Titre: Title:	Enhancing DO-178C Compliance in Safety-Critical Software: Integrating Formal Methods, Controlled Natural Language, and AI-Driven Automation
Auteur: Author:	Rim Zrelli
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Zrelli, R. (2025). Enhancing DO-178C Compliance in Safety-Critical Software: Integrating Formal Methods, Controlled Natural Language, and AI-Driven Automation [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/63404/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Enhancing DO-178C Compliance in Safety-Critical Software: Integrating Formal Methods, Controlled Natural Language, and AI-Driven Automation

RIM ZRELLI
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Février 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Enhancing DO-178C Compliance in Safety-Critical Software: Integrating Formal Methods, Controlled Natural Language, and AI-Driven Automation

présentée par **Rim ZRELLI**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Tarek OULD-BACHIR, président
Gabriela NICOLESCU, membre et directrice de recherche
John MULLINS, membre et codirecteur de recherche
Mohammad HAMDAQA, membre
Eric ALATA, membre externe

DEDICATION

To my parents,
for their boundless love, countless sacrifices, and unwavering support,
without which none of this would be possible.
To my sister,
for her unshakeable belief in me, her love, and for being my source of strength,
making every challenge worthwhile.
To my brothers,
for their constant encouragement and for being my source of strength,
reminding me of the power of family.

And to my sister-in-law and my friends,
for their support and for standing by me through every step of this journey.
With profound gratitude,
this work is dedicated to each of you...

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and deepest gratitude to my supervisor, Professor *Gabriela Nicolescu*, for guiding me through the winding road of research. Her encouragement, belief in my potential, and unwavering support, both financial and emotional, have been invaluable throughout this journey. Completing this thesis would not have been possible without her insightful feedback, guidance, and commitment of time in reviewing my work.

I would also like to thank Professor *John Mullins* for his guidance at the beginning of my doctoral journey. His support helped shape my foundational understanding of the field of formal methods. I am equally grateful for his thoughtful feedback during the review of this thesis.

I extend my heartfelt gratitude to *Felipe Gohring de Magalhães* for his ongoing mentorship, invaluable insights, and for sharing an invaluable amount of time to revise my work.

I am also deeply thankful to the members of my Ph.D. committee, Professor *Tarek Ould-Bachir*, Professor *Mohammad Hamdaqa*, and Professor *Eric Alata*, for their willingness to review my dissertation.

This research was partially supported by *Humanitas*, whose contributions and expertise greatly enriched this study. I am especially grateful to *Maroua Ben Attia* and *Abdo Shabah* for their support and insights.

I would also like to acknowledge *Mitacs* for their financial support, which played a crucial role in enabling this research.

Finally, I am immensely thankful to my colleagues and friends who collaborated with me during my thesis. Some joined me as co-authors, contributing directly to the research, while others offered their support behind the scenes as *"anonymous heroes"*, making this journey both rewarding and memorable.

ABSTRACT

The increasing reliance on Unmanned Aerial Vehicles (UAVs) across industries such as defense, logistics, and environmental monitoring has intensified the need for strict software certification standards to ensure safety and reliability in autonomous operations. The DO-178C standard in aviation provides a rigorous framework for certifying safety-critical software, emphasizing verification, validation, and traceability throughout the software lifecycle. Compliance with DO-178C is crucial for UAVs, as these systems often operate with limited human oversight, requiring reliable autonomous decision-making under diverse and unpredictable conditions. However, achieving full compliance is challenging due to ambiguities in natural language (NL) requirements, the labour-intensive nature of formal verification processes, and the limited integration of automated tools, creating barriers to scalable certification and highlighting the need for innovative methodologies that streamline verification and enhance compliance.

This research addresses these challenges by advancing DO-178C compliance for UAV software through improved verification processes, reducing NL ambiguities, and advancing tool integration. A comprehensive framework is proposed, bridging formal verification with automation to streamline workflows, enhance requirement precision, and reinforce traceability for safe UAV deployment in complex environments. Key contributions include a Controlled Natural Language (CNL) and an AI-driven NL-to-temporal logic translation framework, enabling robust and scalable verification. Additionally, the *Natural2CTL* dataset is validated to train ML models, advancing automated NL-to-CTL translation and minimizing manual verification efforts. These advancements contribute to technological and methodological progress in UAV certification, with broader applications across safety-critical software domains.

The methodology integrates formal methods, automated tools, and linguistic processing across three main components. First, formal verification methods are combined with automated tools to enhance verification, validation, and traceability while automating certification evidence generation. Second, a CNL framework tailored for DO-178C-compliant requirements mitigates NL ambiguities and improves documentation precision. Finally, the study introduces the *Natural2CTL* dataset, enabling AI-based translation of NL requirements into Computation Tree Logic (CTL) specifications to streamline complex software verification. Together, these methods enhance efficiency, accuracy, and scalability in DO-178C certification processes, supporting the reliable deployment of safety-critical UAV systems in dynamic environments.

The integration of formal verification methods with automated tools demonstrated significant reductions in manual effort for generating and managing DO-178C compliance evidence. The CNL framework addressed NL ambiguities, enhancing requirements precision, while the *Natural2CTL* dataset enabled accurate, automated CTL translations. Evaluation metrics such as *Exact Match* and *Equivalent Match* confirmed that the automated approaches meet both syntactic and semantic correctness for certification.

In conclusion, this thesis offers an innovative framework for DO-178C compliance, integrating formal methods, CNL, and AI-driven verification to streamline certification, reduce manual intervention, and enhance traceability. These advancements significantly improve accuracy, efficiency, and scalability, with implications extending to other safety-critical software sectors. This work lays the foundation for future advancements in automated verification, enabling reliable and scalable certification of UAVs and other high-assurance systems in aviation and beyond.

Keywords: DO-178C compliance, Unmanned Aerial Vehicles, Safety-Critical Software, Formal Verification, Controlled Natural Language, Certification, Language Models, NL-to-CTL Translation, Traceability, Autonomous Systems, Software Verification Automation, Aviation Software Standards.

RÉSUMÉ

La dépendance croissante envers les Véhicules Aériens Sans Pilote (UAV) dans des secteurs tels que la défense, la logistique et la surveillance environnementale a intensifié la nécessité de normes strictes de certification logicielle pour garantir la sécurité et la fiabilité des opérations autonomes. La norme DO-178C, largement utilisée dans l'aviation, fournit un cadre rigoureux pour la certification des logiciels critiques, mettant l'accent sur la vérification, la validation et la traçabilité tout au long du cycle de vie logiciel. La conformité à la norme DO-178C est cruciale pour les UAV, qui fonctionnent souvent avec une supervision humaine limitée et nécessitent une prise de décision autonome fiable dans des conditions diversifiées et imprévisibles. Cependant, atteindre une conformité totale reste un défi en raison des ambiguïtés des spécifications en Langage Naturel (NL), de la nature fastidieuse des processus de vérification formelle, et de l'intégration limitée des outils automatisés, ce qui souligne la nécessité de méthodologies innovantes pour simplifier la vérification et améliorer la conformité.

Cette recherche s'attaque à ces défis en proposant une avancée dans la conformité au DO-178C pour les logiciels UAV grâce à des processus de vérification améliorés, une réduction des ambiguïtés du NL et une intégration accrue des outils automatisés. Un cadre méthodologique complet est proposé, combinant vérification formelle et automatisation pour rationaliser les flux de travail, améliorer la précision des exigences et renforcer la traçabilité pour un déploiement sûr des UAV dans des environnements complexes. Les contributions clés incluent un Langage Naturel Contrôlé (CNL) et un cadre de traduction basé sur l'intelligence artificielle pour convertir les exigences NL en logique temporelle, permettant une vérification robuste et évolutive. De plus, le jeu de données *Natural2CTL* est validé pour former des modèles d'apprentissage automatique, facilitant la traduction NL-CTL automatisée et réduisant les efforts de vérification manuels. Ces avancées représentent un progrès technologique et méthodologique dans la certification des UAV, avec des applications potentielles dans d'autres domaines logiciels critiques.

La méthodologie s'articule autour de trois composantes principales. Premièrement, les méthodes de vérification formelle sont intégrées avec des outils automatisés pour améliorer la vérification, la validation et la traçabilité, tout en automatisant la génération des preuves de certification. Deuxièmement, un CNL adapté aux exigences conformes au DO-178C est proposé pour réduire les ambiguïtés du NL et améliorer la précision de la documentation. Enfin, l'étude introduit le jeu de données *Natural2CTL*, qui permet la traduction automatisée des exigences NL en formules CTL (Computation Tree Logic) pour simplifier la vérifica-

tion des logiciels complexes. Ensemble, ces méthodes augmentent l'efficacité, la précision et l'évolutivité des processus de certification DO-178C, facilitant le déploiement fiable des UAV dans des environnements dynamiques.

L'intégration des méthodes de vérification formelle avec des outils automatisés a permis de réduire significativement les efforts manuels nécessaires à la gestion des preuves de conformité au DO-178C. Le cadre CNL a résolu les ambiguïtés du NL en améliorant la précision des exigences, tandis que le jeu de données *Natural2CTL* a permis des traductions CTL automatisées et précises. Les métriques d'évaluation telles que *Exact Match* et *Equivalent Match* ont confirmé que ces approches automatisées répondent aux exigences de correction syntaxique et sémantique pour la certification.

En conclusion, cette thèse propose un cadre innovant pour la conformité au DO-178C, intégrant des méthodes formelles, un CNL et une vérification basée sur l'IA pour rationaliser la certification, réduire l'intervention manuelle et améliorer la traçabilité. Ces avancées augmentent considérablement l'efficacité, la précision et la fiabilité des processus de certification, avec des implications qui s'étendent à d'autres secteurs logiciels critiques. Ce travail établit une base solide pour les futures avancées en vérification automatisée, ouvrant la voie à une certification évolutive et fiable des UAV et autres systèmes à haute assurance dans l'aviation et au-delà.

Mots Clés: Conformité DO-178C, véhicules aériens sans pilote, logiciels critiques pour la sécurité, vérification formelle, langage naturel contrôlé, certification, modèles linguistiques, traduction LN-vers-CTL, traçabilité, systèmes autonomes, automatisation de la vérification logicielle, normes logicielles en aviation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

UAV	Unmanned Aerial Vehicles
RTCA	Radio Technical Commission for Aeronautics
DAL	Design Assurance Level
NL	Natural Language
CTL	Computation Tree Logic
CNL	Controlled Natural Language
NLP	Natural Language Processing
LLM	Large Language Model
ISO	International Organization for Standardization
NAS	National Airspace System
CFR	Code of Federal Regulations
FAA	Federal Aviation Administration
EASA	European Union Aviation Safety Agency
HLR	High-level Requirement
LLR	Low-Level Requirement
SAS	Software Accomplishment Summary
LTL	Linear Temporal Logic
SRS	Software Requirement Specification
MBD	Model-Based Design
RE	Requirements Engineering
CAS	Collision Avoidance System
SRATS	System Requirements Allocated To Software
PDI	Parameter Data Item
BVLOS	Beyond Visual Line of Sight
LoRA	Low-Rank Adaptation
ExacM	Exact Match
EquivM	Equivalent Match
ExacMTOD	Exact Match with Temporal Operator Difference
EquivMTOD	Equivalent Match with Temporal Operator Difference
PC	Partially Correct
NC	Incorrect
SOJS	Structural Operator Jaccard Similarity

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

"Safety should never be a priority.
It should be a precondition."
Paul O Neill

1.1 Research Context

Building on the growing reliance on Unmanned Aerial Vehicles (UAVs) in diverse sectors, ranging from defence to commercial applications, there is an increasing emphasis on the reliability and safety of their software systems. Given the complexities and risks associated with these systems, particularly in dynamic and unpredictable environments, any software failure could result in severe consequences. To mitigate these risks, the aerospace industry has adopted stringent certification standards like DO-178C [1]. This standard ensures that airborne software systems meet the necessary safety requirements, serving as a fundamental safeguard for UAV operations

Given the critical role of DO-178C in aviation safety, this standard, developed by the Radio Technical Commission for Aeronautics (RTCA), provides a structured framework for ensuring that software in airborne systems adheres to specific safety and reliability criteria. Its focus on rigorous verification, validation, and traceability ensures that each stage of the software lifecycle meets the required Design Assurance Levels (DALs), which correspond to the potential impact of software failures on overall system safety. By guiding software engineers through a disciplined process of requirement specification, design, coding, and testing, DO-178C plays a pivotal role in maintaining high safety standards across the entire software lifecycle [2].

For UAVs, compliance with DO-178C is especially important due to the need to guarantee that their software can handle a wide range of scenarios in real-world operations, from navigation and control to collision avoidance and autonomous decision-making [3]. As UAVs often operate autonomously or with minimal human intervention, the reliability of their software is directly tied to their safety and functionality in the field. Ensuring that software is developed according to DO-178C not only supports safer UAV operations but is also a prerequisite for regulatory approval, enabling these vehicles to be deployed in civilian airspace.

The emphasis on thorough verification and traceability throughout the software lifecycle in DO-178C means that every aspect of the software must be demonstrably compliant with

safety requirements. This includes ensuring that high-level system requirements flow down accurately into software requirements and are consistently reflected in the final implementation. Such a disciplined approach ensures that all software components perform as expected under both normal and abnormal conditions, significantly reducing the likelihood of undetected software errors.

Despite its benefits, achieving DO-178C compliance requires significant effort and expertise, given the extensive documentation and validation required to meet the standard's objectives [4,5]. This has made the certification process a critical focus area for research and innovation in aerospace software engineering. By exploring new methodologies to support this process, there is an opportunity to improve the efficiency and reliability of UAV software certification, ultimately contributing to safer skies and broader adoption of autonomous aerial systems.

1.2 Problem Statement

Despite DO-178C's comprehensive framework, achieving full compliance is a challenging endeavour, particularly for complex UAV systems [6]. To fully appreciate the need for innovation in this area, it is important to recognize the specific barriers to effective compliance, including issues with Natural Language (NL) requirements, formal methods, and existing verification tools.

One of the fundamental obstacles to DO-178C compliance is the inherent ambiguity in NL used to specify software requirements. While NL is valued for its accessibility, it often leads to varied interpretations, resulting in misalignment between intended functionality and implementation [7]. These ambiguities complicate verification efforts, as vague or imprecise requirements make it challenging to produce clear and testable specifications. The time and effort needed to clarify requirements contribute to delays in the certification process, ultimately increasing costs and the risk of errors during software verification.

Another key problem involves the integration of formal methods into the certification process. Formal methods, such as model checking [8] and theorem proving [9], offer a mathematically rigorous way to specify and verify software behaviour [10]. These methods can detect inconsistencies and ambiguities early in the software development lifecycle, thereby enhancing the accuracy of verification. Despite their potential, formal methods are underutilized in practice due to the complexity involved in translating NL requirements into formal representations like Computation Tree Logic (CTL) [11]. This translation requires a high level of expertise, which can be a barrier for many organizations aiming to adopt formal verification techniques. The gap between the ease of using NL and the precision required for formal

verification creates a bottleneck in the certification process, particularly for complex UAV systems that must operate autonomously under varied conditions.

The challenges of achieving DO-178C compliance are further compounded by the limitations of existing automated verification tools. While tools like the LDRA suite [12], and SHERLOCK [13] can automate aspects of traceability and structural coverage analysis, they are often used in isolation from formal methods. This lack of integration means that verification tools do not fully leverage the benefits of formal methods, resulting in a fragmented approach to certification. Additionally, much of the effort in managing requirements traceability and generating certification evidence remains manual, making the process labour-intensive and prone to errors. As UAV systems grow in complexity, the need for more advanced automation in verification becomes even more critical to ensure scalability and reduce the manual burden.

Another significant challenge is the translation of NL requirements into CTL for formal verification [14]. This process is essential for enabling rigorous model checking, but it is often performed manually, requiring domain-specific expertise that many organizations may not have readily available. The manual nature of this translation limits the scalability of formal methods and slows down the certification process. To address this issue, there is a need for automated approaches that can accurately translate NL requirements into formal specifications, making it easier to apply model-checking techniques to real-world UAV systems.

In summary, the current processes for achieving DO-178C compliance face several interrelated challenges: the ambiguity of NL requirements, the difficulty of integrating formal methods, the limitations of existing automated tools, and the complexity of translating NL into formal logic. These challenges hinder the efficiency and effectiveness of the certification process, creating a need for innovative methodologies that can bridge these gaps. This research aims to address these problems by developing a Controlled Natural Language (CNL) to reduce requirement ambiguity, creating datasets like Natural2CTL to support automated translations, and leveraging AI-based approaches to streamline verification tasks. By integrating these methods, the study seeks to provide a more efficient and reliable path to DO-178C compliance, ensuring that UAV software systems meet the highest standards of safety and reliability.

Addressing these challenges requires an approach that integrates advanced methods with practical tools, as outlined in the following section on the research approach.

1.3 Research Approach

In response to the challenges outlined, this research adopts a three-pronged approach: integrating formal methods with automated tools, developing a CNL to reduce ambiguity, and employing AI models to automate the translation of NL into formal specifications. Each element of this approach is designed to address specific obstacles in achieving DO-178C compliance, aiming to streamline verification processes and reduce manual effort:

1. **Integration of Formal Methods with Automated Verification Tools**: Building on formal methods, the research integrates automated tools like the LDRA suite to support continuous verification, validation, and traceability of requirements. This integration aims to enhance the efficiency of the verification process, ensuring that software artifacts align with DO-178C's rigorous standards throughout the lifecycle.

2. **Development of a CNL**: A CNL is proposed to mitigate the inherent ambiguity of NL requirements. Inspired by DO-178C guidelines, this CNL provides structured rules for formulating requirements, ensuring clarity and verifiability. This structured language is designed to make requirements easier to interpret, verify, and trace throughout the software development and certification process.

3. **Automated Translation of NL to CTL Using AI Models**: To bridge the gap between human-readable requirements and formal verification, the research leverages advancements in Natural Language Processing (NLP) and Large Language Models (LLMs). The creation of the Natural2CTL dataset supports the training of AI models like Mistral [15], enabling automated translation of NL requirements into CTL. This automation seeks to reduce the manual effort required for formal verification and enhance the precision of the translation process.

This integrative approach is designed to streamline the certification process, reduce the risk of errors, and make formal methods more accessible and scalable for the aerospace industry.

1.4 Research Questions and Thesis Contributions

The following research questions guide this study, each addressing a specific challenge in the context of DO-178C compliance and the integration of automation and formal methods:

- **What is the impact of integrating formal methods with automated verification tools on the efficiency and rigour of the DO-178C certification process?**

This question examines how the use of automated tools alongside formal methods can improve verification processes for UAV software.

- **How can a CNL reduce ambiguity in DO-178C-compliant software requirements?** This question focuses on identifying linguistic constraints that can transform ambiguous NL requirements into precise, clear specifications suitable for certification. It directly addresses the issue of unclear requirements that hinder the verification process.
- **To what extent can the Natural2CTL dataset facilitate the training of AI models for translating NL requirements into formal logic?** This question focuses on the creation and validation of the Natural2CTL dataset, assessing its role in enabling AI-based models to perform accurate NL-to-CTL translations and supporting the broader goal of automating formal verification.
- **How can automated translation from NL to CTL improve formal verification processes for safety-critical systems?** This question explores the effectiveness of LLMs like Mistral in translating NL requirements into CTL, aiming to enhance accuracy and reduce reliance on domain-specific expertise in formal verification.

This thesis makes the following contributions:

1. Integration of Formal Methods with Automated Tools: The research demonstrates how formal methods, when combined with automated tools, can streamline the verification, validation, and traceability processes required by DO-178C, particularly for complex UAV software.
2. Development of a CNL for DO-178C Requirements: This CNL provides a structured approach to writing software requirements, reducing ambiguity and enhancing the verifiability of requirements.
3. Creation of the Natural2CTL Dataset: This dataset, consisting of over 2,000 pairs of NL requirements and their CTL equivalents, supports the training of LLMs for automated NL-to-CTL translation.
4. Fine-Tuning LLMs for NL-to-CTL Translation: The study optimizes models like Mistral, achieving higher accuracy in generating formal specifications from NL requirements. Comparative analyses with models like GPT-4 demonstrate significant improvements in translation precision.

These contributions collectively aim to improve the efficiency, reliability, and scalability of software certification processes in the aerospace industry.

1.5 List of Publications

The following is a list of our publications related to this dissertation.

- Zrelli, R.*, Misson, H. A.*, Kamkuimo, S., Ben Attia, M., de Magalhães, F. G., & Nicolescu, G. (2024). Integrating Formal Methods and Automated Tools for DO-178C Compliance in UAV Software. Submitted to the Journal of Aerospace Information Systems. (*: Equal Contribution)
  This paper is presented in Chapter 5.

- Zrelli, R., Amaral Misson, H., Ben Attia, M., Gohring de Magalhães, F., Shabah, A., & Nicolescu, G. (2024, March). Natural2CTL: A Dataset for Natural Language Requirements and Their CTL Formal Equivalents. In International Working Conference on Requirements Engineering: Foundation for Software Quality (pp. 205-216). Cham: Springer Nature Switzerland.
  This paper is presented in Chapter 7.

- Zrelli, R., Misson, H.A., Ben Attia, M., Magalhaes, F.G.d., Shabah, A., & Nicolescu, G.: Advancing Formal Verification: Fine-tuning LLMs for Translating Natural Language requirements to CTL specifications. In: Proceedings of the 35th International Workshop on Rapid System Prototyping (2024). Accepted, to appear.
  This paper is presented in Chapter 8.

- Zrelli, R., Misson, H.A., Ben Attia, M., Magalhaes, F.G.d., Shabah, A., & Nicolescu, G. (2024). Bridging Natural Language and Formal Specifications: A Multi-Approach Methodology for NL-to-CTL Translation Using LLMs. Submitted to the International Journal on Software and Systems Modeling (SoSyM).
  This paper is presented in Chapter 9.

Furthermore, collaborations during this research produced the following publications:

- Kamkuimo, S. A., Magalhaes, F., Zrelli, R., Misson, H. A., Attia, M. B., & Nicolescu, G. (2023). Decomposition and Modeling of the Situational Awareness of Unmanned Aerial Vehicles for Advanced Air Mobility. Drones, 7(8), 501.

- Misson, H. A., Zrelli, R., Ben Attia, M., Magalhaes, F. G., & Nicolescu, G. (2023, September). ReDaML: A Modeling Language for DO-178C High-Level Requirements in Airspace Systems. In Proceedings of the 34th International Workshop on Rapid System Prototyping (pp. 1-7).
- Misson, H.A., Zrelli, R., Ben Attia, M., Magalhaes, F.G.d., Shabah, A., & Nicolescu, G. (2024). Bridging the Gap: A DO-178C Compliant Framework for Requirements-to-Architecture Transition. Submitted to the Aerospace Systems journal.

1.6 Thesis Outline

The remainder of this dissertation is organized as follows. Chapter 2 provides background information, while Chapter 3 surveys related work on DO-178C compliance, the challenges of using NL for requirements, and recent advancements in AI-based formal verification. Chapter 4 provides a high-level overview of the research process followed in this dissertation. Chapter 5 presents the integration of formal methods with automated verification tools. Chapter 6 details the development of a CNL for aerospace requirements, along with a rule-based system for verifying compliance. Chapter 7 describes the creation and validation of the Natural2CTL dataset, which supports automating the translation of NL requirements into CTL. Chapter 8 explores the fine-tuning of LLMs for NL-to-CTL translation, comparing different models to optimize accuracy. Chapter 9 introduces alternative NL-to-CTL translation methodologies, including few-shot learning and hybrid BERT-GPT models, and evaluates their effectiveness for scalable formal specification generation. Finally, Chapter 10 summarizes the dissertation's key contributions, discusses its limitations, and suggests directions for future research.

CHAPTER 2 Background

2.1 Chapter Overview

In this chapter, we present the key concepts and foundational knowledge necessary to understand the research contributions of this thesis. This includes a discussion of airworthiness certification standards, with a particular focus on DO-178C, as well as an exploration of formal methods, Controlled Natural Language (CNL), Natural Language Processing (NLP), Computation Tree Logic (CTL), and Large Language Models (LLMs). This chapter is designed for readers who may be unfamiliar with these concepts, offering a comprehensive overview that lays the groundwork for the methodologies discussed in subsequent chapters.

2.2 Airworthiness Certification Standards

2.2.1 Overview of Certification Standards

The International Organization for Standardization (ISO)[1] defines software certification as the process by which a third party provides written assurance that a product, process, or service meets specified characteristics [16]. In the context of aircraft equipment and systems certifications, these characteristics refer to regulations that ensure the airworthiness of aircraft operating within the National Airspace System (NAS) [17]. Compliance with these regulations is demonstrated through a rigorous process where the responsible entities must show that all necessary measures have been taken to guarantee the safety, reliability, and integrity of the aircraft's systems. In the United States, for example, these requirements are codified under Title 14 of the Code of Federal Regulations (14 CFR) [18].

To streamline the certification process and ensure that aviation products meet legal requirements, various industry stakeholders have developed a comprehensive set of standards. These standards serve as guidelines, detailing how to achieve compliance with federal regulations. The standards cover a wide range of processes, including system development, safety assessment, and design assurance. Among these, DO-178C [1] is central to the certification of software used in airborne systems, while DO-254 [19] addresses the certification of airborne electronic hardware, and DO-297 provides guidelines for integrated modular avionics [20].

The core purpose of these standards is to ensure that all components of an aircraft, ranging from software to hardware, operate reliably under various operational conditions, minimizing

[1] https://www.iso.org

the risk of failures that could affect overall flight safety. Although DO-178C and related standards were initially developed for the broader aviation industry, they are now also applied to the certification of UAVs. As UAVs become more prevalent in both commercial and defence sectors, it is crucial that they meet the same rigorous safety and reliability standards as manned aircraft, especially since they often operate autonomously or with minimal human oversight. Given the complexity of their software, which may include functionalities such as autonomous navigation and collision avoidance, adhering to stringent certification standards is essential to ensure their operational safety.

2.2.2 DO-178C for Software Certification

DO-178C [1], formally titled *Software Considerations in Airborne Systems and Equipment Certification*, serves as the primary standard for certifying software used in airborne systems. Developed by the RTCA[2] and EUROCAE[3], this standard provides guidance on ensuring that software for airborne equipment complies with airworthiness requirements. The standard is particularly relevant for software that controls critical aircraft functions, as it establishes a structured framework for software development and verification, emphasizing the importance of safety throughout the software lifecycle.

The core purpose of DO-178C is to ensure that software embedded in airborne systems performs its intended functions safely and reliably [1]. To achieve this, DO-178C outlines a comprehensive set of objectives across various stages of the software lifecycle, including planning, development, verification, configuration management, and quality assurance. Each stage is designed to produce evidence that supports the software's compliance with airworthiness requirements. This evidence is documented meticulously and reviewed by certification authorities such as the Federal Aviation Administration (FAA)[4] or the European Union Aviation Safety Agency (EASA)[5].

Design Assurance Levels

One of the foundational principles of DO-178C is the concept of DALs, which categorize software based on the potential impact of its failure on the safety of the aircraft. Levels range from DAL A (catastrophic consequences) to DAL E (no safety effect). The rigour of the verification activities required increases with the criticality of the software's function,

[2]https://www.rtca.org/
[3]https://www.eurocae.net/
[4]https://www.faa.gov/
[5]https://www.easa.europa.eu/

meaning that higher DALs demand more stringent reviews and testing.

Table 2.1 presents the number of objectives for each DAL level. The first column of the table refers to the software level at which abnormal behaviour at this level is associated with the failure condition represented in the second column. The third column presents the number of objectives that must be satisfied according to the DAL level. Among these, certain objectives must be satisfied independently using evidence proving the satisfaction of these objectives, which must be verified by a person other than the one who developed it.

Table 2.1 DO-178C Objectives by DAL Level.

DAL level	Failure condition	Objectives	With independence
A	Catastrophic	71	30
B	Hazardous	69	18
C	Major	62	5
D	Minor	26	2
E	No safety effect	0	0

Software Development Processes under DO-178C

DO-178C defines specific processes that structure the software lifecycle, which includes planning, development, and integral processes like verification and configuration management. While the standard does not prescribe specific development methods, it outlines required activities, such as requirements capture, design, coding, and testing, that must be systematically followed. This flexibility allows for the adaptation of various development methodologies as long as they align with the certification requirements.

- Planning Process: The initial step involves creating plans that define how the development activities will comply with DO-178C objectives. Key planning documents include the Plan for Software Aspects of Certification, Software Development Plan, and Software Verification Plan.
- Development Process: This stage covers requirements gathering, system design, coding, and software integration. High-Level Requirements (HLRs) and Low-Level Requirements (LLRs) are specified and refined during this process, forming the foundation for subsequent design and coding. The process is designed to ensure that all software functionalities align with the defined safety criteria.
- Verification Process: Verification activities include reviews, analysis, and testing to ensure that the software meets all specified requirements and functions correctly under

various conditions. This includes testing for structural coverage, which involves verifying that all parts of the software code have been executed at least once during testing. Achieving complete code coverage is particularly crucial for DAL A and B software, where even a minor error could lead to catastrophic outcomes.

Traceability and Documentation

Traceability is another critical element in DO-178C, ensuring that each requirement is traceable through all stages of development and testing. This means that every high-level requirement must be mapped to corresponding low-level requirements, design elements, and verification activities. Such traceability ensures that changes in requirements are consistently reflected throughout the software lifecycle, supporting bidirectional traceability and facilitating thorough reviews.

The standard mandates the generation of several key documents as part of the certification evidence, including the Software Accomplishment Summary (SAS), Software Configuration Index (SCI), and test results. These documents provide a comprehensive record of compliance with DO-178C objectives and are essential for certification authorities to evaluate the safety and readiness of the software.

Application of DO-178C to UAVs

Although originally developed for manned aircraft, the application of DO-178C has been extended to UAVs. This shift was driven by regulatory developments, such as FAA Order 8130.34D [21], which introduced certification requirements for UAV software. The adoption of DO-178C for UAVs reflects the growing importance of ensuring that autonomous systems operate safely in civilian airspace.

The certification of UAV software presents unique challenges, including the need for the software to manage complex, real-time decision-making processes autonomously. This makes the thorough verification and traceability emphasized by DO-178C particularly crucial in the UAV context. The methodologies and frameworks provided by DO-178C ensure that even as UAVs take on increasingly complex roles, their software meets the highest standards of safety and reliability.

Challenges in DO-178C Implementation

Despite its comprehensive framework, achieving compliance with DO-178C can be a demanding process. The standard's emphasis on extensive documentation, rigorous verification, and

the need for traceability between software artifacts often results in significant time and resource investments. Additionally, the interpretation of DO-178C guidelines can vary, making it challenging for software developers to align their processes with certification expectations consistently. This has led to a growing interest in exploring new methodologies and tools that can streamline the certification process while ensuring adherence to DO-178C's stringent requirements.

2.3 Formal Methods and Verification Approaches

2.3.1 Formal Methods

Formal methods [22] are a set of mathematically based techniques and tools used to specify, design, and verify complex systems. These techniques provide a rigorous framework for understanding the properties and behaviours of software and hardware, ensuring that they operate correctly according to their specifications. Formal methods differ from traditional testing methods in that they can prove the absence of certain types of errors, rather than just detecting their presence during test runs. This makes them particularly valuable for safety-critical systems, where failure could have catastrophic consequences.

Importance of Formal Methods in Verification

In the context of safety-critical applications like aerospace and automotive systems, formal methods offer a high degree of confidence in the correctness of software components [23]. This is crucial when even small errors can lead to significant risks, such as system failures or safety breaches. By using formal methods, engineers can provide proof that a system will behave as expected under all possible conditions that the formal model accounts for. This capability is especially critical in domains governed by stringent certification requirements, such as those outlined in the DO-178C standard.

Formal methods encompass a range of techniques, including model checking [8], theorem proving [24], and formal specification languages. These methods enable systematic exploration of system states and behaviours, identifying edge cases that may not be covered during traditional testing. This is particularly useful in complex systems like UAVs, where software must handle numerous operational scenarios, including unexpected or rare conditions.

Model Checking

Model checking [25] is one of the most widely used formal verification techniques and plays a key role in verifying the correctness of both hardware and software systems. It involves creating a model of the system and systematically exploring all possible states of that model to verify whether a given property holds true in each state. Properties are often expressed using temporal logics like Linear Temporal Logic (LTL) [26] or CTL [11, 27]. For instance, properties like *"the system should never reach a deadlock state"* can be verified automatically by checking all reachable states of the system model.

Model checking has become an essential tool in the verification of systems that are too complex for exhaustive testing. Its automated nature makes it accessible to engineers, as it does not require deep expertise in mathematical proofs but still offers the rigour needed for safety-critical applications. However, challenges such as the state-space explosion—where the number of states grows exponentially with the size of the system—can limit its applicability. Advances in symbolic model checking [28], and abstraction techniques [29] have been developed to address these challenges, making model checking more practical for industrial applications.

Theorem Proving and Formal Specifications

Theorem proving [24] is another technique within formal methods that involves using logic to construct a formal proof that a system adheres to its specifications. Unlike model checking, which is automated, theorem proving often requires significant expertise, as the user guides the proof process. This method is particularly useful when the properties to be verified are too complex to encode directly into a model, or when a high degree of customization is required in the verification process.

Formal specifications [30] serve as the foundation for both model checking and theorem proving, offering a precise language for describing the expected behaviours of a system. These specifications are often written using formal languages like Z [31], B-Method [32], or Alloy [33], which provide a structured way to define system properties. By precisely capturing system requirements and behaviours, formal specifications reduce ambiguity, making it easier to verify that the system's implementation meets its intended design.

2.3.2 Computation Tree Logic

CTL [11, 27] is a branching-time temporal logic that allows the specification of properties of systems that can evolve in different ways over time. It is particularly useful in formal

verification, especially for model checking, due to its expressive power in describing how system states change along various execution paths.

Syntax of CTL

The syntax of CTL [8] combines logical operators with temporal operators, enabling precise descriptions of system behaviours over time. The key path quantifiers—**"A"** (for all paths) and **"E"** (there exists a path)—alongside temporal operators like **"G"** (globally), **"F"** (eventually, or in the future), **"X"** (next), and **"U"** (until), provide a flexible language for specifying system properties. These operators allow CTL to express properties such as whether a condition will always hold, or whether it holds at some point in the future.

Formal Semantics of CTL

CTL uses a mathematical model called a Kripke structure to interpret system behaviours, which allows for precise descriptions of how a system evolves over time [8].The Kripke structure is a tuple $M = (S, R, L)$ where:

- S is a set of states.
- $R \subseteq S \times S$ is a transition relation, representing the possible state transitions.
- $L : S \to 2^P$ is a labeling function, where P is a set of atomic propositions, and $L(s)$ gives the set of propositions true in state s.

For a state s in M, the satisfaction relation $M, s \models \varphi$ indicates that φ holds true at state M. The semantics of the CTL operators are defined as follows:

- $M, s \models EX\varphi$: There exists a successor state s' such that $(s, s') \in R$ and $M, s' \models \varphi$.
- $M, s \models AX\varphi$: For all successor states s' such that $(s, s') \in R$, $M, s' \models \varphi$.
- $M, s \models EG\varphi$: There exists a path $\pi = s_0, s_1, ...$ starting at $s_0 = s$ such that φ holds at every state along the path.
- $M, s \models AG\varphi$: For every path π starting at s, φ holds at every state along the path.
- $M, s \models EF\varphi$: There exists a path π starting at s such that φ holds at some state along the path.

- $M, s \models AF\varphi$: For every path π starting at s, there exists some state along the path where φ holds.

To illustrate CTL's expressiveness, consider the following examples:

- $AG\ (request \rightarrow AF\ response)$: This specifies that on all paths, if a request occurs, it is eventually followed by a response.
- $EF\ (crash)$: There exists a path where the system eventually enters a state where a crash condition holds, representing a possible failure.

These examples demonstrate how CTL allows for the specification of both universal and existential conditions, making it highly suitable for verifying safety and liveness properties in concurrent systems.

These formal semantics allow CTL to effectively capture various aspects of system behaviour, making it a powerful tool in the field of formal verification. It is widely used in model-checking applications to verify whether a given system model satisfies the desired properties or specifications. By specifying both existential and universal quantifiers over paths, CTL provides a balance between expressiveness and computational efficiency, which is crucial for analyzing complex systems. This capability is particularly relevant in domains such as aerospace and autonomous systems, where rigorous verification is essential for certification.

2.4 Language and Requirement Specifications

2.4.1 Natural Language in Requirement Specification

NL is widely used for specifying software requirements due to its accessibility and ability to express complex ideas in a form that is easily understood by diverse stakeholders, including those without a technical background [34]. This makes NL a common choice for creating Software Requirement Specifications (SRS), as it allows end-users and clients to understand and validate the requirements effectively [35]. The intuitive nature of NL ensures that critical system functionalities and user needs are articulated clearly during the initial stages of system development.

However, the use of NL in requirement specifications presents significant challenges, primarily due to its inherent ambiguity [7]. Ambiguity occurs when a word or phrase can have multiple interpretations, leading to inconsistencies in how different stakeholders understand a requirement. Such misunderstandings can result in discrepancies between what stakeholders intend and what developers implement, potentially leading to costly revisions later in the

development process. This issue is particularly problematic in safety-critical systems, such as those used in aerospace, where misinterpretations in requirements could compromise system safety.

Additionally, NL-based requirements are often criticized for being vague or lacking in precision. Vagueness in requirement statements can leave gaps in understanding, making it challenging to derive clear, detailed behaviours for system implementation [36]. When requirements lack specificity, developers may need to make assumptions, which increases the risk of errors during the software development process.

To address these challenges, various methods have been developed to enhance the precision of NL requirements. One such approach is the integration of CNL, which retains the readability of NL while imposing structured rules that reduce ambiguity and vagueness [37]. In parallel, NLP tools have been applied to analyze NL requirements and identify potentially ambiguous terms, offering suggestions for more precise phrasing [38]. These techniques enable the use of NL for communication while ensuring that requirements remain suitable for detailed technical specifications and subsequent verification.

2.4.2 Controlled Natural Language

CNLs [39] are simplified versions of NLs that use a limited vocabulary and a set of clearly defined grammar rules to reduce ambiguity. CNLs have been developed to address the inherent challenges, posed by NL in technical documentation, particularly in the context of requirements engineering. By restricting language constructs, CNLs facilitate easier interpretation of requirements and ensure that requirements are clear and unambiguous.

CNLs aim to balance the flexibility of natural language with the precision needed for formal specifications, making them ideal for high-assurance domains like aerospace. By using a simplified vocabulary and grammar, CNLs ensure that requirements are interpreted consistently by all stakeholders, reducing the risk of misunderstandings that can occur with traditional, unrestricted NL requirements.

CNLs provide several key benefits in the context of software requirements specification [40]. These include improved clarity, reduction of semantic ambiguities, and easier automation of the verification process. By using a restricted subset of vocabulary and grammatical structures, CNLs ensure that each requirement is unambiguous and thus more amenable to automated analysis. This makes CNLs a suitable choice for generating requirements that align with stringent standards like DO-178C, which demands rigorous definition and traceability throughout the software lifecycle.

Moreover, CNLs help facilitate collaboration between technical and non-technical stakeholders. Since CNLs maintain a form that is understandable to non-experts while also being structured enough for technical interpretation, they enable effective communication throughout the software development process [41]. This characteristic is especially valuable when documenting requirements for safety-critical systems, where clarity and precision can directly impact the safety and functionality of the final product.

2.5 AI and Automation in Software Verification

2.5.1 Natural Language Processing

NLP [42] is a branch of artificial intelligence that deals with the interaction between computers and human languages. It focuses on enabling machines to process and analyze large amounts of NL data in a way that is both meaningful and useful. NLP combines computational linguistics with machine learning and deep learning techniques to allow computers to understand, interpret, and generate human language.

NLP techniques include tasks such as tokenization, syntactic parsing, named entity recognition, and sentiment analysis. For software engineering purposes, NLP can be applied to parse requirement documents, detect inconsistencies, and even translate certain aspects of NL into more formal representations [43]. For example, tokenization breaks down text into individual words or phrases, while parsing analyzes the grammatical structure of sentences. These techniques help in extracting meaningful information from the requirements, making it easier to translate them into logical expressions.

NLP's relevance to software verification lies in its ability to streamline traditionally time-consuming and error-prone tasks. By automating the initial analysis of requirements, NLP reduces the manual effort required from engineers and ensures a more consistent interpretation of specifications. This capability is particularly crucial in safety-critical domains like aerospace, where unambiguous requirements are necessary to meet stringent standards such as DO-178C. By enhancing the clarity and precision of requirement specifications, NLP contributes to more efficient verification processes, ultimately supporting the development of safer and more reliable software systems.

Building on the principles of NLP, LLMs represent a significant advancement, offering enhanced capabilities in interpreting and generating human language

2.5.2 Large Language Models

LLMs [44] are advanced types of artificial intelligence models designed to understand and generate human language by processing vast amounts of text data. Built upon deep learning architectures like transformers, LLMs can analyze linguistic patterns and generate coherent responses to a wide range of queries. Examples of well-known LLMs include GPT-4, BERT, and Mistral. These models have revolutionized the field of NLP by enabling computers to engage with human language in a nuanced and contextually aware manner.

LLMs are trained on extensive datasets that include books, articles, websites, and various other textual sources, allowing them to learn the complexities of language, including grammar, syntax, and even cultural references. The transformer architecture, which underpins these models, enables LLMs to focus on different parts of an input sequence when generating responses, allowing for a deep understanding of context and improved relevance in their outputs.

The ability of LLMs to interpret and generate complex textual descriptions makes them particularly valuable for tasks that require understanding the intent behind software requirements and translating that intent into formal verification models. This capability is crucial in domains like aerospace, where compliance with rigorous standards such as DO-178C demands precise and verifiable documentation throughout the software development lifecycle. By facilitating the transition from NL requirements to formal specifications, LLMs help streamline the certification process, ensuring that safety-critical systems are both accurate and reliable.

2.6 Chapter Summary

This chapter has introduced the fundamental concepts that underlie this research, from airworthiness certification standards like DO-178C to advanced methods for software verification. By exploring formal methods, CNLs, NLP, and LLMs, we have laid the groundwork for understanding the approaches proposed in this dissertation.

CHAPTER 3 LITERATURE REVIEW

3.1 Chapter Overview

The purpose of this chapter is to provide a comprehensive overview of existing research related to the key contributions of this thesis. Each section of this chapter is designed to align with a specific research contribution, offering a focused discussion on the challenges, methodologies, and tools that form the foundation of this study.

3.2 Review of Integrating Formal Methods with Automated Verification Tools

The development of software for UAVs necessitates strict adherence to safety and certification standards due to the criticality of these systems and the potentially severe consequences of software failures. The DO-178C standard serves as a key framework for ensuring software reliability in airborne systems, offering detailed guidelines for verification, validation, and traceability throughout the software lifecycle [2]. Evolving from DO-178B, DO-178C introduces refinements and supplements to accommodate advancements in software engineering practices, such as model-based development (DO-331) and formal methods (DO-333). These updates address the need to manage the growing complexity of modern avionics software and enhance the efficiency and reliability of the certification process. Compliance with DO-178C presents particular challenges in UAV systems, given their increased reliance on software for complex functions like autonomous navigation and collision avoidance [4] [5].

3.2.1 Formal Methods and Their Role in DO-178C Compliance

Formal methods have long been recognized for their ability to provide rigorous, mathematically grounded verification of software properties like correctness, safety, and reliability. They are particularly valuable within the DO-178C framework for their ability to exhaustively verify software behaviour against its specifications. Studies such as [45] and [46] underscore the importance of formal methods in early error detection and precise requirements definition. This rigour is essential in UAV software, where adherence to safety requirements is paramount for functions like autonomous navigation.

Other studies, such as [47] and [48], demonstrate the application of model checking and abstract interpretation to verify specific avionics system components, offering high-assurance evidence that is crucial for meeting DO-178C certification standards. These approaches

provide a degree of rigour that surpasses traditional testing methods, ensuring thorough verification of system behaviours. The potential of formal methods in UAV systems is further emphasized in [46], which applies model checking to validate compliance with aviation rules, thereby offering a systematic pathway to certification for autonomous operations. Despite these advantages, the substantial expertise and manual effort required to implement formal methods often restrict their use in larger, more complex projects, underscoring the need for their integration with automated tools to enhance scalability and efficiency.

3.2.2 Advancements in Model-Based Design (MBD)

MBD has become popular in avionics software development due to its ability to accelerate the development process which may streamline compliance with rigorous standards like DO-178C. The study [49] discusses how MBD supports compliance by enabling traceability across software life cycles, which is especially critical for DO-178C. Additionally, the work in [50] introduces domain-specific modelling languages that simplify the complex requirements of DO-178C, improving traceability and reducing development complexity.

The applicability of MBD in avionics is further explored in [51], which evaluates how various MBD approaches support the development and certification of avionics software. The study highlights the effectiveness of frameworks in reducing complexities and ensuring that model-based methods are aligned with the rigorous documentation and traceability requirements of DO-178C. It emphasizes the advantages of using models as primary artifacts throughout the software lifecycle to facilitate compliance.

In [6], a practical case study is presented, focusing on the development of a landing gear control system using a methodology tailored for compliance with DO-178C and its supplements, DO-331 and DO-332. The study provides a detailed account of how a requirements specification and design were constructed, using model-based techniques to ensure that the system's architecture and low-level requirements were fully aligned with DO-178C guidelines. It offers insights into the challenges encountered during the development process and demonstrates the practical application of MBD to meet certification standards.

3.2.3 Automated Verification Tools for DO-178C Compliance

Automated tools play a vital role in reducing the cost and effort required for DO-178C compliance. Authors in [13] present SHERLOCK, an automated traceability tool that supports continuous certification by ensuring that all trace links are maintained. The study [52] discusses a MATLAB-based tool that integrates model-based approaches with automation to

improve compliance with DO-178C/DO-331.

The paper [53] explores how automated testing can streamline compliance for software-defined radio systems, demonstrating that integrating automation with manual verification significantly enhances efficiency. Similarly, the paper [54] showcases how a combination of simulation and formal verification tools can automate evidence generation for complex UAV systems.

3.2.4 Integration of Formal Methods with Automated Tools

Combining formal methods with automated tools offers a holistic approach to managing the complexities of DO-178C compliance. This integrated approach is particularly valuable for UAV systems, where complex software functionalities demand rigorous verification. For example, [55] outlines a methodology that combines formal methods with qualified tools to facilitate the certification of intelligent UAV systems. By leveraging formal verification techniques like model checking, the approach ensures deterministic behaviour in critical functions, such as sense-and-avoid systems, while using model-based development to streamline the design phase. This structured integration provides the precision required for certifying autonomous operations.

Similarly, authors in [56] explore the potential of adapting Agile methodologies within the constraints of DO-178C. It discusses how Agile practices, such as iterative testing and continuous integration, can be harmonized with formal verification to improve the development process. This study highlights that while traditional safety-critical development practices may seem rigid, incorporating Agile methods alongside formal tools can enhance flexibility without compromising compliance standards.

Another approach [5] provides a case study on certifying adaptive learning systems using a combination of formal methods and qualified tools. This hybrid approach helps to address the challenges of verifying adaptive behaviours in UAVs, offering a viable path toward certification under the stringent requirements of DO-178C.

3.2.5 Comparative Analysis & Addressing Gaps

The methodologies used for achieving DO-178C compliance in UAV software development present a range of strengths and limitations. Formal methods, as discussed in [46] and [47], are known for their ability to deliver rigorous verification of complex behaviours, providing a high level of assurance in software correctness. However, their complexity, coupled with the need for specialized expertise, often limits their scalability in larger projects, posing a barrier

to widespread adoption.

