average-probability monitor	95.72	0.95	0.95	0.95
SqueezeNet	MNIST	Max monitor	95.34	0.94	0.95	0.95
ResNet 50	RoboCup@work	No-monitor	99.32	1.0	1.0	1.0
ResNet 50	RoboCup@work	Evidential-zoom monitor	1.0	1.0	1.0	1.0
ResNet 50	RoboCup@work	Average-probability monitor	99.94	1.0	1.0	1.0
ResNet 50	RoboCup@work	Max monitor	99.74	0.99	1.0	1.0
MobileNet V2	RoboCup@work	No-monitor	95.31	0.97	0.97	0.97
MobileNet V2	RoboCup@work	Evidential-zoom monitor	99.16	1.0	1.0	1.0
MobileNet V2	RoboCup@work	Average-probability monitor	98.81	0.98	0.99	0.99
MobileNet V2	RoboCup@work	Max monitor	98.74	0.99	1.0	1.0
SqueezeNet	RoboCup@work	No-monitor	99.62	0.98	0.98	0.98
SqueezeNet	RoboCup@work	Evidential-zoom monitor	1.0	0.99	1.0	1.0
SqueezeNet	RoboCup@work	Average-probability monitor	1.0	0.99	0.99	0.99
SqueezeNet	RoboCup@work	Max monitor	99.83	0.99	1.0	1.0

Table 5.11: Summary of all experiments conducted

6

Conclusions

The usage of autonomous systems across various domains raised many concerns related to safety and reliability. Failure of safety-critical systems leads to different hazards like injuries, death, and property damage. After encountering a series of such failure experiences, researchers and engineers started finding solutions for achieving the goal of trustworthy autonomy. Learning Enabled Components (LECs) are the unit software building blocks for any autonomous system. LECs are data-driven models and are non-deterministic, which means aleatoric uncertainty is an inherent property of these models. In addition to aleatoric uncertainty, the noise present in real-world data makes these models more uncertain (epistemic uncertainty). So verifying these models during the development time is not sufficient. So the researchers in the trustworthy autonomy field are considering runtime verification as a solution to address the uncertainty in predictions.

From the literature, we can observe many different mathematical techniques available to quantify and measure uncertainty. Researches are exploring these approaches and developing new runtime verification techniques. Evidential uncertainty analysis is one such technique used to create a runtime monitor in this R&D. To test the performance of these monitors, as proof of concept, some Convolutional Neural Network (CNN) models were selected, and the performance results were evaluated. To argue that these monitors increase the reliability of LECs, in this R&D, an assurance case is written using Goal Structured Notation (GSN). The contributions made towards this R&D are explained in section 6.1. The lessons learned by the end of this research work are discussed in section 6.2. Proving the reliability of a LEC is a very challenging problem. We did our best to approach this problem with very little prior literature work, and there is still a lot of future work to achieve your reliability goal. The future scope is explained in section 6.3.

6.1 Contributions

The following are the contributions for this research work:

- **Literature:** A detailed background information on assurance cases, their types, uses are provided, along with details about step-by-step development of assurance cases is explained with a use-case example. A literature review on the state of the art practices, techniques, and runtime monitors for developing safe and reliable LECs is provided.

- **Software reliability techniques:** A detailed illustration of dependability, software reliability metrics and testing methods are provided.
- **DNN development following design choices:** The best practices for making the design choices during the development and testing phase of LECs are explained supported by the researchers' prior work in this field.
- **CNN models development and testing:** Three different CNN models are selected for proof of concept and are developed following the best design choices. The performance results of the model are evaluated for the same, and inferences were explained from the LEC reliability perspective.
- **Runtime monitors:** Three different runtime monitors: evidential-zoom monitor, max monitor, average-probability monitor are developed, and the model's reliability is tested before and after using these three monitors. Performance results for each monitor are evaluated, and inferences are explained.
- **Assurance case for the reliability of DNN:** An assurance case for the goal: "Using runtime monitors will increase the reliability of LECs." A strong argumentation using the software reliability techniques is made to satisfy this goal. The assurance case is written using the GSN notation.