In contrast, automated verification tools, such as those detailed in [52] and [13], provide significant benefits in terms of time and cost efficiency. These tools excel in managing repetitive verification tasks, facilitating faster compliance with DO-178C requirements. However, their capabilities may be less suited for addressing intricate verification challenges, particularly when deep, formal analyses are required to guarantee the correctness of complex software behaviours.

MBD, as discussed in [51] and [50], provides a more intuitive development process that simplifies traceability but may require careful integration with verification tools to ensure compliance. Balancing these methods in a hybrid approach often yields the best results, as suggested in [56].

The certification of adaptive and autonomous UAV systems introduces additional complexities, given their dynamic nature and the necessity for real-time performance and deterministic behaviour. As highlighted in [4], existing methodologies often struggle to meet the stringent real-time constraints imposed by DO-178C, particularly concerning memory management and timing analysis. These aspects are critical to the safe operation of autonomous systems in unpredictable environments, making compliance especially challenging for UAVs that rely heavily on software-driven decision-making.

The complexity of certification becomes even more pronounced when dealing with adaptive learning behaviours, where systems evolve based on operational data. The study in [5] illustrates the difficulties in certifying such systems, as they challenge traditional verification approaches designed for static behaviours. This highlights the need for novel methodologies that can effectively verify adaptive behaviours throughout the software lifecycle, ensuring that any evolution in system behaviour remains in compliance with the rigorous standards of DO-178C.

To address these challenges, the integration of formal verification with model-based simulations has been proposed as a potential solution. For example, [54] suggests that combining formal methods with simulation-based approaches can significantly strengthen the certification process, especially for systems where adaptive behaviours pose verification challenges. This approach allows for a thorough analysis of system behaviours early in the development process, helping to identify and mitigate risks before they become more difficult to address later. Nevertheless, the substantial time and expertise required for formal methods remain a barrier, emphasizing the need for early-stage incorporation of formal verification activities to ensure issues are addressed before they become entrenched.

Moreover, a significant gap identified in [51] is the lack of integrated solutions that comprehensively cover the entire avionics software lifecycle. This study emphasizes the importance of enhanced traceability and coverage analysis, which are critical for ensuring that each stage of software development aligns with DO-178C's requirements. This gap suggests a need for methodologies that provide holistic lifecycle support, from requirement definition through to coding and verification, particularly for systems with complex adaptive functionalities.

Our approach directly addresses these gaps by integrating formal methods with automated tools like the LDRA tool suite, aiming to achieve both the rigour of formal verification and the efficiency of automation. This approach leverages automated tools for handling routine verification tasks while utilizing formal methods for in-depth analyses where precision is paramount. By doing so, we offer a balanced strategy that supports the certification of adaptive UAVs, maintaining evidence generation and traceability throughout the entire software lifecycle and ensuring a more streamlined certification path.

Overall, the literature highlights significant advancements in the use of formal methods and automation for achieving DO-178C compliance, but also reveals persistent gaps, especially concerning the certification of adaptive systems. Addressing these gaps will require ongoing innovation in the integration of verification methodologies and tools, as well as validation through real-world case studies. The trend in recent research points towards the development of integrated solutions capable of managing the increasing complexity of safety-critical software while adhering to stringent standards like DO-178C. As UAVs continue to evolve with advanced capabilities, the demand for such comprehensive methodologies will grow, driving further advancements in this domain.

3.3 CNLs for Reducing Requirement Ambiguity

The challenge of ambiguity in NL requirements has been a significant focus in the field of software Requirements Engineering (RE). This section explores the complexities of using NL for requirement specifications, the development and potential of CNLs as a solution, and recent advancements in automating the requirements specification process.

3.3.1 Ambiguity in NL Requirements

NL has long been the preferred medium for specifying software requirements, primarily due to its accessibility and ease of understanding for stakeholders from diverse backgrounds [34]. However, the inherent ambiguity of NL presents major challenges in the context of requirements engineering, especially for safety-critical domains like aerospace, where precise inter-

pretations are crucial to ensure compliance with rigorous standards such as DO-178C.

However, the use of NL introduces significant challenges, primarily due to its inherent ambiguity. In their comprehensive survey on ambiguity in NL-based requirements [7], authors emphasize that misinterpretations can result in inconsistencies between the intended and actual implementation of requirements. The study emphasizes the critical need for strategies that can manage these ambiguities without sacrificing the ease of understanding offered by NL.

The complexity of NL arises from issues such as polysemy, where a single word can have multiple meanings, and syntactic ambiguity, where the structure of sentences allows for multiple interpretations [40]. These ambiguities often necessitate time-consuming clarifications between stakeholders and engineers, which can delay the software development process. The research [57] highlights the effort required to translate ambiguous NL requirements into more structured forms using the Semantic Business Vocabulary and Rules (SBVR) standard. While SBVR offers a structured approach, its adoption often involves manual intervention, highlighting the need for more automated solutions to manage ambiguity.

3.3.2 Development of CNL

CNLs have emerged as a promising approach to mitigate the ambiguity of NL requirements. These languages restrict the grammar and vocabulary of NL to create a more structured and less ambiguous version that remains user-friendly [37]. In their study on building a CNL named *Rimay* [58], authors emphasize that CNLs strike a balance between the accessibility of NL and the precision required for effective communication of software requirements in domains like finance.

Expanding on this, The study [59] introduced a model-driven and template-based approach to CNLs, focused specifically on minimizing ambiguity in requirements specifications for complex systems. Their approach leverages Model-Driven Engineering (MDE) to create structured templates that guide the formulation of requirements, using controlled language to further reduce interpretative variability. By integrating templates within CNLs, this study provides an adaptable framework that aligns CNLs closely with domain-specific requirements, enabling more precise and consistent language use.

CNLs are designed to maintain the intuitive nature of NL while introducing a degree of structure that makes the requirements easier to parse and verify. For example, the work [60] further explores the value of tailoring CNLs to specific domains, such as space projects. They argue that domain-specific adaptations are necessary to ensure that CNLs remain

effective in reducing ambiguities. Each domain has unique terminologies and requirements, making a one-size-fits-all approach less practical. However, most existing CNLs are tailored to particular industries and lack a generalizable framework that could be adapted for use with stringent standards like DO-178C.

Further, the literature notes a gap in integrating automated tools that ensure adherence to CNL rules during the requirements specification process. While CNLs provide a framework for writing unambiguous requirements, automated tools can help verify compliance with these rules, ensuring consistency across large-scale projects. The lack of such integration points to research opportunities, particularly in developing a rule-based system that verifies adherence to CNL principles during the documentation process.

3.3.3 Automation in Requirements Specification and Verification

Recent advances in NLP and automation have introduced tools that can assist in the generation and analysis of software requirements, offering potential solutions to the challenges posed by NL requirements. For example, *ReqGen* tool [61] leverages domain-specific ontologies to generate NL-based requirements automatically, reducing the time and effort required during the initial stages of requirements elicitation. Similarly, studies like [62] explore how NLP techniques can be used to parse requirements documents and identify potential ambiguities. These studies suggest that automating the initial analysis of requirements can streamline the process, making it more efficient.

However, while these tools excel at automating requirements generation and analysis, they do not inherently ensure that the generated requirements meet the precision and verifiability standards required by industry standards like DO-178C. In this regard, the paper [63] advocates for the use of a CNL approach to mitigate issues such as ambiguity and inconsistency that are common in NL-based requirements. The paper also explores the integration of RE with MDE processes, which could indirectly support consistency and traceability in various domains, including those with high safety requirements. The emphasis on integrating these methodologies underscores the importance of precision in requirement specifications and highlights how structured approaches can enhance the quality of requirements.

Additionally, the development of a Wiki-based tool prototype, as discussed in the study [63], aims to validate the effectiveness of CNL-based approaches for improving the quality of requirements. This validation process could be particularly relevant to safety-critical systems, where ensuring the reliability of requirements is crucial. While the paper does not focus specifically on automated, rule-based systems, the principles it discusses—such as reducing ambiguity through controlled language—align with broader efforts to enhance the rigour and

precision of requirements for safety-critical applications.

These studies collectively highlight the promise of automation in requirements specification but also underscore the need for integrating such tools with mechanisms that ensure precision and adherence to industry-specific standards like DO-178C. This gap presents an opportunity for research aimed at developing a CNL framework supported by rule-based verification, specifically designed to address the stringent needs of aerospace software certification.

3.3.4 Synthesis and Addressing Gaps

The literature reveals a consensus on the benefits of CNLs for reducing ambiguity in NL requirements and the potential of automation to enhance the requirements engineering process. Studies such as those by [58] and [60] emphasize the effectiveness of domain-specific CNLs, while others like [61] highlight the role of NLP in automating requirement generation. However, a clear gap exists in integrating these two approaches to create a robust, automated framework that ensures adherence to CNL rules while maintaining the flexibility of NL.

One notable finding is that while CNLs can be effective in reducing ambiguity, they often require manual oversight to ensure adherence to their structured rules, which limits their scalability in complex projects. Automated systems that focus on compliance with CNL guidelines could address this limitation by ensuring that every requirement meets predefined standards of clarity and precision without needing extensive manual review.

Critical Analysis and Research Gaps

The existing literature highlights significant progress in the application of CNLs and automation in reducing the ambiguity of software requirements. However, these approaches often remain isolated, focusing either on domain-specific adaptations or on the automation of requirements generation without ensuring compliance with industry-specific standards like DO-178C.

A critical gap identified is the absence of a comprehensive framework that combines the clarity of CNLs with the efficiency of automated tools, tailored specifically for the needs of the aerospace sector. Additionally, many studies, such as those focusing on NLP tools, emphasize the initial stages of requirements generation but lack mechanisms for ongoing verification and traceability, which are crucial for compliance in safety-critical domains.

This research aims to address these gaps by developing a CNL framework specifically designed for DO-178C compliance, combined with a rule-based verification system. This system automates the evaluation of requirements, ensuring they adhere to the precision and traceability

standards required in aerospace. By integrating CNLs with automation, the research offers a novel approach that balances the need for clarity with the demands of large-scale, safety-critical software projects.

The literature review highlights the key challenges in using NL for software requirements, particularly its inherent ambiguity, and explores how CNLs have emerged as a viable solution. Despite the progress in CNL development and NLP-based automation, significant gaps remain in creating generalizable, scalable frameworks suitable for DO-178C-compliant software. Our research seeks to bridge these gaps by developing a CNL tailored to DO-178C requirements and integrating it with a rule-based verification system. This approach aims to contribute to more efficient and reliable certification processes in aerospace, ensuring that software requirements remain both precise and verifiable.

3.4 Automating NL-to-CTL Translation with AI Models

This section reviews the key methodologies, models, and datasets that have emerged to address the complex process of translating NL requirements into CTL representations, a task that is particularly challenging in safety-critical domains. By detailing the progression from early structured approaches to contemporary AI-driven solutions, this section highlights the advancements that have been made and identifies remaining gaps that motivate the development of the Natural2CTL dataset.

3.4.1 NL-to-CTL Translation and Motivation for Natural2CTL Dataset

The transformation of NL requirements into formal specifications is pivotal in ensuring that software systems, particularly those in safety-critical domains such as aerospace and autonomous systems, operate reliably and comply with rigorous verification standards. CTL stands as a crucial formalism in model checking for concurrent and reactive systems due to its unique branching temporal structure [64]. However, the translation of NL requirements into CTL remains challenging due to the inherent ambiguity of NL and the specialized temporal and branching semantics required in CTL [65].

Significant progress has been made in transforming NL requirements into various formal logics. One notable approach, discussed in [14], demonstrates techniques for converting English comments into CTL properties to streamline verification processes in hardware design. However, such methodologies often struggle to handle the temporal constraints required by CTL, a challenge also noted in [66], which examines how informal NL requirements can be structured into formal specifications.

Other studies focus on language models to aid in translating NL to simpler forms of temporal logic, such as First-Order Logic (FOL) and LTL. For instance, [67] explores the synthesis of LTL formulas from NL requirements but identifies persistent issues in capturing the nuances of temporal dependencies. While LTL captures linear temporal progressions effectively, it lacks the branching capability inherent in CTL, which limits its applicability to more complex, concurrent system verifications.

Despite these advancements, a critical gap remains: the absence of a systematically organized dataset that pairs NL requirements with corresponding CTL representations. Datasets are crucial for training AI models to handle the subtleties of NL-to-formal logic translation, particularly in temporal logics. The SpecNFS study [68] highlights the importance of annotated datasets in translating NL specifications into formal representations. SpecNFS demonstrates that even with state-of-the-art language models, achieving high accuracy in semantic parsing for formal verification tasks remains challenging, underscoring the need for domain-specific, annotated data to drive research and improve model efficacy.

To address these limitations, we developed the Natural2CTL dataset, a structured resource integrating CTL's rigour with the accessibility of NL. This dataset compiles NL requirements with their CTL equivalents, supporting both rigorous validation and structured representation within translation processes. Natural2CTL serves as a foundational resource for AI-driven NL-to-CTL translation, facilitating advancements in RE and formal verification by providing the standardized data necessary for robust, adaptable models capable of addressing the complexities of NL-to-CTL translation.

By introducing the Natural2CTL dataset, we aim to catalyze progress in AI-driven formal verification, overcoming key limitations in current resources and methodologies. With the necessary data infrastructure, Natural2CTL provides a pathway to more precise and adaptable verification systems capable of handling complex temporal structures, advancing the capabilities of automated verification within safety-critical industries.

3.4.2 Early Approaches to NL-to-Formal Logic Translation

Early efforts in NL-to-formal logic translation focused primarily on structured, template-based methods. Tools like PROPEL used property pattern templates to simplify the NL-to-formal translation process, relying on predefined structures to make translation more accessible [69]. While useful in controlled scenarios, these template-based systems faced limitations due to their rigidity, which restricted the range of NL expressions they could handle.

Subsequent advancements explored the potential of NLP techniques to offer greater flexibility.

Systems such as ARSENAL utilized domain-specific semantic parsing to create formal models from NL requirements [70]. Similarly, methodologies that generated CTL properties from comments in Hardware Description Language (HDL) code facilitated verification in hardware design contexts [14]. Despite these innovations, such approaches often remained constrained by domain-specific requirements, relying on pre-defined templates or ontologies that limited their broader applicability.

3.4.3 The Role of LLMs in Formal Verification

Recent years have seen the emergence of LLMs, which have opened new pathways for automating NL-to-formal specification translation. LLMs have proven effective at translating NL into simpler formal logics, including FOL and LTL. For instance, the study in [71] explored the potential of LLMs for *"autoformalization"*, demonstrating GPT-4's capability to translate NL mathematics into formal representations. Another study [72] applied GPT-4 to automate SystemVerilog assertions for Register Transfer Level (RTL) design verification, highlighting the applicability of LLMs in hardware verification contexts.

These studies illustrate LLMs' potential for enhancing formal verification, although primarily with simpler logics like FOL and LTL. CTL, however, poses unique challenges due to its branching-time structure, which accommodates concurrent and reactive systems—an aspect that LTL cannot fully capture. This distinction underscores the additional complexity involved in NL-to-CTL translation, particularly in handling temporal operators and alternative future states.

3.4.4 Recent Advances in NLP-driven Formal Specification

Significant progress has been made in NLP-driven formal specification tools that go beyond earlier approaches. Tools like *nl2spec* and *Lang2LTL* have advanced automated translation of NL into temporal logics such as LTL through fine-tuning models like T5. For example, *nl2spec* focuses on converting unstructured NL into LTL formulas with user guidance, providing a domain-agnostic and flexible methodology for specification generation [73]. *Lang2LTL* applies similar techniques to Signal Temporal Logic (STL), demonstrating success in generating temporal logic specifications for control systems and robotics [74].

While these tools have demonstrated efficacy in temporal logic translation, their scope remains limited to linear temporal logics like LTL and STL, which lack the branching capabilities required for CTL. Our research, therefore, seeks to address the distinctive challenges of NL-to-CTL translation by focusing on capturing both temporal and branching semantics—a

requirement essential for systems involving concurrent or reactive behaviours.

A recent study [75] introduces SYNTHTL, a method that integrates LLMs with model checkers to translate NL specifications into LTL. Their approach decomposes complex NL statements into smaller components and incorporates human verification (oracle) during the translation process to ensure accuracy. While SYNTHTL highlights the potential of LLMs for automating LTL translations, their method focuses on LTL, which is inherently less suited for branching-time scenarios that CTL handles. Moreover, SYNTHTL requires human oversight in verifying the generated LTL formulas, whereas our approach aims to minimize human intervention by refining models that can generate syntactically and semantically accurate CTL formulas that are directly applicable to model-checking tools. Thus, our research provides an automated approach to handling the additional complexity of CTL, demonstrating the feasibility of LLMs for high-assurance verification with minimal human oversight.

Other studies have combined rule-based and machine-learning approaches to enhance precision. For instance, the hybrid system for assertion generation from NL specifications [76] combines both approaches to generate SystemVerilog assertions from NL. While successful in its domain, this system is not designed to handle the intricacies of CTL.

3.4.5 Addressing Challenges in CTL Translation

Despite advancements in NLP and LLMs, translating NL into CTL remains a challenging task due to NL's variability and the difficulty of ensuring syntactic precision in formal languages. The study [65] emphasized the importance of flexible translation methods to handle domain-specific requirements and the complexity of adapting templates for different logics. Authors, in their study on probing LLMs for understanding temporal expressions [77], found that while LLMs show promise, they still struggle with accurately capturing complex temporal operators, a key requirement in CTL.

The work in [78] and [74] highlighted the need for more sophisticated LLM-driven approaches to handle temporal logic, specifically addressing gaps in translating NL to STL and LTL. These approaches demonstrated significant progress in automated formal specification, but CTL remains less explored in this context.

3.4.6 Synthesis and Research Gaps

This review highlights substantial advances in automating NL-to-formal logic translation, especially with LLMs, rule-based models, and hybrid approaches that improve accuracy in translating NL into structured logic representations. Yet, significant gaps persist, particularly

in transitioning from simpler temporal logics like LTL to branching temporal logics like CTL. Existing datasets and methodologies are largely tailored to linear logics, underscoring the necessity for specialized datasets, such as Natural2CTL, and more sophisticated translation models.

Our research addresses these challenges by providing a unique contribution through the Natural2CTL dataset, specifically designed to support AI-driven CTL translation. The dataset supplies annotated examples essential for training models capable of managing the nuanced demands of CTL's branching and temporal structures. This resource, coupled with advanced LLMs and rigorous evaluation metrics, establishes a foundational framework for accurate, autonomous NL-to-CTL translation, bridging the gaps between RE, AI, and formal verification.

3.5 Chapter Summary

In this chapter, we provided a comprehensive review of the literature surrounding methodologies and tools essential for achieving DO-178C compliance in safety-critical software systems. Key areas explored include the integration of formal methods with automated tools, advancements in MBD, the role of CNLs in reducing ambiguity in software requirements, and challenges in translating natural language requirements into formal representations. We identified gaps in the scalability and generalizability of existing approaches, particularly in the context of adaptive UAV systems, and highlighted the need for more integrated solutions that balance formal rigour with automation efficiency. This foundation paves the way for the next chapter, where we define the research objectives and outline the thesis organization.

CHAPTER 4 METHODOLOGY

4.1 Chapter Overview

This chapter defines the core objectives and structural layout that guide this thesis, building on the foundational context and challenges introduced in Chapter 1. The main goal is to address complexities in achieving DO-178C compliance for UAV software, with a focus on improving verification processes, reducing requirement ambiguity, and facilitating the translation of NL requirements into formal specifications.

To achieve these goals, the chapter presents targeted research objectives developed in response to the key research questions identified in Chapter 1. These objectives are structured around critical challenges, including integrating formal methods with automated verification tools, reducing ambiguities in NL requirements, and leveraging AI-driven models for translating NL into formal language. Each objective introduces a methodical approach to address these issues and underpins the primary contributions of this thesis, such as the development of a CNL framework, the Natural2CTL dataset, fine-tuned LLMs, and an evaluation of various approaches for NL-to-CTL conversion.

4.2 Methodology Overview

This section provides an overview of the methodologies adopted to achieve the research objectives and answer the research questions. Designed to address the critical challenges of DO-178C compliance for UAV software, the methodology focuses on enhancing verification processes, reducing requirement ambiguity, and facilitating NL-to-formal language translations. Each objective aligns with a key methodological approach, ranging from the integration of formal methods to the development of advanced AI-based translation techniques.

The methodology is organized into four primary objectives, each with specific sub-objectives that collectively support the overall goals of the thesis. These objectives and their associated contributions are outlined below and visually summarized in Figure 4.1.

1. **Objective 1: Integration of Formal Methods with Automated Verification Tools**

   This objective explores integrating formal methods with automated tools to support verification, validation, and traceability throughout the software lifecycle, enhancing

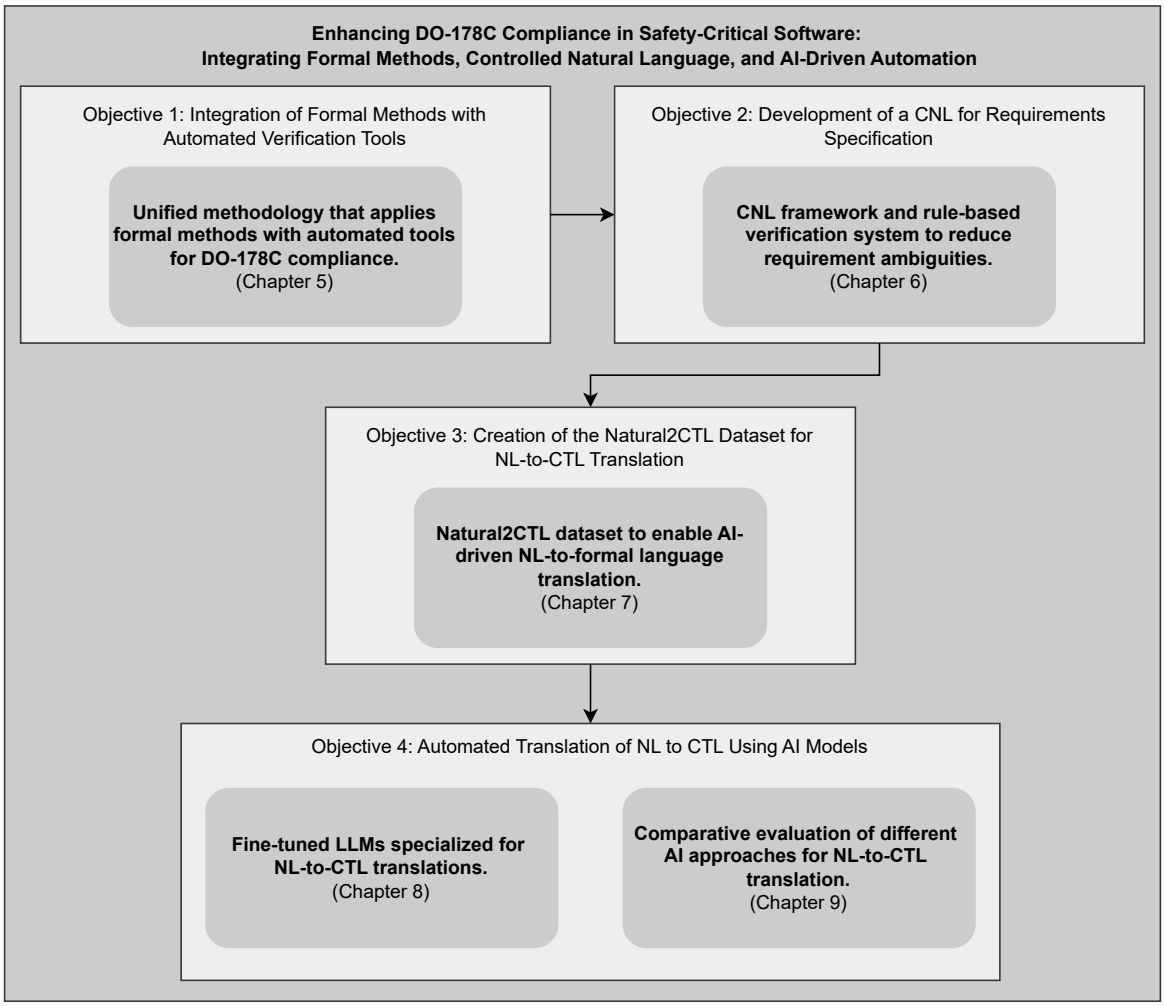


Figure 4.1 Overview of the thesis methodology.

the rigour and efficiency of DO-178C compliance processes. The methodology here focuses on:

- Formal Methods and Automated Tools: Investigating how formal verification methods (such as model checking) can be combined with tools like the LDRA suite to streamline compliance.
- Case Study Validation: The integration is demonstrated using a UAV collision avoidance system as a practical case study, showcasing the real-world applicability and compliance benefits of this approach.

The details of this methodology and its results are covered in Chapter 5.

> ***Contribution 1:*** A unified approach to applying formal methods, combined with automated tools, within DO-178C frameworks, reducing manual verification effort while ensuring compliance.

2. **Objective 2: Development of a CNL for Requirements Specification**

   To address the ambiguity inherent in NL requirements, this objective focuses on developing a CNL tailored to DO-178C needs, providing structured linguistic rules that enhance clarity and verifiability. This objective involves:

   - Linguistic Constraints and Rules: Defining a set of grammatical constraints that standardize NL requirements, ensuring that each specification is unambiguous and meets verification standards.
   - Rule-Based Verification System: Implementing a rule-based system that automatically verifies CNL requirements against DO-178C standards, supporting compliance checks.

   The CNL framework and rule-based verification system represent a significant step toward reducing ambiguity in safety-critical requirements, as discussed in Chapter 6.

> ***Contribution 2:*** *A CNL framework and a verification system reducing ambiguities in DO-178C requirements, facilitating compliance verification.*

3. **Objective 3: Creation of the Natural2CTL Dataset for NL-to-CTL Translation**

   This objective aims to bridge the gap between NL requirements and formal representations, creating a dataset that supports the training of AI models for translating NL requirements into CTL. The methodology here includes:

   - Data Collection and Annotation: Compiling and annotating NL-CTL requirement pairs to build a robust dataset for training.
   - Evaluation and Validation: Conducting expert reviews, inter-rater reliability assessments, and case studies to ensure the accuracy and applicability of the dataset.

   The Natural2CTL dataset is detailed further in Chapter 7.

> ***Contribution 3:*** A dataset enabling advanced training for AI-based NL-to-formal translations, supporting automation in formal verification processes.

4. **Objective 4: Automated Translation of NL to CTL Using AI Models**

   This objective seeks to develop AI-driven approaches for translating NL requirements into CTL, aiming to streamline the formal verification process for DO-178C compliance. To achieve this, two sub-objectives are pursued, each focused on distinct AI methodologies to enhance translation accuracy and efficiency:

   - **Fine-Tuning LLMs for NL-to-CTL Translation:** This sub-objective explores fine-tuning advanced LLMs, such as Mistral, on the Natural2CTL dataset to adapt these models specifically for the domain of NL-to-CTL translation. Fine-tuning enhances the model's ability to accurately interpret and translate NL requirements into CTL, addressing domain-specific nuances and improving overall translation precision.

     Details of this approach and its evaluation are provided in Chapter 8.

     > ***Contribution 4:*** A suite of fine-tuned LLMs specialized for translating NL requirements into CTL, contributing to higher accuracy and reliability in automated formal verification.

   - **Evaluation of Alternative Translation Techniques:** In addition to fine-tuning LLMs, this sub-objective investigates the effectiveness of various alternative methodologies for NL-to-CTL translation, including few-shot learning with GPT-4 and the hybrid BERT-GPT approach. This sub-objective involves a comparative analysis to assess the accuracy, scalability, and feasibility of each method.

     This study is discussed in Chapter 9, serving as a practical guide for selecting optimal AI methods for formal verification tasks.

     > ***Contribution 5:*** Comparison of different AI approaches capable of translating requirements from NL into CTL, offering valuable insights into the most effective models and approaches.

4.3 Chapter Summary

In this chapter, we presented a detailed overview of the thesis, outlining the research objectives formulated in response to our research questions. We highlighted the main contributions and briefly summarized the studies conducted. We provided a guide to the chapters where each contribution is detailed, establishing a clear roadmap for the reader.

CHAPTER 5 INTEGRATING FORMAL METHODS AND AUTOMATION FOR DO-178C COMPLIANCE

5.1 Chapter Overview

The development of safety-critical software for UAVs requires adherence to rigorous standards, with DO-178C as a primary regulatory framework to ensure safety, reliability, and consistency in airborne systems. However, achieving compliance with DO-178C presents challenges, particularly given the complexity and adaptive capabilities of modern UAV systems, which often demand high levels of precision, traceability, and verification across multiple phases of the software lifecycle. This chapter introduces an integrative methodology that combines formal methods with automated verification tools to address these challenges, specifically tailored to facilitate DO-178C compliance for UAV software. The work presented in this chapter was conducted in equal collaboration with my colleague *Henrique Amaral Misson*.

In previous chapters, we examined the foundational aspects of DO-178C and explored various compliance methodologies. This chapter [1] builds on these insights by proposing a structured approach that leverages the mathematical rigour of formal methods alongside the efficiency and scalability of automated tools. Formal methods offer robust techniques for specifying and verifying critical system behaviours, while automated tools enhance traceability and provide continuous validation throughout the development lifecycle. Together, these methods form a unified framework that is not only capable of addressing DO-178C requirements but is also adaptable to the specific demands of UAV software.

The core objective of this chapter is to demonstrate how this integrated approach can streamline the certification process by ensuring that software artifacts are both verifiable and traceable from the definition of initial requirements through the final system integration. This approach is particularly significant for UAV systems, where safety concerns necessitate early error detection, rigorous validation, and efficient certification evidence generation. By combining formal methods with automated tools, this methodology addresses gaps in traditional compliance approaches, offering a scalable solution that balances rigour with practical efficiency.

[1] Part of the content of this chapter is submitted for publication as: Zrelli, R.*, Misson, H. A.*, Kamkuimo, S., Ben Attia, M., de Magalhães, F. G., & Nicolescu, G.: "Integrating Formal Methods and Automated Tools for DO-178C Compliance in UAV Software". In: *Journal of Aerospace Information Systems (JAIS)* (2024). (*: Equal Contribution)

5.2 Study Design and Objective Setting

This section outlines the structured approach taken to integrate formal methods with automated tools, specifically for advancing DO-178C compliance in UAV software systems. Recognizing the unique challenges of ensuring safety in autonomous UAV operations, the study adopts a methodology that leverages both the mathematical rigour of formal methods and the efficiency of automated verification tools. By using this approach, we aim to mitigate potential safety risks and enhance the robustness of requirements, design, and implementation phases.

The core objectives guiding this study are to develop an approach that not only enhances the precision and clarity of system requirements but also facilitates traceability and compliance throughout the software lifecycle. To evaluate the effectiveness of this integrated approach, we apply the methodology to a real-world UAV Collision Avoidance System (CAS), thus demonstrating its practical value in a safety-critical context.

Figure 5.1 provides an overview of the study design, illustrating the stages from initial formal specification to final integration and evaluation within a real-world application scenario. This framework is structured to address the critical areas of DO-178C compliance, focusing on accurate requirement specification, automated traceability, and continuous verification.

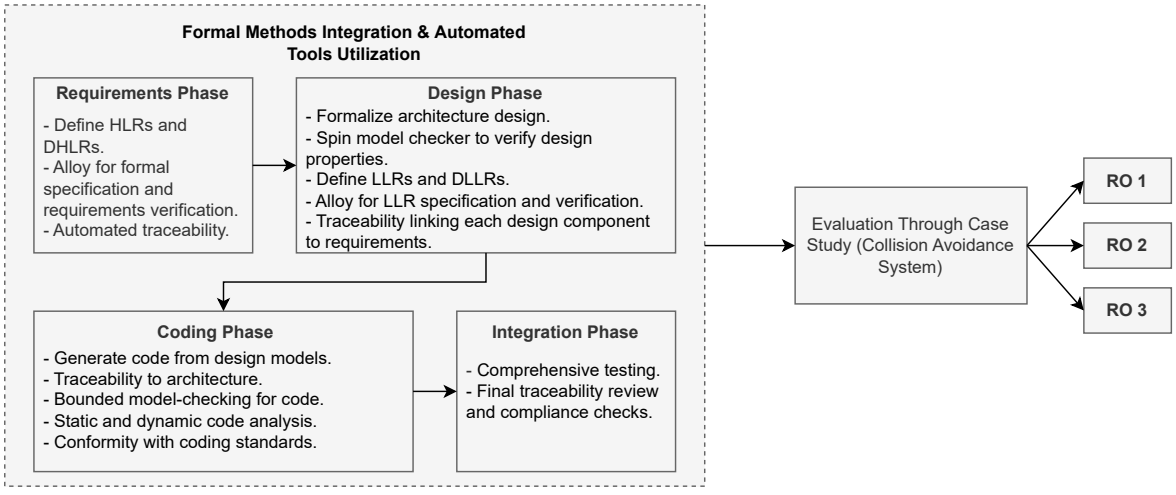


Figure 5.1 Schematic diagram of the study.

The primary objectives are as follows:

1. **To integrate formal methods for rigorous specification and verification of UAV software development (RO1)**: This objective involves leveraging formal

methods to develop unambiguous, mathematically precise specifications. Formal specification and verification are particularly critical for capturing safety requirements and ensuring the deterministic behaviour of UAV systems.

2. **To automate verification tasks to reduce manual effort and support traceability (RO2)**: By utilizing automated tools, this objective aims to streamline the verification process, enabling continuous monitoring and traceability across all development stages.

3. **To evaluate the proposed methodology through a case study of a collision avoidance system (CAS) (RO3)**: A case study on a UAV CAS will serve to validate the effectiveness of this integrated methodology, assessing its impact on both compliance efficiency and safety outcomes.

Each objective is strategically aligned with the goals of DO-178C to ensure that the methodology provides a scalable, certifiable solution for safety-critical UAV applications. Through this study, we aim to contribute a unified approach that addresses the challenges of DO-178C compliance in increasingly complex UAV software environments.

5.3 Methodology

5.3.1 Proposed Methodology

The proposed methodology integrates formal methods with automated verification tools to ensure a comprehensive, traceable, and certifiable software development process for UAV systems, adhering to the rigorous standards set by DO-178C. The methodology's primary objective is to systematically address the key challenges associated with developing safety-critical software for UAVs by ensuring that each phase of the software development lifecycle—from requirements through integration—meets DO-178C compliance requirements.

The core of this methodology lies in combining the mathematical rigour of formal methods with the efficiency and automation provided by specialized verification tools. This hybrid approach ensures that each phase of the development process is both verifiable and traceable, producing the necessary certification artifacts required by aviation authorities. The methodology systematically spans the requirements, design, coding, and integration phases, ensuring comprehensive coverage of DO-178C objectives, from the initial specification to the final system integration.

Formal Methods for Rigorous Specification and Verification

The use of formal methods in this methodology ensures that all system behaviours are specified with mathematical precision, reducing the likelihood of ambiguity in the requirements and design phases. Formal methods are applied to define and verify key aspects of the system through the following techniques:

- Formal Specification: System requirements (SRATS) and software requirements (HLRs and LLRs), are modelled using formal languages such as Alloy. These formal models enable the precise expression of system properties, eliminating ambiguities that could lead to misinterpretations during implementation and ensuring consistency between the specifications.
- Formal Verification: The methodology employs model-checking techniques to verify the correctness of software applications against the specified requirements. Tools such as SPIN and ESBMC are used to perform an exhaustive exploration of the system's state space, ensuring that the specified properties hold across all possible execution paths. This is particularly valuable for detecting deadlocks, invalid states, coding errors and violations and unanticipated interactions between subsystems at an early stage of development.

The formal methods employed ensure that the critical components of the UAV software are verified not only against functional requirements but also against safety properties, such as freedom from deadlock, and ensuring correctness during the requirements and design phases which are essential for compliance with DO-178C.

Automated Tools for Continuous Verification and Traceability

To complement formal methods, the methodology incorporates automated verification tools that facilitate the management of requirements, execution of static and dynamic code analysis, and the creation of end-to-end traceability across the software development lifecycle. These tools play a pivotal role in ensuring compliance with DO-178C objectives by reducing manual effort, enhancing accuracy, and supporting the generation of certification artifacts. The use of qualified tools ensures that all phases, from requirements specification to final integration, are continuously monitored for consistency, traceability, and adherence to standards.

Automated tools provide several key benefits in the context of safety-critical software development:

- Automated Traceability: Traceability across the software lifecycle is established and maintained, ensuring bidirectional links between requirements, design elements, source code, and test artifacts. This traceability is essential for demonstrating compliance with DO-178C, as every software artifact must be directly linked to its originating requirement. Automated traceability minimizes the risk of inconsistencies and ensures that updates in requirements propagate through all development stages, thereby maintaining alignment with the certification objectives.

- Static and Dynamic Analysis: Automated verification using static and dynamic analysis tools ensures that the code adheres to coding standards and meets the design requirements. This is achieved through automated code reviews, dynamic testing, and unit testing, providing continuous feedback on the quality and compliance of the software.

- Certification Evidence Generation: Automation greatly simplifies the creation of certification artifacts, including traceability matrices, verification reports, and SAS. These artifacts are critical for demonstrating compliance with DO-178C objectives and are required during the certification process. Automated tools ensure the consistency and completeness of these outputs, reducing the likelihood of human error and streamlining certification activities.

While the methodology is tool-agnostic and adaptable to various software ecosystems, examples of tools that can support these activities include LDRA's suite [12] (TBmanager, TBvision, TBrun), Simulink [79] for model-based development, and formal verification platforms like Alloy Analyzer [80] or SPIN [81]. These tools provide complementary functionalities, such as traceability management, structural coverage analysis, and integration of formal verification results.

5.3.2 Formal Methods and Automated Tools: A Unified Approach

A key innovation of this methodology is its seamless integration of formal methods and automated tools. While formal methods provide the mathematical foundation for specifying and verifying system behaviours, automated tools offer practical solutions for managing traceability and continuously validating system components. The proposed methodology thus combines the rigour of formal verification with the efficiency of automation, ensuring that all software components are developed and tested following DO-178C.

The combination of these two approaches ensures that each phase of the software development lifecycle is covered:

- Requirements Phase: Formal methods are employed to specify HLRs, which ensures accuracy and precision as well as conformity with SRATS. Meanwhile, automation seeks to maintain traceability and conformance to standards.
- Design Phase: Formalization also aims to ensure the consistency of LLRs and architecture, and formal verification, using Model Checking, is used to ensure that these are compliant with their high-level counterparts. Automation aims to help with the traceability and verification of artifacts.
- Coding Phase: Automated static and dynamic analysis tools provide continuous feedback on code quality and ensure adherence to the design.
- Integration Phase: Verifying that all software components function correctly together as a unified system.

This methodology is designed not only for rigorous verification but also for scalability. The combination of formal methods and automation allows the process to be efficiently applied to complex UAV systems, where manual verification would be impractical. Furthermore, the automation provided by the tool streamlines many of the time-consuming aspects of verification, making the methodology scalable to larger projects without sacrificing the rigour needed for DO-178C compliance.

5.4 Process Description

This section outlines a detailed step-by-step approach to transforming system requirements into certified software, adhering strictly to the DO-178C objectives. The process spans all phases of the software development lifecycle: Requirements, Design, Coding, and Integration. For each phase, the methodology employs formal methods and automated verification tools to ensure compliance, traceability, and verification.

5.4.1 Software Requirements Phase

The Software Requirements phase serves as the foundational step in developing DO-178C-compliant software, focusing on capturing and managing precise, unambiguous software high-level requirements. This phase aims to ensure that all functional and non-functional requirements are well-defined, verified, and traceable, forming a reliable basis for subsequent development stages. Meeting DO-178C objectives such as compliance, traceability, accuracy, consistency, and verifiability (objectives A3.1, A3.6, A3.2, A3.4) is paramount throughout this phase.

Figure 5.2 illustrates the entire Software Requirements phase, showing the sequence from requirements collection to formal specification, analysis, and verification. Each step is depicted with key outputs that feed into the next stage of development, emphasizing the bidirectional traceability and formal verification processes. This diagram highlights the integration of formal methods and automated tools, ensuring that the process remains aligned with DO-178C objectives throughout.

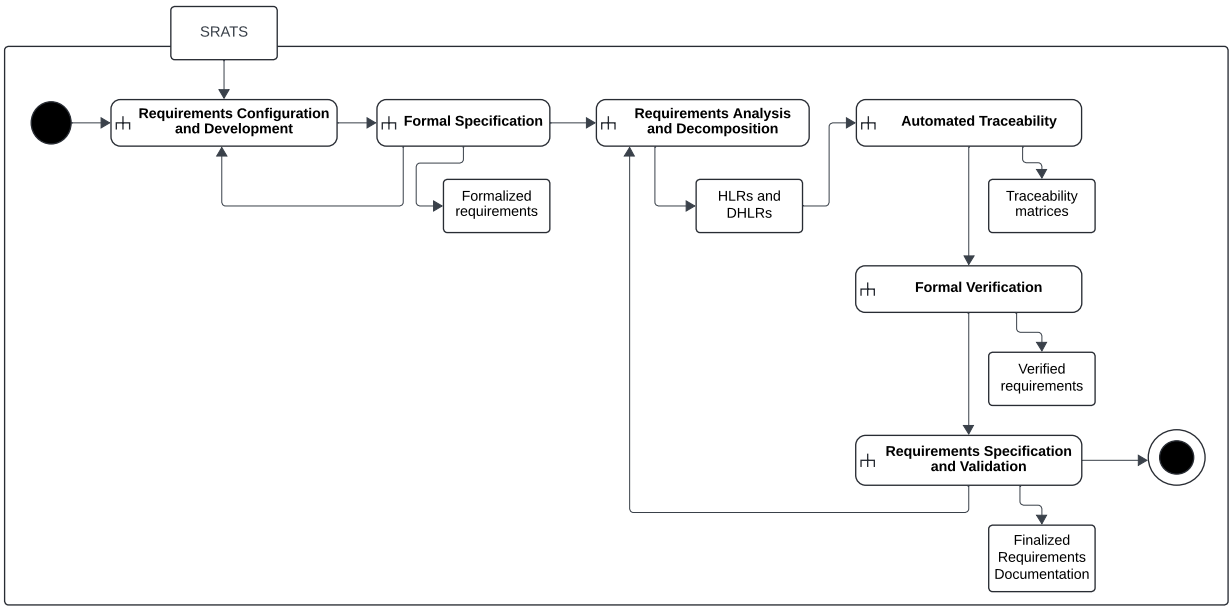


Figure 5.2 Expanded view of the software requirements process.

Requirements Configuration and Development The first step of the process consists of collecting and organizing the system's functional and non-functional requirements. This activity involves documenting the system's behaviour, performance, and constraints to ensure all aspects of the system are adequately covered. The configuration of requirements must enable proper traceability and refinement in subsequent phases. Clear and thorough requirements documentation ensures the system's intended functionality is well-defined from the outset.

Formal Specification In this step, the requirements are formalized to eliminate ambiguities and ensure a mathematically sound foundation for the HLR that will be developed. Formal specification provides a precise and verifiable model of the requirements, reducing the risks of misinterpretation and errors in later stages of development. Formal specification enables the system behaviour to be represented unambiguously, ensuring it aligns with the

project's functional and safety goals. Formal specification methods like Alloy [80, 82], Z notation [83] or B method [84] can be used to specify requirements using a well-defined syntax and semantics.

Requirements Analysis and Decomposition This activity involves analyzing the SRATS and breaking them down into more detailed, manageable components, such as HLRs and DHLRs. This decomposition helps to clarify system functionalities, making it easier to design, implement, and test. Possible derived requirements may be required to supplement the expected behaviour from a software perspective. If it has been developed, it should be sent back to the system team so that they can determine if no impact was added to the system.

Automated Traceability Automated traceability ensures that all system and high-level requirements (SRATS and HLRs) are traceable through subsequent phases of the software lifecycle, including design, coding, and testing. Establishing bidirectional links between requirements and future development artifacts ensures that each requirement is properly implemented and verified, preventing inconsistencies or overlooked requirements as the project progresses. Maintaining traceability early in the lifecycle is crucial for demonstrating compliance with DO-178C and ensuring comprehensive coverage.

This activity focuses on continuously linking requirements to artifacts, ensuring that any changes made during the requirements phase are reflected across all relevant documents and models. In turn, this establishes a robust foundation for traceability in later phases, when the design and testing will build on these requirements.

Formal Verification Early in the lifecycle, it is important to verify that the requirements are correct and complete, as well as to detect and resolve any potential conflicts or errors before they propagate to later stages. Formal verification techniques, such as model checking [8] or theorem proving [9], could be applied to ensure the system's properties meet their intended specification. This helps detect inconsistencies, deadlocks, or invalid states.

Model-checking techniques can be used to verify that the specified requirements are accurate and consistent with system behaviour. This ensured compliance with DO-178C objectives A3.2 (accuracy) and A3.4 (verifiability). The early application of formal verification allows us to detect potential errors before they impact later development phases.

Requirements Specification and Validation The final activity in this phase involves specifying the detailed requirements in a form that is consistent, complete, and verifiable.

This includes reviewing the requirements documentation, refining the specification where necessary, and validating it to ensure alignment with both system goals and regulatory standards. The validated requirements are then used as the basis for the subsequent design phase.

Peer reviews and formal validation techniques must be applied to ensure that the requirements are accurate and complete. Model checking can also be used to validate high-level requirements, ensuring alignment with DO-178C objectives A3.2 (accuracy and consistency), A3.4 (verifiability) and A3.5 (conformity to standards). This helps ensure that all system requirements are covered.

For a quick reference, Table 5.1 summarizes the key activities, the tools used, the outputs generated, and the DO-178C objectives addressed during the Software Requirements Phase.

Table 5.1 Summary of Activities, Outputs, and DO-178C Objectives Addressed in the Software Requirements Phase.

Activity	**Output**	**DO-178C Objective Addressed**
Requirements Configuration and Development	Comprehensive system requirements	no objective addressed
Formal Specification	Formalized, unambiguous requirements	no objective addressed
Requirements Analysis and Decomposition	Detailed HLRs, DHLRs	A2.1 (Developed HLRs), A2.2 (Defined DHLRs)
Automated Traceability	Traceability matrices	A3.1 (Compliance), A3.6 (Traceability)
Formal Verification	Verified requirements, validation reports	A3.2 (Accuracy), A3.4 (Verifiability), A3.5 (Conformity to standards)
Requirements Specification and validation	Validated requirements documentation	A3.2 (Accuracy), A3.4 (Verifiability), A3.5 (Conformity to standards)

5.4.2 Software Design Phase

The Software Design phase is a critical stage where the software application architecture and detailed design are defined, ensuring that the software is designed to match all of the requirements gathered in the preceding phase and is detailed enough to assist developers in writing code. This phase ensures that the software architecture is systematically designed and that LLRs are created and traceable to HLRs. The goal is to ensure that the software design is robust, modular, and verifiable, complying with DO-178C objectives. The phase

also incorporates formal verification techniques to ensure logical consistency and safety in the design.

Figure 5.3 illustrates the entire Software Design phase, showing the sequence from subsystem identification to design verification and review. Each step is depicted with key outputs that feed into the subsequent development stages, emphasizing the traceability and formal verification processes.

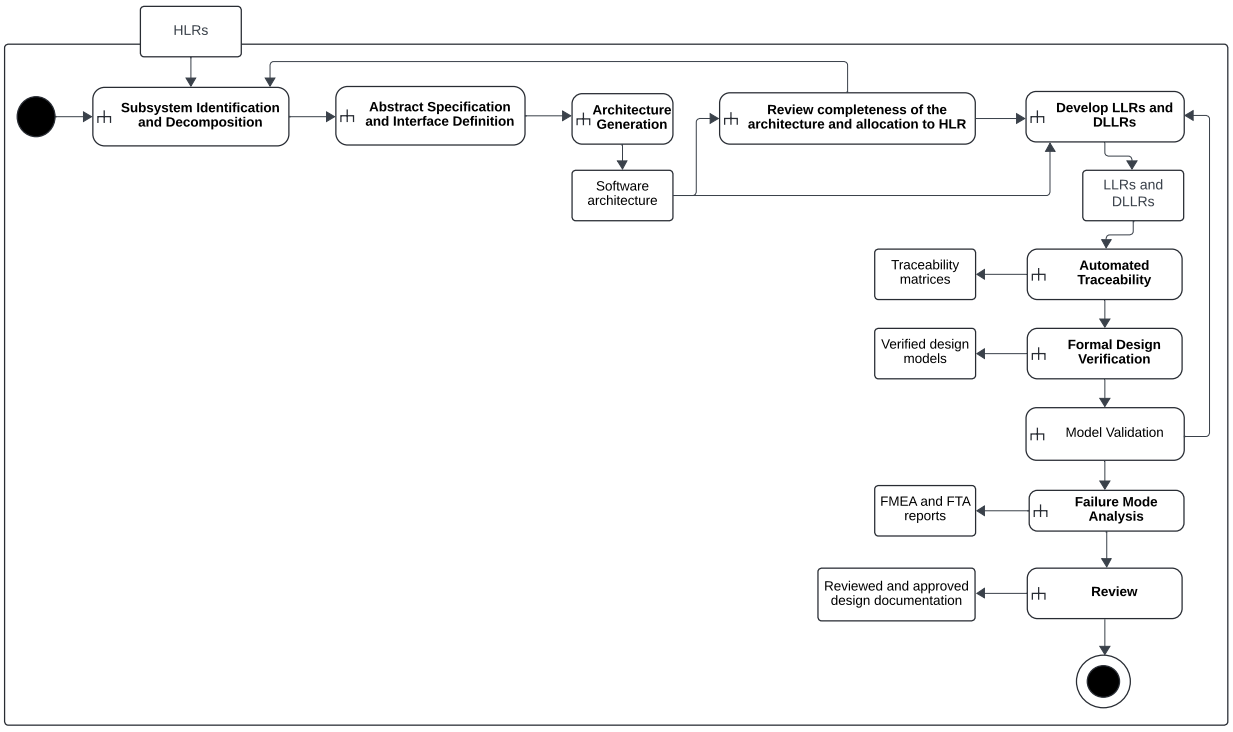


Figure 5.3 Expanded view of the software design process.

Subsystem Identification and Decomposition This activity involves breaking down the system into smaller, manageable subsystems, facilitating easier implementation and verification. The purpose is to modularize the system, making it easier to design, implement, and test individual components without compromising the overall architecture. Subsystem identification provides the foundation for establishing clear boundaries and responsibilities for each subsystem, ensuring modularity and scalability.

In our case study, the decomposition produced subsystem models and diagrams, establishing a well-defined structure for the overall system architecture.

Abstract Specification and Interface Definition The abstract specification focuses on determining the detailed structure of each subsystem identified in the previous activity. This includes outlining how each subsystem will function, as well as specifying inputs and outputs that direct component interactions. The task also includes producing detailed interface documentation that captures both control and data flow across subsystems. This documentation serves as a blueprint for the system's interactions, ensuring that the behaviour of each subsystem is consistent with the overall system design. By precisely specifying these interfaces, this step promotes consistency, simplifies integration, and establishes the framework for eventual implementation and verification.

Architecture Generation The Architecture Generation activity is responsible for building the structure of the software application, with a focus on describing and displaying subsystem interactions using thorough diagrams. The architecture should be documented in a consistent and easily accessible design so that developers can use it as a foundation for both implementation and maintenance. Clear and well-organized documentation enables future developers to efficiently maintain and expand the software architecture, which is especially important for long-term projects.

Review Completeness of the Architecture and Allocation to HLR Once the architecture is generated, it is essential to review the architecture's completeness and ensure it aligns with the HLRs. Following DO-178C specifications, the architecture must be linked with HLRs, ensuring a clear mapping or traceability between them. This ensures that each architectural component is justified and tied to specific requirements, which aids certification efforts. The review ensures that no critical elements are omitted and that the architecture is well-suited to address the needs outlined by the HLRs.

LLR and DLLR Development This activity focuses on converting HLRs into comprehensive specifications that will guide software implementation. The LLRs give the amount of detail required for developers to accurately produce the software by defining functional behaviour, algorithms, and constraints for each component. The LLRs also include a data dictionary, which defines data types, variables, and structures, ensuring that data is handled consistently across the code. In cases where additional behavioural details are needed to support or clarify the primary requirements, DLLRs are created.

By producing precise and comprehensive LLRs and DLLRs, this process assures that the software meets both design expectations and system requirements.

Automated Traceability Similar to the requirements phase, maintaining traceability between design elements and requirements is critical. Automated traceability ensures that all design components are linked to their respective HLRs, LLRs, and other development artifacts. This guarantees that the design accurately reflects the system's requirements and that any changes to the design can be traced back to their originating requirement.

Formal Design Verification This activity seeks to confirm the accuracy, consistency, and completeness of LLRs and DLLRs, if applicable, through a rigorous verification procedure. This activity ensures that the LLRs appropriately reflect the HLRs' intent and that all system behaviours are well-defined and free of ambiguities. Model-checking techniques are frequently used to perform formal verification, giving a mathematical foundation for determining conformity between LLRs and HLRs.

This technique aids with the early detection of inconsistencies, unwanted actions, and potential design faults by investigating all conceivable states and behaviours within the design. This level of formal verification improves the reliability of the software design and increases confidence that the implementation will precisely meet the requirements.

Model Validation Before proceeding to the coding phase, it is essential to validate the design models to ensure they accurately reflect the requirements. This step aims to confirm that the LLRs completely capture the intended functionality and accurately represent the HLRs. This activity involves validating the LLRs by reviewing the verification results to ensure consistency and completeness. During model validation, any discrepancies, missing requirements, or omitted behaviours cause the LLRs and DLLRs to be re-evaluated (step back to "Develop LLRs and DLLRs activity"). Corrections are then made to ensure that the model reflects the original system intent. This activity ensures a solid basis for code implementation by carefully validating the model representing low-level requirements, reducing the chance of issues occurring later in the development process.

Failure Mode Analysis The process identifies and mitigates potential failure modes in the software design to ensure overall system safety and dependability. This procedure typically involves performing Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). FMEA thoroughly investigates each component and function to identify possible failure modes, causes, and effects, allowing the design team to prioritize and fix important vulnerabilities. FTA, on the other hand, examines how a series of failures can lead to a critical or catastrophic event, outlining dependencies and likely root causes in a systematic, hierarchical manner. Together, FMEA and FTA provide a comprehensive perspective of

potential failure situations, guaranteeing that the software design meets the necessary safety requirements.

Review The final activity in the design phase involves reviewing the design to ensure it meets all quality and safety standards. This step ensures that the design is accurate, complete, and ready for implementation. Reviews help identify any potential design flaws or areas that require further refinement.

For a concise summary, Table 5.2 highlights the key activities, tools used, outputs generated, and the DO-178C objectives addressed during this phase.

5.4.3 Software Coding Phase

The Software Coding process involves translating design specifications into source code that meets high safety and reliability standards for safety-critical systems. In this phase, developers implement code based on the comprehensive descriptions created throughout the design stages, guaranteeing strict compliance with DO-178C criteria. The coding phase emphasizes traceability, allowing each line of code to be linked back to its appropriate LLRs and, as a result, HLRs, thereby confirming compliance with the system's intended functionality and safety objectives.

To maintain high quality, automated tools and formal methods are used to check code consistency, find errors, and guarantee coding standards are followed. This approach guarantees that the final code is robust, verifiable, and perfectly aligned with the stated architecture and requirements.

Figure 5.4 provides a visual representation of the Software Coding phase. It illustrates the sequential flow from code generation to formal verification and testing, emphasizing the integration of automated analysis and validation processes.

Code Generation This activity focuses on generating source code directly from design models, ensuring that the implementation adheres precisely to the specified design and functional requirements. The fundamental goal of this step is to preserve consistency and traceability so that each code component can be easily connected back to its appropriate design elements. Code can be generated using qualified tools that automate the process and reduce human error, or it can be written manually. A hybrid technique can also be used, combining automatic generation and manual development. This activity lays the basis for accurate, reliable code, assisting the certification process by meeting DO-178C traceability and quality standards.

Table 5.2 Summary of Activities, Outputs, and DO-178C Objectives Addressed in the Software Design Phase.

Activity	**Output**	**DO-178C Objective Addressed**
Subsystem Identification and Decomposition	Subsystem models, decomposition diagrams	no objective addressed
Abstract Specification and Interface Definition	Interface definitions, abstract specifications	no objective addressed
Architecture Generation	Software architecture models	A2.3 (Architecture development)
Review completeness of the architecture and allocation to HLR	Verified architecture allocated to HLRs	A4.8 (Architecture's compatibility with HLRs), A4.9 (Architecture's consistency) , A4.11 (Architecture's verifiability)
Develop LLRs and DLLRs	Documented LLRs and DLLRs	A2.4 (Developed LLRs), A2.5 (Defined DLLRs)
Automated Traceability	Traceability matrices	A4.6 (Traceability)
Formal Design Verification	Verified design models	A4.1 (LLRs compliance with HLRs), A4.2 (LLRs accuracy), A4.4 (LLRs verifiability)
Model Validation	Validated design models	A4.1 (LLRs compliance), A4.2 (LLRs accuracy)
Failure Mode Analysis	FMEA and FTA reports	no objective addressed
Review	Reviewed and approved design documentation	A4.1 (LLRs' compliance with HLRs), A4.5 (LLRs' conformity to standards), A4.8 (Architecture's compatibility with HLRs), A4.9 (Architecture's consistency), A4.12 (Architecture's conformity to standards).

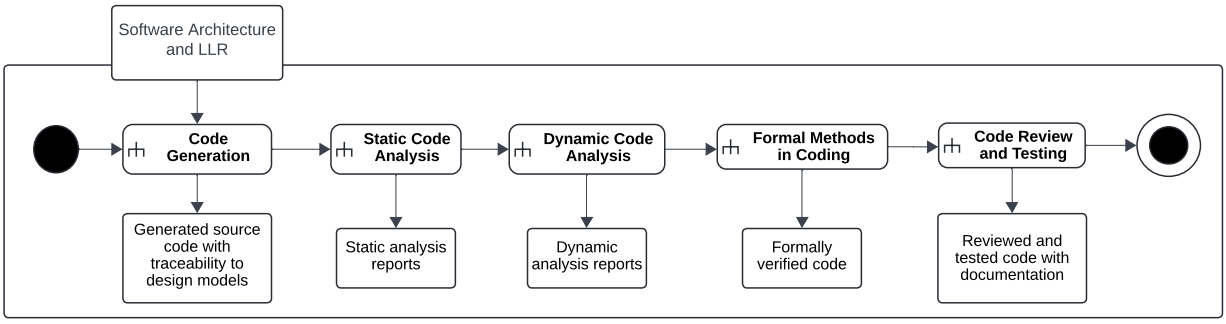


Figure 5.4 Expanded view of the software coding process.

Static Code Analysis Static code analysis is a key activity in the coding process that checks source code without running it, revealing potential flaws early in development. This analysis identifies code standard violations, data flow inconsistencies, and other quality issues, allowing developers to correct them before they affect runtime behaviour. This activity helps to create reliable, high-quality software that is easier to maintain and more compliant with DO-178C regulations by verifying the code adheres to set safety and quality criteria.

Dynamic Code Analysis Dynamic code analysis examines the code's behaviour during execution, ensuring that it works properly under a variety of scenarios and satisfies performance and safety expectations. This activity entails running the software and observing its actual reactions, which include looking for runtime faults, timing difficulties, and memory utilization. Code coverage is an important aspect of dynamic analysis since it ensures that all pathways and circumstances within the code are checked, allowing us to ensure that no functionality is left untested. Dynamic code analysis, which simulates real-world events, certifies the code's suitability for deployment in a safety-critical environment, hence supporting DO-178C's emphasis on dependable and verifiable software behaviour.

Formal Methods in Coding Formal methods in coding add mathematical rigour to the software verification process, allowing for the exact detection of potential errors within the code. Using formal verification approaches, such as bounded model checking, developers can systematically investigate a finite subset of the software execution across all potential inputs. This method enables thorough verification of critical properties, such as correctness, safety, and conformance to design requirements. Bounded model checking, in particular, can reveal subtle issues that typical testing approaches may miss by investigating edge cases and unexpected input possibilities in a controlled, repeatable manner. Through formal approaches, this activity provides an additional degree of confidence in code reliability and conformance

with DO-178C standards, ensuring that the software is error-free and behaves as intended before moving into the testing and final verification stages.

Code Review and Testing The code review and testing activity is the final step in the coding process, and it involves a thorough examination to ensure that the code meets all quality standards and functional requirements. This step includes performing systematic code reviews to ensure that the implementation appropriately reflects the planned functionality. Following the review, testing is carried out to check for any remaining errors and ensure that the software functions properly under expected settings.

5.4.4 Software Integration Phase

The Software Integration phase is the final step in the development lifecycle, where all software components are integrated, tested, and verified to ensure they work together as intended. This phase aims to validate that the compiled and integrated code meets the system requirements, functions correctly in an integrated environment, and is ready for deployment. The activities in this phase ensure compliance with key DO-178C objectives related to integration, verification, and validation.

Source Code Analysis and Compilation The first step in the integration phase is to analyze and compile the source code. This ensures that all individual modules are correctly compiled and linked to form executable code without any errors. Compilation checks help identify any issues related to code compatibility, linking errors, or violations of system specifications.

During this activity, industry-standard compilation and linking procedures are used to verify the integrity of the source code, ensuring that it produces a valid executable. This step aligns with DO-178C objective A2.7, which focuses on ensuring that the software is compiled and linked correctly to maintain integrity throughout the integration process.

Automated Integration Testing Once the software components are compiled, they are integrated and tested to ensure that the different modules interact correctly. The goal of automated integration testing is to validate that all components function together seamlessly and meet the overall system requirements.

Automated integration testing verifies that the integrated modules perform as expected when operating as part of a unified system. This activity helps detect any integration issues, such as incorrect data handling between modules, timing errors, or unexpected interactions. By

performing comprehensive testing of module interactions, this activity addresses DO-178C objective A5.7, which focuses on ensuring that the software integration process is complete and correct.

End-to-End Verification It ensures that the entire system meets its specified requirements and functions correctly when deployed in a fully integrated environment. This step involves executing comprehensive tests that validate the system's performance, reliability, and compliance with requirements.

The end-to-end verification process ensures that the integrated software is free from defects, performs as expected, and adheres to the system requirements. It involves rigorous testing across different scenarios to ensure that the system meets DO-178C objectives A6.1 (compliance with HLRs), A6.2 (robustness with HLRs), A6.3 (compliance with LLRs), and A6.4 (robustness with LLRs).

Parameter Data Items (PDI) Generation As the final step in the integration phase, PDIs are generated. PDIs refer to configuration files and system parameters that ensure the correct deployment and operation of the software in its target environment. The generation of PDIs is critical for configuring the system correctly and ensuring that all necessary data items are included for proper functionality.

This activity produces the necessary PDIs to configure the system for operation, ensuring that the software is tailored to its intended use environment. The generation of PDIs (DO-178C objective A5.8) is an essential part of delivering a complete, deployable system.

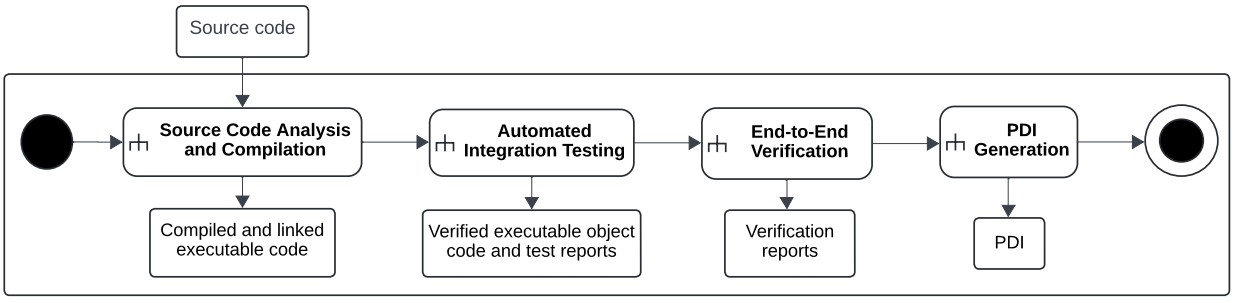


Figure 5.5 Expanded view of the software integration process.

The Software Integration phase, as depicted in Figure 5.5 ensures that all software components are properly compiled, tested, and verified before deployment. Each step in this phase—from code analysis and compilation to end-to-end verification—ensures that the system operates as a unified whole and meets the specified requirements.

5.5 Case Study: Collision Avoidance System for UAVs

5.5.1 Overview of Collision Avoidance System

The CAS for UAVs is a crucial technology designed to ensure the safety and reliability of autonomous UAV operations by enabling the detection, evaluation, and avoidance of potential collision threats in real-time. As UAVs become more prevalent in both commercial and defence sectors, safely integrating them into increasingly congested airspaces is paramount. The CAS addresses this challenge by autonomously replicating and enhancing traditional "see-and-avoid" techniques used by human pilots, enabling UAVs to operate safely in dynamic airspace environments.

The CAS continuously monitors the UAV's surroundings, identifying both cooperative and non-cooperative aircraft or objects. It plays a particularly vital role in Beyond Visual Line of Sight (BVLOS) operations, where human operators are unable to maintain direct visual contact with the UAV. In such cases, the CAS autonomously assesses collision risks, prioritizes threats, and determines appropriate avoidance maneuvers to ensure the UAV maintains a safe flight path. This capability is crucial for UAVs operating in airspace shared with manned aircraft, where real-time, data-driven decision-making is required to avoid mid-air collisions.

As a Design Assurance Level B (DAL-B) system under the DO-178C standard, the CAS is classified as critical to preventing hazardous situations, demanding robust software development, verification, and validation processes. The DAL-B classification emphasizes the system's importance in ensuring reliable performance under all operational conditions, as any failure could lead to catastrophic consequences.

The key functions of the CAS include:

- Traffic Detection: The system continuously monitors the UAV's surroundings, detecting both cooperative and non-cooperative aircraft or objects.
- Track Traffic: The previous traffic detected is tracked to establish a reliable track history for each object.
- Collision Risk Assessment: Once traffic is detected, the CAS evaluates the potential for a collision based on trajectory predictions, considering speed, direction, and proximity.
- Threat Prioritization: In situations where multiple collision threats are identified, the system prioritizes these threats based on proximity and the immediacy of potential impact.

- Avoidance Maneuver Determination: CAS autonomously determines the most appropriate avoidance maneuver (e.g., altitude adjustment or lateral deviation) to prevent a collision with the highest-priority threat.
- Maneuver Command: The system sends maneuver commands to the UAV's flight control system to execute the avoidance action in real-time.

Inspired by NASA's advancements in non-cooperative collision avoidance [85], the CAS implementation seeks to enhance UAVs' ability to autonomously navigate congested airspaces with minimal human intervention. This system aligns with efforts to establish higher safety standards for UAVs in civil airspace, ensuring they meet regulatory safety requirements, particularly as outlined in DO-178C. By autonomously detecting, evaluating, and avoiding collisions, CAS serves as a critical enabler for UAV operations, facilitating their safe and reliable integration into future airspace systems.

5.5.2 Implementation of Methodology

The proposed methodology was applied to the development and verification of the CAS for UAVs, focusing on the key phases of Requirements, Design, and Coding to ensure compliance with DO-178C. Given the system's critical DAL-B classification, each phase was executed with rigour, integrating formal methods and automated tools to ensure traceability, verifiability, and safety.

The implementation of the methodology was tailored to the specific needs of the CAS development, leveraging tools that aligned with the project's objectives and constraints. While specific tools, such as the LDRA tool suite, were employed for this case study, the methodology itself is designed to be adaptable, enabling the use of alternative tools and platforms that meet project-specific requirements. This section highlights the functionality of the tools used in this implementation while underscoring the broader applicability of the proposed approach. Each phase was iteratively refined based on model-checking results, verification activities, and updates to system requirements, ensuring that the CAS development was robust and aligned with certification standards.

Requirements Phase

The first phase of the methodology focuses on defining and evaluating the CAS requirements for DO-178C compliance. This entailed identifying the system-level requirements (SRATS), including functional and non-functional aspects critical to CAS performance and safety. To

ensure these requirements were clear, unambiguous, and precise, a formal model of the SRATS was developed using the Alloy language, and the Alloy Analyzer was used to conduct rigorous consistency checks.

A notable example is SRATS_002, which specifies the spatial boundaries of CAS surveillance based on detection range, azimuth, and elevation fields of regard. The Alloy model for SRATS_002 is as follows:

```
fact SRATS_002 {
all sys: CollisionAvoidanceSystem |
    sys.surveillanceVolume.traffic = { t: Traffic |
        t.x <= sys.detectionRange and
        (t.y >= sys.minAzimuthFieldOfRegard and t.y <= sys.
           azimuthFieldOfRegard) and
        (t.z >= sys.minElevationFieldOfRegard and t.z <= sys.
           elevationFieldOfRegard)
    }
}
```

Listing 5.1 Alloy Specification for SRATS_002.

This Alloy model formally encapsulates the spatial constraints of the CAS surveillance volume. Assertions, such as *CompleteTrafficDetection*, were used to verify that all traffic meeting these criteria was correctly detected. The Alloy Analyzer validated this assertion without counterexample, confirming the logical consistency of the system model under the tested scenarios.

During the verification process of other assertions, Alloy Analyzer discovered several inconsistencies in the SRATS, revealing issues such as incorrect assumptions about traffic detection range and ambiguities in threat prioritization. These findings were documented as defects and reported to the system engineering team for correction. Each issue was addressed using an iterative refinement strategy, resulting in a more accurate and reliable set of system requirements consistent with the desired system behaviour and safety objectives. A comprehensive list of SRATS is provided in Appendix A for reference.

Following the refinement and verification of the SRATS, the validated requirements were imported into LDRA TBManager to support the HLR development. The SRATS were systematically decomposed into key CAS functions, enabling the creation of software-level HLRs (detailed in Appendix A) that directly traced back to the system requirements.

The initial draft of the HLRs was developed and set up within a versioned baseline in the tool.

As depicted in Figure 5.6, traceability was created indicating that each HLR was directly linked to its associated SRATS. Following the outlined development activities, the next stage was to ensure that the requirements fulfilled DO-178C objectives, specifically those listed in *Table A-3 (Verification of Outputs of Software Requirement Process)* of the DO-178C.

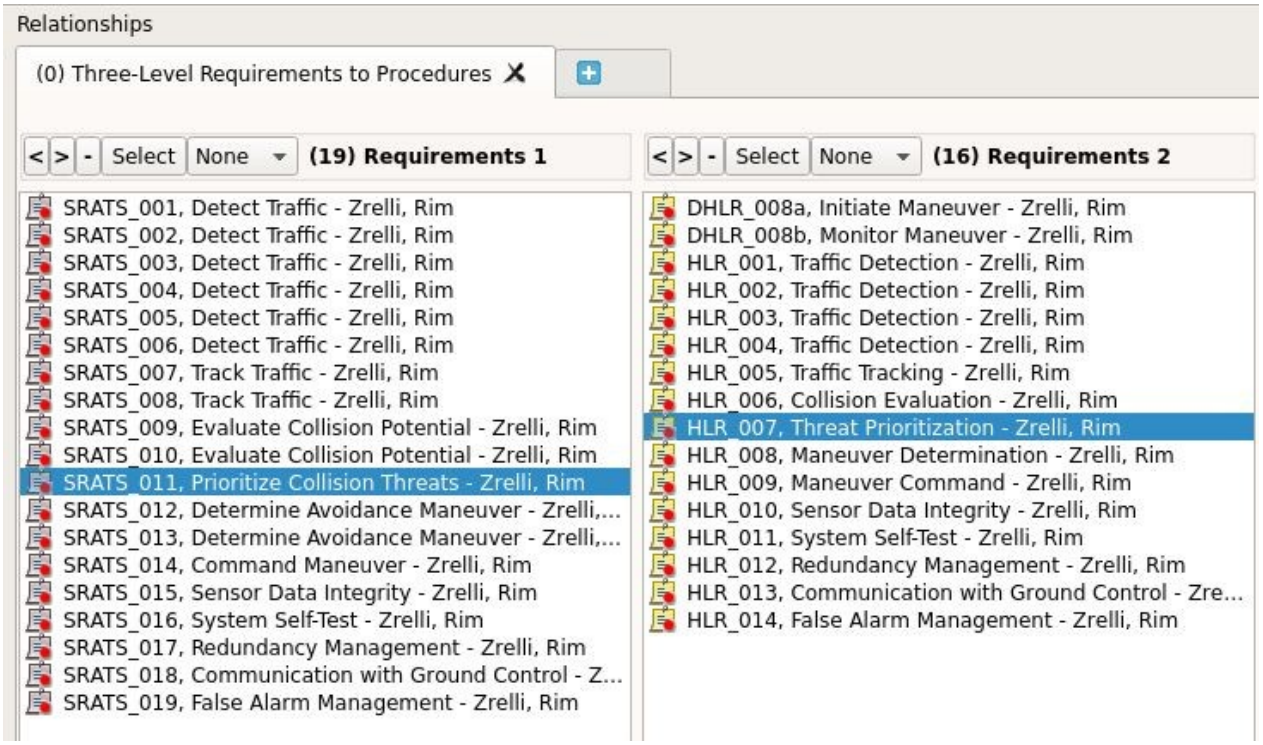


Figure 5.6 TBmanager relationship view between SRATS and HLR.

To ensure that the HLRs adhered to DO-178C objectives for compliance with SRATS, accuracy, consistency, and verifiability, a formal model was constructed in Alloy. This model enabled the detection of inconsistencies and ambiguities, facilitating early refinement and validation of the requirements.

The verification process involved:

- Initial model execution: The initial Alloy execution demonstrated that the HLRs were logically consistent and free of major contradictions. However, one critical assertion, *InadequateCollisionThreatHandling*, failed, exposing gaps in HLR_008 and HLR_009, which relate to the handling of collision threats and associated maneuvers.
- Analysis of the counterexample: The failed assertion revealed a scenario where identified collision threats did not trigger corresponding avoidance maneuvers. This gap was traced to ambiguities in the activation and prioritization conditions for maneuvers:

  - Ambiguity in maneuver activation: HLR_008 lacked specificity about when maneuvers should be initiated for detected threats.
  - Gaps in prioritization logic: HLR_009 did not define how to prioritize maneuvers when multiple threats were present.
  - Dynamic adjustment issues: Conditions for dynamically updating or terminating maneuvers were insufficiently detailed.

- Refinement: Based on this analysis, HLR_008 and HLR_009 were refined to explicitly mandate the initiation of maneuvers for all detected threats unless overridden by a higher-priority threat and to address maneuver termination and prioritization. To bridge the gap between HLRs and LLRs, two new DHLRs (DHLR_008a and DHLR_008b) and their rationale were introduced to clarify the conditions for handling collision threats.

- Reverification: The refined HLRs and DHLRs were re-modelled in Alloy. Assertions, including *InadequateCollisionThreatHandling*, were verified (as depicted in Figure 5.7), and no further counterexamples were found. This confirmed compliance with DO-178C objectives A3.2 (accuracy), A3.4 (verifiability), and A3.5 (conformity to standards).

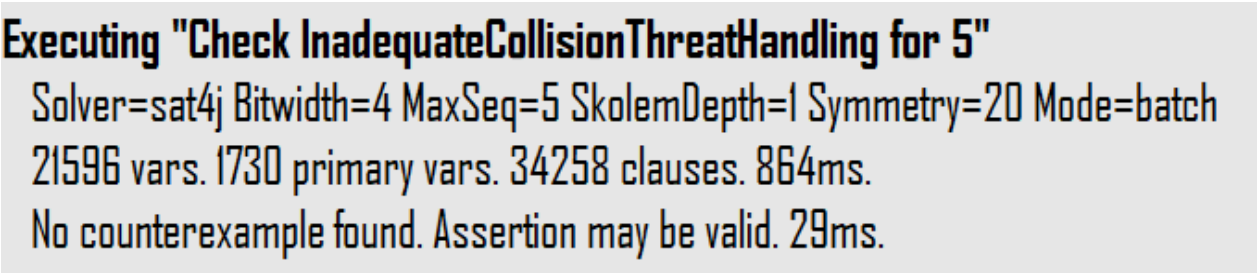


Figure 5.7 Results for checking the InadequateCollisionThreatHandling assertion after refinement.

The assertion *InadequateCollisionThreatHandling* states that some traffic detected as a threat is not being actively maneuvered against when it should be and it is specified as follows:

```
1 assert InadequateCollisionThreatHandling{
2     all sys: CollisionAvoidanceSystem, t: Traffic |
3         t in sys.collisionThreats implies (
4             some m: sys.maneuvers |
5             m.isActive = True and
6             t in m.threat.spaceZone and
```

```
7               (no t2: sys.collisionThreats - t | t2.threatLevel > t.
                   threatLevel and t2.timeToCollision < t.
                   timeToCollision)
8           )
9   }
10  check InadequateCollisionThreatHandling for 5
```

Listing 5.2 InadequateCollisionThreatHandling Assertion.

The validated HLRs were imported back into TBManager to finalize the traceability matrix and link each HLR and DHLR to its corresponding SRATS. This integration facilitated the generation of certification artifacts, such as the Software Requirements Traceability Matrix (SRTM) (Figure 5.8).

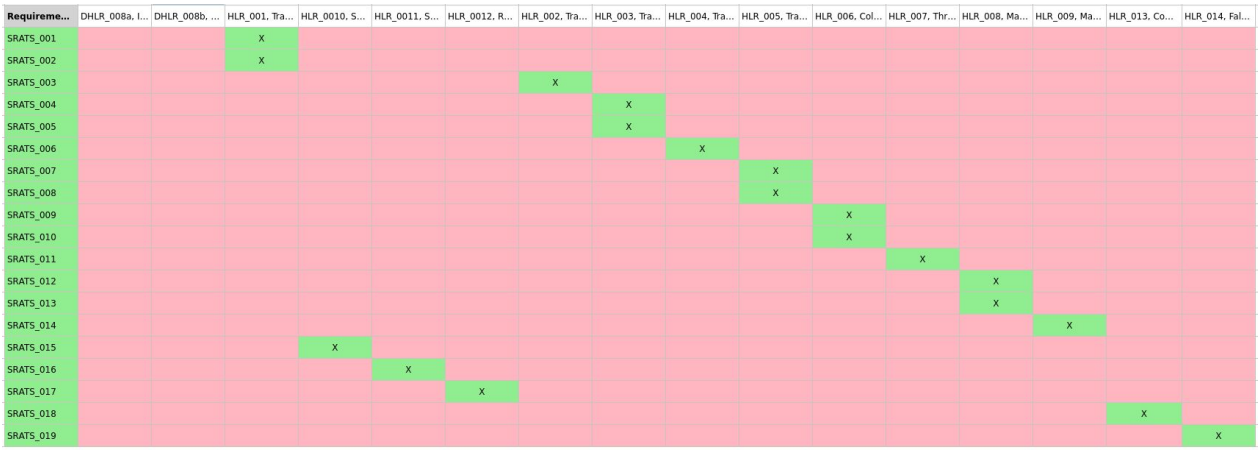


Figure 5.8 Traceability matrix between SRATS and HLRs.

The final activity of the Requirements phase, referred to as *Requirements Specification and Validation*, concentrates on validating the HLRs to ensure their robustness and readiness for further development phases. Previously, a set of HLRs was drafted using the SRATS, and these HLRs underwent preliminary verification activities. In this phase, however, a peer review is carried out by an independent reviewer who was not involved in the original HLR drafting. This impartial evaluation is crucial to ensuring that all standards have been satisfied, that the requirements are accurate and unambiguous and that there is complete traceability between each HLR and the system requirements, which includes all relevant parts.

If any issues or inconsistencies are discovered during the peer review, the original author logs a defect for rectification, which initiates an iterative process to enhance the requirements. The

reviewer follows a rigorous checklist to confirm that each DO-178C objective is met, including the standards' clarity, consistency, and completeness. This systematic approach ensures that the HLRs meet stringent quality and compliance criteria, reducing the likelihood of errors that could spread to later phases of development. Once all of the checklist elements have been satisfactorily completed, the evaluation is formally documented.

Checklist High-Level Requirements

Review ID:	Peer Review (PR) HLR 001
Review Date:	2024-09-29
Reviewer(s):	Name: Henrique Amaral Misson; Role: Sw developer; Is the Author?: ☐
Project Name:	CAS_CollisionAvoidance
Data Item(s) under review:	The purpose of this PR is to review the high-level requirements of the above-mentioned program. Data to be reviewed: Refer to the "HighLevelRequirements" group in TBManager with implemented Baseline defined in the field below. Number of HLRs to review: 16
Software Level:	B
Version / Baseline:	Baseline_005_Revew_HLR
Is it verified ?	Yes

Checklist Item	Yes / No	Comments	Correction
High-level requirements comply with system requirements? (A-3.1)	☑ TRUE		
High-level requirements are accurate and consistent? (A-3.2)	☑ TRUE	No. Refer to defects for more details	Corrected during implementation
High-level requirements are compatible with target computer? (A-3.3)	☑ TRUE		
High-level requirements is verifiable? (A-3.4)	☑ TRUE	No. Refer to defects for HLR that was not verifiable.	Corrected during implementation
High-level requirements conforms to standards? (A-3.5)	☑ TRUE		
High-level requirements are traceable to system requirements? (A-3.6)	☑ TRUE		
Algorithms are accurate? (A-3.7)	☑ TRUE	Not Aplicable - No algorithms defined	
Are all derived requirements, if applicable, are justified?	☑ TRUE		

Figure 5.9 Checklist used for peer review of HLRs during the validation.

Figure 5.9 illustrates the peer review checklist used during this validation phase. This checklist ensures that all DO-178C objectives related to HLRs are rigorously evaluated, including compliance with system requirements (A3.1), accuracy and consistency (A3.2), and traceability (A3.6). Any identified issues are documented in the "Comments" section and addressed iteratively during the correction process.

Throughout this process, configuration management is strictly maintained in TBManager, where all review activities, changes, baselines, and defects are recorded. This allows for full documentation of the requirements development process, and TBManager can provide a report summarizing the results of the *Requirements Specification and Validation* phase. The final output of this phase, known as Requirements Data, consists of both validated HLRs and DHLRs, which will serve as the foundational specifications for the design and coding phases.

The requirements phase concluded with the generation of validated Requirements Data, consisting of SRATS, HLRs, and DHLRs. These outputs formed a robust foundation for the design and coding phases, fulfilling all Table A-3 objectives of DO-178C (Figure 5.10). This iterative and tool-supported process mitigated risks early, ensuring that subsequent phases adhered to the highest standards of safety and certification rigour.

Objectives	Artifacts	Assets	...	Objective Name	Objective Standard	Objective S...	Placeholders
Table A-3 1	0	3	...	High-level requirements comply with system requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-3 2	16	3	...	High-level requirements are accurate and consistent	DO-178C - Populated	Fulfilled	100.00%
Table A-3 3	0	1	...	High-level requirements are compatible with target computer	DO-178C - Populated	Fulfilled	100.00%
Table A-3 4	16	3	...	High-level requirements are verifiable	DO-178C - Populated	Fulfilled	100.00%
Table A-3 5	0	1	...	High-level requirements conform to standards	DO-178C - Populated	Fulfilled	100.00%
Table A-3 6	16	2	...	High-level requirements are traceable to system requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-3 7	0	1	...	Algorithms are accurate	DO-178C - Populated	Fulfilled	100.00%

Figure 5.10 DO-178C objectives for verification of outputs of software requirement process in TBmanager.

Design Phase

The design phase of the methodology centred around defining the architecture and the LLRs of the CAS to ensure that it adhered to both operational and safety goals, as outlined in the HLRs.

The design phase begins with an in-depth review of the *Requirements Data* generated during the requirements phase. The goal of this analysis is to decompose the HLRs into modular subsystems, each responsible for a critical function of the CAS. Key subsystems identified include traffic detection, tracking, collision evaluation, threat prioritization, and maneuver determination. For each subsystem, essential inputs, outputs, and interactions with other components and external equipment are carefully described and documented. Once these components and their interfaces have been identified, a preliminary architecture model is developed. This model, created in the Architecture Analysis and Design Language (AADL) [86] using the Osate tool [87], encapsulates both internal relationships and system-level views of CAS components. The design adheres to the ARINC653 [88] industry-standard partitioning scheme through an AADL library.

One of the critical subsystems identified during the design phase is *Traffic Detection*. This subsystem is responsible for continuously monitoring the UAV's surroundings. The architecture of this subsystem was modelled in AADL, as shown in Figure 5.11, capturing its inputs (sensor data from onboard radar and external data sources), outputs (a list of detected traffic), and interactions with other CAS components, such as *Traffic Tracking* and *Collision Evaluation*.

The architectural representation (Figure 5.11) encapsulates the core operations of this subsystem, including surveillance volume calculation and traffic detection. The figure highlights traceability to the associated HLRs (HLR_001 - HLR_004), ensuring that the subsystem aligns with the requirements established in the previous phase. For a detailed view of the

architectural diagrams of all the identified components of the CAS Architecture refer to Appendix A.

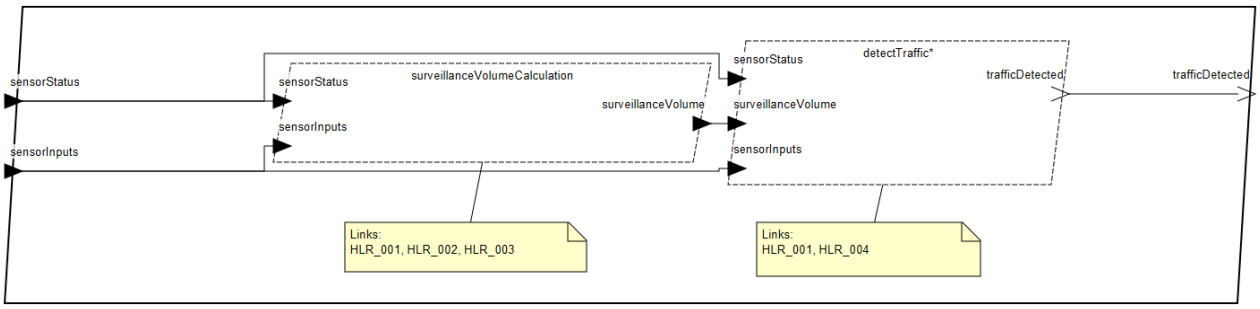


Figure 5.11 AADL model of the traffic detection subsystem.

An independent peer review was conducted to confirm that the architecture aligns with the HLRs and meets the DO-178C objectives. This review involved a detailed checklist evaluation to verify the architecture's compliance with key DO-178C objectives, including consistency, verifiability, and traceability to HLRs. Any defects, such as missing component inputs, were identified and corrected during this process.

In addition, formal verification was performed using model-checking approaches to ensure the architecture's accuracy and reliability. The architecture was modelled in the Promela language and analyzed using the Spin model checker, which converts features of interest into LTL statements. These LTL attributes specify important system behaviours and limitations, guaranteeing that the architecture design meets the necessary safety and functional requirements. The verification process focused on critical system properties, such as liveness properties, ensuring that critical actions (such as avoidance maneuvers) are eventually executed when necessary and safety properties, preventing the activation of incorrect or dangerous maneuvers, thereby avoiding unintended consequences.

Key properties analyzed through LTL claims included:

- c1: A maneuver command is eventually executed for detected collision threats. $[] <> CommandManeuver@SendCmdTrue$.
- c2: Detection of traffic leads to a maneuver command. $[](DetectTraffic@SendTrafficDetected) \rightarrow <> (CommandManeuver@SendCmdTrue)$.
- c3: Collision evaluation follows traffic detection. $[](DetectTraffic@SendTrafficDetected) \rightarrow <> (CollisionEvaluation@EvaluateCollisionPotential)$.

The results showed that no assertion violations or invalid end states were detected, demonstrating that the model satisfies all safety and functional properties in the explored scenarios. However, some claims exhibited unreached states. The analysis of these results indicates that while the core architecture is robust, the unreached states suggest potential areas for refinement or the need for additional test scenarios. For example, claim *c1* revealed missed conditions where detected threats did not lead to a maneuver command, suggesting the need for enhanced conditions in the model. Claim *c3* highlighted edge cases where collision evaluation did not consistently follow traffic detection, pointing to potential gaps in the prioritization logic. These findings were documented and addressed iteratively, ensuring continuous improvement of the architecture model and alignment with the safety-critical objectives of DO-178C.

While architectural design activities occur outside the LDRA tool suite, the verified architecture model and supporting documentation were imported into TBManager. This integration ensured that all verification results were tracked and included in the overall configuration management system, facilitating traceability and certification artifact generation.

The LLRs were derived to define the operational behaviour of each subsystem with precision. For instance:

- LLR-001 specifies that when the sensor status is active, the TrafficDetection function retrieves the sensor inputs, initiating the data acquisition process.
- LLR-002 establishes the rules for validating sensor inputs, ensuring parameters such as DetectionRange, AzimuthFOR, and ElevationFOR fall within safe operational ranges.

A comprehensive list of the LLRs is provided in Appendix A, which includes LLR-001 through LLR-037.

Formal verification was conducted to validate the accuracy, consistency, and compliance of the LLRs with the HLRs. This process, conducted in two stages, ensured adherence to DO-178C objectives A4.1 (compliance), A4.2 (accuracy), and A4.4 (verifiability).

1. Execution of the Alloy Model Representing All LLRs
   A comprehensive Alloy model was created to represent all LLRs as a cohesive system. This model encoded the functional logic, interactions, and constraints described in the LLRs. Using the Alloy Analyzer, the model was executed to validate internal consistency and interaction across subsystems. The analysis confirmed that the LLRs collectively formed a logically sound design. Instances were generated successfully

without contradictions, demonstrating that the specified constraints did not conflict and that subsystems interacted as expected under the architectural model.

2. Assertion-Based Compliance Checks with HLRs
   Once the internal consistency of the LLRs was validated, assertions were defined for each HLR to ensure compliance. These assertions encapsulated key behaviours derived from the HLRs and were checked against the Alloy model. For example:

   - Assertion HLR_001_TrafficDetection verified that the inputs conformed to the correct field of regard (AzimuthFOR and ElevationFOR) for detecting traffic. It is formally expressed as:

```
assert HLR_001_TrafficDetection {
    all td: TrafficDetection |
        some td.sensorInput =>
        td.sensorInput.DetectionRange > 0 and
        td.sensorInput.AzimuthFOR >= -110 and td.sensorInput
           .AzimuthFOR <= 110 and
        td.sensorInput.ElevationFOR >= -15 and td.
           sensorInput.ElevationFOR <= 15
}
check HLR_001_TrafficDetection for 5
```

   Listing 5.3 HLR_001_TrafficDetection Assertion.

   The Alloy Analyzer executed this assertion, and no counterexamples were found (as shown in Figure 5.12), confirming compliance with HLR_001. The absence of counterexamples demonstrated that the system adhered to operational parameters for the sensor input's field of regard, ensuring comprehensive spatial coverage and reliable traffic detection.

   - Assertion HLR_008_ManeuverDetermination ensured that appropriate maneuvers were determined in response to detected collision threats. The validation of this assertion highlighted the system's ability to proactively determine maneuvers upon detecting potential collisions.

   The results of these checks confirmed that all assertions were validated successfully, with no counterexamples identified. The LLRs adhered to their corresponding HLRs, ensuring alignment with operational and safety goals. These results validated the robustness of the LLRs and their compliance with DO-178C objectives, minimizing risks of errors propagating into the coding phase.

```
Executing "Check HLR_001_TrafficDetection for 5"
  Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20 Mode=batch
  142476 vars. 2948 primary vars. 275239 clauses. 4130ms.
  No counterexample found. Assertion may be valid. 8ms.
```

Figure 5.12 Results for checking the HLR_001_TrafficDetection assertion.

Throughout the design phase, automated traceability tools (TBmanager) were employed to ensure that all LLRs could be traced back to their respective HLRs. This traceability guaranteed that any changes made during the design process were reflected throughout the system, maintaining alignment with DO-178C objective A4.6 (traceability). TBmanager was used to generate and maintain traceability matrices, linking HLRs to their corresponding LLRs. These matrices ensured that each requirement was traceable from its specification to its detailed design. As refinements were made during verification activities, the matrices were continuously updated to reflect the most current relationships between HLRs and LLRs. The integration of automated tools allowed for real-time compliance checks against DO-178C requirements. This approach ensured that the evolving design consistently adhered to certification standards, minimizing the risk of misalignment between HLRs and LLRs during iterative updates.

Figure 5.13 illustrates the traceability matrix generated, which links HLRs to their associated LLRs. The green markers indicate relationships between requirements, ensuring complete coverage and traceability as mandated by DO-178C standards.

Requireme...	LLR-001, Sen...	LLR-002, Sen...	LLR-003, Get...	LLR-004, Get...	LLR-005, Get...	LLR-006, Extr...	LLR-007, Con...	LLR-008, Veri...	LLR-009, Out...	LLR-013, Det...	LLR-014, Est...	LLR-015, Upd...	LLR-016, Mai...	LLR-017, Trac...	LLR-018, Coll...	LLR-019, Thr...	LLR-020, Tim...
HLR_001	X	X	X	X	X	X	X	X	X								
HLR_002		X															
HLR_003		X															
HLR_004										X							
HLR_005											X	X	X	X			
HLR_006															X	X	
HLR_007																	X
HLR_008																	
HLR_009																	
HLR_010																	
HLR_011																	
HLR_012																	
HLR_013																	
HLR_014																	

Figure 5.13 Traceability matrix between HLRs and LLRs.

The FMEA and FTA activities were crucial in identifying potential failure points and assessing their impact on the CAS. These analyses strengthened the robustness and reliability of

the system, uncovering failure modes that could compromise safety and functionality.

The FMEA focused on identifying potential failure modes across each subsystem of the CAS, evaluating their causes, and analyzing their effects. This systematic approach prioritized failure modes based on their severity and proposed mitigations to address vulnerabilities. For example, in the *Traffic Detection subsystem*, failure modes such as *"Traffic detected late"* or *"Traffic detected at incorrect height"* were analyzed. These failures were assessed for their impact on the UAV's ability to respond to collision threats in time. Recommendations included implementing dual-sensing technologies, periodic sensor calibration, and improving data validation protocols (see Appendix B for the complete FMEA report).

Complementing the FMEA, the FTA provided a graphical representation of fault scenarios and their contributing factors. By tracing failures back to their root causes, the FTA highlighted dependencies and vulnerabilities in the system architecture. For the *Traffic Detection* subsystem, the FTA (Figure 5.14) identified key contributors to detection failures, including sensor failure, software errors, communication failures, configuration errors, and physical obstruction.