6.2 Lessons Learned

The lesson learned during this R&D work are:

- We can say that assurance cases are used to "satisfy" the goals rather than "proving" the goals. Assurance cases will also help in identifying problems during the product development phase.
- GSN is currently a well-maintained graphical notation standard, which is why it is used across different research fields. Confirmation bias, which is typical among humans, will limit the soundness of the assurance cases.
- DNNs are non-deterministic models, making them highly uncertain to noisy real-world data. Hence, these models can be certified for safety or reliability only by verifying them during runtime.
- Following the best practices for making design choices during the development of LECs will increase their reliability to some extent.
- Hyperparameter tuning is one effective technique that helps models to converge. Considering higher initial learning rates will make the model convergence difficult. Considering adaptive learning rates can be another alternative if reducing the training time is our priority.
- Using runtime monitoring techniques can increase the reliability of the LECs, but one should also consider developing the monitors more computationally efficient.

6.3 Future Work

The direction of possible future research is given in this section.

- Assurance measures: The concept of assurance measures is untouched in this work which is a part that completes the trustworthy/assured autonomy architecture mentioned in section 1.1.
- Evaluating GSN: Iterating the assurance case to improve the soundness of argumentation by performing GSN evaluation. This step is a capstone for every assurance case development cycle that helps identify the flaws in the argumentation. This step can be performed many times until the stakeholders are satisfied with our argument.
- Developing computationally efficient runtime monitors: As the autonomous systems have limited computational resources, zooming each image multiple times and predicting those images to find confidence increases the time and space complexity, which is inefficient from the software development perspective.
- Can we quantify the tradeoff between reliability and computational resources? How much we can compromise on reliability to accommodate limited computational resources.
- Simulation testing is part of LEC safety assurance, which can be implemented and tested with more complex LECs.
- Testing the monitors in real-world scenarios and evaluating them increases the soundness of the argumentation.
- Training adversarially robust models to increase the reliability and testing them in real-world scenarios.
- Evaluating the reliability of the model after training it for adversarial robustness.
- Can satisfying safety requirements increase reliability? What is the relation between safety and reliability in the context of LECs? Writing an assurance case to prove the same argument.
- Identifying all the reliability requirements considering many scenarios where LECs are used.
- Evaluating the LECs with different black box software testing methods should strongly support the argumentation for reliability than the white-box techniques. Identifying such strategies and improving the soundness of the argumentation.
- Are fault tree analysis techniques applicable in the context of LECs? Can these techniques help us find new inferences about the black-box nature of LECs?

References

[1] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher *et al.*, "A survey of uncertainty in deep neural networks," *arXiv preprint arXiv:2107.03342*, 2021.

[2] "Ntsb's report on the 2018 uber crash," 2018. [Online]. Available: https://www.ntsb.gov/news/press-releases/Pages/NR20191119c.aspx

[3] S. Neema, "Artificial intelligence colloquium: Assurance for machine learning," 2019. [Online]. Available: https://www.youtube.com/watch?v=5AJAJHJGupQ

[4] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," *arXiv preprint arXiv:1806.01768*, 2018.

[5] E. Asaadi, E. Denney, J. Menzies, G. J. Pai, and D. Petroff, "Dynamic assurance cases: A pathway to trusted autonomy," *Computer*, vol. 53, no. 12, pp. 35–46, 2020.

[6] "Iso/iec/ieee international standard - systems and software engineering–systems and software assurance –part 1:concepts and vocabulary," *ISO/IEC/IEEE 15026-1:2019(E)*, pp. 1–38, 2019.

[7] S. ISO, "9001: 2015," *Quality management systems—Requirements*, 2015.

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

[9] C. Holloway, "The friendly argument notation (fan)," 2020.

[10] E. Heavner and C. M. Holloway, "Assurance arguments for the non-graphically-inclined: Two approaches," 2017.