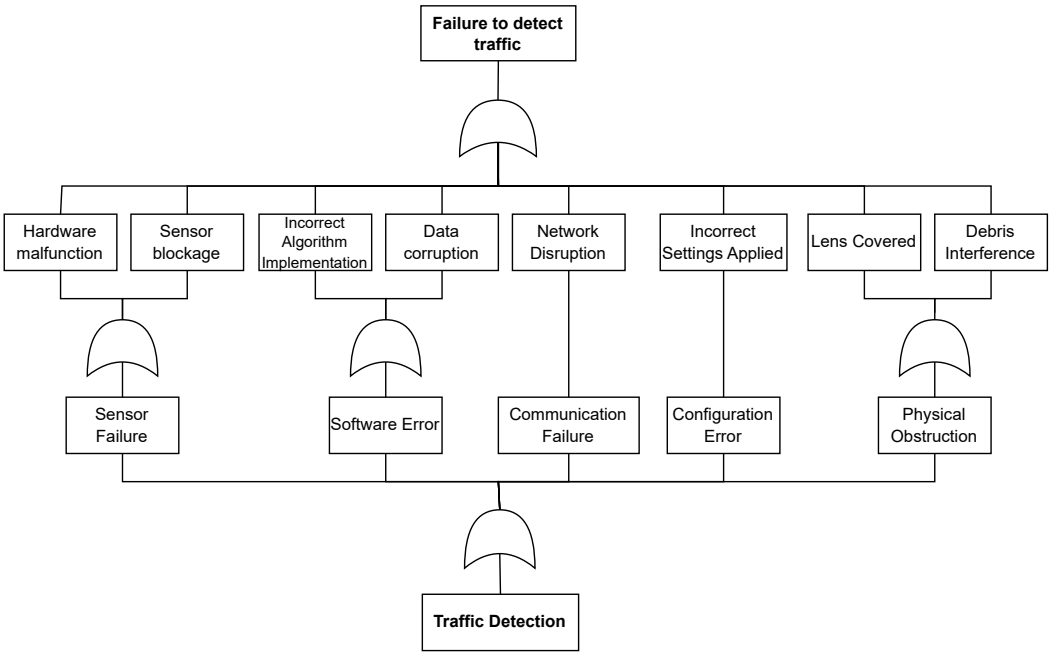


Figure 5.14 FTA for the Traffic Detection subsystem.

The final activity in the design phase involved a comprehensive review of the CAS architecture and design documentation to ensure conformance to all safety and quality standards. This step aimed to validate the readiness of the system design for the subsequent coding phase,

ensuring that all critical safety and operational requirements were met.

To verify the completeness, accuracy, and consistency of the design documentation, independent peer reviews were conducted. These reviews focused on validating that:

- The design aligned with the safety-critical objectives outlined in DO-178C.
- All design artifacts were traceable to their respective HLRs and LLRs.
- The architecture and LLRs were robust and sufficiently detailed to guide the coding phase.

Like the HLR and architecture reviews, the peer reviews employed detailed checklists to ensure systematic evaluation.

By the conclusion of this phase, all defects identified during the reviews and verification activities were resolved. Figure 5.15 presents a summary of the DO-178C objectives fulfilled during the design phase. This table, generated using TBManager, demonstrates 100% compliance with objectives related to LLRs and software architecture, including traceability, consistency, and verifiability. The robust alignment between design activities and certification requirements further validated the readiness of the design for the coding phase.

Objectives	Artifacts	Assets	...	Objective Name	Objective Standard	Objective S...	Placeholders
Table A-4 1	0	3	...	Low-Level requirements comply with high-level requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-4 2	34	2	...	Low-level requirements are accurate and consistent	DO-178C - Populated	Fulfilled	100.00%
Table A-4 3	0	1	...	Low-level requirements are compatible with target computer	DO-178C - Populated	Fulfilled	100.00%
Table A-4 4	34	2	...	Low-Level requirements are verifiable	DO-178C - Populated	Fulfilled	100.00%
Table A-4 5	0	1	...	Low-level requirements conform to standards	DO-178C - Populated	Fulfilled	100.00%
Table A-4 6	50	2	...	Low-level requirements are traceable to high-level requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-4 7	0	1	...	Algorithms are accurate	DO-178C - Populated	Fulfilled	100.00%
Table A-4 8	0	3	...	Software architecture is compatible with high-level requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-4 9	0	3	...	Software architecture is consistent	DO-178C - Populated	Fulfilled	100.00%
Table A-4 10	0	1	...	Software architecture is compatible with target computer	DO-178C - Populated	Fulfilled	100.00%
Table A-4 11	0	3	...	Software architecture is verifiable	DO-178C - Populated	Fulfilled	100.00%
Table A-4 12	0	2	...	Software architecture conforms to standards	DO-178C - Populated	Fulfilled	100.00%
Table A-4 13	0	2	...	Software partitioning integrity is confirmed	DO-178C - Populated	Fulfilled	100.00%

Figure 5.15 DO-178C objectives for verification of outputs of the software design process in TBmanager.

By the end of the design phase, the CAS architecture and LLRs have been thoroughly verified and validated, ensuring that they meet all necessary safety and operational requirements. This provided a robust foundation for the subsequent coding phase, where the verified design would be translated into executable code.

Coding Phase

The Coding phase of the software development process focuses on translating the system design into executable source code. Given the critical nature of the software, a hybrid method was used, combining automated code creation with manual development.

The CAS's functional behaviour was initially modelled in MATLAB Simulink [79], following the specifications outlined in LLRs, DLLRs, and the overall architecture. Simulink models provided a detailed representation of each subsystem, encapsulating their individual behaviours, data flows, and interactions. Figure 5.16 illustrates the Simulink model of the CAS, showcasing its modular subsystems and data flows. It highlights how functional components were structured and interlinked to achieve the overall operational objectives of the system.

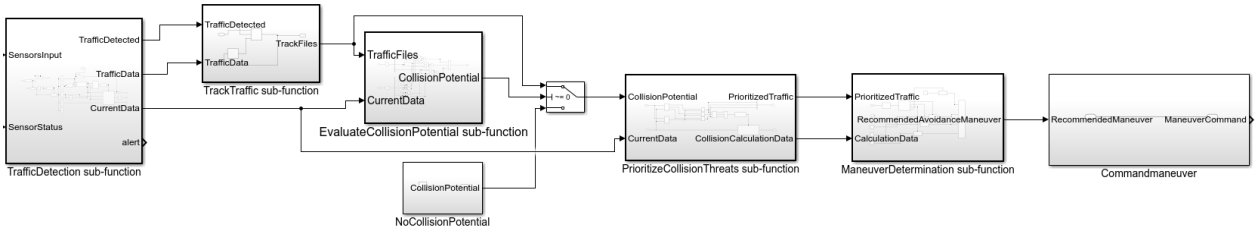


Figure 5.16 Simulink model of the CAS showing subsystems and data flow.

Within Simulink, tests were run to identify and eliminate any typos, logical errors, or inconsistencies in the model, creating a solid basis for code creation. Once validated, the model was translated to C code with Embedded Coder, which created separate .c and .h files for each subsystem, resulting in modular code that is consistent with the system's architectural design.

Although automated code generation formed the majority of the implementation, manual coding was required to integrate the subsystems into a unified software application. This involved programming the interactions and dependencies between subsystems, such as data interchange, control flow, and module timing. These adjustments supplement the automated programming and provide more flexibility in managing complicated interactions inside the system. The generated code complies with DO-178C standards for safety-critical systems, giving close attention to traceability, modularization, and design descriptions.

To preserve traceability, all manual code adjustments were rigorously documented, linking each modification to its corresponding requirements in the LLRs and architectural design. This comprehensive documentation ensured continued adherence to DO-178C objectives, particularly A5.5 (traceability of source code to design).

To integrate the generated and manually adjusted code into the project's configuration and verification framework, all code files were imported into TBManager. This integration enabled comprehensive traceability by mapping code functions back to the LLRs and, subsequently, to the original SRATS through HLRs. As depicted in Figure 5.17, TBManager provides a relational view illustrating the connections between the three levels of requirements and the associated source code procedures.

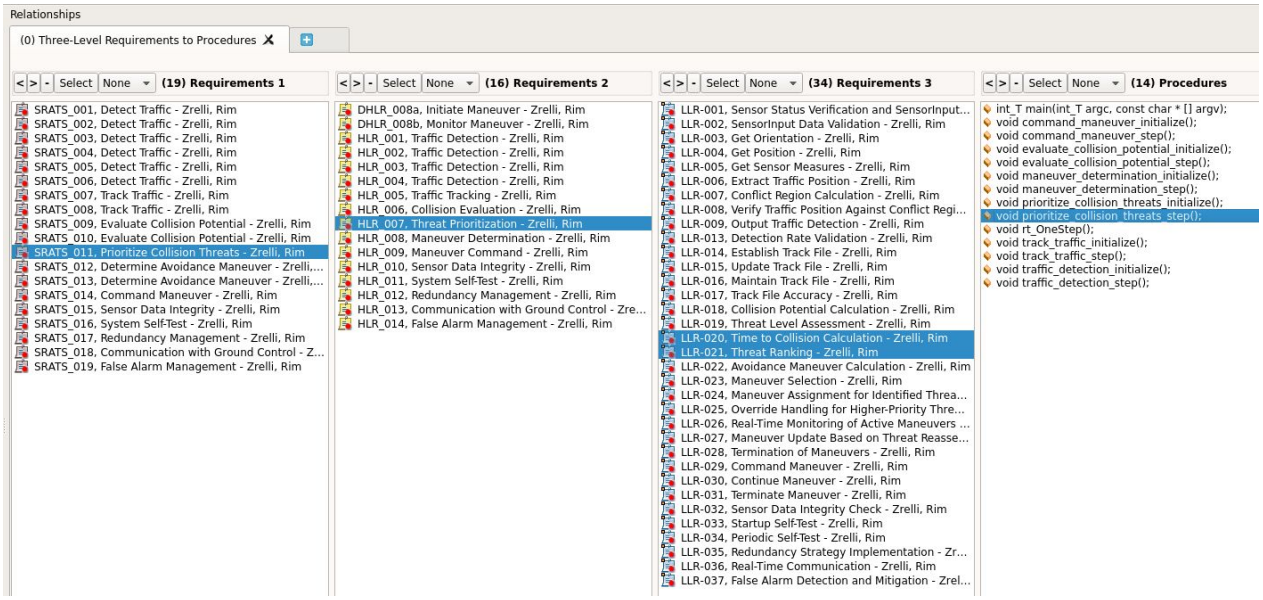


Figure 5.17 TBmanager relationship view showing the traceability between three requirement levels (SRATS, HLR and LLR) and the source code functions.

This traceability structure establishes a clear lineage from high-level system specifications to the implemented code, ensuring that each code function is precisely aligned with the design. It allows for extensive verification, simplifies maintenance tasks, and ensures compliance with DO-178C objectives, particularly A5.5 (traceability of source code to requirements). By maintaining this level of traceability, the project guarantees that the implementation remains consistent with the original design and functional objectives while supporting efficient identification and resolution of potential issues during future updates or audits.

To ensure the correctness and reliability of the code, both static and dynamic code analyses were conducted.

The generated and manually refined code underwent static analysis to detect potential coding standard violations, evaluate code quality, identify data flow issues, and detect runtime anomalies. LDRA's TBvision tool was employed to conduct this analysis, generating comprehensive static analysis reports that flagged potential violations and ensured adherence to

DO-178C objectives A5.4 (conformity to standards) and A5.6 (accuracy and consistency). The analysis began by verifying the source code for conformance with MISRA C 2012 [89], a widely used coding standard for embedded systems. The choice is made as *Embedded Coder* that generated source code uses this standard.

The static analysis began with an initial scan where TBvision found multiple breaches classified as "required", which must be addressed to comply with the standard. This initial assessment marked the start of an iterative procedure in which violations were systematically addressed and then reanalyzed to assure compliance. During these cycles, the interactive refinement of the code resolved the required violations, allowing for increasing conformity with the standard. In the final revisions, all the violations were treated. Achieving coding standard compliance ensures the source code adheres to a predetermined set of guidelines. Part of the results of the code review are summarized in Figure 5.18. Procedures marked as "PASS" indicate that the referred procedures fully adhered to the standard.

Procedure Results +/-

Code Review Result	Procedure	Source File	Unique Violations	Failure Density (Viols/R.Line %)
PASS	Global Program			
PASS	command_maneuver_step	command_maneuver.c	0%	0%
PASS	command_maneuver_initialize	command_maneuver.c	0%	0%
PASS	evaluate_collision_potential_step	evaluate_collision_potential.c	0%	0%
PASS	evaluate_collision_potential_initialize	evaluate_collision_potential.c	0%	0%
PASS	rt_OneStep	main.c	0%	0%

Figure 5.18 Code review report generated by LDRA.

Once compliance with coding standards was achieved, the focus shifted to assessing software quality through a structured evaluation of key metrics: clarity, maintainability, and testability.

- **Clarity**: Ensures the code is easily understandable, facilitating peer reviews, debugging, and future modifications.
- **Maintainability**: Evaluates the ease of implementing changes in response to evolving requirements, ensuring adaptability and resilience over the software lifecycle.
- **Testability**: Measures how readily the code can be tested, which is crucial for ensuring functional accuracy and identifying defects efficiently.

Using LDRA's analysis tools, each metric was thoroughly evaluated, and the results are depicted in Figure 5.19. The analysis revealed that most source files scored exceptionally well

in maintainability and testability, indicating a robust design. While the clarity metric showed areas for improvement in specific files, these did not impact compliance or functionality, but they highlight opportunities for further refinement to enhance code readability.

File Results

File	All Metrics	Clarity	Maintainability	Testability
command_maneuver.c	91%	71%	100%	100%
evaluate_collision_potential.c	91%	71%	100%	100%
main.c	91%	79%	100%	93%
maneuver_determination.c	100%	100%	100%	100%
prioritize_collision_threats.c	96%	86%	100%	100%
track_traffic.c	96%	86%	100%	100%
traffic_detection.c	96%	93%	91%	93%

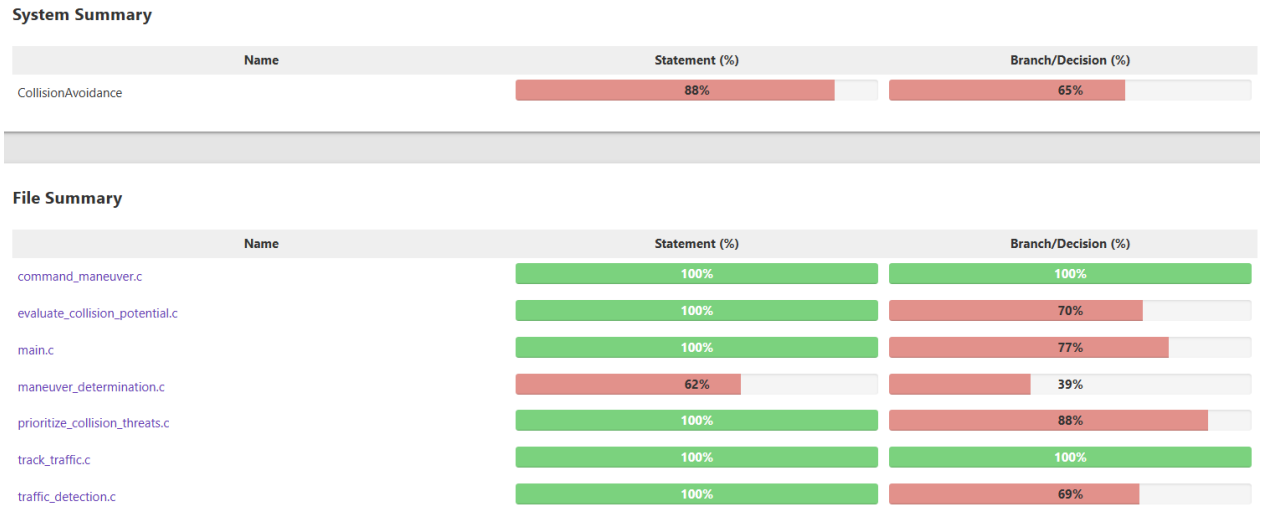
Figure 5.19 Code quality report generated by LDRA.

The dynamic analysis of the code concentrates on determining structural coverage, which evaluates the extent to which software code is exercised under certain test conditions. This analysis provides a dynamic metric essential for ensuring the thoroughness of testing. These criteria are matched to the criticality of the code under test, ensuring proportionate efforts to achieve acceptable coverage. Automated tools are strongly suggested for this process due to their efficiency and accuracy, particularly in complex systems.

The structural coverage analysis was carried out using LDRA's TBvision tool. The process entailed instrumenting the source code, in which LDRA added additional code to monitor execution during testing. We provided simulated inputs to the application execution representing UAV position data and sensor detections, modelling various scenarios of traffic presence or absence. These inputs tested various paths and circumstances within the code. The analysis produced extensive results, as shown in Figure 5.20, that included percentage coverage for key metrics including Statement and Branch/Decision coverage for each file, as well as specific information about untested code paths.

The evaluation was conducted to meet the requirements of DO-178C Level B software, confirming that the coverage metrics were appropriate for this criticality level. The assessment confirmed if the required structural coverage parameters for this safety level were met. While the procedure needed careful planning and iterative testing to resolve coverage gaps, LDRA's thorough insights helped to improve code quality and ensure compliance with stringent avionics software standards. This emphasizes the role of dynamic analysis in showing software dependability and functional safety in critical systems.

The code coverage results provide useful information about the UAV software system's performance and reliability. While 100% Statement and Branch/Decision coverage across all files is desirable, the existing results identify areas for improvement and serve as a benchmark. For



System Summary

Name	Statement (%)	Branch/Decision (%)
CollisionAvoidance	88%	65%

File Summary

Name	Statement (%)	Branch/Decision (%)
command_maneuver.c	100%	100%
evaluate_collision_potential.c	100%	70%
main.c	100%	77%
maneuver_determination.c	62%	39%
prioritize_collision_threats.c	100%	88%
track_traffic.c	100%	100%
traffic_detection.c	100%	69%

Figure 5.20 Code coverage analysis report generated by LDRA.

example, in *maneuver_determination.c*, which obtained 62% Statement and 39% Branch/Decision coverage, the analysis highlighted particular regions where Simulink-generated support functions like *rtIsNaNF* were not exercised during testing. These findings demonstrate the LDRA analysis's effectiveness in identifying potential optimization opportunities, even inside automatically generated code.

The initial results of the code coverage analysis performed using TBvision, with a set of input data, revealed lower-than-expected Statement and Branch/Decision coverage percentages for several files. However, LDRA allows us to exercise the code using test cases run in TBrun, allowing us to refine and expand our testing. We estimate that by using TBrun to run more or more specialized test cases, we will improve the coverage results for the bulk of the files, resulting in more extensive code validation.

Overall, the results demonstrate the effectiveness of using automated techniques in UAV software development. The coverage attained shows that essential areas of the code were thoroughly tested, with detailed reports providing actionable insights for targeted improvements.

Formal verification was employed during the coding phase to ensure that the CAS code adhered to critical safety properties, aligning with DO-178C objectives for verifiability (A5.3) and accuracy (A5.6). Bounded Model Checking [90] was utilized to validate the correctness of code segments by modelling the code logic and verifying it against pre-defined safety properties. This approach provided mathematical assurance of the system's logical soundness and operational reliability.

To this intention, the Efficient SMT-Based Bounded Model Checker (ESBMC) was chosen for its compatibility with the C programming language and its capability to simulate program execution across all possible input scenarios. ESBMC is designed to detect a wide array of potential programming issues, including out-of-bounds array accesses, null or invalid pointer dereferences, memory alignment violations, integer overflows, and floating-point anomalies (e.g., NaNs and division by zero). Additionally, it aids in identifying memory leaks, a critical factor in ensuring code robustness and compliance with DO-178C standards for resilience and dependability.

To use ESBMC for verifying the correctness of our code, the first step is to specify input handling with nondeterministic values that ESBMC can explore during bounded model checking. Specifically, sensor values inputs were defined as *nondet_double()*, ensuring that the model checker can test a wide range of potential values. Furthermore, $\_\_ESBMC\_assume$ is used to apply constraints on the sensor inputs, limiting the values to a suitable range depending on the system's predicted behaviour. This is important because ESBMC works by investigating every possible combination of inputs, and it needs deterministic inputs to verify every possible program execution path.

The verification process leveraged ESBMC's ability to analyze and prove that all states within the program were reachable under the forward condition. This comprehensive analysis was exemplified by the successful verification output shown in Figure 5.21, which demonstrates that the CAS code satisfied all specified properties after exhaustive exploration of execution paths, with a maximum bounded state depth of $k = 33$.

```
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.084s
Solving with solver Boolector 3.2.0
Encoding to solver time: 0.084s
Runtime decision procedure: 0.000s
BMC program time: 0.520s

VERIFICATION SUCCESSFUL

Solution found by the forward condition; all states are reachable (k = 33)
```

Figure 5.21 Bounded Model Checking results using ESBMC for CAS code.

The final step in the coding phase involved a thorough review and automated testing to ensure that the code met all safety and performance standards. LDRA was employed to conduct automated testing and generate test reports, ensuring the code's robustness and compliance.

- Code review: The generated and modified code was systematically reviewed by experts to ensure that it adhered to the system's functional and safety requirements. The review process focused on verifying that the code conformed to the architectural models and complied with DO-178C objective A5.1 (compliance with LLRs).

- Automated testing: Automated test cases were performed using TBrun, where a sequence of test cases was designed and executed to test the source code thoroughly. TBrun enables the execution of procedure calls using the specified test data, guaranteeing that the source code works as intended. The test data was defined using test cases drawn from HLRs and LLRs, ensuring complete coverage of the system's requirements. In TBrun, a white-box analysis was performed, with an emphasis on a thorough structural evaluation of the code under test. This analysis used Coverage Analysis to determine the usefulness of the test data in running the unit tests and guaranteeing the code's functionality.

  A total of 44 test cases were run, with the results indicating that 100% of them passed properly, as depicted in Figure 5.22. A manual review was also performed to confirm that the expected outputs were reached, adding another layer of validation to the tests' accuracy and completeness. This comprehensive method not only evaluated the software's conformance to functional requirements but also helped to ensure the UAV system's overall reliability and safety. These tests ensured that each component functioned correctly and that the code complied with DO-178C objectives A5.1 (compliance with LLRs) and A5.2 (compliance with software architecture).

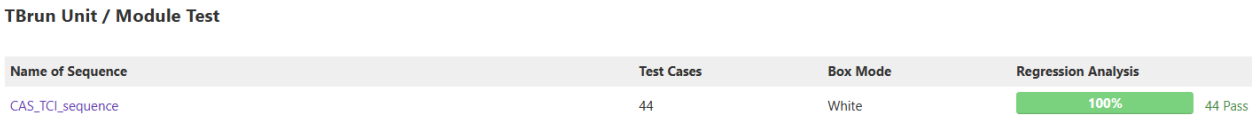


Figure 5.22 TBrun Unit/Module test report.

By the end of the coding phase, the CAS source code was fully generated, verified, and validated through a combination of formal methods, manual reviews, and automated testing. Figure 5.23 provides a summary of how the code satisfied DO-178C coding objectives. This rigorous process ensured the code adhered to all required safety and performance standards, forming a robust and compliant foundation for the subsequent Software Integration phase.

Objectives	Artifacts	Assets	...	Objective Name	Objective Standard	Objective S...	Placeholders
Table A-5 1	42	2	...	Source Code complies with low-level requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-5 2	8	2	...	Source Code complies with software architecture	DO-178C - Populated	Fulfilled	100.00%
Table A-5 3	8	2	...	Source Code is verifiable	DO-178C - Populated	Fulfilled	100.00%
Table A-5 4	3	3	...	Source Code conforms to standards	DO-178C - Populated	Fulfilled	100.00%
Table A-5 5	42	1	...	Source Code is traceable to low-level requirements	DO-178C - Populated	Fulfilled	100.00%
Table A-5 6	8	2	...	Source Code is accurate and consistent	DO-178C - Populated	Fulfilled	100.00%
Table A-5 7	0	1	...	Output of software integration process is complete and correct	DO-178C - Populated	Fulfilled	100.00%
Table A-5 8	0	2	...	Parameter Data Item File is correct and complete	DO-178C - Populated	Fulfilled	100.00%
Table A-5 9	0	1	...	Verification of Parameter Data Item File is achieved	DO-178C - Populated	Fulfilled	100.00%

Figure 5.23 DO-178C objectives for verification of outputs of the software coding process in TBmanager.

5.5.3 Data Collection and Analysis

This section synthesizes the verification and validation (V&V) findings to highlight the CAS's compliance with DO-178C objectives. It consolidates evidence supporting the accuracy, safety, and robustness of the development lifecycle, emphasizing key metrics, traceability, and certification readiness.

Verification and Validation Findings

The V&V process, conducted iteratively across development phases, employed formal methods, peer reviews, and automated tools, yielding high-confidence results. The following points summarize critical outcomes:

1. **Comprehensive system verification**:
    - Formal verification of SRATS, HLRs, and LLRs using the Alloy Analyzer validated logical soundness and consistency. Assertions such as *CompleteTrafficDetection* ensured the CAS accurately detected and tracked traffic under defined constraints (e.g., azimuth and elevation fields of regard).
    - Architectural verification through Spin confirmed compliance with key safety properties, including liveness (ensuring maneuvers are eventually executed for collision threats) and safety (preventing unsafe maneuvers).
2. **Code integrity**:
    - Static code analysis using LDRA TBvision highlighted initial mandatory MISRA C 2012 violations, resolved iteratively to achieve near-complete compliance with

only minor advisory warnings. These results ensured the source code aligned with DO-178C objectives for accuracy, consistency, and adherence to coding standards.

- Formal methods applied to the executable code (via ESBMC) verified memory safety and correctness, identifying and resolving potential anomalies such as null pointer dereferences and integer overflows. These checks fortified the code's resilience against runtime faults.

3. **Dynamic testing**:

   - TBvision and TBrun were used to perform dynamic code analysis for coverage, where particular data inputs were supplied to assess the code's execution. The structural (or code) coverage, which evaluates the sections of the code that were exercised during testing, was the main focus of this investigation. The findings showed significant coverage, with most of the code functions being successfully executed with the given inputs. To guarantee that all functionalities are fully tested and to attain more extensive code coverage, the analysis also pointed out areas that require improvement.

     Alongside those analyses, test cases were created and run in accordance with the system's requirements, guaranteeing that every test matched particular HLRs and LLRs. By matching the test cases to these specifications, the testing procedure proved that the system satisfies its stated goals by validating the code's functionality and demonstrating thorough requirements coverage.

Traceability and Compliance Evidence

Traceability underpinned the CAS development, ensuring all requirements remained aligned through automated tools. Evidence includes:

1. **Traceability matrices**:

   - We generated bidirectional traceability matrices linking SRATS to HLRs (Figure 5.8), HLRs to LLRs (Figure 5.13), and LLRs to source code (Figure 5.24). The matrices demonstrated full coverage and allowed efficient impact analysis for requirement changes.

2. **Coverage reports**:

   - The *Project Coverage Summary* (Figure 5.25) provides a comprehensive overview of the coverage achieved across all levels of requirements (SRATS, HLRs, and

Requireme...	int_T main(in...	void comma...	void comma...	void evaluat...	void evaluat...	void maneuv...	void maneuv...	void prioritiz...	void prioritiz...	void rt_OneS...	void track_tr...	void track_tr...	void traffic_d...	void traffic_d...
LLR-001													X	X
LLR-002													X	X
LLR-003													X	X
LLR-004													X	X
LLR-005													X	X
LLR-006													X	X
LLR-007													X	X
LLR-008													X	X
LLR-009													X	X
LLR-013													X	X
LLR-014											X	X		
LLR-015											X	X		
LLR-016											X	X		
LLR-017											X	X		
LLR-018				X	X									
LLR-019				X	X									
LLR-020								X	X					
LLR-021								X	X					
LLR-022						X	X							
LLR-023						X	X							
LLR-024						X	X							
LLR-025						X	X							
LLR-026						X	X							
LLR-027						X	X							
LLR-028						X	X							
LLR-029		X	X											

Figure 5.24 Traceability matrix between LLRs and code.

LLRs) and the associated test cases. The report highlights the relationships and traceability between requirements and test cases, with each set organized into specific groups. The report confirmed full traceability from SRATS to HLRs and LLRs, ensuring that every software requirement was adequately addressed and tested during the verification phase. The report shows a Project Coverage of 100%, indicating that all requirements and test cases across the Groups have been covered. Achieving this level of coverage validates the completeness of the verification activities and ensures compliance with certification standards.

3. **Defect management**:

   - Automated defect reporting tracked issues across phases, with key findings recorded in a consolidated *Defect Summary Report*. Figure 5.26 illustrates resolved defects, categorized by severity, confirming adherence to iterative improvement principles.

The aggregated results underscore the readiness of the CAS for DO-178C certification, supported by a comprehensive suite of evidence. Verified traceability artifacts clearly demonstrate alignment with DO-178C objectives, ensuring that requirements, design elements, and implementation remain consistently connected throughout the development lifecycle. Formal verification results further validate the logical integrity and correctness of the system, ad-

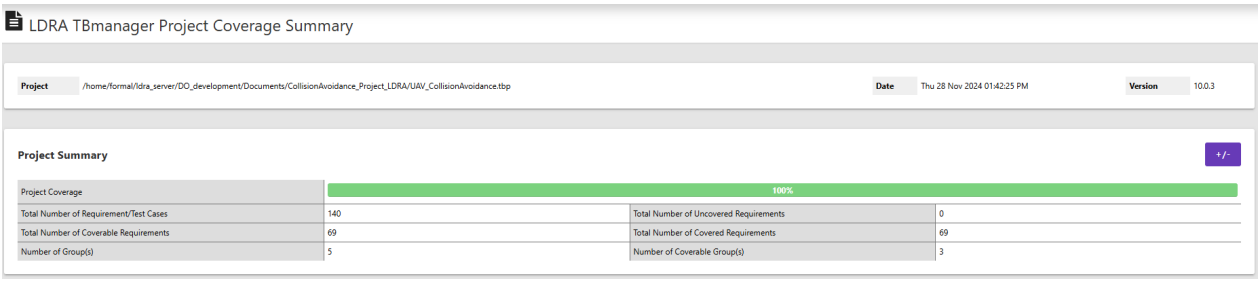


Figure 5.25 Project coverage summary for CAS requirements and test cases.

Defects

Parents	Defect Report Number	Priority	DR Description	Status
(2) HLR_008, HLR_009	PR_HLR_Defect_001	Medium	Defects found during Alloy Specifi...	Closed
(12) HLR_001, HLR_002, HLR_003, HLR...	PR_HLR_Defect_002	Low	A peer-review was done and Alloy...	Closed
(2) DHLR_008a, DHLR_008b	PR_HLR_Defect_003	Low	A peer-review was done and Alloy...	Closed
(1) LLR-002	PR_LLR_Defect_001	Medium	Traceability? This LLR traces HLR_...	Closed
(2) LLR-013, LLR-016	PR_LLR_Defect_003	Medium	Those LLR are too high-level. The...	Closed
(1) LLR-014	PR_LLR_Defect_004	Medium	Why "establish" meas? Try to use ...	Closed
(1) LLR-017	PR_LLR_Defect_005	Medium	This LLR is too high level. - The u...	Closed
(2) LLR-025, LLR-027	PR_LLR_Defect_006	Medium	Defect found during Alloy Specific...	Closed
(26) LLR-001, LLR-003, LLR-004, LLR-0...	PR_LLR_Defects_007	Low	A peer-review was done and an Al...	Closed
(2) SRATS_012, SRATS_013	PR_SASR_Defect_001	Medium	Defects found during Alloy Specifi...	Closed
(17) SRATS_001, SRATS_010, SRATS_01...	PR_SASR_Defects	Low	A peer-review was done and an Al...	Closed
(1) LLR-015	PR_LLR_Defect_008	Medium	What is the X updates per second...	Closed

Figure 5.26 Defect summary report.

dressing critical safety and functional properties. Additionally, extensive static and dynamic analysis outputs affirm the runtime reliability of the system, confirming its conformance to industry standards and enhancing confidence in its operational robustness.

5.6 Results and Discussion

5.6.1 Evaluation Against DO-178C Objectives

The methodology's effectiveness in achieving compliance with DO-178C was evaluated by systematically addressing its objectives through rigorous verification and validation activities. The process integrated formal methods with automated tools, ensuring that development phases, from requirements to coding, adhered to certification standards. Table 5.3 provides a consolidated view of the verification activities, tools employed, outcomes generated, and corresponding DO-178C objectives.

Key milestones in this evaluation include:

1. **Verification of Requirements and traceability**: The development began with the formal verification of SRATS, which were refined to eliminate inconsistencies and

ambiguities. Tools such as Alloy ensured that HLRs and detailed LLRs maintained logical integrity and precise alignment with system objectives. Traceability was managed comprehensively across all requirement levels using TBmanager, which facilitated the generation of traceability matrices. These matrices confirmed bidirectional traceability essential for compliance with DO-178C objectives A3.6 and A4.6.

2. **Design phase evaluation**: The architectural models, developed using AADL and verified with SPIN, were validated for logical consistency and compliance with system constraints. Liveness and safety properties were analyzed through LTL claims to confirm the correctness of subsystem interactions. The iterative use of model-checking tools provided continuous refinement of design artifacts, addressing objectives A4.8, A4.9 and A4.11.

3. **Code verification and static/dynamic analysis**: The coding phase was fortified through automated static and dynamic analysis using TBvision. Static analysis identified and resolved coding standard violations, ensuring conformity with MISRA C 2012, code quality (clarity, maintainability and testability) and alignment with DO-178C objective A5.4. Dynamic testing validated runtime behaviour, addressing code coverage. Formal verification, supported by ESBMC, further ensured the logical consistency and verifiability of the implemented code, as required by A5.3.

4. **Review processes and certification artifacts**: Peer reviews were employed during the requirements, design, and coding phases providing an independent evaluation of artifacts against DO-178C criteria. These reviews, coupled with automated traceability and documentation tools, ensured that certification deliverables such as traceability matrices and accomplishment summaries adhered to DO-178C objectives.

The comprehensive approach applied to the CAS project demonstrates the efficacy of combining formal methods and automation to address DO-178C requirements. By leveraging a structured methodology, the development process achieved early defect detection, robust traceability, and seamless generation of certification artifacts. These outcomes underscore the scalability and effectiveness of the methodology in certifying complex safety-critical systems, paving the way for its application in broader domains.

5.6.2 Impact of Methodology

The proposed methodology represents a significant advancement in addressing the challenges of developing DO-178C-compliant UAV software. By integrating formal methods with auto-

Table 5.3 Summary of Verification and Traceability Activities, Tools, Outputs, and DO-178C Objectives Addressed.

Activity	**Tool Used**	**Outcome**	**DO-178C Objective**
SRATS Verification	Alloy	Inconsistencies resolved, refined SRATS	
HLR Verification	Alloy	Logical consistency, compliance with SRATS, refinement of HLRs, defining DHLRs	A3.1, A3.2, A3.4
Design Model Verification	Spin	Verified design models, Logical consistency. Compliance with system constraints	A4.8, A4.9, A4.11
LLR Verification	Alloy	Logical consistency. Compliance with HLRs verified	A4.1, A4.2, A4.4
Code Verification	ESBMC	Verified code	A5.3, A5.6
Automated Static Analysis	TBvision	Code reviewed (Coding standards and quality review)	A5.4
Automated Dynamic Testing	TBvision / TBrun	Validation of runtime behaviour (Code coverage and MC/DC planner)	A5.3
Traceability Management (SRATS-HLRs)	TBmanager	Generated traceability matrices. Ensured bidirectional traceability	A3.6
Traceability Management (HLRs-LLRs)	TBmanager	Maintained traceability from HLRs to LLRs. Consistent updates across the lifecycle	A4.6
Traceability Management (LLRs-Code)	TBmanager	Maintained traceability from LLRs to code functions	A5.5
Peer Review	Review Checklists	Review records, resolved issues	A5.1, A5.2

mated verification tools, this approach enhances both the rigour and efficiency of the software lifecycle, directly impacting compliance, traceability, and overall system safety.

Enhancements to Certification Compliance

The integration of formal methods into the requirements and design phases ensures early detection of ambiguities and inconsistencies. For example, Alloy models were employed to validate logical soundness in both SRATS and HLRs, while Spin confirmed liveness and safety properties in the software architecture. These formal verification activities streamline adherence to DO-178C objectives such as verifiability, accuracy and consistency. This contrasts sharply with traditional methods that rely heavily on manual reviews, which can miss subtle errors, particularly in complex systems.

The methodology also leverages qualified tools to automate and guarantee traceability, structural coverage analysis, and certification evidence generation. This automation addresses the objectives of traceability, verifiability and conformity to standards reducing manual effort and improving the reliability of deliverables. Comparatively, the study on software-defined radio systems [53] highlights the potential for automated testing in certification processes but underscores the limitations of partial tool integration. In contrast, our methodology integrates both formal and automated approaches seamlessly, yielding a more robust certification process.

Efficiency Gains in Safety-Critical Development

The unified framework of formal methods and automated tools enhances efficiency by reducing redundant tasks and manual oversight. For example, TBmanager automates traceability matrix generation, linking requirements to code functions and verification results. Similarly, automated testing reduces the time for structural coverage validation and compliance checks. This efficiency is particularly beneficial in meeting the stringent timelines often associated with UAV certification projects, as evidenced in the study [5], which highlighted the resource intensity of certifying adaptive learning agents under DO-178C.

In terms of lifecycle coverage, the use of MBD methodologies complements formal verification by providing a visual and systematic representation of requirements and architecture. The work by [50] also acknowledges the scalability of MBD approaches, especially when integrated with domain-specific modelling languages. However, their findings emphasize the necessity of aligning models with certification requirements, a gap our methodology addresses by embedding formal validation within the MBD process.

Addressing Complex System Challenges

The proposed methodology effectively mitigates the challenges posed by system complexity, especially for UAVs with adaptive or autonomous functionalities. As demonstrated in the case study on the CAS, formal methods ensure the deterministic behaviour of critical functions like maneuver determination. This aligns with recommendations by the study [54], who advocate for model checking and abstraction techniques to ensure compliance with DO-178C in autonomous systems.

Comparative Insights and Limitations

Compared to alternative methodologies, such as the process-oriented build tool for airborne software development discussed by [52], our approach demonstrates greater integration and automation. While the process-oriented tool excels in managing configurations and streamlining testing, it does not incorporate the formal rigour necessary for early-stage error detection. By combining these strengths, our methodology presents a balanced approach to achieving both precision and scalability.

However, the reliance on qualified tools poses a limitation, as the certification process becomes tool-dependent. This highlights a need for further exploration into tool qualification strategies to mitigate dependency risks. Additionally, while formal methods offer unparalleled rigour, their application requires specialized expertise, which can increase the initial learning curve for teams transitioning from traditional practices.

5.6.3 Case Study Insights

The implementation of the CAS for UAVs presented several challenges and required iterative refinements to address discrepancies uncovered during formal verification and automated analyses. This section outlines the key insights gained during the development process, focusing on lessons learned from applying the integrated methodology of formal methods and automated tools, challenges encountered, and refinements made to achieve compliance with DO-178C standards.

Challenges Encountered

One of the primary challenges was to manage virtual subsystems faced during code generation using Embedded Coder from the Simulink model. When attempting to generate code for the entire model, which comprised several subsystems, many of these were treated as

virtual subsystems, resulting in no code being generated for them. The approach used to solve this problem was to generate code for every subsystem separately, making sure that all required code was generated. Each subsystem's independently generated code was subsequently manually combined to provide a comprehensive and useful system implementation. While this approach required additional effort in integration, it ensured that all subsystems were properly represented in the final code.

Another significant challenge was related to performing test coverage analysis for the generated code. Initially, there were difficulties in configuring the build environment within LDRA to support the analysis. After this configuration was in place, preliminary results showed that coverage levels were below expectations and that numerous statements and branch/decision points were not covered. Addressing this required analyzing the code in detail and identifying additional possibilities for sensor inputs that could be streamed to improve coverage. The outcomes demonstrated significant improvements once the strategy was put into practice and several iterations were carried out in Tbvision and TBrun. However, some functions generated by Simulink to support the logic—such as utility functions, remained unexercised, which prevented the branch/decision coverage from reaching optimal levels.

Additionally, the initial code analysis revealed numerous coding standard violations and traceability gaps, emphasizing the need for iterative corrections. Early phases involved extensive learning to utilize LDRA's features effectively, including structural coverage analysis and traceability management. The absence of complete familiarity with tool functionalities posed delays, highlighting the importance of training and incremental tool adoption in complex projects.

Iterative Refinements

To resolve uncovered issues, iterative refinements were applied at multiple levels:

1. Refining prioritization logic: Formal verification using Alloy and model-checking tools pinpointed unresolved states in prioritization. Specific refinements included:
    - Clearly defining activation thresholds for collision maneuvers.
    - Introducing dynamic conditions for prioritizing simultaneous threats, ensuring robustness against edge cases.
2. Subsystem integration: The separately generated code for each subsystem was then manually integrated, allowing for a complete and functional implementation of the system.

3. Enhanced test generation: A systematic approach to generating test cases was adopted, leveraging path analysis capabilities in TBvision and TBrun to cover untested branches. Automated testing was integrated with dynamic analysis tools to iteratively close coverage gaps, ultimately achieving compliance with DO-178C objectives for MC/DC coverage.

Lessons Learned

1. Iterative refinements mitigate early risks:
   Iterative verification cycles proved vital in uncovering latent issues early. For instance, addressing the *InadequateCollisionThreatHandling* assertion during the requirements phase prevented logical inconsistencies from propagating to later stages, highlighting the importance of early-stage corrections in reducing downstream rework.

2. Automation enhances certification readiness:
   Automated tools, particularly the LDRA suite, accelerated key compliance activities by streamlining traceability and artifact generation. Notably, automated test generation significantly improved Modified Condition/Decision Coverage (MC/DC) and reduced manual verification workload, though the initial learning curve emphasized the need for dedicated training.

3. Formal methods foster precision:
   The precision offered by formal methods like Alloy and SPIN was pivotal in validating logical consistency and resolving ambiguities across requirements and design. SPIN's validation of liveness properties for collision maneuvers demonstrated the role of model-checking in providing robust safety assurances under diverse operational conditions.

4. Collaboration drives clarity:
   Effective collaboration between domain experts and developers ensured ambiguities in requirements were resolved efficiently. The shared language provided by formal modelling tools such as Alloy bridged communication gaps, ensuring alignment on safety-critical objectives and fostering a unified understanding of system behaviours.

5.6.4 Practical Implications: Best Practices for DO-178C Compliance

This section consolidates key insights from the methodology and case study implementation into actionable best practices for achieving DO-178C compliance. These practices address both technical and procedural challenges inherent in the development of safety-critical software.

Establish Rigorous Requirements Engineering Practices

The foundational step in DO-178C compliance is developing robust, unambiguous requirements. The methodology demonstrated the importance of iterative refinements during the SRATS and HLR verification phases, employing formal methods like Alloy to uncover latent ambiguities and inconsistencies early. Best practices include:

- Quantifiable requirements: Ensure requirements are measurable with precise tolerances to facilitate unambiguous verification.
- Clear terminology and rationale: Use consistent language and separate explanatory rationale to prevent misinterpretation during implementation and review.

Leverage Automation for Compliance Efficiency

Automated tools significantly streamline the verification process and compliance activities. Key strategies include:

- Automated traceability: Use tools to maintain bidirectional traceability across requirements, design, and code. This not only satisfies DO-178C objectives but also reduces manual effort and errors.
- Enhanced structural coverage analysis: Automated test case generation was critical in achieving MC/DC coverage, especially in addressing previously uncovered branches during dynamic analysis.

Integrate Formal Methods for Robust Verification

The use of formal methods such as SPIN for architectural model verification and bounded model checking for resolving design-level ambiguities ensured logical consistency and validated safety properties. These methods should be strategically integrated to:

- Validate critical properties: Analyze liveness, safety, and interaction properties to reinforce system reliability under operational scenarios.
- Ensure early detection of errors: Incorporate formal verification early in the lifecycle to mitigate downstream defects, as evidenced by resolving critical assertions in the prioritization logic.

Balance Modularity and Integration

The modular design of the CAS introduced challenges during the integration phase, particularly due to virtual subsystems in Simulink models. Effective practices include:

- Subsystem validation: Perform independent verification of each module before integration, followed by rigorous validation of interaction flows to ensure consistency with architectural requirements.
- Integration testing automation: Leverage tools capable of automating integration testing to validate interaction scenarios across multiple subsystems.

Continuous Learning and Collaboration

The iterative nature of the methodology revealed the importance of ongoing training and interdisciplinary collaboration:

- Training on specialized tools: Invest in comprehensive training for formal tools and automated suites to overcome initial learning curves, ensuring effective utilization.
- Cross-disciplinary communication: Encourage collaboration between domain experts and developers to align safety-critical objectives with practical implementation constraints.

Prioritize Certification-Ready Artifacts

The methodology underscored the need for generating certification-ready artifacts at every phase. Key practices include:

- Integrated documentation: Produce comprehensive artifacts, such as traceability matrices, requirement refinement logs, and structural coverage reports, ensuring they are aligned with DO-178C objectives.
- Iterative artifact validation: Use verification tools to iteratively validate certification artifacts, addressing discrepancies proactively.

By applying these best practices, organizations can navigate the complexities of DO-178C compliance with greater efficiency and confidence. The integration of formal methods, automation, and iterative refinement into the lifecycle not only ensures compliance but also enhances the safety and reliability of safety-critical software. These practices, validated through

the CAS case study, highlight their scalability and applicability across broader domains in avionics software certification.

5.7 Chapter Summary

In this chapter, we explored the integration of formal methods and automated verification tools to address the challenges of achieving DO-178C compliance in UAV software development. The proposed methodology combined the mathematical rigour of formal methods with the efficiency and scalability of automated tools for traceability, validation, and certification evidence generation. Each phase of the software lifecycle was addressed through a unified framework that ensures both precision and compliance. A case study involving a UAV CAS demonstrated the real-world applicability of this approach, highlighting its capacity for rigorous validation and practical efficiency in meeting DO-178C standards.

CHAPTER 6 DEVELOPING A CONTROLLED NATURAL LANGUAGE AND RULE-BASED VERIFICATION SYSTEM

6.1 Chapter Overview

However, the inherent ambiguity and variability of NL requirements remain a persistent challenge in ensuring precise, consistent, and verifiable specifications that meet DO-178C standards. To address this gap, this chapter introduces a CNL tailored for the DO-178C standard, coupled with a rule-based verification system designed to ensure CNL compliance and support the automation of verification tasks.

This chapter aims to reduce the ambiguities in NL requirements through a constrained linguistic framework that maintains readability while improving accuracy and consistency. By constraining NL with syntactical and lexical rules that enforce clarity, the CNL can reduce the risk of misinterpretation and non-compliance in airborne software systems. Additionally, a rule-based verification system was developed to enforce these linguistic constraints, automatically flagging any non-compliant requirements and ensuring that all CNL-based requirements align with DO-178C's stringent compliance criteria.

This chapter builds on the hypothesis that integrating CNL with an automated rule-based verification system can enhance DO-178C compliance by facilitating unambiguous and verifiable requirements. To test this hypothesis, we designed an empirical evaluation to assess the effectiveness of the CNL and rule-based system using case studies and error analysis. This evaluation measures the impact of the CNL on requirement clarity and the ability of the rule-based system to enforce compliance with minimal human intervention.

By aligning CNL's linguistic structure with DO-178C's compliance needs and automating its verification through a rule-based system, this chapter aims to contribute an approach to improving specification quality and compliance efficiency in airborne software development. This work holds the potential to not only enhance verification accuracy but also streamline the compliance process, ultimately contributing to safer and more reliable airborne systems.

6.2 Study Design and Objective Setting

The study design for developing the CNL and rule-based verification system follows a structured approach to provide a clear and enforceable framework for DO-178C-compliant software requirements. This system combines linguistic constraints with an automated verification

process to ensure that requirements are precise, unambiguous, and aligned with regulatory standards. Figure 6.1 provides a schematic overview of the study design, detailing the stages from CNL development and rule creation to system evaluation and case studies. Each phase was carefully designed to uphold the rigorous standards required for aerospace software compliance. Each step in the process is designed to enhance the clarity, verifiability, and regulatory alignment of requirements, ensuring that the CNL and rule-based system provide a robust solution for DO-178C compliance in aerospace software development.

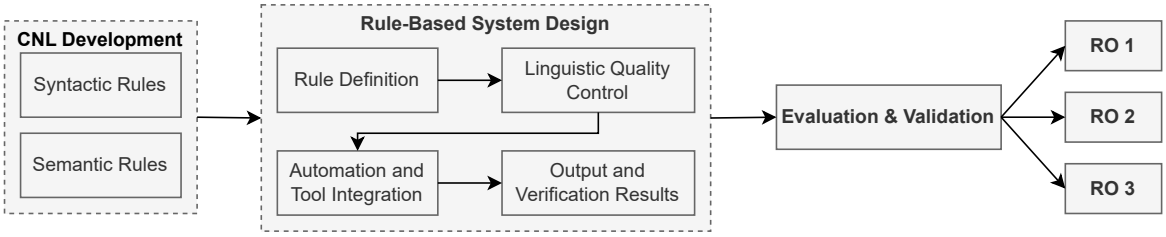


Figure 6.1 Schematic diagram of the study.

The primary objectives guiding this study are as follows:

- **To develop a CNL tailored for DO-178C-compliant software requirements (RO1)**: The study aims to address the ambiguity common in NL requirements by creating a structured language that applies strict linguistic constraints aligned with DO-178C guidelines. This objective focuses on defining syntactic and semantic rules that ensure clarity and enforceable consistency, reducing misinterpretation risks and enhancing compliance with regulatory standards.

- **To create a rule-based verification system that automatically checks for compliance with CNL constraints (RO2)**: This objective emphasizes the automation of requirement verification through a rule-based system. By embedding specific DO-178C compliance requirements within the verification rules, the system systematically checks each requirement for alignment with CNL constraints. This automation aims to reduce the manual workload in verification processes, improving consistency and reducing error rates.

- **To evaluate the effectiveness of the CNL and rule-based verification system in reducing ambiguity and improving verification efficiency (RO3)**: The third objective focuses on assessing the practical impact of the CNL and rule-based system on the quality of requirements and verification processes. This includes measuring the reduction in ambiguities, assessing verification time and accuracy improvements,

and analyzing user feedback. This evaluation will combine quantitative metrics with qualitative insights to validate the system's effectiveness in a DO-178C compliance context.

6.3 Design and Development of the CNL for DO-178C Compliance

6.3.1 Linguistic Constraints and Rules in CNL

In developing a CNL to ensure DO-178C compliance, it is essential to implement a series of linguistic constraints and rules that ensure clarity, verifiability, and traceability in software requirements. These rules are designed to address common ambiguities and inconsistencies in NL requirements, supporting alignment with DO-178C standards for aerospace software development. By formalizing these rules, the CNL provides a structured approach to requirements documentation, reducing interpretative discrepancies and ensuring all stakeholders can uniformly interpret and verify requirements.

To provide an organized and comprehensive approach for the CNL rule implementation, Table 6.1 categorizes each linguistic constraint and rule by function. The structure includes five distinct categories, each represented by an abbreviation that aids in cross-referencing the rules throughout this chapter. The abbreviations signify different rule types and highlight the primary purpose of each category within the CNL framework:

1. **SYN** (Syntactic Rules): These rules establish a consistent sentence structure in requirements documentation, minimizing ambiguity through a clear Actor-Action-Object format.

2. **SEM** (Semantic Rules): Focusing on the use of consistent vocabulary and avoiding ambiguous terms, semantic rules help maintain clarity of meaning across requirements.

3. **CON** (Conditions and Constraints): This category specifies how conditions and constraints should be articulated in the CNL.

4. **TRA** (Traceability Mechanisms): Traceability rules support tracking requirements from high-level design through to implementation, which is crucial for maintaining structured documentation.

5. **ADV** (Additional Rules for Compliance): Drawing from best practices in aerospace guides [91, 92], these additional rules further enhance clarity, conciseness, and interpretative accuracy.

Table 6.1 Rules and Constraints for Requirement Specification.

ID	Rule/Constraint	Description	Example
SYN1	Actor Action Object Structure	Each requirement follows a structured format of actor (component), action (function), and object (target). This rule minimizes ambiguity and promotes readability.	"The autopilot system shall maintain altitude within 100 feet."
SYN2	Quantitative Expressions	Requirements should use specific, quantitative items (e.g., "within 5 ms") rather than vague expressions to ensure measurability and verifiability.	"The system shall respond within 100 ms $\pm$ 5 ms."
SEM1	Defined Terms and Consistent Vocabulary	Employ a consistent vocabulary and avoid synonyms or ambiguous terminology. DO-178C compliance requires clarity and prevents misinterpretation.	Define "Altitude" as per the project glossary, ensuring consistency across all requirements.
SEM2	Avoid Implementation Details in High-Level Requirements	High-level requirements should describe "what" rather than "how," focusing on functionality rather than implementation specifics.	"The system shall calculate altitude based on sensor input," instead of specific algorithms.
CON1	Explicit Conditionals (IF THEN Structures)	Conditionals must be specific, with preconditions clearly articulated, allowing testers to verify requirements in specific scenarios.	"If the temperature exceeds 250°C, the system shall initiate cooldown procedures within 10 seconds."

Continued on next page

ID	Rule/Constraint	Description	Example
CON2	Timing and Response Constraints	Timing constraints, critical in aerospace, should be specified explicitly (e.g., response time $\leq$ x ms for alerts) to ensure DO-178C compliance.	"The response time shall not exceed 5 ms for critical alerts."
TRA1	Unique Requirement IDs	Assign unique identifiers to requirements to enable traceability throughout the software development lifecycle.	"HLR-001: The autopilot system shall maintain altitude."
TRA2	Cross Referencing Derived Requirements	Requirements derived from others should include references to the originating requirement, facilitating a clear link between high-level and derived requirements.	"DHLR-001.1: Derived from 'HLR-001' for altitude control."
ADV1	Active Voice	Requirements should use active voice to improve clarity. Specify the actor directly and avoid using ambiguous language.	"The system shall display an alert within 2 seconds."
ADV2	Avoid Vague Terms	Avoid vague terms such as "some," "any," or "several" to maintain specificity and reduce ambiguity.	Replace "adequate response" with "response within 2 seconds."
ADV3	Single Sentence per Requirement	Each requirement should be expressed as a single, complete thought to reduce complexity and support traceability.	"The system shall issue an alert within 10 seconds when the battery charge falls below 20%."
ADV4	Positive Statements	Requirements should specify expected behavior in positive terms to reduce ambiguity and cognitive load during verification.	"The system shall complete shutdown within 5 seconds if the temperature exceeds the safe threshold."

Continued on next page

ID	Rule/Constraint	Description	Example
ADV5	Avoid Pronouns	Pronouns like "this," "these," or "it" can introduce ambiguity. Use explicit references for clarity.	"The system shall initiate diagnostic checks upon request."

6.3.2 Examples of CNL Transformation

To illustrate the effectiveness of the CNL framework developed for DO-178C compliance, this section presents examples (Table 6.2) of how ambiguous or inconsistent NL requirements are transformed into CNL-compliant statements. Each transformation aims to enhance clarity, verifiability, and traceability, aligning with DO-178C's stringent requirements for unambiguous and testable documentation. These examples demonstrate how the application of syntactic and semantic rules within the CNL can reduce interpretation discrepancies and ensure that requirements meet certification standards.

6.3.3 Mapping CNL Rules to DO-178C Requirements

To ensure effective alignment of CNL rules with DO-178C requirements, this mapping identifies how each CNL rule type supports the objectives stipulated by DO-178C for aerospace software development. DO-178C emphasizes specific objectives, including accuracy, verifiability, traceability, completeness, and conformity to standards, each essential for certifying software in aviation systems. By structuring CNL rules around these objectives, we establish a pathway to regulatory compliance through precise, testable, and traceable requirements.

1. **Accuracy and Consistency**: DO-178C emphasizes that all requirements must be accurate, reflecting the correct, intended functionality without ambiguity, and consistent, with no contradictions across requirements. This objective prevents discrepancies or conflicting interpretations within the system documentation.

    - SYN1 (Actor-Action-Object Structure) and SYN2 (Quantitative Expressions) enforce a structured, uniform requirement format with a clear actor, action, and object and specific quantitative terms, reducing ambiguity and supporting DO-178C's need for requirements that are direct and easy to interpret.
    - SEM1 (Defined Terms and Consistent Vocabulary) promotes consistency by enforcing a uniform vocabulary, preventing discrepancies that could lead to misinterpretation.

- ADV1 (Active Voice) further reduces ambiguity in actor responsibilities by specifying who performs each action.

2. **Verifiability**: To meet DO-178C standards, each requirement must be verifiable, meaning it should have clear, objective criteria that can be tested or otherwise validated through systematic review or analysis.

   - SYN2 (Quantitative Expressions) and CON1 (Explicit Conditionals) directly support verifiability by requiring measurable criteria, such as response times or conditions, facilitating objective testing and validation, which is essential to DO-178C's verifiability objectives.
   - ADV4 (Positive Statements) enhances verifiability by specifying expected behaviours directly, reducing ambiguities that could complicate testing or validation.
   - ADV2 (Avoid Vague Terms) further enhances verifiability by enforcing precise terminology, ensuring requirements are testable without subjective interpretation.

3. **Traceability**: Traceability within DO-178C is essential for tracking requirements across the development lifecycle, from high-level specifications to implementation and testing, ensuring every aspect of the software is linked to the original system objectives.

   - TRA1 (Unique Requirement IDs) and TRA2 (Cross-Referencing Derived Requirements) facilitate traceability by assigning a unique identifier to each requirement and establishing cross-references for derived requirements, supporting DO-178C's need for bidirectional traceability across lifecycle stages.
   - SEM1 (Defined Terms and Consistent Vocabulary) indirectly supports traceability by ensuring terminology remains consistent across documentation, making it easier to follow requirements across various stages and artifacts within the project lifecycle.
   - ADV5 (Avoid Pronouns) contributes to traceability by using explicit terms rather than pronouns, reducing potential ambiguity.

4. **Completeness**: Completeness is critical under DO-178C, ensuring that all necessary functionality, safety, and performance requirements are accounted for without omissions, supporting overall system integrity and reliability.

   - CON1 (Explicit Conditionals) helps fulfill completeness by articulating all relevant conditions, covering the range of operational scenarios necessary for software verification and validation.

- SEM2 (Avoid Implementation Details in High-Level Requirements) helps maintain completeness by focusing requirements on the "what" rather than the "how," preserving high-level functionality and intent without delving into design specifics.

5. **Measurability and Testability**: Requirements should include quantitative measures where applicable, such as specific performance thresholds or response times, to enable objective assessment. Measurability and testability are vital for ensuring each requirement has a clear, objective criterion for confirming its fulfillment.

    - SYN2 (Quantitative Expressions) and CON2 (Timing and Response Constraints) directly support DO-178C's objective of testability by specifying measurable terms, enabling clear, objective assessments, and meeting the standard's criteria for measurable requirements.
    - ADV4 (Positive Statements) further enhances testability by framing each requirement as a direct, actionable statement, reducing potential ambiguities and streamlining validation processes.

6. **Conformance to Standards**: Conformance to standards is critical for ensuring documentation and requirement structures adhere to pre-established criteria, essential for regulatory review and certification.

    - ADV3 (Single Sentence per Requirement) and SEM1 (Defined Terms and Consistent Vocabulary) support standard conformance by promoting concise, consistent terminology, eliminating synonyms or complex sentences that could complicate regulatory review.
    - TRA1 (Unique Requirement IDs) further supports conformance by providing an organized structure to the requirement set, enabling each item to be easily referenced, discussed, and validated in alignment with DO-178C's structured documentation mandates.

7. **Derived Requirements Management**: Any derived requirements developed during software design and development should be identified, traced, and justified. This ensures that all derived functions align with original system goals and safety considerations.

    - TRA2 (Cross-Referencing Derived Requirements) directly aligns with DO-178C's requirements management expectations by establishing cross-references for all derived requirements, ensuring that derived items maintain a clear link to system-level goals, fulfilling DO-178C's need for transparent derivation management.

6.3.4 Rule Selection and Validation Process

The selection and validation of CNL rules were conducted through an iterative, structured process designed to ensure that the rules align with DO-178C standards for consistency, traceability, verifiability, and compliance. This approach involved initial rule selection, expert review, empirical validation, and documentation of the final rule set, each step reinforcing the applicability and effectiveness of the rules in addressing DO-178C objectives. Figure 6.2 provides a schematic flow to visually illustrate the selection and validation stages.

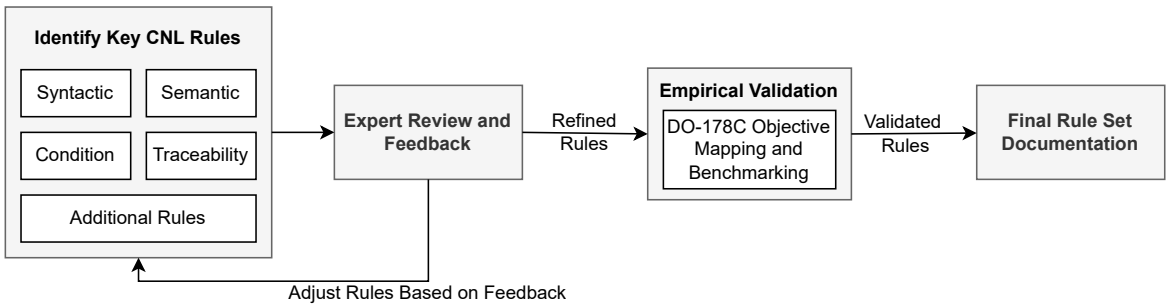


Figure 6.2 Rule selection and validation process for developing CNL compliance with DO-178C.

1. **Initial Rule Selection**
   The initial draft of the CNL rules was developed by identifying key qualities outlined in DO-178C, including clarity, consistency, and testability. Rules were created to ensure syntactic and semantic uniformity, enforce conditions and constraints, and integrate mechanisms for traceability. This preliminary rule set was structured to include:

   - Syntactic rules for standardizing sentence structures, such as using the Actor-Action-Object format and quantitative expressions.
   - Semantic rules to ensure consistent vocabulary and avoid implementation details in high-level requirements.
   - Condition and Constraint rules to specify operational scenarios and timing constraints.
   - Traceability rules for unique identifiers and cross-referencing derived requirements.
   - Additional Rules for Aerospace Compliance derived from best practices in requirements engineering for aerospace systems.

These foundational rules were mapped to core DO-178C objectives to pre-assess their potential compliance impact and verify they addressed ambiguities, traceability, and testability effectively.

2. **Expert Review and Feedback**
   The preliminary rule set underwent a rigorous review by subject matter experts with specialized knowledge in DO-178C and requirements engineering. The expert review focused on identifying potential gaps in clarity, completeness, and alignment with DO-178C guidelines. Experts provided feedback, resulting in adjustments and additional rule recommendations, such as:

   - Strengthening verifiability by defining explicit thresholds and tolerances for measurable values, directly supporting DO-178C's emphasis on quantitative metrics.
   - Promoting traceability through the inclusion of a robust glossary to standardize terminology and improve term consistency across all requirements.
   - Enhancing clarity by reinforcing active voice usage to clarify responsibilities and reduce ambiguity.

   This stage established a rule set that better aligned with DO-178C objectives while remaining feasible for consistent application in aerospace software requirements.

3. **Empirical Validation and Refinement**
   Following the expert review, the revised rule set was subjected to empirical validation, which involved mapping the rules directly to DO-178C objectives and comparing them against established guidelines for aerospace requirements [91, 92]. This phase included:

   - Testing the practicality of rules by applying them to real-world case studies to assess their effectiveness in reducing ambiguity and supporting objective, verifiable requirements.
   - Checking for completeness by ensuring that each DO-178C objective was addressed and identifying any need for additional rules, such as discouraging the use of subjective qualifiers (e.g., "adequately") to eliminate interpretative ambiguity.
   - Benchmarking against best Practices by consulting external guides on aerospace requirement standards, which led to the integration of additional rules like avoiding vague terms and using positive statements to improve requirement clarity and verification.

The empirical validation phase provided data-driven insights that informed final adjustments, resulting in a rule set thoroughly aligned with DO-178C's regulatory requirements for aerospace software.

4. **Final Rule Set Documentation**
   After iterative adjustments, the validated CNL rules were documented as a finalized set with practical examples and application guidelines. This documentation specifies each rule's intended function, associated DO-178C objectives, and practical examples to facilitate consistent application across aerospace projects. Key documentation components include:

   - Rule descriptions and DO-178C mappings to highlight each rule's purpose, compliance benefits, and direct relation to DO-178C standards.
   - Application guidelines that provide usage examples for common requirement scenarios to clarify rule intent.

   This final documentation ensured the CNL framework was readily adoptable, supporting both compliance and readability in aerospace software requirements.

6.4 Rule-Based Verification System for CNL Requirements

A rule-based verification system is an automated tool designed to verify that NL requirements conform to the rules defined within the CNL framework. This system assesses each requirement against predefined syntactic, semantic, and structural rules. By identifying deviations from CNL rules, the system provides a streamlined approach for detecting ambiguities and inconsistencies in NL requirements, which aids in maintaining the quality of requirements documentation.

6.4.1 System Overview

The rule-based verification system does not enforce compliance directly with DO-178C but rather ensures adherence to CNL rules that support DO-178C objectives. This system flags NL requirements that do not align with the established CNL framework, allowing users to correct non-compliant requirements and thus indirectly support DO-178C compliance. The system processes requirements, applies rule checks, and outputs a structured report identifying any non-compliant elements along with the specific CNL rules they violate.

The system architecture consists of three main components:

1. Input Parser: This component ingests NL requirements in a structured format (CSV) and prepares them for analysis. The parser standardizes text by removing extraneous characters and normalizing format, allowing for consistent rule application.

2. Rule Engine: The core of the system, where each CNL rule is applied to the parsed requirements. This engine checks for compliance with specific syntactic, structural, and semantic rules that are implemented within the system, identifying deviations that may introduce ambiguity or inconsistency.

3. Output Generator: This component compiles verification results, listing any non-compliant requirements with corresponding rule violations, and exports the findings in a structured format (CSV). This output is designed to facilitate the review and refinement process for requirements authors.

The rule-based verification system was designed to achieve the following goals:

- **Consistency**: By applying CNL rules uniformly, the system helps maintain a standardized format across requirements, reducing variability that could arise from manual verification.
- **Efficiency**: Automated verification significantly reduces the time required to check requirements for compliance with CNL rules, enhancing productivity and supporting large-scale requirements analysis.
- **Scalability**: The system can handle extensive datasets, making it adaptable to projects with numerous requirements while ensuring consistent rule application.

6.4.2 Implemented Rules in the Rule-Based System

The following CNL rules are directly implemented within the rule-based system: SYN1: Actor-Action-Object Structure, ADV1: Active Voice, ADV2: Avoid Vague Terms, ADV3: Single Sentence per Requirement, and ADV5: Avoid Pronouns.

To compensate for certain CNL rules that are challenging to implement in an automated system, additional rules are included to indirectly support quality control. These supplementary rules enhance the system's ability to maintain requirements clarity and consistency:

- Use correct grammar, spelling, and punctuation: This rule supports **SEM1 (Defined Terms and Consistent Vocabulary)** and **ADV4 (Positive Statements)**. Correct grammar ensures consistent terminology and helps in identifying positive statements, which indirectly supports the expression of unambiguous requirements.

- Include tolerances when specifying qualitative or performance values: This rule addresses **SYN2 (Quantitative Expressions)** and **CON2 (Timing and Response Constraints)**. By ensuring that requirements include specific tolerances, this rule improves the precision and verifiability of performance-related requirements.
- Use standard units and symbols for measurements and physical properties: This rule reinforces **SYN2 (Quantitative Expressions)** and **SEM1 (Defined Terms and Consistent Vocabulary)**. Standardizing units and symbols ensures clarity and prevents misinterpretation, supporting both quantitative expressions and consistent terminology.

Certain CNL rules were not included in the rule-based system due to limitations in NLP capabilities and contextual interpretation requirements. The rule SYN2 (Quantitative Expressions) requires interpreting a wide range of numerical data and units, ensuring they are specific and measurable. Implementing comprehensive parsing for all quantitative expressions is complex and may lead to inconsistencies without advanced context-aware NLP capabilities. The rule SEM1 (Defined Terms and Consistent Vocabulary), for its effective implementation, it requires access to a project-specific glossary and the ability to cross-reference domain-specific terms. The rule CON2 (Timing and Response Constraints) involves understanding the context of timing constraints and their relevance to specific scenarios, which requires a level of situational awareness and domain knowledge that is difficult to automate reliably.

6.4.3 Technical Framework and Tools

The rule-based verification system is implemented using the following technologies:

- Pandas [93]: Used for efficient data handling, especially for loading, organizing, and outputting requirements from CSV files.
- Spacy [94]: A natural language processing library used to parse sentence structures, facilitating checks for active voice and other syntactic requirements.
- Regular Expressions [95]: Applied to detect and replace specific patterns in requirements text, such as ambiguous terms and non-standard units.
- Pint [96]: This library standardizes measurement units, which is essential for ensuring that quantitative values are clearly defined and consistent across requirements.

- LanguageTool-python [97]: An API used to identify and correct grammatical errors, supporting the enforcement of clear grammar and avoiding vague expressions.

The pseudocode 1 summarizes the core functionality of the rule-based system.

6.5 Evaluation and Use Cases

This section presents an empirical evaluation of the CNL and rule-based verification system, utilizing selected functional requirements from the *NASA Access 5 Collision Avoidance* Project [85] as a test case. The primary aim is to demonstrate how the CNL framework and rule-based system can reduce ambiguity, enforce consistency, and facilitate compliance for DO-178C certification. This application demonstrates both the capabilities and limitations of the rule-based system, as well as necessary manual interventions to ensure compliance with CNL rules.

6.5.1 Application Scenario: CAS in Unmanned Aircraft

The evaluation centers on functional requirements related to the CAS for Unmanned Aircraft Systems (UAS), specifically designed to operate safely within the NAS while avoiding other airborne traffic. The CAS requirements specify a series of tasks the system must perform—from detecting and tracking traffic to commanding and executing avoidance maneuvers. Each requirement is assessed to determine if it can be directly processed by the rule-based system or if manual transformation is needed to meet CNL standards.

The CAS functions selected for evaluation include:

- Detecting Traffic: Ensuring detection within the system's surveillance volume.
- Tracking Detected Traffic: Developing reliable tracks on detected aircraft.
- Evaluating Collision Potential: Determining if any tracked elements pose a collision threat.
- Prioritizing Collision Threats: Ranking threats by severity.
- Determining an Avoidance Maneuver: Recommending a safe course of action.
- Commanding the Maneuver: Issuing a command for the recommended maneuver.
- Executing the Commanded Maneuver: Performing the maneuver to avoid a collision.

Algorithm 1 Rule-Based Verification System for CNL Compliance Checking

Require: CSV file containing NL requirements
Ensure: Compliance report identifying non-compliant requirements

Import necessary libraries: pandas, spacy, re, pint, language_tool_python
Load models and initialize tools:
    Load Spacy English model for NLP parsing
    Initialize Pint UnitRegistry for unit handling
    Initialize LanguageTool API for grammar and spelling checks
Define a dictionary (*unit_mapping*) for standardized units of measurement
Implement functions to check compliance with each CNL rule:
    check_actor_action_object(text): Verifies compliance with SYN1 by ensuring an Actor-Action-Object structure.
    check_active_voice(text): Checks ADV1 compliance by verifying that the sentence uses active voice.
    check_vague_terms(text): Checks ADV2 compliance by flagging any ambiguous terms (e.g., "some," "any").
    check_single_sentence(text): Ensures ADV3 compliance by verifying the requirement is a single sentence.
    check_pronouns(text): Checks ADV5 compliance by flagging any pronouns in the requirement text.
    check_grammar(text): Verifies grammar, spelling, and punctuation to support SEM1 and ADV4 indirectly.
    check_tolerances(text): Checks if tolerances are included in quantitative expressions to support SYN2 (indirectly).
    check_units(text): Verifies standardized measurement units to support SYN2 and SEM1 (indirectly).
Load dataset:
    Load NL requirements from CSV file into a pandas DataFrame
for each requirement in DataFrame **do**
    Check compliance with **check_actor_action_object(requirement.text)**
    Check compliance with **check_active_voice(requirement.text)**
    Check compliance with **check_vague_terms(requirement.text)**
    Check compliance with **check_single_sentence(requirement.text)**
    Check compliance with **check_pronouns(requirement.text)**
    Check compliance with **check_grammar(requirement.text)**
    Check compliance with **check_tolerances(requirement.text)**
    Check compliance with **check_units(requirement.text)**
    Log any rule violations for each requirement
end for
Generate and save compliance report:
    Compile all non-compliance messages for each requirement
    Save compliance report to a new CSV file
return Compliance report

6.5.2 Output of the Rule-Based Verification System

In this subsection, we outline the results of the rule-based system's analysis of selected requirements. The rule-based system automatically flagged requirements that did not comply with the defined CNL rules. Figure 6.3 summarizes the specific rule violations identified for each requirement, demonstrating the system's ability to detect issues related to pronouns, vague terms, and other ambiguities.

ID	Requirement	Violations
CA F1	The Collision Avoidance System shall detect traffic within its surveillance volume.	ADV5: Contains pronoun: "its"
CA F2	The Collision Avoidance System shall track the detected traffic.	None
CA F3	The Collision Avoidance System shall evaluate the potential for collision with each traffic element being tracked.	None
CA F4	The Collision Avoidance System shall prioritize the traffic posing a collision threat.	None
CA F5	The Collision Avoidance System shall determine an avoidance maneuver that prevents a collision.	None
CA F6	The Collision Avoidance System shall command an appropriate avoidance maneuver.	ADV2: Contains vague term: "appropriate"
CA F7	The Collision Avoidance System shall perform the commanded maneuver.	None
CA F8	The CAS shall establish a track on detected cooperative traffic in the surveillance volume within TBD seconds of initial detection.	SYN2: Standard units without preceding numerical values: ['seconds']
CA F9	The Collision Avoidance System shall continuously monitor all dected trafic.	ADV2: Contains vague term: "continuously"; Grammar/Spelling Errors: ['Possible spelling mistake found.']
CA F10	The Collision Avoidance System shall evaluate the potential for collision with each traffic element being tracked. It shall assess existing collision threats as part of the collision evaluation.	ADV3: Requirement is not a single sentence.; ADV5: Contains pronoun: "It"
CA F11	False tracks shall account for no more than 1.2% of all tracks established by the CAS.	Missing tolerances for values: ['1.2']

Figure 6.3 Rule-Based system verification results for selected requirements.

The results from the rule-based verification system, shown in Figure 6.3, highlight several common types of rule violations in the selected requirements. First, the system flagged the use of pronouns, such as "its" in CA F1 and "It" in CA F10, which fall under rule ADV5. Pronouns can introduce ambiguity by making it unclear which specific element is being referenced, potentially leading to misinterpretation. Additionally, vague terms like "appropriate" in CA F6 and "continuously" in CA F9 were flagged under rule ADV2. These terms lack precision and are open to interpretation, making it difficult to verify and test the requirements objectively. The system advises replacing such terms with specific, measurable language to ensure clarity.

Another violation type, identified under rule SYN2, involves units specified without numerical

values, as seen in CA F8 with “within TBD seconds.” Such placeholders lack the quantifiable detail needed to evaluate the requirement, as DO-178C compliance standards emphasize precise, testable metrics. Lastly, for multi-sentence structures like CA F10, the system flagged rule ADV3, which promotes single-sentence requirements to enhance clarity and traceability. Requirements containing multiple statements are recommended to be split to maintain consistency and simplify interpretation. Together, these flagged violations illustrate how the rule-based verification system identifies areas for improvement in requirement precision and verifiability.

6.5.3 Potential Rule Violations Not Covered by the Rule-Based System

While the rule-based system effectively detects many syntactic and lexical violations, certain CNL rules involve nuances that require manual review due to limitations in natural language processing. Specific types of violations that were not detected by the rule-based system include:

- Semantic Consistency (SEM1): Ensuring consistent vocabulary and terminology across all requirements, particularly with project-specific terms, requires access to a defined glossary or domain-specific knowledge that the system lacks.
- Contextual Constraints (CON2): Requirements involving timing constraints, such as "within TBD seconds," need context-aware interpretation, which is challenging for the rule-based system to evaluate without predefined values.
- Implicit Requirements: Requirements that imply actions or responsibilities without explicit statements, such as CA F8 and CA F10, necessitate human interpretation to confirm if all conditions are covered.

These examples illustrate that while the rule-based system can automate the detection of certain rule violations, manual intervention is necessary to address context-specific or semantically complex requirements. This manual oversight helps ensure comprehensive compliance with DO-178C.

6.5.4 Requirement Transformation

In this subsection, we demonstrate how selected requirements were transformed to meet CNL and DO-178C standards. The transformations are categorized based on whether they were detected by the rule-based system or manually evaluated to correct additional ambiguities or inconsistencies.

1. CA F1 (Initial): The Collision Avoidance System shall detect traffic within its surveillance volume.
    - **Transformation**: The Collision Avoidance System shall detect traffic within the defined surveillance volume.
    - **Explanation**: The pronoun "its" was replaced with "the defined" to improve clarity by explicitly referencing the surveillance volume, thus reducing ambiguity.
2. CA F3 (Initial): The Collision Avoidance System shall evaluate the potential for collision with each traffic element being tracked.
    - **Transformation**: No changes are needed, as the requirement follows the Actor-Action-Object structure and uses specific terminology, meeting CNL guidelines directly.
3. CA F5 (Initial): The Collision Avoidance System shall determine an avoidance maneuver that prevents a collision.
    - **Transformation**: The Collision Avoidance System shall determine an avoidance maneuver to maintain a minimum separation distance of 500 feet from any detected traffic.
    - **Explanation**: The term *"prevents a collision"* was specified with a measurable criterion—"maintain a minimum separation distance of 500 feet"—to ensure the requirement is precise and verifiable, aligning with DO-178C's measurability standards.
4. CA F10 (Initial): The Collision Avoidance System shall evaluate the potential for collision with each traffic element being tracked. It shall assess existing collision threats as part of the collision evaluation.
    - **Transformation**: The Collision Avoidance System shall evaluate the potential for collision with each traffic element being tracked and assess existing collision threats.
    - **Explanation**: The requirement was reformulated into a single sentence to improve clarity and eliminate the pronoun "It," ensuring the requirement meets CNL rules ADV3 and ADV5.

The transformation process illustrates that while the rule-based system effectively automates compliance for straightforward requirements, complex or context-dependent requirements necessitate manual modifications to meet CNL and DO-178C standards. This hybrid approach,

leveraging both automated and manual verification—supports a robust, efficient compliance strategy by combining the precision of rule-based checks with the flexibility of human review for nuanced requirements.

6.6 Discussion and Interpretation

6.6.1 Impact of CNL and Rule-Based Verification on DO-178C Compliance

The empirical evaluation demonstrated that the CNL framework, combined with the rule-based verification system, effectively mitigates several ambiguities and inconsistencies inherent in NL requirements. By enforcing syntactic and lexical rules, the CNL reduces the risk of misinterpretation, contributing to more precise and verifiable specifications. This aligns with the chapter's hypothesis that integrating a CNL and automated verification can facilitate DO-178C compliance by streamlining requirement precision and verification.

The rule-based verification system was shown to detect and flag common issues, such as vague terms, pronouns, and missing numerical specifications, which are obstacles to clear and testable requirements. By identifying these issues automatically, the system reduces manual review time and ensures that requirements adhere more closely to structured CNL guidelines. This automated approach not only enhances consistency but also allows project teams to address potential non-compliance issues earlier in the development process, supporting efficient DO-178C certification efforts.

However, while the rule-based system is proficient at detecting syntactic and lexical non-compliances, certain types of semantic and contextual requirements violations remain outside its automated detection capabilities. This limitation highlights the inherent complexity of NLP in high-stakes, context-sensitive fields like aviation software. For example, rules related to semantic consistency, contextual constraints, and implicit requirements require human interpretation and domain knowledge. Therefore, a hybrid approach, automated rule-based verification complemented by manual review, proves to be the most effective strategy for ensuring comprehensive compliance with DO-178C.

6.6.2 Strengths of the CNL Framework and Rule-Based Verification System

The CNL framework and rule-based system offer several advantages for enhancing requirement quality in aviation software projects:

- Clarity and Consistency: The application of syntactic rules, such as the Actor-Action-Object structure, and lexical rules, such as avoiding pronouns and vague terms, im-

proves the clarity of requirements. This is critical for ensuring that all stakeholders, including developers, testers, and certification authorities, can interpret requirements consistently.

- Efficiency: Automating the detection of rule violations reduces the manual burden on reviewers, allowing for quicker identification of non-compliant requirements. This efficiency gain is particularly valuable in large-scale aerospace projects where hundreds or thousands of requirements need to be verified.
- Traceability and Documentation: By applying CNL rules systematically, the approach supports structured and traceable documentation, which is essential for DO-178C compliance. Unique identifiers, precise terminology, and clearly defined conditions contribute to traceability across the development lifecycle, from high-level requirements to testing and verification.
- Reduced Risk of Certification Delays: By addressing ambiguities and inconsistencies early, the CNL framework helps mitigate risks of costly and time-consuming certification delays. This proactive approach aligns with DO-178C's objective of ensuring accurate, testable, and traceable requirements from the outset.

These strengths underscore the value of the CNL framework and rule-based system as foundational tools for improving the quality and compliance of requirements in aviation software.

6.6.3 Limitations and Areas for Improvement

While the evaluation highlighted the strengths of the CNL and rule-based verification system, it also revealed limitations in their ability to comprehensively address all types of DO-178C compliance needs. Key limitations include:

- Inability to detect contextual constraints: Requirements that specify timing or response conditions, such as "within TBD seconds", require contextual understanding to evaluate compliance effectively. Current NLP capabilities are limited in interpreting such requirements without predefined values or a broader understanding of operational scenarios. Future work could explore integrating domain-specific ontologies or machine learning models that can infer likely contexts for requirements.
- Semantic consistency and glossary dependence: The system cannot ensure semantic consistency, especially when project-specific terms require alignment with an established glossary. Integrating a project glossary or terminology database could improve the system's capacity to enforce consistent vocabulary across requirements.

- Implicit requirements and ambiguous scenarios: Some requirements imply actions or responsibilities without explicitly stating them, such as CA F8's reference to tracking without specifying a response time. The rule-based system cannot detect such implicit gaps. A solution could involve adding rules that flag implied actions or unspecified conditions, prompting manual review for requirements that could introduce ambiguity.

Addressing these limitations could involve future enhancements to the rule-based verification system, including advanced NLP models capable of handling context-sensitive rules or integrating machine learning algorithms trained on domain-specific requirements data.

6.6.4 Synthesis

The results from the empirical evaluation support the primary hypothesis of this chapter: that a CNL, when integrated with an automated rule-based verification system, can significantly enhance the clarity, consistency, and verifiability of requirements, thereby aiding DO-178C compliance. The analysis reveals that the system is well-suited for identifying issues that impact requirement clarity and testability, fulfilling the objectives of improving accuracy, reducing ambiguity, and increasing efficiency in requirement verification.

Moreover, the evaluation demonstrates that the system can reduce verification time through automation while maintaining a high level of compliance. Although manual intervention remains necessary for certain complex requirements, the rule-based system provides a foundational level of quality control that streamlines the review process and reduces the overall verification burden.

The evidence suggests that the CNL framework and rule-based system not only facilitate compliance but also contribute to a more systematic, reliable, and scalable approach to requirement management in aviation software. This supports the long-term goal of creating safer and more dependable airborne systems by improving the quality and precision of foundational specifications.

Future research could focus on expanding the rule-based system to cover semantic and context-based rules, possibly through machine learning algorithms that are trained to recognize domain-specific nuances. Additionally, integrating real-time glossary updates and leveraging cloud-based collaborative verification platforms could enhance the system's adaptability and relevance in dynamic, large-scale aerospace projects.

In conclusion, the CNL and rule-based verification system offer a promising framework for advancing DO-178C compliance in aviation software by improving requirement consistency and verifiability. While limitations remain, particularly in the detection of context-specific

and implicit requirements, the hybrid approach of combining automated and manual review ensures a robust and flexible compliance solution.

6.7 Chapter Summary

In this chapter, we presented the development of a CNL tailored to meet DO-178C compliance requirements and a complementary rule-based verification system. The CNL introduces structured linguistic rules to reduce ambiguities and improve the clarity and consistency of NL requirements, while the rule-based system automates the detection of non-compliance with these rules. This framework was evaluated using case studies, highlighting its ability to streamline the verification process and reduce the risk of misinterpretation in safety-critical software requirements. The next chapter will delve into the creation of the *Natural2CTL* dataset and its role in enabling the translation of NL requirements into formal specifications.

Table 6.2 Examples of CNL Transformation for DO-178C Compliance.

Original Requirement	**Transformed Requirement (CNL)**	**Explanation**
The system should operate safely under high-altitude conditions if possible.	The system shall maintain operational safety at altitudes of up to 30,000 feet under all specified environmental conditions.	The term "if possible" introduces ambiguity, implying optionality incompatible with safety-critical requirements. In the CNL transformation, specific altitude and environmental conditions are defined, ensuring that the requirement is both clear and verifiable, which is essential for DO-178C's emphasis on precision and objectivity.
The software might need to restart during power fluctuations.	The software shall automatically restart within 1 second in the event of a power fluctuation exceeding ±5% of nominal voltage.	Words like "might" create uncertainty and lack enforceability in the requirement. By specifying a measurable condition (power fluctuation exceeding ±5%) and a response time (1 second), the CNL version ensures compliance with DO-178C's objectives for verifiability and predictability, facilitating the requirement's testing and validation.
The system should be able to detect and report low battery levels promptly.	The system shall issue a low-battery alert within 10 seconds when the battery charge falls below 20% capacity.	Terms like "promptly" are inherently subjective and could be interpreted in various ways. The CNL transformation specifies a response time and a precise battery threshold, enabling objective testing and validation. This approach ensures alignment with DO-178C's requirement for requirements to be measurable and testable.
The system must not exceed safe operating temperatures under normal conditions.	The system shall maintain internal temperature below 80°C under all specified operating conditions.	The term "safe operating temperatures" is vague and open to interpretation. The transformed CNL requirement defines a specific temperature threshold (80°C), making the requirement clear, enforceable, and verifiable. This specificity fulfills DO-178C's demand for unambiguous, measurable requirements.
If the pilot requests, the system could initiate a diagnostic check.	The system shall initiate a diagnostic check within 5 seconds upon receiving a request from the pilot.	The original statement's use of "could" suggests an optional behaviour, which lacks clarity and enforceability. The CNL statement removes this ambiguity by specifying a definite action (initiation of a diagnostic check) and a clear response time, thereby ensuring compliance with DO-178C's emphasis on deterministic and testable requirements.

CHAPTER 7 NATURAL2CTL: A DATASET FOR NATURAL LANGUAGE REQUIREMENTS AND THEIR CTL FORMAL EQUIVALENTS

7.1 Chapter Overview

This chapter [1] presents the Natural2CTL dataset, a curated collection of NL requirements and their formal CTL equivalents, specifically developed to bridge the gap between NL requirements and formal specifications. This dataset addresses a critical issue in software verification: translating inherently ambiguous NL requirements into precise CTL formulations, which is essential for rigorous verification across a variety of safety-critical domains. The chapter details the systematic data collection, selection, and annotation processes for the dataset, along with validation methods aimed at ensuring accuracy and robustness. Through this initiative, Natural2CTL provides a foundational resource for enhancing automated translation and formal verification, supporting advancements in both RE and formal methods.

7.2 Study Design and Objective Setting

The study's design for developing the Natural2CTL dataset involved a structured approach to create a comprehensive and high-quality dataset that aligns with industry and research needs in RE and formal verification. Figure 7.1 provides a schematic overview of the study design, detailing the process from data collection and annotation to evaluation. Each step was carefully executed to maintain high standards of accuracy and relevance.

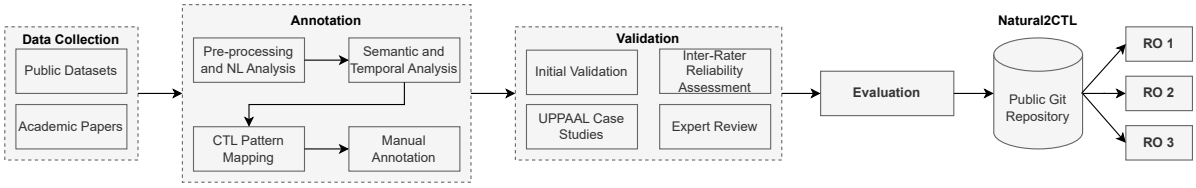


Figure 7.1 Schematic diagram of the study.

[1] Part of the content of this chapter is published in: Zrelli, R., Misson, H.A., Ben Attia, M., Magalhaes, F.G.d., Nicolescu, G.: "Natural2CTL: A Dataset for Natural Language Requirements and Their CTL Formal Equivalents". In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2024).

The primary objectives guiding this study are as follows:

- **To construct a comprehensive and reliable NL-to-CTL dataset (RO1)**: The study aims to create a well-structured dataset that pairs NL requirements with corresponding CTL specifications. This objective emphasizes selecting functional requirements with clear temporal dynamics that align well with CTL patterns, reducing ambiguity in the representation and increasing the dataset's relevance for formal verification applications.
- **To ensure accuracy and consistency through rigorous annotation and validation processes (RO2)**: Accurate translation of NL requirements to CTL specifications is critical. This objective focuses on developing a precise annotation process, including semantic analysis and temporal alignment. Each requirement undergoes multiple stages of validation to ensure high fidelity to the original requirements and suitability for formal verification workflows.
- **To validate the dataset's effectiveness through multiple evaluation stages (RO3)**: The dataset undergoes validation to verify its quality and applicability. This includes initial validations, inter-rater reliability assessments, UPPAAL case studies, and expert reviews. These evaluations help establish the dataset as a trusted resource for both academic and industrial applications, ensuring its relevance for automated RE and formal methods.

In line with these objectives, the dataset construction follows a series of methodological steps: data collection, annotation, iterative validation, and evaluation, with each step designed to enhance accuracy and practical relevance. The selection and annotation process emphasizes requirements that align well with CTL's temporal and logical structure, ensuring that each entry in the dataset supports effective use in formal verification tasks.

7.3 Dataset Development & Overview

7.3.1 Data Collection

To construct a comprehensive dataset, we embarked on an extensive process of data collection and annotation, aiming to provide a repository that connects software requirements with their corresponding CTLs. The approach outlined below is fusing academic rigour and practicality that is useful to the industry.

Our primary sources for software requirements included public datasets ([98], and [99]), academic papers ([100], [101], [76], [73], [14],and [102]), and Kaggle [103], a popular platform for datasets. These sources provided a rich variety of functional and non-functional requirements essential for our study.

Our initial effort involved a literature search to identify papers containing software requirements paired with their CTL counterparts ([76], [73], [14], and [102]). This was an essential step to ensure that a portion of our CTL translations aligned with existing research standards and ensuring that they were externally validated, complementing our manual annotations.

We adopted a targeted approach in selecting requirements. Emphasis was placed on labels such as functional, availability, fault tolerance, performance, and security. These were chosen for their clarity and directness in depicting software behaviour, making them suitable for effective CTL representation.

Our decision to concentrate on these specific requirements stemmed from their relevance and capability to yield precise and clear CTL specifications. Labels such as maintainability, portability or scalability give fundamental insights into software design. However, their qualitative and subjective nature may not be a perfect fit for the CTL structure. We gave precedence to requirements demonstrating distinct temporal behaviour. This focus stemmed from CTL's ability to capture system dynamics over time. By selecting requirements that emphasized time-anchored system behaviours, we ensured they were amenable to CTL formulations, thus maintaining fidelity within the CTL framework.

In selecting requirements for translation into CTL, it was crucial to distinguish between those that were suitable and those that were not. For instance, consider the requirement from the Kaggle dataset [103]: 'The average number of recycled parts records per day shall be 50,000.' This requirement, representative of the scalability label, illustrates a challenge for CTL translation. CTL focused on specifying system behaviour over time, is less suited for capturing quantitative performance metrics like this one. Such requirements, while important, do not align with the temporal and behavioural focus of CTL and were therefore excluded from our dataset. This decision reflects our prioritization of requirements that could be effectively represented in CTL, focusing on clear temporal and behavioural aspects. This example highlights the rationale behind our selective approach and underscores the importance of aligning the dataset's content with the strengths and limitations of CTL as a formal specification language.

The selection process entailed an initial review of available requirements, followed by an analysis of their suitability for CTL translation. This approach guaranteed the inclusion of only the most relevant and clear requirements, aligning with the foundational principles

of temporal logic in system specification. This systematic and focused approach to data collection was fundamental in assembling a dataset that not only meets academic standards but is also practically relevant to the industry. The Natural2CTL dataset thus stands as a comprehensive repository, bridging the gap between NL and formal logic specifications.

7.3.2 Annotation Process

The annotation of the Natural2CTL dataset was a meticulous process that aimed to transform NL requirements into precise CTL specifications. This journey began with a methodical approach to manual annotation, carefully considering the variability and ambiguity inherent in NL.

Understanding the formal semantics of CTL was pivotal [8]. CTL's systematic representation incorporates temporal operators to capture system behaviours over time. The Keypath quantifiers—"A" (for all paths) and "E" (there exists a path)—alongside temporal operators like "G" (globally), "F" (Eventually, or in the future), "X" (next), and "U" (until), provide a flexible language for specifying system properties. For instance, "$AG(p)$" denotes 'p' holding globally on all paths, while "$EF(q)$" indicates the existence of a path where 'q' eventually becomes true.

The "Property Pattern Mappings for CTL" [104, 105] provided a structured approach for translation. This method helped bridge NL requirements to their formalized representations, covering patterns such as Absence, Existence, Bounded Existence, Universality, Precedence, and Response. These patterns were instrumental in reducing the variability in interpreting NL requirements and ensuring consistent CTL translations. To provide clarity:

- *Absence*: Identifies instances where a particular property or event never occurs.
- *Existence*: Emphasizes scenarios where a specific property or event does happen.
- *Bounded Existence*: Highlights situations where an event or property occurs within specified limits.
- *Universality*: Describes the consistent or continuous occurrence of an event or a property over a given timeframe.
- *Precedence*: Focuses on sequences where one event always precedes another.
- *Response*: Underlines situations where a certain event always responds to another.
- *Precedence 1 cause-2 effect*: Describes sequences where one event precedes two other outcomes.

- *Precedence 2 cause-1 effect*: Highlights situations where two initiating events lead to a single resulting event.
- *Response Chain 1 stimulus-2 response*: Specifies cases where one event triggers a chain reaction leading to two other events.
- *Response Chain 2 stimulus-1 response*: Emphasizes scenarios where two events in sequence trigger a single subsequent event.
- *Constrained Chain*: Identifies circumstances where one event or sequence of events responds to another under specific constraints.

For a more in-depth exploration of the patterns and their respective sub-variants, more details can be found at `http://patterns.projects.cis.ksu.edu/`.

Our process (Figure 7.2) began with pre-processing to clarify ambiguities in the NL requirements, followed by temporal analysis to identify time-anchored behaviours. State variables and actions were then defined, leading to a deep semantic interpretation to contextualize the requirement. We identified CTL patterns that matched the interpreted semantics, adapting or combining patterns as necessary to accurately reflect the requirement's intent. To embody the inherent variability of CTL and ensure the robustness of our translations, we implemented a validation step to assess the accuracy of the CTL properties. Any deviations led to a re-evaluation of the pattern identification or adaptation process. This approach ensured that the final CTL specification was a true representation of the original NL requirement, even for atypical cases that did not conform to standard patterns.