[11] E. Bills, S. Mastrangelo, and F. Wu, "Documenting medical device risk management through the risk traceability summary," *Biomedical instrumentation technology / Association for the Advancement of Medical Instrumentation*, vol. 49, pp. 26–33, 03 2015.

[12] R. Wei, T. Kelly, X. Dai, S. Zhao, and R. Hawkins, "Model based system assurance using the structured assurance case metamodel," 05 2019.

[13] "Ieee standard–adoption of iso/iec 15026-2:2011 systems and software engineering–systems and software assurance–part 2: Assurance case," *IEEE Std 15026-2-2011*, pp. 1–28, 2011.

[14] R. Bloomfield and P. Bishop, "Safety and assurance cases: Past, present and possible future–an adelard perspective," in *Making systems safer*. Springer, 2010, pp. 51–67.

[15] J. Rushby, "The interpretation and evaluation of assurance cases," *Comp. Science Laboratory, SRI International, Tech. Rep. SRI-CSL-15-01*, 2015.

[16] A. C. W. Group *et al.*, "Goal structuring notation community standard (version 2)," 2018.

[17] P. N. Leveson, "White paper on the use of safety cases in certification and regulation."

[18] N. Leveson, "White paper on the use of safety cases in certification and regulation," 2011.

[19] S. Wilson, J. McDermid, P. Fenelon, and P. Kirkham, "No more spineless safety cases: A structured method and comprehensive tool support for the production of safety cases," in *2nd international conference on control and instrumentation in nuclear installations (INEC'95)*, 1995.

[20] T. P. Kelly, "Arguing safety: a systematic approach to managing safety cases," Ph.D. dissertation, University of York York, UK, 1999.

[21] A. Avizienis, J.-C. Laprie, and B. Randell, "Fundamental concepts of dependability," *Department of Computing Science Technical Report Series*, 2001.

[22] M. R. Lyu *et al.*, *Handbook of software reliability engineering.* IEEE computer society press CA, 1996, vol. 222.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] T. Wen, G. Xie, Y. Cao, and B. Cai, "A dnn-based channel model for network planning in train control systems," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[26] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2020.

[27] G. Schwalbe and M. Schels, "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)*, 2020.

[28] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[29] R. McAllister, Y. Gal, A. Kendall, M. Van Der Wilk, A. Shah, R. Cipolla, and A. Weller, "Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning." International Joint Conferences on Artificial Intelligence, Inc., 2017.

[30] I. Kononenko, "Bayesian neural networks," *Biological Cybernetics*, vol. 61, no. 5, pp. 361–370, 1989.

[31] T. Dreossi, S. Ghosh, X. Yue, K. Keutzer, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Counterexample-guided data augmentation," *arXiv preprint arXiv:1805.06962*, 2018.

[32] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," *arXiv preprint arXiv:1905.02175*, 2019.

[33] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," *arXiv preprint arXiv:1712.01785*, 2017.

[34] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[35] L. Pulina and A. Tacchella, "Challenging smt solvers to verify neural networks," *Ai Communications*, vol. 25, no. 2, pp. 117–135, 2012.

[36] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," *arXiv preprint arXiv:1805.02242*, 2018.

[37] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1599–1614.

[38] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 109–119.

[39] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[40] P.-J. Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne, "Learning how to explain neural networks: Patternnet and patternattribution," *arXiv preprint arXiv:1705.05598*, 2017.

[41] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, vol. 2, no. 11, p. e7, 2017.

[42] V. Petsiuk, A. Das, and K. Saenko, "Rise: Randomized input sampling for explanation of black-box models," *arXiv preprint arXiv:1806.07421*, 2018.

[43] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[44] J. Kim, A. Rohrbach, T. Darrell, J. Canny, and Z. Akata, "Textual explanations for self-driving vehicles," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 563–578.