The annotation process was iterative, with a focus on addressing requirements that presented ambiguous temporal dynamics. To ensure the precision of CTL translations, we engaged in a thorough analysis of the formal aspects of CTL. Our collective expertise facilitated a comprehensive internal review process, where each translation was scrutinized and refined to maintain accuracy and adherence to the semantic depth of the requirements.

We adopted a modular translation approach, breaking down complex requirements into manageable components for individual translation and subsequent integration. This practice significantly reduced errors and ambiguities, enabling a more granular and precise translation process.

The annotation phase was more than just a simple translation process. It unfolded an effort to balance and synchronize the inherent flexibility of NL and the rigorous determinism of CTL. Our objective was to ensure that the fidelity of each original requirement is preserved.

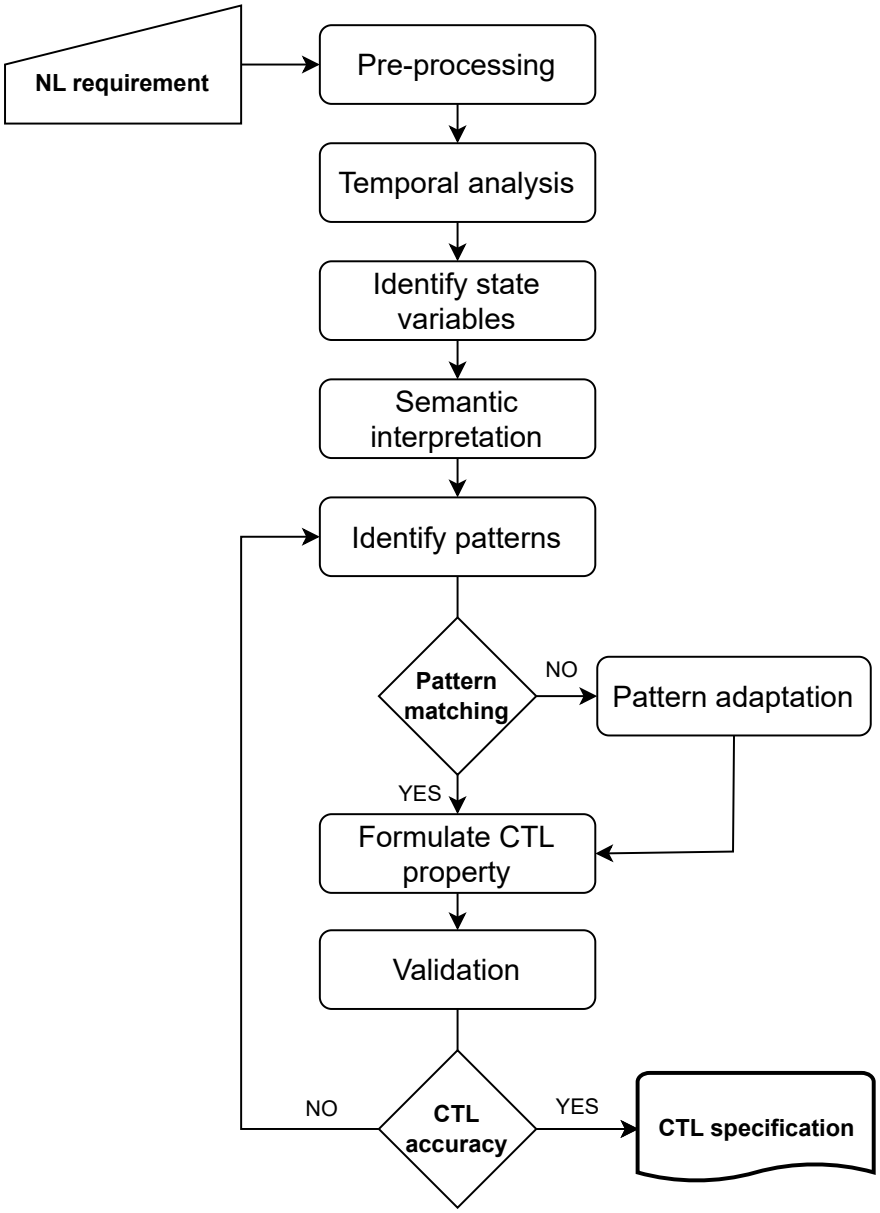


Figure 7.2 Flowchart demonstrating the translation process from NL to CTL.

7.3.3 Dataset Characteristics & Overview

Our dataset comprises a comprehensive compilation of 2,095 entries, sourced from various application domains. It encapsulated different software systems like traffic management systems, which address smart traffic control and vehicular networks, and Cloud Computing. It also reflects the synchronized collaboration inherent in distributed systems in our dataset. Database Systems especially concerning transactions and consistency, as well as E-commerce Systems, further enrich our data with insights into online shopping dynamics. Moreover,

telecommunication nuances like handovers between base stations form an essential part of the collection.

The dataset's richness is evident not just in the variety of systems it covers but also in the range of requirements it addresses. The dataset represents both functional and non-functional aspects. Requirements range from performance-related criteria and safety precautions to security measures and usability features. These requirements explore the intricacies of the system, addressing aspects like startup and shutdown behaviours, communication protocols, mechanisms for error handling, and adaptive responses to unexpected inputs.

A significant portion of our dataset is dominated by liveness properties as evident from the response pattern's widespread usage. This emphasizes the focus on system behaviours that guarantee certain actions or events following previous conditions or triggers. Our dataset's dimensions, however, extend beyond just these liveness-related characteristics. It robustly encompasses safety properties, ensuring that the system avoids undesirable or potentially hazardous states. By covering both these aspects, the dataset provides a well-rounded perspective on system expectations. Additionally, the presence of fairness and stability properties augments the dataset's comprehensive nature, ensuring coverage of potential system behaviours. This ensures not just the absence or presence of certain actions but also their equitable distribution and consistency over time.

For a more granular insight into the dataset's structure and composition, we present below a representative Table 7.1 illustrating the transformation of each requirement from an NL stipulation to its formal CTL specification.

7.4 Dataset Evaluation

7.4.1 Initial Evaluation

To establish the dataset's credibility, we initiated a first validation phase involving two Ph.D. candidates, each bringing distinct expertise in formal methods and CTL. The first, with a background in machine learning and formal verification, complemented the second candidate's practical experience in the industry, particularly in formal analysis and verification of airborne software.

Throughout a focused four-week period, both validators independently selected dataset entries at a ratio of 1 in every 5 for evaluation. This approach resulted in each validator independently reviewing 400 entries. Due to the random nature of the selection process, there were instances where both validators independently chose and reviewed the same entry. Each entry was rigorously assessed and annotated as 'correct,' 'partially correct,' or

Table 7.1 Excerpt from the Dataset Illustrating the Structure of our CSV File

ID	requirement_text	CTL	Pattern
01	The Disputes System shall prevent the creation of duplicate dispute requests 100% of the time.	$AG(\neg\ duplicateDisputeRequest)$	Absence
02	The system shall filter data by: Venues and Key Events.	$AG(filterByVenues) \wedge AG(filterByKeyEvents)$	Universality
03	When given a takeoff command, the _InternalSimulator_ shall move the UAV to the takeoff altitude corresponding to UAV's current longitude and latitude.	$AG(takeoffCommand \rightarrow AF(moveToTakeoffAltitude))$	Response
04	The TCS shall provide an interface between the TCS and an external hard copy printer.	$AF(interfaceWithExternalPrinter)$	Existence

'incorrect,' with validators providing potential corrections where necessary. For guidance in their evaluations, they used the Property Pattern Mappings for CTL [105], which formed the foundation of our annotations.

The feedback from this initial validation phase revealed diverse interpretations of requirements, reflecting variations in CTL translation patterns. Particularly, the use of time operators showed significant variability. Approximately 20% of the entries reviewed were flagged for rectification, indicative of the complex nature of translating NL requirements into CTL.

7.4.2 Inter-Rater Reliability and Review Process

In our commitment to ensure the robustness of our dataset, we focused on assessing inter-rater reliability. This crucial step involved having both validators re-annotate the entries initially evaluated by their counterpart, applying a structured ordinal rating system. The translations were categorized into three levels: 'correct' indicated a CTL translation that accurately and fully represented the original NL requirement; 'partially correct' denoted translations that were generally on the right track but required minor adjustments or refinements; and 'incorrect' pointed to translations that deviated from accurately capturing the requirement's essence. To quantify the consistency in ratings, we calculated Krippendorff's Alpha [106]. A score of 0.7659 was achieved, which falls within the range of substantial agreement. This score is crucial as it implies a significant level of consistency and reliability in the ratings provided by our validators.

After the reliability assessment, we engaged in a detailed review process to address instances where the validators' annotations differed. In such cases, we held joint discussions with both validators to reach a consensus on the most accurate CTL representation for each entry. These discussions were particularly crucial in instances where discrepancies might have indicated potential misinterpretations or ambiguities in the original translations. As a result, we carefully revisited and, where necessary, revised our translations to better align with the validated interpretations, ensuring a higher degree of accuracy and consistency across the dataset.

7.4.3 Validation and Interpretative Variability: UPPAAL Case Study

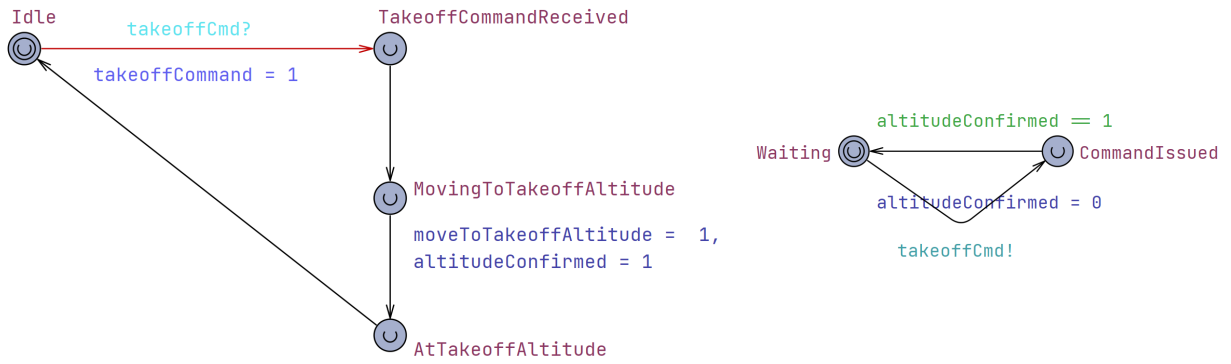
To rigorously validate the CTL translations in our dataset, we conducted a series of case studies using UPPAAL [107], a model-checking tool renowned for verifying real-time systems. UPPAAL was chosen for its advanced capabilities in simulating and verifying real-time systems, making it a more suitable choice for our complex dataset scenarios. This decision was driven by UPPAAL's ability to handle a broader range of temporal logic expressions and its more intuitive model-building environment. These studies were crucial in testing various CTL interpretations and highlighting the inherent variability in formalization.

We selected the requirement: "*When given a takeoff command, the Internal Simulator shall move the UAV to the takeoff altitude corresponding to UAV's current longitude and latitude.*" Two CTL translations were considered:

1. $AG(takeoffCommand \rightarrow AF(moveToTakeoffAltitude))$, represents a response pattern, suggesting a direct causation.
2. $AF(moveToTakeoffAltitude)$, embodies an existence pattern, indicating eventual achievement of the state without a direct causal link.

Both CTL expressions were verified in UPPAAL (Figure 7.3). The analysis aimed to determine which CTL more accurately represented the requirement's intended meaning. The first CTL was more contextually aligned with the requirement's intended meaning. This choice was reinforced by the fact that the second CTL, despite being valid in terms of CTL syntax, did not capture the causative relationship implied in the requirement. This distinction emphasizes the need for careful consideration of the context and semantics in CTL translations.

This example is an illustration of a broader validation process applied to multiple entries in the dataset, ensuring the contextual accuracy of our CTL annotations. The insights gained from this case study, and others like it, have been instrumental in refining our dataset,



(a) UPPAAL model representation for the UAV takeoff requirement

(b) Verification results for the two CTL translations.

Figure 7.3 UPPAAL model and verification results for UAV Takeoff requirement

ensuring each entry's CTL translation is contextually sound and logically consistent. The selection criteria for the requirement included its ability to demonstrate the diverse range of CTL interpretations possible from a single NL requirement.

7.4.4 Expert Industry Evaluation

Following the UPPAAL case studies, we engaged an industry expert with extensive experience in model checking and formal methods. The expert independently translated 100 randomly selected entries from our dataset into CTL. The translations were then analyzed by the annotator and the two initial validators. This analysis aimed to compare the expert's translations with our original annotations, focusing on aspects like exact match, minimal corrections needed, and complete deviations. This step was crucial in identifying areas where our initial translations could be further refined.

The analysis of the expert's independent translations yielded promising outcomes, with 84% of the translations aligning closely with our original CTL annotations, thereby affirming the accuracy and reliability of our initial work. The need for adjustments in approximately 16% of the entries primarily stemmed from challenges in interpreting nuanced requirements and resolving ambiguities inherent in complex sentence structures within the original requirements. In cases where significant disparities existed between the expert's translations and

our own, we undertook an in-depth, case-by-case analysis. This involved meticulously scrutinizing each CTL expression to dissect the differences in interpretations and determine which version most faithfully captured the essence of the NL requirement. These findings were instrumental in guiding our subsequent refinement efforts.

7.4.5 Future Validation Plans

Building on the validation work already undertaken, our plans focus on establishing a more comprehensive and continuous evaluation process for the Natural2CTL dataset. To achieve this, we intend to form a dedicated validation committee, consisting of experienced academics and industry professionals with expertise in formal methods, CTL, and RE. This committee, including a seasoned professor in formal methods, will collaboratively review and refine the dataset.

Beyond the expert committee, we are committed to fostering community engagement around the Natural2CTL dataset. The dataset is publicly accessible at `https://github.com/RimZrelli/PublicDataset/`, and we will actively seek feedback from the broader research community and potential users. This feedback will be invaluable in identifying real-world challenges, usage scenarios, and opportunities for improvement. We envision this as a dynamic process where the dataset is not only a static resource but evolves and adapts in response to the valuable input from its diverse user base.

7.5 Applications and Limitations

The Natural2CTL dataset represents a significant step forward in integrating RE with formal methods, offering a unique resource that maps NL requirements to their CTL counterparts. This dataset holds substantial value as a foundation for training machine learning models, specifically to automate the translation of NL requirements into CTL. Given the growing influence of artificial intelligence in software engineering, the importance of such a dataset cannot be overstated.

Beyond its application in ML-driven translation, Natural2CTL serves as an invaluable pedagogical tool, providing students, researchers, and industry professionals with a structured dataset to explore the complexities involved in translating NL requirements into formal logic. This resource can support professional training initiatives, enrich understanding of RE challenges and the nuances of formalization, and contribute to rigorous academic studies investigating the intricacies of requirement translation.

The dataset's broader implications are equally noteworthy. As a standardized reference,

Natural2CTL offers a benchmark for evaluating new methodologies in formal translation, facilitating comparative studies and fostering innovation. It provides a consistent platform for testing and validating software tools designed for requirements analysis and translation, allowing these tools to be assessed against a realistic dataset reflective of practical scenarios. In this way, Natural2CTL supports the iterative development and improvement of RE tools capable of navigating the complexities inherent in real-world software specifications.

However, as with any pioneering initiative, Natural2CTL faces certain limitations. The manual annotation process for over 2,095 entries, while ensuring high accuracy, inevitably introduces potential ambiguities in translation. Additionally, the dataset's scope, though substantial, remains modest when compared to the larger-scale datasets commonly used in ML. This size constraint may limit its applicability in certain high-volume ML scenarios, underscoring the potential benefit of future dataset expansion efforts.

7.6 Chapter Summary

In this chapter, we presented the Natural2CTL dataset, designed to bridge the gap between ambiguous NL requirements and their CTL formalizations. We detailed the processes employed in data collection, annotation, and validation to ensure accuracy and relevance. These steps culminated in a high-quality dataset that supports automated NL-to-CTL translations, paving the way for advancements in formal verification of safety-critical systems. Validation efforts, including expert reviews, inter-rater reliability assessments, and UPPAAL case studies, reinforced the dataset's reliability and applicability. The next chapter delves into the fine-tuning of LLMs for translating NL requirements into CTL specifications, focusing on enhancing accuracy and scalability in formal verification processes.

CHAPTER 8 ADVANCING FORMAL VERIFICATION: FINE-TUNING LLMs FOR TRANSLATING NATURAL LANGUAGE REQUIREMENTS TO CTL SPECIFICATIONS

8.1 Chapter Overview

This chapter [1] presents an investigation into the effectiveness of fine-tuning LLMs to translate NL requirements into CTL specifications, a task critical for formal verification in safety-critical systems. Translating NL requirements into precise, machine-interpretable formal specifications remains a major challenge in software engineering, where ambiguity or inconsistency in requirements can lead to severe consequences. Traditionally, this translation process relies on formal methods experts, making it time-intensive and prone to variability.

In response to these challenges, this chapter examines the potential of three state-of-the-art LLMs: LLAMA3, Mistral, and Qwen2—for automating the NL-to-CTL translation. The models are fine-tuned using the Natural2CTL dataset, which includes 2,095 NL requirements paired with validated CTL specifications, aiming to improve their translation accuracy and semantic alignment. This study leverages fine-tuning techniques designed to align LLM outputs with expert-level precision, reducing dependency on manual translation efforts.

To assess the performance of the fine-tuned models, we use a multi-metric evaluation framework, incorporating accuracy, validation loss, semantic similarity, and Structural Operator Jaccard Similarity (SOJS). Emphasis is placed on the Mistral model, given its promising results during preliminary testing. The ultimate objective of this chapter is to establish an automated translation process that approximates the accuracy and reliability typically associated with expert manual translations, thereby enhancing the efficiency and scalability of formal verification workflows.

8.2 Study Design and Objective Setting

The design of this study is centred around fine-tuning three LLMs (LLAMA3, Mistral, and Qwen2) to translate NL requirements into CTL specifications accurately. Figure 8.1 provides an overview of the study design, detailing the sequence of data preparation, model fine-tuning,

[1] Part of the content of this chapter is published in: Zrelli, R., Misson, H.A., Ben Attia, M., Magalhaes, F.G.d., Nicolescu, G.: "Advancing Formal Verification: Fine-tuning LLMs for Translating Natural Language Requirements to CTL specifications". In: *Proceedings of the 35th International Workshop on Rapid System Prototyping* (2024).

evaluation, and comparative analysis steps that support our objectives. Each component of the process is crafted to achieve high precision and alignment with expert standards, given the critical nature of accurate formal translations.

The primary objectives guiding this study are as follows:

- **To assess the effectiveness of fine-tuning LLMs for domain-specific applications (RO1)**: Fine-tuning each model on a specialized dataset (Natural2CTL) aims to enhance the syntactic precision and semantic fidelity of CTL translations, making these models suitable for direct deployment in verification workflows.
- **To evaluate model performance across diverse metrics (RO2)**: A multi-metric evaluation framework is applied to assess each model's performance, with accuracy, semantic similarity, SOJS, and validation loss as core metrics. This framework provides a comprehensive understanding of each model's strengths and limitations, enabling a detailed comparison of translation efficacy across the three LLMs. Given its initial strong performance, the Mistral model receives additional fine-tuning and analysis.
- **To compare LLM-generated translations with human performance(RO3)**: By evaluating how automated LLM-based translation compares with human expertise, we benchmark model-generated CTL outputs against translations provided by non-expert human translators. This objective explores the potential and limitations of LLMs in approximating expert-level translation fidelity, emphasizing the areas where human oversight remains necessary.

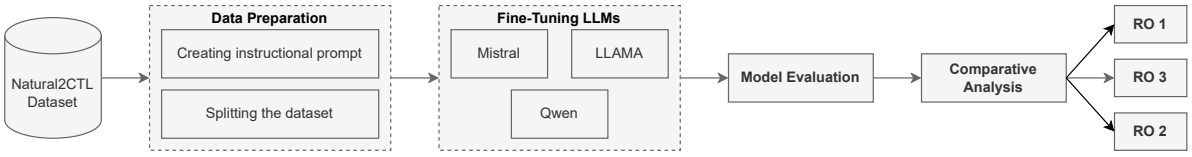


Figure 8.1 Schematic diagram of the study.

In line with these objectives, the study proceeds as follows: The Natural2CTL dataset undergoes a rigorous data preparation process, including instructional prompt creation and dataset stratification to ensure a balanced representation of CTL patterns. Each model is then fine-tuned through multi-fold cross-validation, with detailed parameter adjustments aimed at optimizing memory usage, accuracy, and overall model performance. Subsequently, the models are evaluated and their outputs analyzed, comparing each model's performance across the predefined metrics. Finally, a comparative analysis between model outputs and

human translations is conducted to assess the feasibility of LLMs as reliable translation tools for formal verification tasks.

Figure 8.1 illustrates the methodological steps, highlighting the flow from data preparation and model fine-tuning to evaluation and analysis, with each step aligning with the study's objectives.

8.3 Methodology

In this section, we describe the dataset preparation and the fine-tuning process for the LLMs used in our study.

8.3.1 Dataset

We utilized the Natural2CTL dataset [108], which comprises 2,095 entries of NL requirements paired with their CTL specifications. This dataset, inspired by the property patterns described by [105], ensures that each CTL representation closely matches its NL counterpart. The entries encompass a diverse range of software requirements with temporal behaviours apt for CTL translation. Each entry was validated by domain experts to support translation accuracy.

In this study, the dataset was prepared to ensure robust model training and evaluation. Initially, the dataset was loaded from a CSV file. We then created a diverse set of instructional phrases to guide the translation of NL requirements into CTL specifications. Examples of these instructional phrases include: "Transform the following sentences into CTL expressions," "Translate the following statements into CTL formulas", and "Convert the next sentences into CTL statements". This variety was introduced to ensure comprehensiveness and provide diverse training prompts, allowing the model to learn better and perform more robust inference across different instructions.

The dataset was divided using a stratified shuffle split method, allocating 90% for training and 10% for testing to maintain a balanced representation of various CTL patterns. This stratification ensures that each subset is representative of the overall distribution of CTL patterns, crucial for reliable model training and evaluation. Unique patterns appearing only once were exclusively retained in the training set to prevent potential class imbalances in the testing phase.

The test subset was equally divided into validation and test sets, enabling rigorous evaluation through model tuning and final performance assessment on unseen data. Finally, the training,

validation, and test datasets were saved into separate CSV files. Leveraging the Hugging Face datasets library, these files were then uploaded to the Hugging Face Hub.

8.3.2 Fine-Tuning LLMs

In our approach, we fine-tuned three state-of-the-art LLMs: LLAMA3, Mistral, and Qwen2. The objective was to optimize the translation of NL requirements into CTL. The fine-tuning process utilized multi-fold cross-validation to ensure robust evaluation and enhance model reliability. The selection of these models was based on their proven capabilities in handling complex NL tasks and their adaptability to domain-specific fine-tuning. The specific versions of the models used for fine-tuning were: mistral-7b-instruct-v0.3-bnb-4bit, Qwen2-7B-Instruct-bnb-4bit and llama-3-8b-Instruct-bnb-4bit.

To ensure robust evaluation and enhance model reliability, we employed multi-fold cross-validation. Initially, a 12-fold cross-validation was performed for all models. Based on performance metrics, the Mistral model, which showed superior results, was further optimized using 50-fold cross-validation.

The training procedure involved several steps:

1. Model Initialization: The pre-trained models were loaded with specified configurations, including a maximum sequence length of 2048 and using 4-bit quantization to reduce memory usage. These configurations ensured efficient handling of large input sequences and optimal memory usage.

2. Adapter Integration: LoRA (Low-Rank Adaptation) adapters were integrated to update a subset of model parameters, facilitating efficient fine-tuning.

3. Prompt Formatting: A prompt was created to guide the NL-CTL translation. The prompt included clear instructions as detailed in the dataset preparation; task descriptions which explicitly stated the requirement to convert NL sentences into corresponding CTL formulas, emphasizing the need to consider conditions, stability, and permitted values; and examples. Each entry in the dataset was formatted with specific instructional phrases, requirements text, and expected CTL output. The prompt also incorporated an end-of-sequence token to prevent continuous generation.

4. Training Execution: The models were trained using the SFTTrainer framework, which managed batch processing, gradient accumulation, and parameter updates. Key hyperparameters included learning rate, batch size, and epoch count.

Parameter optimization was conducted by adjusting hyperparameters such as learning rates and weight decay. The optimized settings included: Learning Rate: 2e-5, Weight Decay: 0.01, Batch Size: 2 (with gradient accumulation steps of 4), and Epochs: limited to 2 to prevent overfitting.

For a concise overview, the pseudocode 2 summarizes the fine-tuning process.

Algorithm 2 Fine-Tuning LLMs

Require: Pre-trained LLMs (LLAMA3, Mistral 7B, Qwen2), training dataset, validation dataset, tokenizer
Ensure: Fine-tuned LLM
1: Initialize model and tokenizer with pre-trained weights
2: Integrate LoRA adapters for parameter-efficient fine-tuning
3: Split dataset into k-folds for cross-validation
4: **for** each fold k **do**
5: Split training dataset into train and validation sets
6: Format prompts for NL-to-CTL translation
7: Train model using SFTTrainer with specified hyperparameters
8: Evaluate model on the validation set
9: Record validation loss
10: **end for**
11: Calculate the average validation loss across all folds
12: Display the average validation loss
13: Save fine-tuned model and tokenizer
14: **return** Fine-tuned LLM

8.4 Experimental Design

To assess the quality of the CTL specifications generated by our models, we employed a set of comprehensive evaluation metrics that consider both syntactic accuracy and semantic correctness. These metrics were selected based on their relevance to formal verification tasks, where precision in logic expression is paramount. Below is an outline of the metrics used, along with the justification for their selection.

1. **Validation Loss** measures how well the model generalizes to unseen data during training. A lower validation loss indicates better generalization and reduced overfitting.

2. **Accuracy** is a critical metric in assessing the correctness of the generated CTL formulas. In our context, accuracy is measured by comparing the generated CTL expressions against gold standard formulas, both syntactically and semantically. Given the com-

plexity of NL-to-CTL translations, accuracy is further broken down into six evaluation categories, which allow for finer granularity in assessing the performance of the models:

- Exact Match (ExacM): The predicted CTL expression exactly matches the gold standard.
- Equivalent Match (EquivM): The predicted CTL expression is logically equivalent to the gold standard but not syntactically identical.
- Exact Match with Temporal Operator Difference (ExacMTOD): The predicted CTL expression contains minor differences in temporal operators (e.g., AF instead of AX) but still captures the correct logical meaning.
- Equivalent Match with Temporal Operator Difference (EquivMTOD): The predicted CTL is logically equivalent to the gold standard with minor variations in temporal operators.
- Partially Correct (PC): The predicted CTL captures some aspects of the requirement but misses key components or logical operators.
- Incorrect (NC): The predicted CTL fails to capture the intended meaning of the requirement.

These categories were determined through a manual evaluation process conducted by a formal methods expert, who assessed each predicted CTL formula for both syntactic and semantic alignment. Accuracy is calculated as the number of correct predictions (ExacM, EquivM, ExacMTOD, and EquivMTOD) divided by the total number of predictions, as shown in Equation 8.1."

$$\text{Accuracy} = \frac{\text{ExacM} + \text{EquivM} + \text{ExacMTOD} + \text{EquivMTOD}}{\text{Total Predictions}} \tag{8.1}$$

Including different matching categories (e.g., temporal operator differences) ensures that the evaluation captures subtle but meaningful distinctions between correct and partially correct translations. This distinction is important in formal verification, where small temporal variations can be acceptable depending on the context.

3. **Semantic similarity** measures how close the generated CTL is to the gold standard in meaning, even when syntactic structures differ. We calculate semantic similarity using sentence embeddings generated by the *'all-MiniLM-L6-v2'* model [109] from the Sentence Transformers library. Cosine similarity between the embeddings of the predicted and gold standard CTL specifications is used to capture the semantic relevance of the predictions.

This metric is useful in cases where syntactic deviations are permissible but the overall meaning of the CTL expression remains intact. A higher semantic similarity score indicates that the generated CTL specification successfully captures the intent behind the NL requirement, even if the expression is not an exact match.

4. **Structural Operator Jaccard Similarity (SOJS)**: To evaluate the structural fidelity of the generated CTL formulas, we use SOJS. This metric focuses on comparing key CTL operators and structural elements, such as logical and temporal operators, which are essential for ensuring the formal correctness of the specification.

   The SOJS metric works by tokenizing both the predicted and the reference CTL formulas based on the core components of temporal logic (e.g., AG, EF, U, $\rightarrow$) and standard logical operators (e.g., &, |, !). These tokens capture the formal elements that define the structure of CTL expressions.

   The Jaccard similarity is then computed as the ratio of the intersection of tokens (shared elements between the predicted and gold formulas) to the union of tokens (the total distinct elements in both formulas), as shown in Equation 8.2. This approach ensures that the comparison focuses on the syntactic structure relevant to formal verification, rather than lexical differences.

$$\text{SOJS}(gold, pred) = \frac{tokens(gold) \cap tokens(pred)}{tokens(gold) \cup tokens(pred)} \tag{8.2}$$

   where tokens(gold) and tokens(pred) represent the sets of syntactic elements extracted from the gold-standard and predicted CTL formulas, respectively.

   SOJS is crucial in formal verification tasks where maintaining the correct logical operators is essential for ensuring that the temporal and branching semantics of the system are properly represented. A high SOJS score indicates that the model has correctly captured the formal structure of the requirement, which is a critical component in model checking and verification processes.

5. **Human Non-Expert Comparison**: To evaluate how non-expert users without specialized knowledge in formal methods perform in translating NL requirements into CTL, we conducted an empirical study. This study involved 82 undergraduate computer science students who had minimal or no prior experience with CTL or formal methods. The purpose of this comparison was to assess the accessibility of NL-to-CTL translation for non-experts and to benchmark the model's performance against novice human translations.

The participants were given a course on temporal logic and CTL and were then asked to translate selected NL requirements from the dataset. These NL requirements represented a variety of prevalent CTL patterns, ensuring a comprehensive evaluation across common temporal logic patterns.

Participants were asked to manually translate these NL requirements into CTL expressions. To assess their translations, a professor specialized in formal methods classified their responses into three categories:

- **Correct (C)**: The translation aligns with the intended CTL formula.
- **Partially Correct (PC)**: The translation captures some of the intended logic but contains significant errors or omissions.
- **Incorrect (NC)**: The translation fails to capture the intended meaning of the requirement.

This categorization provided valuable insights into how effectively non-experts grasp the concepts of temporal logic and the challenges they face in translating NL requirements into CTL. The motivation behind using non-experts for this comparison was to evaluate how well the automated models support users who lack formal methods expertise. The goal was to explore whether these models can lower the entry barrier for non-experts by simplifying the translation process.

8.5 Results & Discussion

This section presents the outcomes of our evaluation of the models fine-tuned for translating NL requirements into CTL specifications. We compare the performance of LLAMA3, Mistral, and Qwen2 using different evaluation metrics, and discuss the implications of these results in formal verification.

8.5.1 NLP Stage Evaluation

We used different metrics to evaluate the syntactic and semantic precision of CTL translations from NL, including validation loss, accuracy, semantic similarity, and SOJS. Table 8.1 provides a comparative analysis of the performance of LLAMA3, Mistral, and Qwen2 models.

Initially, all three models were evaluated using 12-fold cross-validation to assess their performance in translating NL requirements into CTL specifications.

The LLAMA3 model demonstrated the lowest validation loss (0.14) among the 12-fold models, indicating its strong ability to generalize to unseen data. However, its accuracy (54.29%)

Table 8.1 Model Performance Metrics.

Model	Validation Loss	Accuracy (%)	Semantic Similarity	SOJS Score
mistral-7b-instruct-v0.3-bnb-4bit [12 Folds]	0.16	66.67	0.75	0.70
mistral-7b-instruct-v0.3-bnb-4bit [50 Folds]	0.07	71.43	0.79	0.72
Qwen2-7B-Instruct-bnb-4bit [12 Folds]	0.29	44.76	0.74	0.66
llama-3-8b-Instruct-bnb-4bit [12 Folds]	0.14	54.29	0.75	0.69

and SOJS score (0.69) were slightly lower than those of the Mistral model. This suggests that while LLAMA3 performs well in generalizing the training data, it may not capture the syntactic and structural nuances as effectively as needed for precise CTL translations.

The Mistral model achieved superior performance across most metrics in the 12-fold evaluation, with a validation loss of 0.16, accuracy of 66.67%, semantic similarity of 0.75, and SOJS score of 0.70. These results indicate that Mistral captures both the syntactic structure and semantic meaning of NL requirements effectively, making it a strong candidate for further optimization. Based on these promising results, we decided to further optimize this model using 50-fold cross-validation.

After 50-fold cross-validation, the Mistral model demonstrated even better performance, achieving the best metrics overall: a validation loss of 0.07, accuracy of 71.43%, semantic similarity of 0.79, and SOJS score of 0.72. This improvement highlights the effectiveness of extensive cross-validation and fine-tuning in enhancing model performance. Table 8.2 exemplifies Mistral's variable translation outcomes.

In contrast, Qwen2 lagged in overall performance, with a validation loss of 0.29 and an accuracy of 44.76%. While it showed some effectiveness, it struggled to capture the intricacies of CTL specifications compared to the other models.

8.5.2 Comparison with Human Translations

To contextualize the performance of our models, we compared the fine-tuned model outputs with translations generated by non-expert human participants. The human translations served as a benchmark to evaluate the accessibility of NL-to-CTL translation for users without formal methods expertise.

Table 8.2 Examples of Successful and Failed Translations of Mistral.

NL Requirement	The system will display a blank set of ratings if there are no ratings yet defined.
Gold CTL	AG(!ratingsDefined → AF(blankRatingsDisplayed))
Prediction (EquivM)	AG(!exists(ratings)) → AF(displayBlankRatings)
NL Requirement	If the retractGears or the extendGears function becomes active, the GearsInTransitEvent shall be raised.
Gold CTL	AG((retractGearsActive | extendGearsActive) → AX(GearsInTransitEventRaised))
Prediction (ExacMTOD)	AG((retractGearsActive | extendGearsActive) → AF(GearsInTransitEventRaised))
NL Requirement	The DPU-EDAC shall install an ISR to be invoked when an SBE occurs and scrub the location containing the SBE.
Gold CTL	AG(SBE → AF(installISR & scrubLocation))
Prediction (EquivMTOD)	AG(SBEOccurs → AX(installISR & scrubSBELocation))
NL Requirement	The MMI on/off control shall be designed to prevent accidental activation/deactivation.
Gold CTL	AG(MMIControl → AX(!accidentalActivation | !accidentalDeactivation))
Prediction (PC)	AG(!accidentalActivation & !accidentalDeactivation)
NL Requirement	This mechanism shall allow calls with a higher assigned priority to override (pre-empt) existing calls of a lower priority.
Gold CTL	AG(highPriorityCall → AF(preemptLowPriorityCall))
Prediction (NC)	AG(callPreemptionRequest → AF(callPreemptionGranted))

The results showed that human-generated translations were correct in 39.33% of cases, partially correct in 13.72%, and incorrect in 46.95%. The comparison reveals that the Mistral model significantly outperforms human translations, particularly those produced by individuals with only basic training in CTL. This highlights the potential of our automated methods to not only supplement but in many cases surpass the accuracy of non-expert human translators in the context of formal verification.

These findings underscore the value of employing advanced LLMs in formal verification processes. The ability of the Mistral model to achieve a high rate of correct translations indicates that with targeted training and optimization, automated methods can approach expert-level accuracy. However, it is important to acknowledge the role of human supervision in this

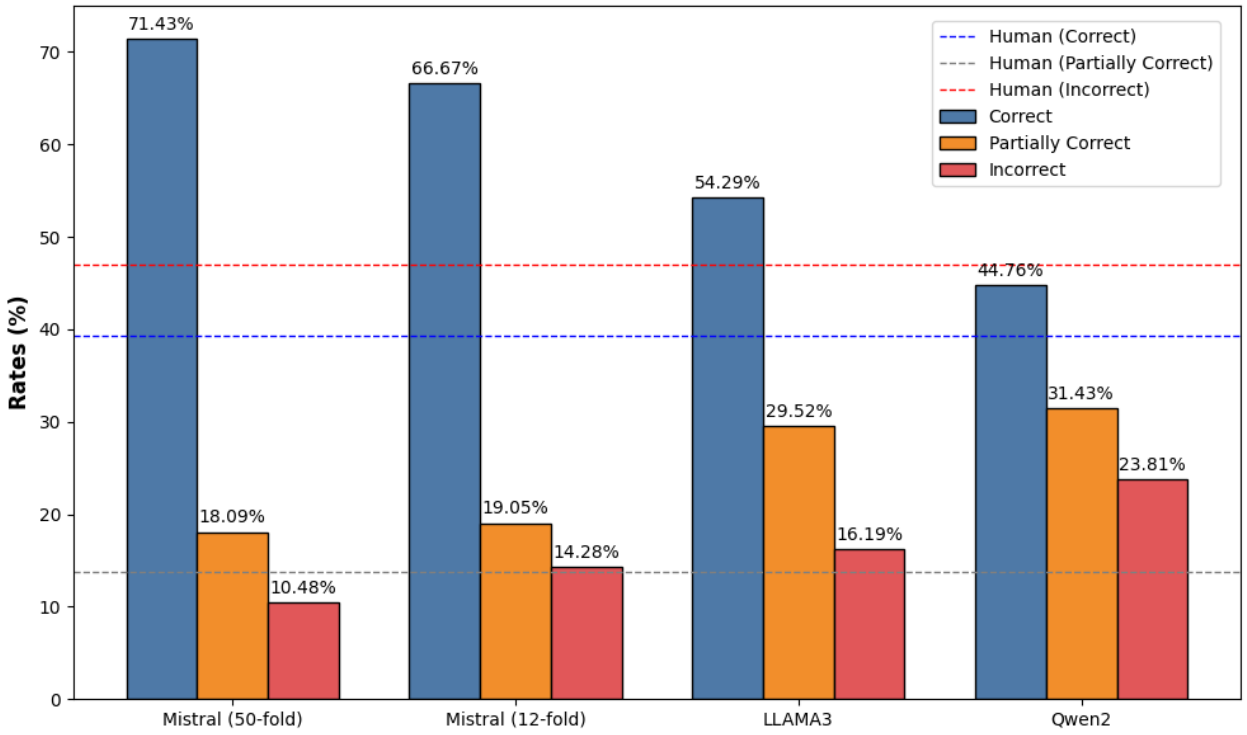


Figure 8.2 Evaluation of model-generated CTL specifications versus non-expert human performance.

process. The nuanced nature of CTL translations and the potential for syntactic variations that still maintain semantic correctness suggest that a collaborative framework integrating both automated tools and human expertise is ideal.

Figure 8.2 illustrates the comparative performance of the models against human translations, highlighting the improvements achieved by the Mistral model post-50-fold cross-validation.

8.5.3 Insights and Practical Implications

Our study provides key insights into the application of LLMs for translating NL requirements into CTL specifications.

The Mistral model, particularly after extensive cross-validation and fine-tuning, consistently outperformed other models and even non-expert human translators. This demonstrates the potential of LLMs to handle complex translation tasks in formal verification, reducing dependency on specialized human expertise.

The comparison with human translations highlights that automated methods can achieve and even surpass the accuracy of translations produced by individuals with limited formal

methods training. However, the variability in human performance underscores the need for continuous improvements and refinements in automated approaches to ensure consistent accuracy.

The generated CTL specifications' usability in actual model-checking scenarios is a crucial aspect of our study. The high accuracy and semantic similarity scores indicate that the translated specifications are not only syntactically correct but also retain the intended logical meaning. This means that the CTL formulas generated by our models can be directly employed in model-checking tools, thereby streamlining the verification process for embedded systems.

While advanced LLMs show promise, the integration of human supervision remains crucial. A collaborative framework that leverages the strengths of both automated tools and human expertise can ensure the reliability and correctness of CTL translations, particularly in critical applications within embedded systems.

Our findings suggest that, while deep specialist knowledge remains vital in formal verification, LLMs can significantly reduce the cognitive load and time required for those less experienced in formal methods. Rather than replacing expertise, these models provide a support system that assists users in refining CTL specifications. However, given the nuanced nature of LLM outputs and potential semantic variations, human oversight remains crucial to maintain high standards of accuracy.

8.6 Chapter Summary

This chapter detailed our exploration into fine-tuning LLMs to automate the translation of NL requirements into CTL specifications. Leveraging the Natural2CTL dataset, the study optimized these models for precision in formal verification workflows. The multi-metric evaluation framework, incorporating validation loss, accuracy, semantic similarity, and SOJS, confirmed the Mistral model as the most promising candidate after extensive cross-validation, outperforming both human non-expert translations and competing LLMs. These advancements highlight the potential of LLMs in formal verification, offering efficiency, scalability, and reduced reliance on manual translation. The next chapter transitions to a multi-approach methodology, incorporating fine-tuning, few-shot learning, and hybrid models for NL-to-CTL translation, exploring their comparative effectiveness and practical applications in formal verification workflows.

CHAPTER 9 BRIDGING NATURAL LANGUAGE AND FORMAL SPECIFICATIONS: A MULTI-APPROACH METHODOLOGY FOR NL-TO-CTL TRANSLATION USING LLMs

9.1 Chapter Overview

This chapter [1] explores translating NL requirements into CTL specifications by introducing two novel methodologies: few-shot learning with GPT-4 and a hybrid integration of BERT for pattern identification with GPT-4 for translation. Building on the findings in the previous chapter, which focused on optimizing NL-to-CTL translations through fine-tuning LLMs, particularly the Mistral model, this chapter broadens the methodological scope to investigate alternative approaches that do not rely solely on extensive fine-tuning.

The primary purpose of this chapter is to evaluate the effectiveness, efficiency, and adaptability of these alternative approaches in NL-to-CTL translation tasks. Few-shot learning with GPT-4 is designed to leverage the model's ability to generalize from a small set of examples, thus enabling accurate translation of diverse NL requirements without extensive training. The hybrid BERT-GPT approach, on the other hand, combines BERT's strengths in pattern recognition with GPT-4's generative capabilities. By first identifying the logical pattern underlying each NL requirement with BERT and then using GPT-4 to produce the corresponding CTL formula, this integration aims to enhance accuracy in capturing complex logical structures.

Through a comparative analysis of these methods against the previously fine-tuned Mistral model, this chapter aims to assess the advantages and limitations of each approach, specifically regarding their precision, semantic alignment, and applicability in formal verification tasks. By examining the nuanced strengths of few-shot and hybrid methods, this study contributes valuable insights into how LLMs can be effectively adapted for automated formal specification generation in RE and formal methods.

[1] Part of the content of this chapter is submitted for publication as: Zrelli, R., Misson, H.A., Ben Attia, M., Magalhaes, F.G.d., Nicolescu, G.: "Bridging Natural Language and Formal Specifications: A Multi-Approach Methodology for NL-to-CTL Translation Using LLMs". In: *International Journal on Software and Systems Modeling (SoSyM)* (2024).

9.2 Study Design and Objective Setting

The objective of this chapter is to extend the exploration of translating NL requirements to CTL specifications by evaluating alternative methodologies that address scalability and data-dependency challenges inherent in formal verification tasks. Unlike the fine-tuning-focused approach of the previous chapter, this study introduces two distinct methodologies—few-shot learning with GPT-4 and a hybrid BERT-GPT integration—to explore flexible, low-data approaches to NL-to-CTL translation. The key objectives guiding this study are as follows:

- **To evaluate alternative methodologies for NL-to-CTL translation with a focus on minimal data dependency (RO1)**: This chapter investigates few-shot learning and BERT-GPT hybrid integration as alternative methodologies that reduce the need for extensive fine-tuning while aiming for syntactic accuracy and semantic fidelity in CTL translations. This objective supports the broader goal of providing scalable, adaptable translation methods suitable for varied data availability and resource constraints in formal verification contexts.
- **To implement a comparative evaluation framework across diverse performance metrics (RO2)**: Using accuracy, SOJS, and semantic similarity metrics, we evaluate and compare the few-shot learning and BERT-GPT hybrid approaches against the previously fine-tuned Mistral model. This framework provides insights into the strengths and limitations of each approach, with attention to their applicability in settings that prioritize efficient, adaptable translation processes over full-scale fine-tuning.
- **To analyze model-generated translations against human performance to establish benchmarks (RO3)**: By benchmarking model-generated CTL translations with those of non-expert human translators, we aim to assess the degree of alignment between automated and human interpretations, especially where semantic accuracy may vary across translation methods. This comparison provides a baseline for understanding where human oversight may still be essential for achieving the rigorous standards required in formal verification for safety-critical systems.

In alignment with these objectives, the study proceeds by first processing and preparing the Natural2CTL dataset, focusing on balanced data representation across varied CTL translation patterns. We then implement three distinct approaches: fine-tuning Mistral on the dataset, setting up GPT-4 with few-shot learning using select examples, and integrating BERT for pattern identification with GPT-4 in a hybrid model. A structured evaluation follows, where model outputs are assessed across predefined metrics. Finally, a comparative

analysis is conducted to examine the translation performance across LLM approaches and the baseline human performance. Figure 9.1 provides an illustrative flow of this methodological process.

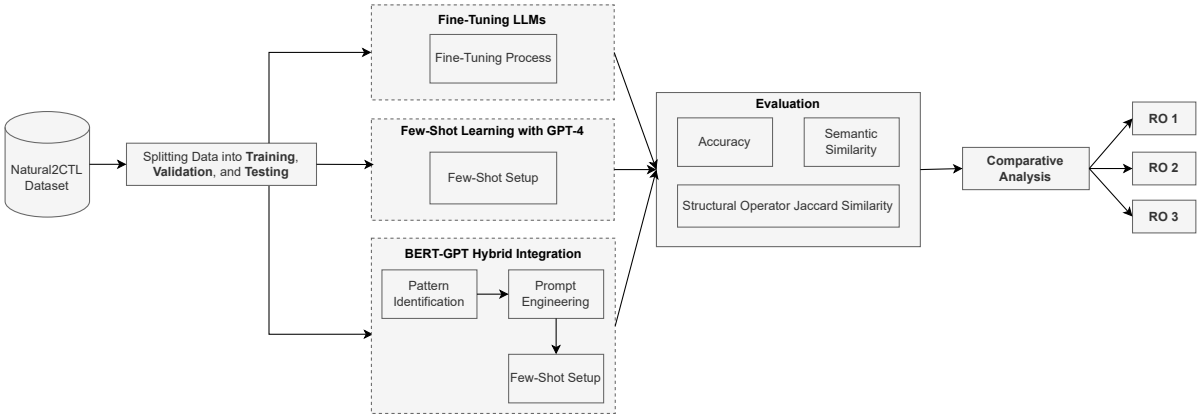


Figure 9.1 Schematic diagram of the study.

9.3 Dataset

The Natural2CTL dataset, as comprehensively discussed in Chapter 8, forms the foundation of this study, consisting of 2,095 NL requirements paired with CTL specifications. The dataset includes requirements that emphasize temporal behaviours, such as safety and liveness properties, across various domains.

The dataset's development involved rigorous annotation processes and pattern-based mappings to ensure accurate and representative CTL translations. Detailed information on the dataset's creation, including the systematic annotation methodology, validation strategies, and pattern types, is provided in the aforementioned chapter, where its robustness and applicability in formal verification are discussed.

9.3.1 Dataset Preparation for the Fine-Tuning Approach

For the fine-tuning process, the dataset was structured to ensure balanced and representative training and evaluation. The dataset was first loaded from a CSV file, where a diverse set of instructional phrases was introduced. These phrases were designed to guide the translation of NL requirements into CTL specifications. Examples include prompts like "Transform the following sentences into CTL expressions" and "Translate the following statements into CTL formulas". The inclusion of varied instructions aimed to improve the model's ability to handle

different expressions and contexts, enhancing its generalization capability during inference.

The dataset was then divided using a stratified shuffle split method [110], which allocated 90% of the data for training and 10% for testing. This method ensured that each subset accurately reflected the distribution of CTL patterns found in the dataset. To prevent class imbalances in the evaluation phase, any unique CTL patterns that appeared only once were kept exclusively in the training set. The rationale behind this decision was to ensure that the model had sufficient exposure to these rare patterns during training, allowing it to learn and handle them effectively. While this choice could limit the evaluation of the model's generalization on these rare patterns, the decision was made to provide the model with enough data to recognize and process them.

Next, the test subset was split evenly into validation and test sets. This structure provided a clear separation between the data used for model tuning and the data reserved for final performance evaluation on unseen examples. After segmentation, the training, validation, and test datasets were saved into separate CSV files and then uploaded to the Hugging Face Hub using the Hugging Face datasets library. This facilitated seamless integration with the model training pipeline.

9.3.2 Dataset Preparation for the Few-Shot Learning Approaches

For the few-shot learning approaches, we leveraged the test set segmented during the fine-tuning process. This ensured consistency across the evaluation phases, allowing for a fair comparison between the fine-tuned models and the few-shot methods.

1. **Direct Few-Shot Learning**: In this approach, we selected 15 examples from the train set to showcase a broad range of CTL patterns and requirement scenarios. The selection process prioritized diversity and complexity, reflecting the types of translation challenges encountered in real-world formal verification tasks. The examples included both simple and complex NL requirements, covering varied temporal behaviours and logical structures. Few-shot learning is inherently designed to achieve strong model performance using minimal examples. Extensive empirical testing revealed that increasing the number of examples beyond 15 yielded only marginal improvements in accuracy. Thus, 15 examples were identified as an optimal number, providing sufficient diversity to guide the model effectively without requiring additional examples. This approach ensures that the model generalizes well across NL-to-CTL translation tasks while maintaining computational efficiency.

2. **Pattern Identification and Few-Shot Learning**: For this hybrid approach, we uti-

lized the same training, validation, and test sets defined in the Direct Few-Shot Learning approach to ensure consistency in evaluation. While using the same dataset distribution helped maintain alignment across approaches, we acknowledge that employing different data splits for each technique could further reveal the unique strengths and limitations of each method. In this study, BERT was fine-tuned to classify NL requirements based on predefined CTL patterns, and these classifications were then provided to GPT-4 along with the corresponding few-shot examples. By aligning the datasets across methods, we aimed to attribute any observed improvements to the pattern-based integration rather than potential variations introduced by differing dataset distributions.

9.4 Methodology Overview

This study builds upon the findings from Chapter 8, where we fine-tuned three LLMs for NL-to-CTL translation, with the Mistral model achieving the best performance. In this study, we further explore the capabilities of LLMs by introducing and comparing two new approaches: GPT-4 few-shot learning and BERT-GPT hybrid integration. These approaches and the fine-tuned Mistral model are evaluated to gain a deeper understanding of their performance and applicability in automating formal specification generation.

9.4.1 Few-Shot Learning with GPT-4

For the few-shot learning approach with GPT-4, we aimed to leverage the model's ability to generalize from a small number of examples while maintaining translation accuracy across diverse CTL patterns. This approach was designed to showcase how LLMs can handle the complexity of NL-to-CTL translations without extensive fine-tuning.

The process begins with setting up GPT-4 as a CTL expert through an initial system message: *"You are a CTL expert. Your task is to translate requirement statements into self-contained CTL formulas. You return a JSON with fields 'id' and 'ctl'."* This instruction guides GPT-4 in consistently generating outputs aligned with the expected CTL format.

The training examples are selected from the train set, with 15 examples chosen for their diversity across different CTL patterns and logical complexities. These examples are presented in a structured conversational format, where the user provides a requirement, e.g., *"Translate the following requirement statement into CTL: 'requirements_text',"* and GPT-4 responds with the corresponding CTL formula in JSON format, such as "id": X, "ctl": "CTL_formula".

The selected examples, while minimal in number, were determined to be sufficient based

on empirical evaluation, which indicated that increasing the number of examples yielded diminishing returns in performance relative to the API cost [111]. By limiting the number of examples to 15, we achieved a balance that ensures translation accuracy without the need for extensive training data or resources.

During the few-shot learning process, GPT-4 was iteratively presented with these examples, enhancing its ability to generate accurate CTL structures from minimal contextual cues. The setup reinforced GPT-4's understanding of the linguistic and logical nuances necessary for converting NL requirements into formal specifications.

For inference, GPT-4 processes novel requirements using a standardized prompt similar to the one employed during training. The prompt format is kept simple: *"Translate the following requirement statement into CTL: 'requirements_text'."*

By employing this few-shot learning setup, this approach demonstrates how GPT-4 can be scaled for practical formal verification tasks, offering a solution for converting NL requirements into formal specifications without the overhead of large-scale data or model retraining.

9.4.2 BERT-GPT Hybrid Integration for Pattern Identification

To enhance the translation accuracy of NL requirements into CTL specifications, we implemented a hybrid approach that integrates pattern identification using a fine-tuned BERT model with GPT-4's few-shot learning capabilities. This method leverages the strengths of both models: BERT's ability to accurately classify NL requirements into predefined CTL patterns and GPT-4's flexibility in generating contextually accurate CTL formulas based on minimal examples.

The rationale for this hybrid approach is to first identify the logical pattern underlying each NL requirement using a fine-tuned BERT model, and then provide this identified pattern as contextual information to GPT-4 during the few-shot learning process. By doing so, we guide GPT-4 to generate more precise CTL formulas that align with the identified logical structure, thereby reducing the likelihood of translation errors, especially for complex or ambiguous requirements.

The BERT model was fine-tuned on the Natural2CTL dataset, where each NL requirement was labelled with one of 11 predefined CTL patterns[2] (e.g., Absence, Existence, Response) [105]. The fine-tuning process involved training BERT to classify requirements based on these patterns using a multi-class classification setup. While we explored multiple transformer models during initial experimentation, including RoBerta [112], DistilBERT [113],

[2]More details about patterns can be found at `http://patterns.projects.cis.ksu.edu/`

and Electra [114], BERT demonstrated the best overall performance in terms of accuracy.

The fine-tuned BERT model was used to predict the pattern for each new requirement during inference. Once a pattern was identified, this information was incorporated into the prompt given to GPT-4. For instance, if a requirement was classified under the "Existence" pattern, the prompt provided to GPT-4 would include this pattern information, guiding the model to generate a CTL formula that adheres to the logical constraints of that pattern.

In this setup, the pattern identified by BERT is used to enhance GPT-4's few-shot learning process. The integration occurs by appending the pattern information to the user's prompt, which informs GPT-4 about the specific CTL structure it should focus on when generating the output. This hybrid approach is particularly effective for handling requirements that involve nuanced logical relationships, as it combines BERT's classification strength with GPT-4's generative capabilities.

The process can be summarized as follows:

1. **Pattern Identification**: For each NL requirement, BERT predicts the most relevant CTL pattern.

2. **Contextual Prompting**: The identified pattern is incorporated into the GPT-4 prompt.

3. **Few-Shot Inference**: GPT-4 processes the enhanced prompt and generates a CTL formula consistent with the identified pattern.

The pattern identification model was fine-tuned using labelled training data, with cross-validation performed to ensure generalizability across different CTL patterns. During inference, the predicted pattern was appended to the GPT-4 prompt in the following format: *Translate the following requirement statement into CTL: 'requirements_text' I propose that we use this pattern: 'predicted_pattern'.*

The prompt was then processed by GPT-4, which generated the CTL formula based on both the NL requirement and the suggested pattern. The integration of BERT and GPT-4 will not only improve translation accuracy but also provide a scalable approach for handling varied and complex NL requirements in formal verification tasks.

9.4.3 Fine-Tuning the Mistral Model

As detailed in Chapter 8, the Mistral model was fine-tuned specifically for translating NL requirements into CTL specifications. This process involved multiple rounds of cross-validation,

adapter-based parameter optimization, and custom prompt formatting to maximize accuracy and reliability in formal specification generation.

For this study, we include the fine-tuned Mistral model as a baseline to compare its performance against two new approaches—GPT-4 few-shot learning and the BERT-GPT hybrid integration. These new methods provide alternative strategies for NL-to-CTL translation without the intensive fine-tuning required for Mistral, enabling a comparative analysis of performance, adaptability, and computational efficiency across models.

9.5 Evaluation Metrics

To evaluate the accuracy, semantic fidelity, and structural integrity of the CTL formulas generated by each model, we apply the same comprehensive set of metrics as in Chapter "Advancing Formal Verification: Fine-Tuning LLMs for Translating Natural Language Requirements to CTL Specifications". These metrics are essential in assessing translation quality, given the nuances of formal verification.

In this chapter, we exclude the Validation Loss metric, as our focus here shifts to comparing three distinct translation approaches: Mistral fine-tuning, GPT-4 few-shot learning, and the BERT-GPT hybrid. The following metrics guide our assessment:

- Accuracy: As detailed in Chapter 5, this metric measures the correctness of each generated CTL formula by comparing it with gold-standard formulas across six defined categories—Exact Match (ExacM), Equivalent Match (EquivM), Exact Match with Temporal Operator Difference (ExacMTOD), Equivalent Match with Temporal Operator Difference (EquivMTOD), Partially Correct (PC), and Incorrect (NC). The different matching categories, including those focused on temporal operator variations, ensure that both syntactic and semantic nuances are captured.

- Semantic Similarity: This metric evaluates the alignment in meaning between the generated and gold-standard CTL formulas, regardless of syntactic differences. As in Chapter 5, we calculate semantic similarity using cosine similarity on sentence embeddings, with a higher score indicating closer alignment with the NL requirement's intended meaning.

- Structural Operator Jaccard Similarity (SOJS): To gauge the structural fidelity of CTL formulas, SOJS compares key logical and temporal operators, as well as branching structures, to ensure formal verification consistency. A high SOJS score here signals

that the models have accurately captured the structural semantics critical for model checking.

- Human Non-Expert Comparison: We again benchmark our models against translations provided by non-expert participants. Following the methodology described in Chapter 5, participants' translations are categorized as Correct (C), Partially Correct (PC), or Incorrect (NC), offering insights into the accessibility of NL-to-CTL translation for users without formal verification expertise.

These metrics provide a robust basis for comparing the three approaches; each intended to meet the high accuracy and reliability standards for formal verification tasks. By building on the established evaluation framework, this study seeks to identify the strengths and limitations of few-shot and hybrid learning methods relative to the fine-tuned Mistral baseline.

9.6 Results and Comparative Analysis

This section presents a detailed performance comparison across three approaches—fine-tuning the Mistral model, few-shot learning with GPT-4, and the BERT-GPT hybrid integration for pattern identification. The performance was evaluated using the metrics of accuracy, semantic similarity, and SOJS, as outlined earlier. Additionally, we include comparisons with human-generated translations by non-expert users to provide a benchmark for evaluating the accessibility and effectiveness of the automated methods.

9.6.1 Performance Comparison Across Approaches

The fine-tuned Mistral model delivered the highest overall performance, achieving an accuracy of 71.43%, which includes 22.86% ExacM, 44.29% EquivM, 5.71% ExacMTOD, and 27.14% EquivMTOD. The model demonstrated robust performance across both semantic and structural evaluations, with a semantic similarity score of 0.79 and an SOJS score of 0.72. These results suggest that the fine-tuned Mistral model is highly effective at capturing both the syntactic structure and semantic meaning of NL requirements. The strong performance in the equivalence categories (EquivM and EquivMTOD) indicates the model's ability to generate CTL formulas that are logically correct, even when syntactically distinct from the gold standard.

The GPT-4 few-shot learning approach showed significantly lower performance, with an accuracy of 44.76%. Notably, the model produced no ExacM but delivered 31.92% EquivM, 17.02% ExacMTOD and 51.06% EquivMTOD. This suggests that while GPT-4 is capable of

capturing logical equivalence, it struggles with producing syntactically perfect translations. The semantic similarity score of 0.70 further supports this, indicating that GPT-4 captures the intended meaning, albeit less precisely than the Mistral model. The SOJS score of 0.61 reflects GPT-4's relatively lower performance in maintaining the correct structural elements of CTL expressions. These results imply that GPT-4 can serve as a flexible, general-purpose model for NL-to-CTL translation but lacks the precision needed for high-stakes formal verification tasks without further fine-tuning.

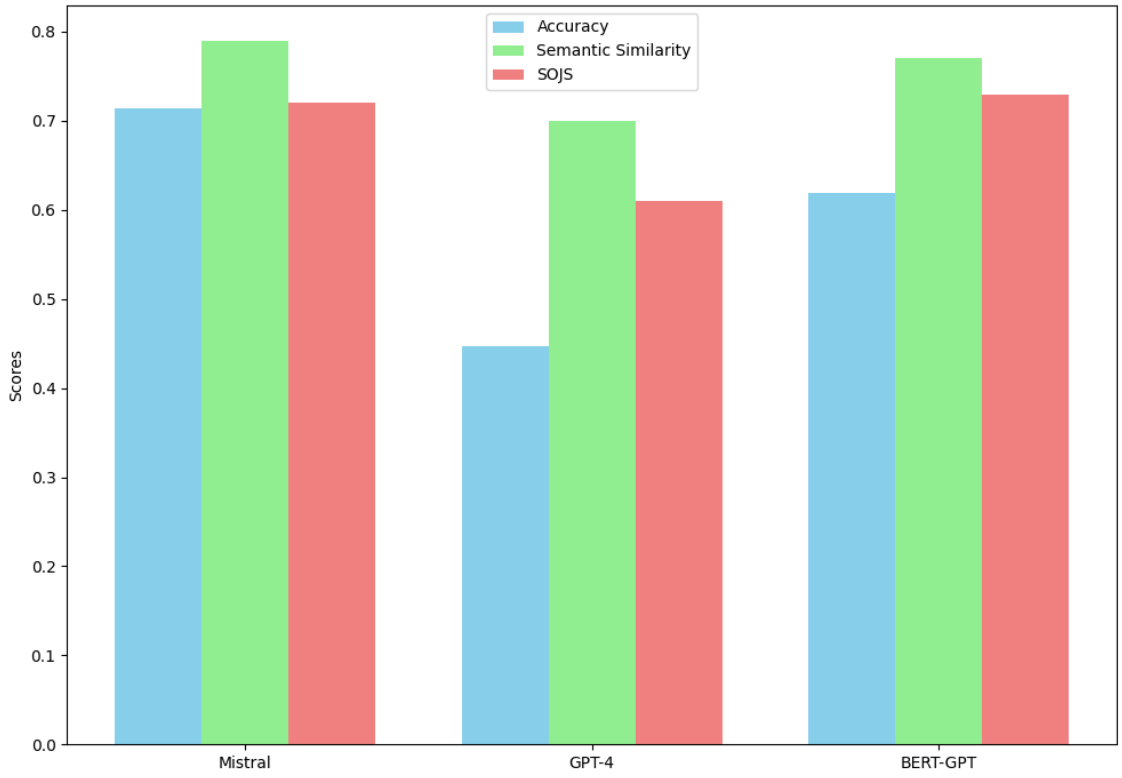


Figure 9.2 Performance comparison across three approaches (Mistral fine-Tuning, GPT-4 few-shot learning, and BERT-GPT hybrid integration) based on accuracy, semantic similarity, and SOJS.

The BERT-GPT hybrid approach provided an intermediate performance between fine-tuning Mistral and GPT-4 alone. The accuracy of 61.9% includes 12.31% ExacM, 58.46% EquivM, 3.08% ExacMTOD, and 26.15% EquivMTOD. The strong performance in the equivalence category (58.46% EquivM) demonstrates the effectiveness of integrating BERT for pattern identification, which helps guide GPT-4 in producing more accurate translations. The semantic similarity score of 0.77 and the SOJS score of 0.73 are close to those of the fine-tuned Mistral model, indicating that the hybrid approach benefits from pattern identification in generating structurally and semantically accurate CTL formulas. These results suggest that

BERT-GPT hybrid integration can be an effective alternative, particularly when computational resources for extensive fine-tuning are limited.

Figure 9.2 visually summarizes these findings, comparing the models' performances across accuracy, semantic similarity, and SOJS. The graph highlights Mistral's superiority across all metrics, followed by the BERT-GPT hybrid model. While GPT-4 demonstrates strong semantic performance, it lags behind in accuracy and structural fidelity.

Table 9.1 illustrates both successful and failed translations generated by the three different models, providing insights into their strengths and weaknesses in NL-to-CTL translation.

In the first row of the table, GPT-4 mistakenly applied an "existence" pattern using the AF operator, whereas the NL requirement clearly demands a "universality" pattern, indicated by the AG operator. This misinterpretation highlights GPT-4's difficulty in correctly handling temporal operators, which results in only a PC translation. In contrast, both Mistral and the BERT-GPT hybrid models successfully generated syntactically correct CTL formulas, with Mistral delivering an ExacM and the BERT-GPT hybrid producing an EquivM.

In the second row, Mistral generates a nearly correct translation but incorrectly substitutes AF (for all paths eventually) for EF (for some paths eventually). This discrepancy in temporal operators places the model's output in the ExacMTOD category, reflecting Mistral's general strength in logical flow but occasional errors in temporal precision. The error could lead to incorrect system behaviour in time-sensitive contexts. Meanwhile, GPT-4's translation misses key elements of the requirement. It simplifies the condition, focusing only on "existingName," and omits the "newStaff" condition entirely. Additionally, the temporal operator EF applies incorrectly to "existingName" rather than the entire conditional logic. This translation is classified as PC, consistent with GPT-4's broader limitations in capturing complex, multi-condition requirements while still producing semantically relevant but incomplete outputs. The BERT-GPT hybrid, though not perfect, captures the full logical structure and applies the correct operator, classifying it under EquivMTOD for its minor operator difference. These examples demonstrate the strengths of Mistral and the BERT-GPT hybrid in handling formal specifications for safety-critical systems, while GPT-4 struggles with temporal precision and complex logical structures.

9.6.2 Comparison with Human Performance

To further contextualize the performance of our models, we compared the generated CTL formulas with those produced by non-expert human participants. The participants achieved an accuracy of 39.33%, which is significantly lower than all the automated approaches. Of

Table 9.1 Examples of Successful and Failed Translations by Different Approaches.

NL Requirement	**Expected CTL**	**Mistral**	**BERT+GPT**	**GPT-4**
The TCS shall be rail transportable.	AG(railTransportable)	AG(railTransportable) (ExacM)	AG(TCSRailTransportable) (EquivM)	AF(railTransportable) (PC)
The system will generate a message while creating a new staff with an existing name, but the system will allow the user to save that new staff if it is required.	AG((newStaff & existingName) $\rightarrow$ EF(warningMessage & saveNewStaff))	AG((newStaff & existingName) $\rightarrow$ AF(warningMessage & saveNewStaff)) (ExacMTOD)	AG(newStaffWith-Existing-Name $\rightarrow$ AF(generateMessage & possibleToSaveNewStaff)) (EquivMTOD)	EF(existingName $\rightarrow$ (generateMessage & canSaveNewStaff)) (PC)

the human-generated CTL formulas, 39.33% were classified as correct, 13.72% as partially correct, and 46.95% as incorrect. These results highlight the difficulty non-experts face in translating NL requirements into formal CTL expressions, even after receiving instruction in temporal logic.

In contrast, the fine-tuned Mistral model significantly outperformed the human participants, with 71.43% of its translations classified as correct, 18.09% as partially correct, and only 10.48% as incorrect. Similarly, the BERT-GPT hybrid approach demonstrated strong performance, with 61.9% correct translations and 14.28% incorrect, positioning it as a viable alternative to Mistral. GPT-4's performance, while lower than the other automated approaches, still surpassed human performance with 44.76% correct translations and 17.14% incorrect.

The comparison underscores the potential of these automated models as tools to assist users who lack expertise in formal methods. The models, particularly the fine-tuned Mistral and BERT-GPT hybrid, provide high levels of accuracy and semantic understanding that significantly reduce the cognitive load on non-expert users.

To better illustrate the results, Figure 9.3 shows the accuracy of each approach in comparison with human performance, highlighting the significant performance gap between automated methods and non-expert human translations.

9.7 Case Studies and Applications

To illustrate the practical application of the generated CTL formulas, we present an example from the test set of our Natural2CTL dataset. This section demonstrates the readiness of the generated CTL formulas for model checking with minimal adjustments, showcasing the capability of our approach for formal verification.

9.7.1 Illustrative Example

- **NL Requirement**: *"The DPU-TIS shall provide the capability for an application program to jam the value of the SC_TIME into the hardware if automatic time synchronization is disabled."*
  This requirement is critical for ensuring correct timing operations in safety-critical systems, particularly when automatic synchronization is turned off.
- **Gold CTL Formula**: $AG(!\,\text{automaticTimeSynchronization} \rightarrow AF\ (\text{jamSC\_TIME}))$
- **Predicted CTL Formula by Mistral**: $AG(!\,\text{autoTimeSynchronization} \rightarrow AF\ (\text{jamSC\_TIME}))$

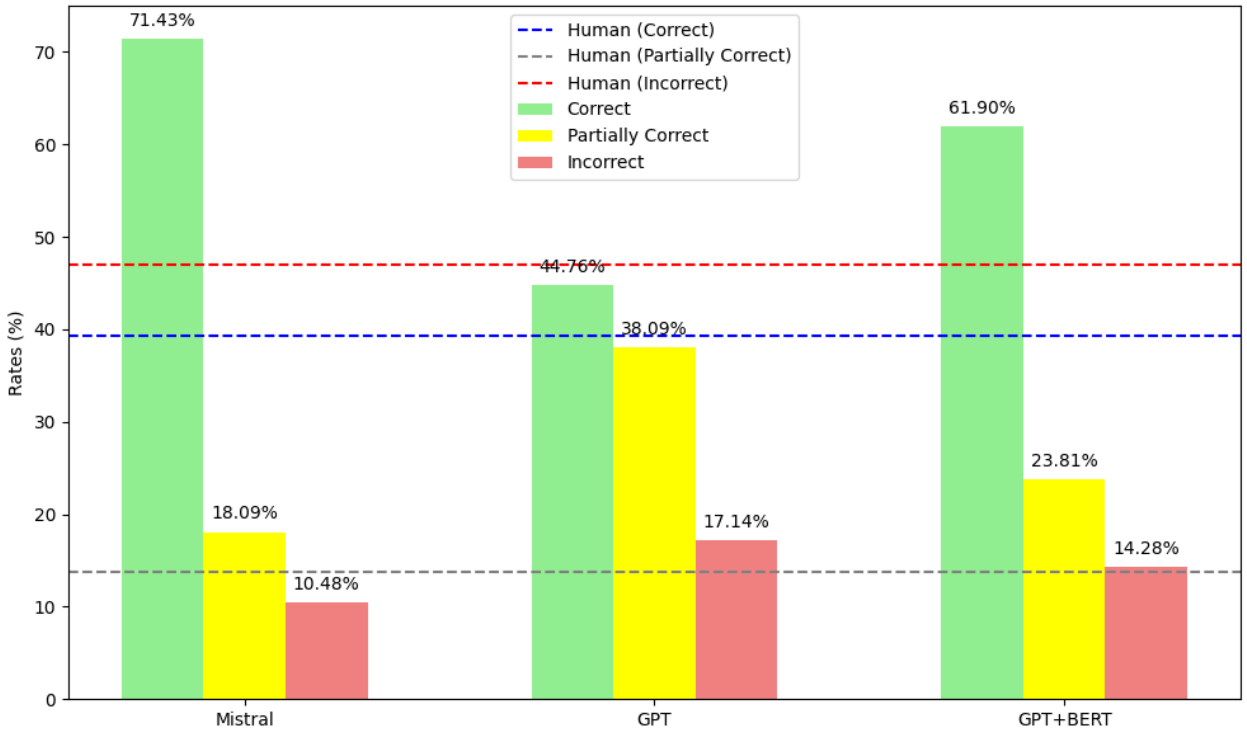


Figure 9.3 Comparison of C, PC, and NC translations generated by Mistral, GPT-4, and BERT-GPT approaches, benchmarked against non-expert human translations.

The Mistral model produced a CTL formula nearly identical to the gold standard. A minor adjustment was made to conform to the state variables used in our NuSMV model, where the generated variable *autoTimeSynchronization* was renamed to match the variable *automaticTimeSynchronization*. Such naming adjustments are typical in formal verification workflows and do not detract from the validity of the generated CTL formula.

9.7.2 Model Checking with NuSMV

We modelled the system in NuSMV [115] to verify the correctness of the generated CTL property. NuSMV was chosen for its robust support of CTL properties, making it suitable for verifying the automatically generated specifications. The model captures the system's state transitions, where the *automaticTimeSynchronization* variable controls whether time synchronization is enabled, and *jamSC_TIME* determines whether SC_TIME is jammed into the hardware.

NuSMV Model The NuSMV model shown in Listing 9.1 captures the key system states, ensuring that both time synchronization and SC_TIME jamming are properly represented.

```
MODULE main
VAR
    automaticTimeSynchronization : boolean;
    jamSC_TIME : boolean;

ASSIGN
    init(automaticTimeSynchronization) := TRUE;
    init(jamSC_TIME) := FALSE;

    next(automaticTimeSynchronization) := case
        automaticTimeSynchronization = TRUE : {TRUE, FALSE};
        TRUE : FALSE;
    esac;

    next(jamSC_TIME) := case
        !automaticTimeSynchronization : TRUE;
        TRUE : jamSC_TIME; -- once jammed, remains jammed
    esac;

-- CTL Property
SPEC AG(!automaticTimeSynchronization -> AF(jamSC_TIME))
```

Listing 9.1 NuSMV model for SC_TIME jamming.

Verification Results The CTL property generated by Mistral, with a minor adjustment to rename the state variable *autoTimeSynchronization* to *automaticTimeSynchronization*, was successfully verified as true in the NuSMV model checker. The verification output (Figure 9.4) confirms that the system satisfies the requirement that SC_TIME is eventually jammed whenever automatic synchronization is disabled.

```
NuSMV > check_ctlspec
-- specification AG (!automaticTimeSynchronization -> AF jamSC_TIME)  is true
```

Figure 9.4 NuSMV verification output confirming correctness of the CTL property.

9.7.3 Interpretation of Results

This example illustrates how the proposed approach could be applied to various similar formal verification tasks across different domains. It demonstrates that the CTL formula predicted by the fine-tuned Mistral model, after a minor adjustment to match the state variables in the NuSMV model, is ready for use in model checking. The property was verified successfully, demonstrating the capability of our approach to generate usable formal specifications with minimal human intervention.

The minor renaming of variables highlights that while automated translation is highly effective, small modifications may still be necessary depending on the state variable nomenclature used in specific system models. However, the overall logical structure and meaning of the formula remain intact, showcasing the robustness of the generated CTL properties.

9.8 Analysis of the results

This study has presented an advancement in automating the translation of NL requirements into CTL specifications, using several approaches—fine-tuning the Mistral model, GPT-4 few-shot learning, and BERT-GPT hybrid integration. A detailed comparison of these models, based on accuracy, semantic similarity, and structural fidelity, reveals that while progress has been made, challenges remain in ensuring both syntactic precision and logical equivalence.

9.8.1 Mistral Fine-Tuning: High Performance with Gaps in Precision

The fine-tuned Mistral model achieved the highest overall performance, with an accuracy of 71.43%. However, a deeper analysis of its output reveals that exact syntactic matching remains a significant challenge, as only 22.86% of the translations were classified as ExacM. This gap highlights that the model still struggles with achieving syntactic precision in generating CTL formulas that exactly replicate the gold standard specifications. Despite this, the high proportion of logically equivalent matches (EquivM) and matches with minor temporal operator differences (ExacMTOD and EquivMTOD) suggests that the model captures the correct logical intent, even if not expressed syntactically perfectly.

The difficulty in achieving perfect accuracy can be linked to the NL's inherent ambiguity and variability. NL requirements often present subtle logical structures that the model must navigate. Even with fine-tuning, the Mistral model tends to produce formulas that diverge slightly in structure but retain the same logical meaning, leading to a higher count of equivalent matches.

The Mistral model frequently captured the correct logical flow but struggled with minor variations in temporal operators, as indicated by its performance in the ExacMTOD and EquivMTOD categories. These challenges, while consistent with broader difficulties in temporal operator handling across models, highlight the need for further refinement in applying CTL's temporal logic with precision in high-stakes verification tasks.

This issue reflects a broader difficulty in translating NL into temporal logics, where temporal operators like "AG" (for all paths globally), "AF" (for all paths eventually) or "AX" (for all paths next) must be applied with precision. Given the model's performance in these categories, we can conclude that more targeted fine-tuning on temporal logic distinctions could further enhance the model's reliability in this domain.

Another important consideration is the proportion of PC and NC translations. The model generated 18.09% PC translations, where some aspects of the logical structure were captured, but key components were missed. Specifically, these missing components often involved critical temporal operators (e.g., misinterpreting "eventually" as "globally") or omitted conditions in branching structures, which are essential for representing the full system behaviour in CTL. For example, in some cases, the model successfully identified the logical flow of the requirement but failed to capture all relevant conditions or misused operators that govern how the system should react over time. This finding underscores the complexity of the NL-to-CTL task, as the model may not easily translate certain logical patterns without further contextual information. This challenge is compounded in requirements involving more abstract or domain-specific concepts, where the model must infer not only the logical operators but also the underlying system behaviours implied by the NL input.

The remaining 10.48% of incorrect translations are indicative of the limitations that still exist in current LLM-based approaches to formal verification. These incorrect predictions suggest that while models like Mistral have made significant strides, there is still room for improvement in generating consistently accurate CTL formulas, particularly in complex or ambiguous cases. This is an area where human oversight remains crucial, as automated models cannot yet fully replace expert judgment in formal verification workflows.

Despite these challenges, the Mistral model exhibits strong performance in terms of semantic similarity and SOJS. These metrics evaluate how closely the generated CTL formulas align with the intended meaning and structural integrity of the gold-standard specifications, even in cases where syntactic differences arise. The high scores achieved in these areas confirm that while syntactic precision remains a challenge, the model is proficient at capturing the overall logic and structural patterns required for formal verification.

The use of SOJS, in particular, highlights the model's strength in maintaining the correct

logical operators and temporal semantics, which are critical for ensuring that system properties are properly represented in model-checking tasks. This suggests that the Mistral model can serve as a valuable tool for supporting formal verification tasks.

9.8.2 GPT-4 Few-Shot Learning: Flexibility at the Cost of Precision

The GPT-4 few-shot learning approach exhibited its flexibility by generating logically sound CTL formulas with a moderate accuracy of 44.76%. However, this flexibility comes at the cost of precision, as the model produced no exact matches (0% ExacM), and its performance relied heavily on logically equivalent matches (31.92% EquivM) and temporal operator variations (68.08% across ExacMTOD and EquivMTOD). While GPT-4 is capable of capturing the logical meaning behind NL requirements, it struggles to produce CTL specifications that are syntactically perfect.

PC and NC classifications reveal that the model continues to struggle with more complex or less common logical structures. The partial correctness indicates that the model can identify fragments of the logic but misses key operators or relationships. The incorrect translations highlight areas where the model fails to grasp the logical structure altogether, often in cases where intricate domain-specific logic is required or where multiple dependencies between conditions exist.

This outcome reflects a broader limitation in GPT-4's few-shot learning capabilities: the inability to generalize sufficiently from a small set of examples to generate exact translations for formal methods. The model's reliance on equivalence and temporal operator substitutions highlights its tendency to capture the logical flow of the requirements but fails to maintain the syntactic precision required for CTL specifications. Furthermore, GPT-4 struggles with structural fidelity, where the correct application of temporal and logical operators is crucial for maintaining the exact branching-time semantics of the system. These structural deviations can lead to erroneous or incomplete verifications in formal verification tasks, where even minor misinterpretations in the CTL formula can result in incorrect system behaviour.

Nonetheless, GPT-4's ability to generate semantically correct, if syntactically imperfect, CTL formulas makes it a valuable tool for rapid prototyping or initial translations of NL requirements. In this context, it can provide fast, preliminary translations that require minimal manual adjustments. However, given its reliance on logical equivalence rather than syntactic precision, human oversight is indispensable. Experts must review and refine these translations to ensure they meet the exacting standards necessary for formal verification, especially in safety-critical domains where even minor errors can lead to incorrect system behaviour.

9.8.3 BERT-GPT Hybrid Integration: A Balance of Precision and Flexibility

The BERT-GPT hybrid model achieved an accuracy of 61.9%, striking a balance between Mistral's high accuracy and GPT-4's flexibility. BERT's pattern identification improves its ability to capture syntactic structure, resulting in a higher number of exact matches (12.31% ExacM) compared to GPT-4. However, the model still faces challenges in generating perfect CTL formulas, particularly for requirements with complex or nested logical structures. Yet, the percentage of ExacM for the BERT-GPT approach is not optimal. ExacM failures suggest that while the BERT-GPT integration improves the model's ability to interpret NL, it struggles with highly specific syntactic constraints.

Like other approaches, the BERT-GPT hybrid struggled with temporal operator distinctions, particularly in the ExacMTOD and EquivMTOD categories. Despite BERT's pattern identification assisting in some cases, challenges with the precise application of temporal logic remain, requiring further refinement.

The PC and NC metrics point to areas where the model's performance falls short, particularly in translating complex or less common logical structures. PC translations suggest that the model captures part of the logic but omits critical elements or uses inappropriate logical operators. This may occur when the model fails to fully understand conditional relationships, dependencies, or sequences of events.

NC translations, on the other hand, indicate a complete failure to grasp the required logical structure. These errors often arise in cases where the NL description includes abstract conditions, rare logical constructs, or domain-specific terminology that the model has not been adequately trained to handle. The model may also struggle with understanding higher-order logical constructs, such as deeply nested quantifiers or multiple layers of conditionals.

The integration of BERT into the model's architecture is aimed at improving pattern recognition by categorizing NL inputs into specific CTL patterns before GPT performs the translation. This step helps guide the model toward more accurate CTL formulations. However, the result analysis shows that this approach has limitations when dealing with highly nuanced or context-dependent logical structures. For example, BERT's pattern recognition might correctly identify a common pattern, but GPT might struggle with applying the appropriate CTL operators in cases where the temporal and logical dependencies deviate from the pattern.

The semantic similarity score of 0.77 and SOJS score of 0.73 indicate that the BERT-GPT hybrid model captures both the logical meaning and the structural integrity of the CTL specifications quite well. These results demonstrate that pattern recognition can significantly

enhance the translation process, particularly for complex logical structures. The model's balance of precision and flexibility makes it a strong candidate for use in formal verification workflows, especially when exact syntactic matching is not strictly required, but logical accuracy is paramount.

9.8.4 Overall Insights

The comparative analysis of these approaches highlights a key trade-off between syntactic precision and semantic understanding. The Mistral model's high accuracy underscores the potential of fine-tuned models for formal verification tasks, where maintaining both syntactic and semantic integrity is crucial. GPT-4's flexibility demonstrates the value of general-purpose models in translating a wide range of NL requirements, though its lack of syntactic precision limits its applicability in high-stakes verification. Meanwhile, the BERT-GPT hybrid shows promise as a middle ground, combining pattern recognition with generative capabilities to strike a balance between precision and generalization.

These results expand on our previous findings in Chapter 8, where we fine-tuned the Mistral model and optimized it for the NL-to-CTL translation task. In this study, we validated the robustness of the fine-tuned Mistral model while also comparing it with GPT-4's few-shot learning and the BERT-GPT hybrid approach. The results confirm that while the Mistral model achieves syntactic precision, no single approach is universally optimal for all formal verification tasks. GPT-4 and the BERT-GPT hybrid offer greater flexibility, making them suitable for cases where logical correctness is key, but they still require expert oversight to ensure accuracy. In safety-critical systems, where exact syntactic precision is vital, fine-tuned models like Mistral remain superior, though human verification is still essential to ensure reliability across different system requirements.

9.9 Addressing Challenges and Limitations

The performance of the different approaches reveals specific challenges, particularly in handling temporal operators, achieving syntactic precision, and dealing with nuanced logical structures inherent in formal verification. These challenges underscore the importance of continued refinement and model development to fully realize the potential of NL-to-CTL automation in high-stakes formal verification tasks.

9.9.1 Challenges in Temporal Operator Handling

One of the most persistent challenges observed across all models is the accurate handling of temporal operators. CTL's temporal operators, such as AF (for all paths eventually), AX (for all paths next), and AG (for all paths globally), require precise application to ensure that system behaviours are correctly modelled. However, the variability in how time-related behaviours are expressed in NL introduces significant complexity to this task.

Across the models, particularly the Mistral and BERT-GPT hybrid, a high proportion of matches fell under the ExacMTOD and EquivMTOD categories. These classifications highlight the models' ability to capture the general temporal intent but reveal difficulties in consistently applying the correct operators, such as distinguishing between F (eventually) and X (next).

The consequences of these misapplications can be critical in formal verification, where even minor differences in operator selection can lead to incorrect system behaviour. Further refinement, particularly in how models are trained to handle time-sensitive requirements, is necessary to improve the precision of NL-to-CTL translations, especially in safety-critical systems.

1. **Ambiguity in NL Descriptions**
   NL often presents temporal concepts in informal or ambiguous terms, which the models must interpret to apply the appropriate CTL operators. Phrases such as "eventually," "next," or "globally" may imply different temporal behaviours depending on the context. For instance, "eventually" could map to AF (for all paths eventually) or AX (for all paths next), depending on whether the event is expected to occur immediately or at some unspecified future point.

   This ambiguity often results in incorrect temporal operator selection, as evidenced by the high percentage of matches classified under EquivMTOD and ExacMTOD across the Mistral and GPT-4 models. While these errors do not always alter the logical intent of the CTL specification, they introduce syntactic inconsistencies that could affect the reliability of formal verification in time-sensitive systems.

2. **Subtle Differences in Temporal Semantics**
   In CTL, small differences in temporal operators can lead to significant variations in system behaviour. For example, AF (for all paths eventually) and AX (for all paths next) differ in how they treat the progression of time, with AF allowing for eventual occurrence and AX indicating an immediate transition. Misinterpreting such distinc-

tions during NL-to-CTL translation can result in logically incorrect specifications, even though the general intent remains intact.

The tendency of models, especially GPT-4, to generate logically equivalent formulas with temporal operator variations suggests that they often grasp the broad temporal intent but struggle with the finer details required for precise formal verification. This limitation is particularly critical in safety-critical systems or high-assurance domains where slight deviations in temporal logic can lead to significant verification errors or failures in system behaviour.

3. **Complex Temporal Dependencies**
   NL requirements often describe implicit temporal dependencies that are not explicitly stated, adding another layer of difficulty. For instance, a statement like *"A must occur after B"* could imply various temporal relationships, such as an immediate next-state transition (AX), eventual occurrence (AF), or conditional dependencies that require more sophisticated interpretation.

   Models like Mistral, the BERT-GPT hybrid, and GPT-4 frequently struggle with such dependencies, which are often absent from standard training datasets. This challenge is reflected in the models' inability to consistently differentiate between subtle time-based conditions in NL, leading to incorrect or partial translations. The difficulty in translating branching-time semantics in CTL, where multiple future paths from a given state must be accounted for, compounds this issue. Path quantifiers like A (for all paths) and E (for some paths) are not always easily inferred from NL, resulting in incorrect temporal mappings.

9.9.2 Syntactic Precision vs. Semantic Understanding

Across all models, a clear trade-off between syntactic precision and semantic understanding emerges. GPT-4's few-shot learning demonstrates considerable flexibility, successfully capturing the broad logical meaning behind NL requirements, but often at the expense of syntactic precision. In contrast, the fine-tuned Mistral model delivers higher accuracy and precision in formal CTL specifications, but its rigid approach limits generalization across diverse contexts. The BERT-GPT hybrid strikes an intermediate balance, leveraging BERT's pattern recognition to improve syntactic accuracy while maintaining flexibility in interpretation. However, none of the models fully achieve both perfect semantic understanding and exact syntactic precision simultaneously. These results highlight the need to tailor model selection based on the specific requirements of the formal verification task at hand.

1. **Partially Correct and Incorrect Translations**
   The models' struggles with syntactic precision are most evident in the high percentage of PC and NC translations. In many cases, models captured fragments of the logical structure but failed to fully assemble the complete CTL specification, particularly for complex or abstract requirements. This gap between capturing intent and achieving syntactic accuracy is especially pronounced in requirements involving nested or domain-specific constructs, where precise operator selection and logical consistency are crucial.

   For instance, the models often fail to apply the correct CTL operators in requirements involving conditional logic or domain-specific terminology, leading to incorrect translations. These errors indicate that while the models can handle common or straightforward temporal logic patterns, they struggle with more nuanced or less frequent logical structures.

2. **Flexibility vs. Precision: A Trade-Off**
   The performance differences between the models highlight the trade-off between flexibility and syntactic precision. GPT-4's few-shot learning demonstrates considerable flexibility in interpreting a range of NL inputs, but its reliance on equivalence and temporal operator substitution limits its ability to produce exact CTL matches. The model's semantic understanding is strong, but it consistently falls short in cases where rigid syntactic structures are required, such as in safety-critical formal verification tasks.

   In contrast, the Mistral model, which was fine-tuned for NL-to-CTL translation, delivers higher accuracy and better syntactic precision. However, its more rigid approach limits its generalization across diverse logical structures and contexts. The BERT-GPT hybrid strikes a balance between these two extremes, leveraging pattern recognition to improve syntactic accuracy while maintaining a degree of flexibility. However, it too struggles with more complex, nuanced logical constructs.

9.9.3 Human Oversight and the Role of Model-Driven Engineering in Formal Verification

Given the challenges in handling temporal operators and ensuring syntactic precision, human oversight remains indispensable in the NL-to-CTL translation process. While AI models like Mistral, GPT-4, and BERT-GPT hybrid offer significant potential for automating the generation of CTL formulas, their outputs are not yet reliable enough for unsupervised use, particularly in high-stakes formal verification tasks such as those involving safety-critical systems or regulatory compliance.

In our previous work [116], we focused on fine-tuning the Mistral model, demonstrating its effectiveness in automating the NL-to-CTL translation task. This study not only validates those findings but also extends them by comparing Mistral with GPT-4 and BERT-GPT hybrid approaches, shedding light on the potential for hybrid models in balancing flexibility and precision. These additional insights underscore the evolving role of AI models in formal verification, highlighting that while models like Mistral provide high accuracy, hybrid approaches offer new pathways for integrating human expertise with automated tools.

The integration of AI models within the broader framework of Model-Driven Engineering (MDE) [117] has the potential to significantly enhance formal verification processes. In MDE, automated tools help generate formal specifications and perform verification to ensure that systems behave as expected. The automation of NL-to-CTL translation represents a critical component of this model-driven verification. AI models can greatly reduce the time and effort required to produce formal specifications, allowing for more frequent and scalable verification throughout the software development lifecycle.

The findings of this study demonstrate that while AI models show promise in automating NL-to-CTL translation, they cannot yet function independently in formal verification workflows. In high-precision tasks, especially in safety-critical systems or regulatory compliance, human oversight is indispensable. AI models serve as valuable tools to speed up the verification process, but the accuracy, reliability, and syntactic correctness of the CTL formulas depend on expert human review and refinement. The collaboration between AI-driven automation and human experts ensures that formal specifications meet the rigorous standards necessary for safety and reliability.

By automating the initial translation of NL requirements into CTL, AI models can significantly speed up the formal verification process, enabling rapid prototyping and early detection of specification errors. However, the complexity and high assurance required in formal methods demand human intervention to guarantee the precision and reliability of the final specifications. In this hybrid approach, AI-driven automation helps scale the verification process, while human expertise ensures the high standards necessary for systems requiring rigorous formal verification.

The effective integration of AI models within MDE can transform formal verification by enhancing scalability and efficiency. Nonetheless, human oversight remains crucial to bridge the gap between automated translation and the nuanced understanding required for high-stakes formal verification. This partnership between AI models and human experts ensures that the generated CTL specifications meet the rigorous demands of safety-critical and regulatory-compliant systems.

9.9.4 Improving Training and Fine-Tuning Strategies

The limitations identified in this study highlight the need for more targeted training and fine-tuning strategies to enhance the models' performance. Expanding the training datasets to include more complex examples of temporal logic, nested logical constructs, and conditional dependencies is essential. Datasets that focus on time-sensitive operations, real-time systems, and domain-specific logic would better prepare the models to handle the nuances that current training corpora lack.

In summary, while the models evaluated in this study show significant promise for automating NL-to-CTL translation, substantial challenges remain, particularly in handling temporal operators and achieving syntactic precision. Addressing these limitations will require further refinement of training and fine-tuning strategies, alongside continued human oversight to ensure the reliability of the generated CTL specifications. As AI models become more integral to formal verification processes, balancing flexibility and precision will be critical in leveraging their full potential in high-assurance systems.

9.10 Chapter Summary

This chapter investigated alternative methodologies for translating NL requirements into CTL specifications, introducing few-shot learning with GPT-4 and a hybrid BERT-GPT approach to complement the fine-tuned Mistral model. While Mistral achieved the highest accuracy and structural fidelity, the hybrid model demonstrated a balanced trade-off between precision and flexibility, and GPT-4's few-shot learning highlighted adaptability with limitations in syntactic precision. These methodologies were evaluated using metrics such as accuracy, semantic similarity, and SOJS, with case studies confirming the practical applicability of the generated CTL formulas for formal verification. The next chapter will conclude this thesis by summarizing key contributions and presenting perspectives for future research.

CHAPTER 10 CONCLUSION AND PERSPECTIVES

In this last part of the thesis, we conclude and summarize our findings. In addition, we will discuss the limitations of our studies and the directions for future work.

10.1 Summary of Works

This thesis presents a comprehensive approach to enhancing DO-178C compliance through the integration of formal methods, CNL, and AI-driven solutions for translating requirements from NL to CTL. Each contribution addresses a distinct challenge identified in the early stages of this research, offering new tools, frameworks, and insights that streamline the compliance and verification processes in aerospace software development.

1. **Integrating formal methods with automated verification tools**

   This work presents a unified approach that integrates formal methods, such as model checking, with automated verification tools like the LDRA suite, specifically tailored for DO-178C compliance. By implementing this integration in a UAV collision avoidance case study, the thesis demonstrates how formal verification and automated tools can reduce the manual effort required for certification while maintaining rigorous standards. This contribution exemplifies a practical solution for managing complex verification tasks, particularly in safety-critical UAV applications, enhancing both the efficiency and reliability of compliance processes.

2. **Development of a CNL for requirements specification**

   To address the common issue of ambiguity in NL requirements, this thesis develops a CNL tailored for DO-178C requirements specification. The CNL framework incorporates structured linguistic constraints to clarify requirement statements, reducing misinterpretation and improving verifiability. Accompanying this framework is a rule-based verification system that ensures CNL requirements adhere to DO-178C standards. Together, these tools provide a streamlined approach for managing complex requirements, enhancing traceability and fostering a more rigorous requirements engineering process in safety-critical software.

3. **Creation of the Natural2CTL dataset for NL-to-CTL translation** To facilitate the translation of NL requirements into formal representations, this thesis introduces the Natural2CTL dataset—a curated collection of NL-CTL pairs specifically designed

to train AI models for NL-to-formal language translation. This dataset represents a foundational resource that bridges the gap between human-readable requirements and machine-interpretable specifications. Its creation involved extensive annotation and validation processes to ensure its applicability in real-world contexts, thus supporting AI-based formal verification efforts and paving the way for broader automation in certification tasks.