[45] J. Rabold, M. Siebers, and U. Schmid, "Explaining black-box classifiers with ilp–empowering lime with aleph to approximate non-linear decisions with relational rules," in *International Conference on Inductive Logic Programming*. Springer, 2018, pp. 105–117.

[46] S. Thrun, "Extracting rules from artificial neural networks with distributed representations," *Advances in neural information processing systems*, pp. 505–512, 1995.

[47] F. B. Fuchs, O. Groth, A. R. Kosiorek, A. Bewley, M. Wulfmeier, A. Vedaldi, and I. Posner, "Neural stethoscopes: Unifying analytic, auxiliary and adversarial network probing," *CoRR abs/1806.05502*, 2018.

[48] R. Fong and A. Vedaldi, "Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8730–8738.

[49] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas *et al.*, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)," in *International conference on machine learning*. PMLR, 2018, pp. 2668–2677.

[50] N. R. Council *et al.*, *Risk analysis and uncertainty in flood damage reduction studies*. National Academies Press, 2000.

[51] F. Hbffinan and J. Hammonds, "Propagation of uncertainty in risk assessment: the need to distinguish between uncertainty due to lack of knowledge and uncertainty due to variability," *Risk Analysis*, vol. 14, no. 5, pp. 707–712, 1994.

[52] M. B. Van Asselt and J. Rotmans, "Uncertainty in integrated assessment modelling," *Climatic change*, vol. 54, no. 1, pp. 75–105, 2002.

[53] D. Kahneman and A. Tversky, "Variants of uncertainty," *Cognition*, vol. 11, no. 2, pp. 143–157, 1982.

[54] J. C. Helton, "Treatment of uncertainty in performance assessments for complex systems," *Risk analysis*, vol. 14, no. 4, pp. 483–511, 1994.

[55] M. Henrion and B. Fischhoff, "Assessing uncertainty in physical constants," *American Journal of Physics*, vol. 54, no. 9, pp. 791–798, 1986.

[56] H. Natke and Y. Ben-Haim, "Uncertainty-a discussion from various points of view," *MATHEMATICAL RESEARCH*, vol. 99, pp. 267–276, 1997.

[57] G. J. Klir, "Uncertainty theories, measures and principles," *Uncertainty: Models and Measures, HG Natke and Y. Ben-Haim, eds., Akademie Verlag, Berlin, Germany*, 1996.

[58] Y. Li, J. Chen, and L. Feng, "Dealing with uncertainty: A survey of theories and practices," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2463–2482, 2012.

[59] C.-H. Cheng, G. N¨hrenberg, C.-H. Huang, H. Ruess, and H. Yasuoka, "Towards dependability metrics for neural networks," in *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2018, pp. 1–4.

[60] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[61] V. Abdelzad, K. Czarnecki, R. Salay, T. Denounden, S. Vernekar, and B. Phan, "Detecting out-of-distribution inputs in deep neural networks using an early-layer output," *arXiv preprint arXiv:1910.10307*, 2019.

[62] J. Jacobson and C. Grante, "Functional safety in systems of road vehicles," 2010.

[63] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[64] P. Koopman and F. Fratrik, "How many operational design domains, objects, and events?" in *Safeai@ aaai*, 2019.

[65] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 3, pp. 636–653, 2010.

[66] C.-H. Cheng, D. Gulati, and R. Yan, "Architecting dependable learning-enabled autonomous systems: A survey," *arXiv preprint arXiv:1902.10590*, 2019.

[67] C.-H. Cheng, G. Nührenberg, and H. Yasuoka, "Runtime monitoring neuron activation patterns," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 300–303.

[68] C. H. Lampert, H. Nickisch, and S. Harmeling, "Attribute-based classification for zero-shot visual object categorization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 3, pp. 453–465, 2013.

[69] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[70] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[73] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[74] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[75] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017, pp. 464–472.

[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[77] N. Aafaq, A. Mian, W. Liu, S. Z. Gilani, and M. Shah, "Video description: A survey of methods, datasets, and evaluation metrics," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–37, 2019.