4. **Fine-tuning of LLMs for NL-to-CTL translation**

   Leveraging the Natural2CTL dataset, this thesis explores the fine-tuning of advanced LLMs, such as Mistral, for the domain-specific task of translating NL requirements into CTL. Fine-tuning these models has significantly improved their accuracy in handling domain-specific language and requirements intricacies, offering a powerful tool for automated formal verification. This contribution highlights how tailored AI models can enhance translation precision, reducing the need for manual intervention in the formal verification process, and presents an innovative method for automating complex compliance tasks in safety-critical software.

5. **Evaluation of alternative AI-based translation techniques**

   Beyond fine-tuning LLMs, this thesis conducts a comparative analysis of alternative methodologies for NL-to-CTL translation, including few-shot learning with GPT-4 and the hybrid BERT-GPT approach. This investigation provides a comprehensive evaluation of each method's accuracy, scalability, and practical feasibility, offering valuable insights for selecting optimal models and approaches for specific compliance requirements. By comparing these techniques, this contribution supplies aerospace developers with informed options for implementing automated translations in verification workflows, supporting a more adaptable and efficient compliance process.

Together, the contributions of this thesis have significant implications for advancing DO-178C compliance and improving the overall landscape of safety-critical software development in the aerospace industry. By offering reliable methodologies, this research supports a more streamlined, cost-effective certification process, reducing manual verification burdens while enhancing traceability and rigour. The integration of formal methods with automated tools not only facilitates compliance but also improves certification reliability, which is particularly valuable for complex UAV systems that require thorough validation. Additionally, the development of a CNL framework and the Natural2CTL dataset provides a solution to the challenge of translating ambiguous NL requirements into verifiable, formal representations,

thus bridging a critical gap in requirements engineering for aerospace applications. The AI-driven translation models introduced here also pave the way for broader adoption of formal methods, enabling industries that traditionally rely on NL specifications to embrace scalable, automated formal verification. Ultimately, these contributions lay the groundwork for safer, more dependable UAV operations and safety-critical software systems across the aerospace sector and beyond, aligning with industry standards and fostering innovations that advance software reliability and compliance.

10.2 Limitations

Despite the contributions made through this thesis in advancing DO-178C compliance methodologies, several limitations exist that define the boundaries of the research findings and suggest areas for future improvement.

- **Methodological scope of verification Tools**

  While this thesis integrates formal methods with automated verification tools validated through a UAV collision avoidance case study, it primarily emphasizes model checking in conjunction with tools like the LDRA suite. Other formal verification techniques, such as theorem proving, fall outside this scope, potentially limiting the findings' applicability to different types of aerospace systems. Further research integrating alternative formal methods could diversify the impact on a broader range of systems.

- **Generalizability of case studies**

  The approach was validated through specific case studies, including the UAV collision avoidance system. Although these case studies illustrate practical applications, the results may not generalize across all aerospace applications or broader safety-critical systems. Additional studies involving varied scenarios and system complexities would be valuable to confirm the wider applicability of these methodologies.

- **Limitations in CNL**

  The CNL framework offers a structured approach for reducing ambiguity in NL requirements, which improves verifiability. However, this structured format inherently restricts linguistic flexibility, which may present challenges for complex, nuanced requirements in broader aerospace applications. Additionally, while the rule-based verification system is tailored to DO-178C, its adaptability to other standards may be limited, necessitating further refinement to address the needs of other certification frameworks.

- **Scope and snnotation of the Natural2CTL dataset**

  While the Natural2CTL dataset provides a foundational resource for AI-driven NL-to-CTL translations, it has limitations in size and scope. The manual annotation of over 2,095 NL-CTL pairs, though carefully executed, may still carry some translation ambiguities and is modest in comparison to large-scale datasets typically used in machine learning. Expanding this dataset could support a more robust foundation for ML applications, particularly those requiring large data volumes for high accuracy.

- **Challenges in AI model fine-tuning and interpretability**

  Fine-tuned models such as Mistral have shown improved accuracy for NL-to-CTL translation, but challenges remain in balancing syntactic precision with semantic understanding. Domain-specific language complexities necessitate fine-grained tuning, which can introduce interpretability concerns and potential biases in outputs. Additionally, while alternative models (e.g., few-shot GPT-4 and BERT-GPT hybrids) were explored, limitations persist in handling complex logical structures. Future work could focus on enhancing both accuracy and transparency to increase the reliability of AI outputs in safety-critical applications.

- **Manual analysis and participant constraints**

  Some conclusions in Chapters 8 and 9 were drawn from manual analysis of AI model outputs, with accuracy assessments partially relying on expert evaluation. The limited size of the test set, constrained by the Natural2CTL dataset's capacity, presents a limitation in generalizability. Future studies could strengthen these findings by expanding the test dataset, enabling more robust statistical conclusions. Additionally, the comparison between human non-expert performance and AI models would benefit from a larger participant pool and the inclusion of domain experts, both of which would increase the reliability and validity of the comparative results.

10.3 Opportunities for Future Research

This thesis advances DO-178C compliance through integrative methodologies and technological solutions. While significant progress has been made, our thesis findings open a wide range of opportunities for future work. In the future, we plan to extend our study in the following directions:

- **Dataset expansion and generalization**
  The Natural2CTL dataset is a valuable resource for training LLMs on NL-to-formal

logic translation. However, expanding this dataset to include additional types of requirements, spanning multiple domains or different levels of complexity, could significantly enhance its applicability. Future work could also focus on generating a larger, more varied dataset to better support machine learning training and increase the generalizability of translation models.

- **Impact of CNL on LLMs**
  Investigating how the use of CNL affects the performance, accuracy, and consistency of LLM outputs in the context of translating requirements into formal specifications. This research could include systematic comparisons between CNL-enhanced and non-enhanced LLM translations, especially focusing on their implications for automated verification. Further research might also extend the CNL to encompass complex, nuanced requirements to evaluate its scalability and adaptability for broader industry applications.

- **Refining AI models for formal verification**
  Future studies could further investigate fine-tuning and model refinement strategies that enhance the interpretability, reliability, and robustness of LLM outputs for NL-to-CTL translation. Research might explore hybrid models combining LLMs with rule-based or ontology-driven approaches for greater precision and contextual accuracy in translating complex requirements.

- **User-centered studies on expert and non-expert collaboration with AI tools**
  Another future research direction is to conduct user studies involving both domain experts and non-experts to evaluate how effectively AI models and automated tools can be incorporated into verification workflows. Studies could compare the performance and usability of AI-driven tools when used by different levels of expertise and identify factors that may improve or hinder collaboration with automated systems.

- **Automated verification for non-conformities**
  Expanding the role of AI in DO-178C compliance to include automated detection of non-conformities early in the development cycle is an important avenue for future research. Machine learning models trained on requirement traceability data could identify potential risks due to code changes or requirement misalignment. This approach could streamline certification processes by flagging traceability or risk-related issues before they propagate.

- **Ethical and regulatory implications of AI in safety-critical systems**
  As AI tools become more integral to certification and compliance, there is a grow-

ing need to explore their ethical and regulatory implications. Research could focus on establishing ethical frameworks that mitigate risks of AI bias, ensure transparency in AI-driven decision-making, and align AI processes with stringent regulatory standards. These frameworks would support the responsible integration of AI into formal verification tasks and ensure adherence to safety-critical standards like DO-178C.

By addressing these areas, future research can continue to enhance DO-178C compliance, making the process more efficient, scalable, and adaptable to new challenges in safety-critical software systems.

REFERENCES

[1] RTCA, *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc. Std., 2012, rTCA/DO-178C. [Online]. Available: https://www.rtca.org

[2] B. Brosgol and C. Comar, "Do-178c: A new standard for software safety certification," in *Presented at the 22nd Systems and Software Technology Conference (SSTC)*, vol. 26, 2010, p. 29.

[3] C. Torens, F.-M. Adolf, and L. Goormann, "Certification and software verification considerations for autonomous unmanned aircraft," *Journal of Aerospace Information Systems*, vol. 11, no. 10, pp. 649–664, 2014.

[4] M. Saleab *et al.*, "Low-level memory and timing analysis of flight code for unmanned aerial systems," *Aerospace Systems*, vol. 7, no. 2, pp. 209–225, 2024.

[5] J. Pyrgies, "Towards do-178c certification of adaptive learning uav agents designed with a cognitive architecture," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering: Companion proceedings*, 2020, pp. 174–177.

[6] A. Paz and G. E. Boussaidi, "Building a software requirements specification and design for an avionics system: An experience report," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1262–1271.

[7] U. S. Shah and D. C. Jinwala, "Resolving ambiguities in natural language software requirements: a comprehensive survey," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 5, pp. 1–7, 2015.

[8] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[9] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM (JACM)*, vol. 12, no. 1, pp. 23–41, 1965.

[10] E. M. Clarke *et al.*, *Handbook of model checking*. Springer, 2018, vol. 10.

[11] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," *25 Years of Model Checking: History, Achievements, Perspectives*, pp. 196–215, 2008.

[12] LDRA Technology, Inc., *LDRA Tool Suite*, Wirral, United Kingdom, 2024, software testing and verification tool. [Online]. Available: https://ldra.com

[13] J. C. Santos, A. Shokri, and M. Mirakhorli, "Towards automated evidence generation for rapid and continuous software certification," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020, pp. 287–294.

[14] C. B. Harris and I. G. Harris, "Generating formal hardware verification properties from natural language documentation," in *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. IEEE, 2015, pp. 49–56.

[15] A. Q. Jiang *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[16] International Organization for Standardization, "Conformity assessment," https://www.iso.org/conformity-assessment.html, accessed: June 13, 2022.

[17] Federal Aviation Administration, "National airspace system (nas)," https://www.faa.gov/air_traffic/nas, accessed: May 31, 2022.

[18] ——, "Code of federal regulations, title 14—aeronautics and space," https://www.ecfr.gov/current/title-14, 2022, accessed: June 13, 2022.

[19] RTCA, *DO-254: Design Assurance Guidance for Airborne Electronic Hardware*, RTCA Std., 2000.

[20] ——, *DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*, RTCA Std., 2000.

[21] Federal Aviation Administration, *FAA Order 8130.34D: Airworthiness Certification of Unmanned Aircraft Systems and Optionally Piloted Aircraft*, U.S. Department of Transportation, 2017, accessed: October 22, 2024. [Online]. Available: https://www.faa.gov/documentLibrary/media/Order/FAA_Order_8130.34D.pdf

[22] J. M. Wing, "A specifier's introduction to formal methods," *Computer*, vol. 23, no. 9, pp. 8–22, 1990.

[23] M. Thomas, "The role of formal methods in achieving dependable software," *Reliability Engineering & System Safety*, vol. 43, no. 2, pp. 129–134, 1994.

[24] D. W. Loveland, *Automated theorem proving: A logical basis*. Elsevier, 2016.

[25] E. M. Clarke, "Model checking," in *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*. Springer, 1997, pp. 54–56.

[26] A. Pnueli, "The temporal logic of programs," in *18th annual symposium on foundations of computer science (sfcs 1977)*. ieee, 1977, pp. 46–57.

[27] E. A. Emerson and J. Y. Halpern, ""sometimes" and "not never" revisited: on branching versus linear time temporal logic," *Journal of the ACM (JACM)*, vol. 33, no. 1, pp. 151–178, 1986.

[28] E. Clarke *et al.*, "Symbolic model checking," in *Computer Aided Verification: 8th International Conference, CAV'96 New Brunswick, NJ, USA, July 31–August 3, 1996 Proceedings 8*. Springer, 1996, pp. 419–422.

[29] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 5, pp. 1512–1542, 1994.

[30] B. Potter, J. Sinclair, and D. Till, *An introduction to formal specification and Z*. Prentice-Hall, Inc., 1992.

[31] J. M. Spivey, *Understanding Z: a specification language and its formal semantics*. Cambridge University Press, 1988, vol. 3.

[32] V. Alagar and K. Periyasamy, "The b-method," in *Specification of Software Systems*. Springer, 2011, pp. 577–633.

[33] D. Jackson, I. Schechter, and H. Shlyahter, "Alcoa: the alloy constraint analyzer," in *Proceedings of the 22nd international conference on Software engineering*, 2000, pp. 730–733.

[34] M. Osborne and C. K. MacNish, "Processing natural language software requirement specifications," in *Proceedings of the Second International Conference on Requirements Engineering*. IEEE, 1996, pp. 229–236.

[35] T. Bures *et al.*, "Requirement specifications using natural languages," *Technical Report D3S-TR-2012-05*, 2012.

[36] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, no. 1, pp. 1–30, 1997.

[37] A. Wyner *et al.*, "On controlled natural languages: Properties and prospects," in *International workshop on controlled natural language*. Springer, 2009, pp. 281–289.

[38] L. Mich, "Nl-oops: from natural language to object oriented requirements using the natural language processing system lolita," *Natural language engineering*, vol. 2, no. 2, pp. 161–187, 1996.

[39] N. E. Fuchs and R. Schwitter, "Attempto controlled english (ace)," *arXiv preprint cmp-lg/9603003*, 1996.

[40] T. Kuhn, "A survey and classification of controlled natural languages," *Computational linguistics*, vol. 40, no. 1, pp. 121–170, 2014.

[41] A. Fatwanto, "Software requirements specification analysis using natural language processing technique," in *2013 International Conference on QiR*. IEEE, 2013, pp. 105–110.

[42] K. B. Cohen and A. Dolbey, "Foundations of statistical natural language processing," *Language*, vol. 78, no. 3, pp. 599–599, 2002.

[43] S. Sun, C. Luo, and J. Chen, "A review of natural language processing techniques for opinion mining systems," *Information fusion*, vol. 36, pp. 10–25, 2017.

[44] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[45] D. Brown *et al.*, "Guidance for using formal methods in a certification context," in *Embedded Real Time Software and Systems Conference*, no. NF1676L-10457, 2010.

[46] M. Webster *et al.*, "Formal methods for the certification of autonomous unmanned aircraft systems," in *Computer Safety, Reliability, and Security: 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings 30*. Springer, 2011, pp. 228–242.

[47] D. Cofer and S. Miller, "Do-333 certification case studies," in *NASA Formal Methods: 6th International Symposium, NFM 2014, Houston, TX, USA, April 29–May 1, 2014. Proceedings 6*. Springer, 2014, pp. 1–15.

[48] D. Cofer and S. P. Miller, "Formal methods case studies for do-333," Tech. Rep., 2014.

[49] K. Dmitriev *et al.*, "A lean and highly-automated model-based software development process based on do-178c/do-331," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–10.

[50] N. Metayer, A. Paz, and G. El Boussaidi, "Modelling do-178c assurance needs: A design assurance level-sensitive dsl," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019, pp. 338–345.

[51] A. Paz and G. El Boussaidi, "On the exploration of model-based support for do-178c-compliant avionics software development and certification," in *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2016, pp. 229–236.

[52] P. Panchal *et al.*, "Comprehensive overview of a process-oriented build tool for airborne safety-critical software development," in *2023 10th International Conference on Recent Advances in Air and Space Technologies (RAST)*. IEEE, 2023, pp. 01–06.

[53] L. Bao, C. Fuhrman, and R. Landry, "Certification considerations of software-defined radio using model-based development and automated testing," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*. IEEE, 2023, pp. 1–8.

[54] M. Webster *et al.*, "Generating certification evidence for autonomous unmanned aircraft using model checking and simulation," *Journal of Aerospace Information Systems*, vol. 11, no. 5, pp. 258–279, 2014.

[55] J. Pyrgies, D. Gigan, and R. Haelterman, "An innovative approach for achieving do-178c certification of an intelligent system implementing sense-and-avoid function in uavs," in *Air Transport Research Society World Conference*, 2017.

[56] J. E. F. Ribeiro, J. G. Silva, and A. Aguiar, "Weaving agility in safety-critical software development for aerospace: from concerns to opportunities," *IEEE Access*, 2024.

[57] A. Umber and I. S. Bajwa, "Minimizing ambiguity in natural language software requirements specification," in *2011 Sixth International Conference on Digital Information Management*. IEEE, 2011, pp. 102–107.

[58] A. Veizaga *et al.*, "On systematically building a controlled natural language for functional requirements," *Empirical Software Engineering*, vol. 26, no. 4, p. 79, 2021.

[59] I. Darif *et al.*, "A model-driven and template-based approach for requirements specification," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2023, pp. 239–249.

[60] A. Condamines and M. Warnier, "Towards the creation of a cnl adapted to requirements writing by combining writing recommendations and spontaneous regularities: example

in a space project," *Language Resources and Evaluation*, vol. 51, no. 1, pp. 221–247, 2017.

[61] Z. Zhao *et al.*, "Automated generating natural language requirements based on domain ontology," *arXiv preprint arXiv:2211.16716*, 2022.

[62] S. Vinay, S. Aithal, and P. Desai, "An nlp based requirements analysis tool," in *International Advance Computing Conference. IEEE*, 2009.

[63] D. de Almeida Ferreira and A. R. da Silva, "A controlled natural language approach for integrating requirements and model-driven engineering," in *2009 Fourth International Conference on Software Engineering Advances.* IEEE, 2009, pp. 518–523.

[64] A. K. Karna *et al.*, "The role of model checking in software engineering," *Frontiers of Computer Science*, vol. 12, pp. 642–668, 2018.

[65] I. Buzhinsky, "Formalization of natural language requirements into temporal logics: a survey," in *2019 IEEE 17th international conference on industrial informatics (INDIN)*, vol. 1. IEEE, 2019, pp. 400–406.

[66] F.-L. Li *et al.*, "From stakeholder requirements to formal specifications through refinement," in *Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015, Essen, Germany, March 23-26, 2015. Proceedings 21.* Springer, 2015, pp. 164–180.

[67] A. Brunello, A. Montanari, and M. Reynolds, "Synthesis of ltl formulas from natural language texts: State of the art and research directions," in *26th International symposium on temporal representation and reasoning (TIME 2019).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[68] S. Ghosh *et al.*, "Specnfs: a challenge dataset towards extracting formal models from natural language specifications," in *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, 2022, pp. 2166–2176.

[69] R. L. Smith, G. S. Avrunin, and L. A. Clarke, "From natural language requirements to rigorous property specifications," in *Workshop on Software Engineering for Embedded Systems (SEES 2003): From Requirements to Implementation*, 2003, pp. 40–46.

[70] S. Ghosh *et al.*, "Automatically extracting requirements specifications from natural language," *arXiv preprint arXiv:1403.3142*, 2014.

[71] Y. Wu *et al.*, "Autoformalization with large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32 353–32 368, 2022.

[72] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "Using llms to facilitate formal verification of rtl," *arXiv e-prints*, pp. arXiv–2309, 2023.

[73] M. Cosler *et al.*, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," *arXiv preprint arXiv:2303.04864*, 2023.

[74] J. X. Liu *et al.*, "Lang2ltl: Translating natural language commands to temporal specification with large language models," in *Workshop on Language and Robotics at CoRL 2022*, 2022.

[75] C. Hahn and C. Trippel, "Translating natural language to temporal logics with large language models and model checkers," in *PROCEEDINGS OF THE 24TH CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN–FMCAD 2024*. TU Wien Academic Press, 2024, p. 119.

[76] F. Aditi and M. S. Hsiao, "Hybrid rule-based and machine learning system for assertion generation from natural language specifications," in *2022 IEEE 31st Asian Test Symposium (ATS)*. IEEE, 2022, pp. 126–131.

[77] S. Thukral, K. Kukreja, and C. Kavouras, "Probing language models for understanding of temporal expressions," *arXiv preprint arXiv:2110.01113*, 2021.

[78] Y. Chen *et al.*, "Nl2tl: Transforming natural languages to temporal logics using large language models," *arXiv preprint arXiv:2305.07766*, 2023.

[79] *MATLAB Simulink User's Guide*, The MathWorks, Inc., Natick, Massachusetts, United States, 2023, https://www.mathworks.com/products/simulink.html.

[80] Alloy Developers, "Alloy specification language," 2024, accessed: 2024-11-05. [Online]. Available: https://alloytools.org

[81] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[82] D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[83] J. P. Bowen, "Z: A formal specification notation," in *Software Specification Methods: An Overview Using a Case Study*. Springer, 2001, pp. 3–19.

[84] J.-R. Abrial, A. Hoare, and P. Chapron, "The b-book," *(No Title)*, 1996.

[85] NASA ACCESS 5, "Collision avoidance functional requirements for step 1," NASA Dryden Flight Research Center, Tech. Rep., February 2006, revision 6.

[86] P. H. Feiler *et al.*, "An overview of the sae architecture analysis & design language (aadl) standard: A basis for model-based architecture-driven embedded systems engineering," in *IFIP World Computer Congress, TC 2*. Springer, 2004, pp. 3–15.

[87] P. Feiler, "Open source aadl tool environment (osate)," in *AADL Workshop, paris*, 2004, pp. 1–40.

[88] Aeronautical Radio, Incorporated (ARINC), *ARINC 653: Avionics Application Software Standard Interface*, Std., 1997, prepared by the Airlines Electronic Engineering Committee (AEEC).

[89] *MISRA C:2012 Guidelines for the use of the C language in critical systems*, MISRA (Motor Industry Software Reliability Association), 2013, iSBN: 978-1-906400-10-1, Available at: https://www.misra.org.uk/.

[90] A. Biere, "Bounded model checking," in *Handbook of satisfiability*. IOS press, 2021, pp. 739–764.

[91] S. R. Hirshorn, L. D. Voss, and L. K. Bromley, "Nasa systems engineering handbook," Tech. Rep., 2017.

[92] I. C. S. S. E. T. Committee, *IEEE Guide for Developing System Requirements Specifications*. IEEE, 1998.

[93] J. Reback *et al.*, "pandas-dev/pandas: Pandas 1.0. 5," *Zenodo*, 2020.

[94] Y. Vasiliev, *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020.

[95] C. Chapman and K. T. Stolee, "Exploring regular expression usage and context in python," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 282–293.

[96] J. Luo *et al.*, "Pint: a modern software package for pulsar timing," *The Astrophysical Journal*, vol. 911, no. 1, p. 45, 2021.

[97] language-tool-python contributors, "language-tool-python: A python wrapper for languagetool," 2024, accessed: 2024-11-05. [Online]. Available: https://pypi.org/project/language-tool-python/

[98] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *2017 IEEE 25th international requirements engineering conference (RE)*. IEEE, 2017, pp. 502–505.

[99] "CM1/Requirements Tracing," University of Ottawa, 2015. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html

[100] S. F. Tjong, "Avoiding ambiguity in requirements specifications," *Faculty of Engineering & Computer Science*, 2008.

[101] E. Masuoka *et al.*, "Modis. volume 1: Modis level 1a software baseline requirements," Tech. Rep., 1994.

[102] T. Diamantopoulos *et al.*, "Software requirements as an application domain for natural language processing," *Language Resources and Evaluation*, vol. 51, pp. 495–524, 2017.

[103] "Software requirements dataset," Kaagle, 2020. [Online]. Available: https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset?datasetId=560206&sortBy=dateRun&tab=collaboration

[104] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Property specification patterns for finite-state verification," in *Proceedings of the second workshop on Formal methods in software practice*, 1998, pp. 7–15.

[105] ——, "Patterns in property specifications for finite-state verification," in *Proceedings of the 21st international conference on Software engineering*, 1999, pp. 411–420.

[106] K. Krippendorff, "Computing krippendorff's alpha-reliability," 2011.

[107] G. Behrmann *et al.*, "Uppaal 4.0," 2006.

[108] R. Zrelli *et al.*, "Natural2ctl: A dataset for natural language requirements and their ctl formal equivalents," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2024, pp. 205–216.

[109] N. Reimers, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[110] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[111] OpenAI, "Gpt-4 api pricing," 2024, accessed: 2024-09-12. [Online]. Available: https://openai.com/pricing

[112] Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[113] V. Sanh, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[114] K. Clark, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[115] R. Cavada *et al.*, "Nusmv 2.4 user manual," *CMU and ITC-irst*, vol. 11, pp. 22–45, 2005.

[116] R. Zrelli *et al.*, "Advancing formal verification: Fine-tuning llms for translating natural language requirements to ctl specifications," in *Proceedings of the 35th International Workshop on Rapid System Prototyping*, 2024, accepted, to appear.

[117] S. Kent, "Model driven engineering," in *International conference on integrated formal methods*. Springer, 2002, pp. 286–298.

APPENDIX A CAS SOFTWARE REQUIREMENTS SPECIFICATION AND DESIGN

SRATS_001 *Detect Traffic*: The Collision Avoidance System shall detect traffic within its surveillance volume.
Rationale: Effective traffic detection is the first step in collision avoidance, ensuring that potential threats are identified early.
Category: Functional
Traceability: Developed into HLR_001

SRATS_002 *Detect Traffic*: The Collision Avoidance System shall calculate its surveillance volume depending on all the following conditions:

- detection range;
- azimuth field of regard;
- elevation field of regard.

Rationale: Defines the area within which traffic detection is performed.
Category: Functional
Traceability: Developed into HLR_001

SRATS_003 *Detect Traffic*: The Collision Avoidance System shall detect cooperative traffic at a range of at least 20 nautical miles.
Rationale: Ensures the system can detect cooperative traffic, which is essential for collision avoidance.
Category: Functional
Traceability: Developed into HLR_002

SRATS_004 *Detect Traffic*: The Collision Avoidance System shall detect cooperative traffic within an azimuth FOR of at least +/-110° referenced from the flight path of the UA.
Rationale: Ensures wide horizontal coverage for detecting traffic.
Category: Functional
Traceability: Developed into HLR_003

SRATS_005 *Detect Traffic*: The Collision Avoidance System shall detect cooperative traffic within an elevation FOR of at least +/-15° referenced from the flight path of the UA.
Rationale: Ensures vertical coverage for detecting traffic.
Category: Functional
Traceability: Developed into HLR_003

SRATS_006 *Detect Traffic*: The average Collision Avoidance System detection rate shall be equal to or greater than 1.0 hertz.
Rationale:A sufficient detection rate is necessary to ensure timely updates and responses to detected traffic.
Category: Functional
Traceability: Developed into HLR_004

SRATS_007 *Track Traffic*: The Collision Avoidance System shall track the detected traffic.
Rationale: Accurate tracking of traffic allows for continuous monitoring and assessment of potential collision threats.
Category: Functional
Traceability: Developed into HLR_005

SRATS_008 *Track Traffic*: The Collision Avoidance System shall track cooperative traffic within an azimuth FOR of at least +/-110° and elevation FOR of at least +/-15° referenced from the flight path of the UA.
Category: Functional
Traceability: Developed into HLR_005

SRATS_009 *Evaluate Collision Potential*: The Collision Avoidance System shall evaluate the potential for collision with each tracked traffic element, including the assessment of existing collision threats.
Rationale: Evaluating collision potential helps in identifying imminent threats and preparing the system to take appropriate actions.
Category: Functional
Traceability: Developed into HLR_006

SRATS_010 *Evaluate Collision Potential*: The Collision Avoidance System shall continuously determine if any detected traffic elements pose a collision threat to the vehicle.
Category: Functional
Traceability: Developed into HLR_006

SRATS_011 *Prioritize Collision Threats*: The Collision Avoidance System shall prioritize the traffic posing a collision threat.
Rationale: Prioritizing threats ensures that the most immediate and dangerous threats are addressed first, optimizing response time and effectiveness.
Category: Functional
Traceability: Developed into HLR_007

SRATS_012 *Determine Avoidance Maneuver*: The Collision Avoidance System shall determine an avoidance maneuver that prevents a collision.
Rationale: Determining the correct avoidance maneuver is essential to ensure the safety of the UAV and other airspace users.
Category: Functional
Traceability: Developed into HLR_008

SRATS_013 *Determine Avoidance Maneuver*: The Collision Avoidance System shall revise the maneuver recommendation when other aircraft are simultaneously maneuvering.
Category: Functional
Traceability: Developed into HLR_008

SRATS_014 *Command Maneuver*: The Collision Avoidance System shall command an appropriate avoidance maneuver.
Rationale: Executing the correct maneuver ensures the UAV can avoid collisions effectively.
Category: Functional
Traceability: Developed into HLR_009

SRATS_015 *Sensor Data Integrity*: The Collision Avoidance System shall verify the integrity and accuracy of the sensor data before using it for detection and tracking.
Rationale: Ensuring that the sensor data is reliable and accurate is critical for all subsequent functions of the Collision Avoidance System.
Category: Performance
Traceability: Developed into HLR_010

SRATS_016 *System Self-Test*: The Collision Avoidance System shall perform self-tests on startup and periodically during operation to ensure all components are functioning correctly.

Rationale: Regular self-tests help in identifying and mitigating failures that could compromise the system's ability to detect and avoid collisions.

Category: Safety

Traceability: Developed into HLR_011

SRATS_017 *Redundancy Management*: The Collision Avoidance System shall have redundancy management capabilities to handle failures in primary detection and tracking components.

Rationale: Redundancy ensures that the system remains operational even if some components fail, thus maintaining safety.

Category: Safety

Traceability: Developed into HLR_012

SRATS_018 *Communication with Ground Control*: The Collision Avoidance System shall communicate status and alerts to the ground control station in real-time.

Rationale: Real-time communication with ground control allows for human intervention when necessary and provides situational awareness to operators.

Category: Non-Functional

Traceability: Developed into HLR_013

SRATS_019 *False Alarm Management*: The Collision Avoidance System shall minimize false alarms to prevent unnecessary maneuvers.

Rationale: Reducing false alarms ensures that the system does not execute unnecessary maneuvers, which could be disruptive or hazardous.

Category: Performance

Traceability: Developed into HLR_014

HLR_001 *Traffic Detection*: The Collision Avoidance System shall detect traffic within its surveillance volume. Note: The surveillance volume is defined by three performance characteristics of the sensor: detection range, azimuth field of regard, and elevation field of regard.
Rationale: Ensuring that traffic within the defined surveillance volume is detected is critical for initiating subsequent collision avoidance processes.
Traceability: Developed from SRATS_001 and SRATS_002. Developed into LLR-001, LLR-002, LLR-003, LLR-004, LLR-005, LLR-006, LLR-007, LLR-008, and LLR-009.

HLR_002 *Traffic Detection*: The Collision Avoidance System shall detect cooperative traffic at a range of at least 20 nautical miles.
Rationale: Ensuring the detection of cooperative traffic within a significant range is essential for collision avoidance.
Traceability: Developed from SRATS_003. Developed into LLR-0010.

HLR_003 *Traffic Detection*: The Collision Avoidance System shall detect cooperative traffic within an azimuth field of regard of at least $\pm 110°$ and an elevation field of regard of at least $\pm 15°$, referenced from the flight path of the UAV.
Rationale: Detecting traffic within these fields of regard ensures comprehensive surveillance around the UAV.
Traceability: Developed from SRATS_004 and SRATS_005. Developed into LLR-0011 and LLR-0012.

HLR_004 *Traffic Detection*: The average Collision Avoidance System detection rate shall be equal to or greater than 1.0 hertz.
Rationale: A sufficient detection rate is necessary to ensure timely updates and responses to detected traffic.
Traceability: Developed from SRATS_006. Developed into LLR-0013.

HLR_005 *Traffic Tracking*: The Collision Avoidance System shall track the detected traffic. Note: The track is established when a state estimate is developed with sufficient confidence, and this estimate includes the traffic element's position and velocity vector.
Rationale: Accurate tracking of detected traffic is essential for evaluating potential collision threats and determining appropriate avoidance maneuvers.
Traceability: Developed from SRATS_007 and SRATS_008. Developed into LLR-0014, LLR-0015, LLR-0016, and LLR-0017.

HLR_006 *Collision Evaluation*: The Collision Avoidance System shall evaluate the potential for collision with each traffic element being tracked. This evaluation includes assessing existing collision threats.
Rationale: Evaluating collision potential is necessary to identify imminent threats and prepare the system for appropriate responses.
Traceability: Developed from SRATS_009 and SRATS_010. Developed into LLR-0018 and LLR-0019.

HLR_007 *Threat Prioritization*: The Collision Avoidance System shall prioritize the traffic posing a collision threat. Prioritization is based on ranking the time to collision of the identified threats.
Rationale: Prioritizing threats ensures that the most immediate and dangerous threats are addressed first, optimizing the system's response time and effectiveness.
Traceability: Developed from SRATS_011. Developed into LLR-0020 and LLR-0021.

HLR_008 *Maneuver Determination*: The Collision Avoidance System shall autonomously determine an avoidance maneuver that prevents a collision.
Rationale: Autonomous determination of avoidance maneuvers is critical to ensure timely and effective responses to imminent collision threats.
Traceability: Developed from SRATS_012 and SRATS_013. Developed into LLR-0022 and LLR-0023.

HLR_009 *Maneuver Command*: The Collision Avoidance System shall command an appropriate avoidance maneuver. Note: The commanded maneuver can include initiating a new maneuver, continuing an ongoing maneuver, or terminating an avoidance maneuver if a collision threat no longer exists.
Rationale: Executing the appropriate maneuver ensures the avoidance of collisions based on real-time evaluations of traffic and threats.
Traceability: Developed from SRATS_014. Developed into LLR-0024, LLR-0025, and LLR-0026.

HLR_010 *Sensor Data Integrity*: The CAS shall verify the integrity and accuracy of the sensor data before using it for detection and tracking.
Rationale: Ensuring that the sensor data is reliable and accurate is critical for all subsequent functions of the CAS.
Traceability: Developed from SRATS_015. Developed into LLR-0027.

HLR_011 *System Self-Test*: The CAS shall perform self-tests on startup and periodically during operation to ensure all components are functioning correctly.
Rationale: Regular self-tests help in identifying and mitigating failures that could compromise the system's ability to detect and avoid collisions.
Traceability: Developed from SRATS_016. Developed into LLR-0028 and LLR-0029.

HLR_012 *Redundancy Management*: The CAS shall have redundancy management capabilities to handle failures in primary detection and tracking components.
Rationale: Redundancy ensures that the system remains operational even if some components fail, thus maintaining safety.
Traceability: Developed from SRATS_017. Developed into LLR-0030.

HLR_013 *Communication with Ground Control*: The CAS shall communicate status and alerts to the ground control station in real-time.
Rationale: Real-time communication with ground control allows for human intervention when necessary and provides situational awareness to operators.
Traceability: Developed from SRATS_018. Developed into LLR-0031

HLR_014 *False Alarm Management*: The CAS shall minimize false alarms to prevent unnecessary maneuvers.
Rationale: Reducing false alarms ensures that the system does not execute unnecessary maneuvers, which could be disruptive or hazardous.
Traceability: Developed from SRATS_019. Developed into LLR-0032.

DHLR_008a For each identified collision threat, the CAS shall initiate a corresponding maneuver unless overridden by a higher-priority threat.

DHLR_008b The CAS shall continuously monitor the status of each active maneuver and update or terminate it based on real-time threat assessment.

LLR_001 *Sensor Status Verification and SensorInput Retrieval*: When the SensorStatus is True, the TrafficDetection shall retrieve the SensorInput. Otherwise, the TrafficDetection shall call the Alert function.
Traceability: Developed from HLR-001.

LLR_002 *SensorInput Data Validation*: When SensorInput is received, TrafficDetection shall validate the data as follows: - DetectionRange value is between 0.2 and 3000 meters. -AzimuthFOR value is between -110 and 110 degrees. - ElevationFOR value is between -15 and 15 degrees.
Traceability: Developed from HLR-001.

LLR_003 *Get Orientation*: When SensorInput data is validated, TrafficDetection shall get the UAV orientation using the OrientationData from SensorInput.
Traceability: Developed from HLR-001.

LLR_004 *Get Position*: When SensorInput data is validated, TrafficDetection shall get the current position using the PositionData from SensorInput.
Traceability: Developed from HLR-001.

LLR_005 *Get Sensor Measures*: When SensorInput data is validated, TrafficDetection shall get the UAV sensor measures using the SensorMeasures from SensorInput.
Traceability: Developed from HLR-001.

LLR_006 *Extract Traffic Position*: When the SensorMeasure is not empty, TrafficDetection shall calculate the TrafficPosition using the SensorMeasure and the CurrentPosition.
Traceability: Developed from HLR-001.

LLR_007 *Conflict Region Calculation*: TrafficDetection shall calculate the ConflictRegion using the CurrentPosition, HorizontalRadius, and VerticalMeasure.
Traceability: Developed from HLR-001.

LLR_008 *Verify Traffic Position Against Conflict Region*: TrafficDetection shall compare the TrafficPosition with the ConflictRegion and do the following: When TrafficPosition is in the ConflictRegion, set PotentialTraffic to True. Otherwise, set PotentialTraffic to False.
Traceability: Developed from HLR-001.

LLR_009 *Output Traffic Detection*: When PotentialTraffic is True, TrafficDetection shall output TrafficDetected, TrafficPosition, and CurrentPosition.
Traceability: Developed from HLR-001.

LLR_010 *Detection Range Validation*: TrafficDetection shall validate that the detection range for cooperative traffic is at least 20 nautical miles.
Traceability: Developed from HLR-002.

LLR_011 *Azimuth FOR Validation*: TrafficDetection shall validate that the azimuth field of regard for cooperative traffic detection is at least $\pm 110°$.
Traceability: Developed from HLR-003.

LLR_012 *Elevation FOR Validation*: TrafficDetection shall validate that the elevation field of regard for cooperative traffic detection is at least ±15°.
Traceability: Developed from HLR-003.

LLR_013 *Detection Rate Validation*: TrafficDetection shall ensure that the detection rate is equal to or greater than 1.0 hertz.
Traceability: Developed from HLR-004.

LLR_014 *Establish Track File*: When TrafficDetected is True, the TrafficTracking shall establish a track file for the detected traffic, including Position and VelocityVector.
Traceability: Developed from HLR-005.

LLR_015 *Update Track File*: The TrafficTracking shall update the track file for each detected traffic element at a rate of at least X updates per second.
Traceability: Developed from HLR-005.

LLR_016 *Maintain Track File*: The TrafficTracking shall maintain the track of a detected traffic element until it is no longer detected or poses no threat.
Traceability: Developed from HLR-005.

LLR_017 *Track File Accuracy*: The TrafficTracking shall ensure that the Position and VelocityVector in the track file have an accuracy within Y meters and Z meters/second respectively.
Traceability: Developed from HLR-005.

LLR_018 *Collision Potential Calculation*: The CollisionEvaluation shall calculate the collision potential for each tracked traffic element using Position, VelocityVector, and TrajectoryData.
Traceability: Developed from HLR-006.

LLR_019 *Threat Level Assessment*: The CollisionEvaluation shall assess the threat level of each traffic element based on the calculated collision potential.
Traceability: Developed from HLR-006.

LLR_020 *Time to Collision Calculation*: The ThreatPrioritization shall calculate the TimeToCollision (TTC) for each tracked traffic element.
Traceability: Developed from HLR-007.

LLR_021 *Threat Ranking*: The ThreatPrioritization shall rank traffic elements based on their TimeToCollision, prioritizing the ones with the shortest TTC.
Traceability: Developed from HLR-007.

LLR_022 *Avoidance Maneuver Calculation*: The ManeuverDetermination shall calculate an avoidance maneuver for each identified collision threat.
Traceability: Developed from HLR-008.

LLR_023 *Maneuver Selection*: The ManeuverDetermination shall select the most appropriate avoidance maneuver from the calculated options.
Traceability: Developed from HLR-008.

LLR_08a-01 *Maneuver Assignment for Identified Threats*: The ManeuverDetermination function shall assign a specific avoidance maneuver to each identified collision threat unless a higher-priority threat takes precedence.
Traceability: Developed from DHLR-008a.

LLR_08a-02 *Override Handling for Higher-Priority Threats*: If a higher-priority threat is detected after an initial maneuver is assigned, the CAS shall override the initial maneuver and activate the higher-priority maneuver.
Traceability: Developed from DHLR-008a.

LLR_024 *Command Maneuver*: The ManeuverCommand function shall issue a command to initiate the selected avoidance maneuver.
Traceability: Developed from HLR-009.

LLR_025 *Continue Maneuver*: The ManeuverCommand function shall issue a command to continue an ongoing avoidance maneuver if the threat persists.
Traceability: Developed from HLR-009.

LLR_026 *Terminate Maneuver*: The ManeuverCommand function shall issue a command to terminate an ongoing avoidance maneuver if the threat no longer exists.
Traceability: Developed from HLR-009.

LLR_008b-01 *Real-Time Monitoring of Active Maneuvers*: The CAS shall monitor each active maneuver in real-time, evaluating whether the associated threat still exists or if new threats emerge.
Traceability: Developed from DHLR-008b.

LLR_008b-02 *Maneuver Update Based on Threat Reassessment*: If the threat assessment changes, the CAS shall update the maneuver accordingly. This update can involve adjusting the existing maneuver or initiating a new one.
Traceability: Developed from DHLR-008b.

LLR_008b-03 *Termination of Maneuvers*: The CAS shall terminate an active maneuver when the associated threat is no longer detected or when the maneuver is superseded by a higher-priority action.
Traceability: Developed from DHLR-008b.

LLR_027 *Sensor Data Integrity Check*: The CAS shall perform integrity checks on sensor data before using it for detection and tracking.
Traceability: Developed from HLR-010.

LLR_028 *Startup Self-Test*: The CAS shall perform a self-test on startup to ensure all components are functioning correctly.
Traceability: Developed from HLR-011.

LLR_029 *Periodic Self-Test*: The CAS shall perform self-tests periodically during operation to ensure ongoing functionality of all components.
Traceability: Developed from HLR-011.

LLR_030 *Redundancy Strategy Implementation*: The CAS shall implement redundancy strategies to handle failures in primary detection and tracking components.
Traceability: Developed from HLR-012.

LLR_031 *Real-Time Communication*: The CAS shall transmit status and alerts to the ground control station in real-time.
Traceability: Developed from HLR-013.

LLR_032 *False Alarm Detection and Mitigation*: The CAS shall implement false alarm detection and mitigation strategies to minimize unnecessary maneuvers.
Traceability: Developed from HLR-014.

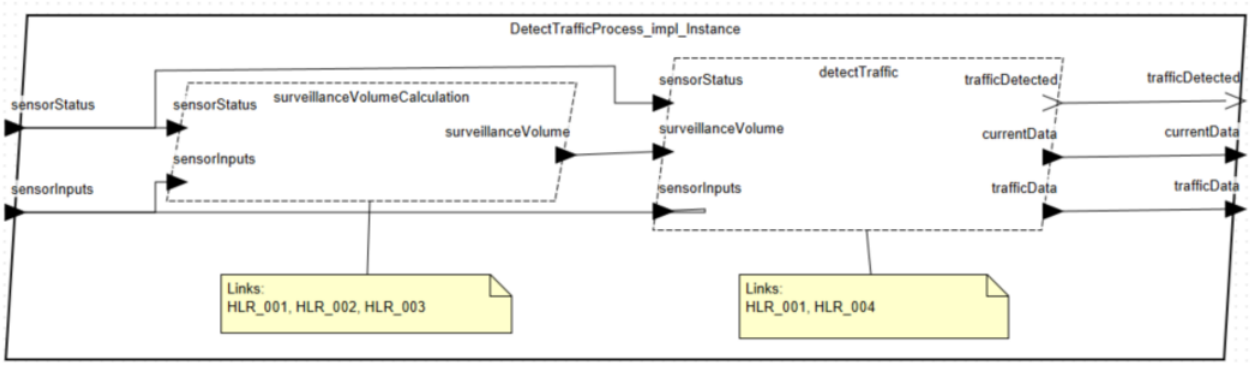


Figure A.1 AADL model of the traffic detection subsystem

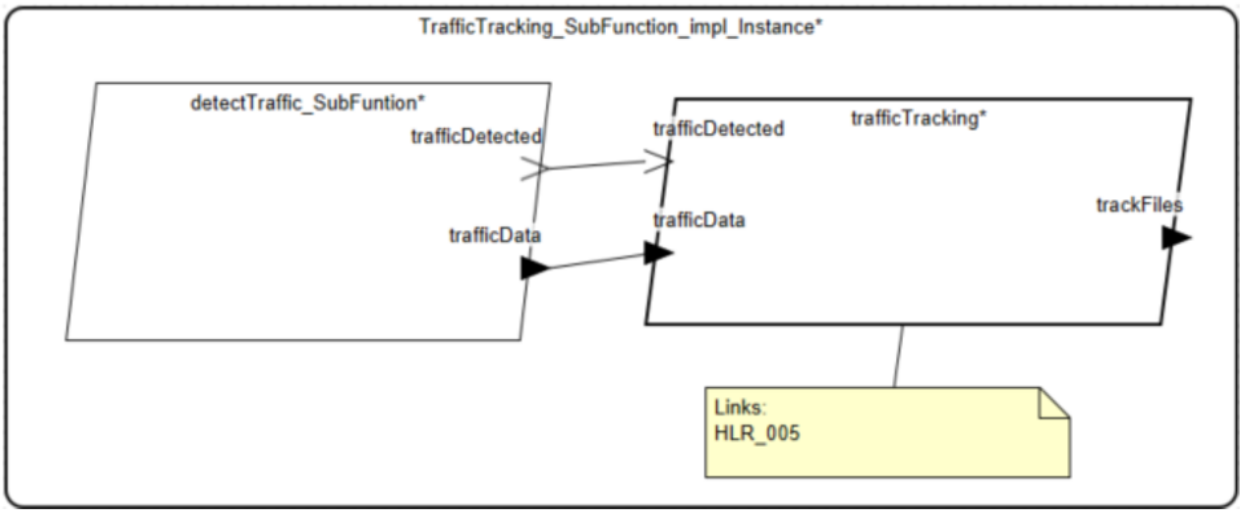


Figure A.2 AADL model of the traffic tracking subsystem

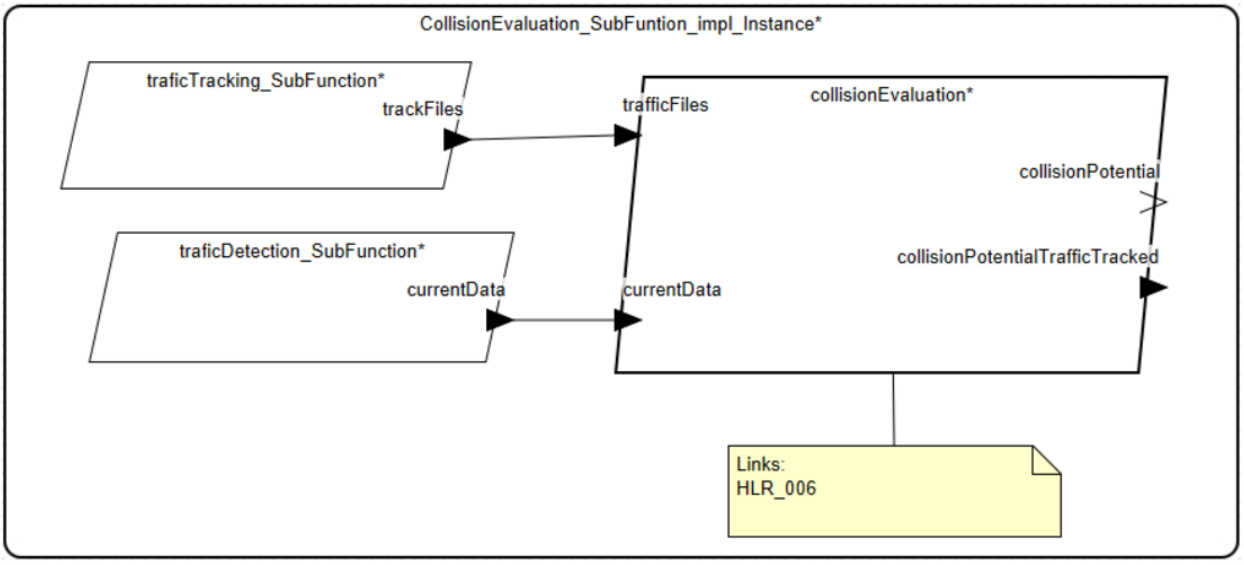


Figure A.3 AADL model of the collision evaluation subsystem

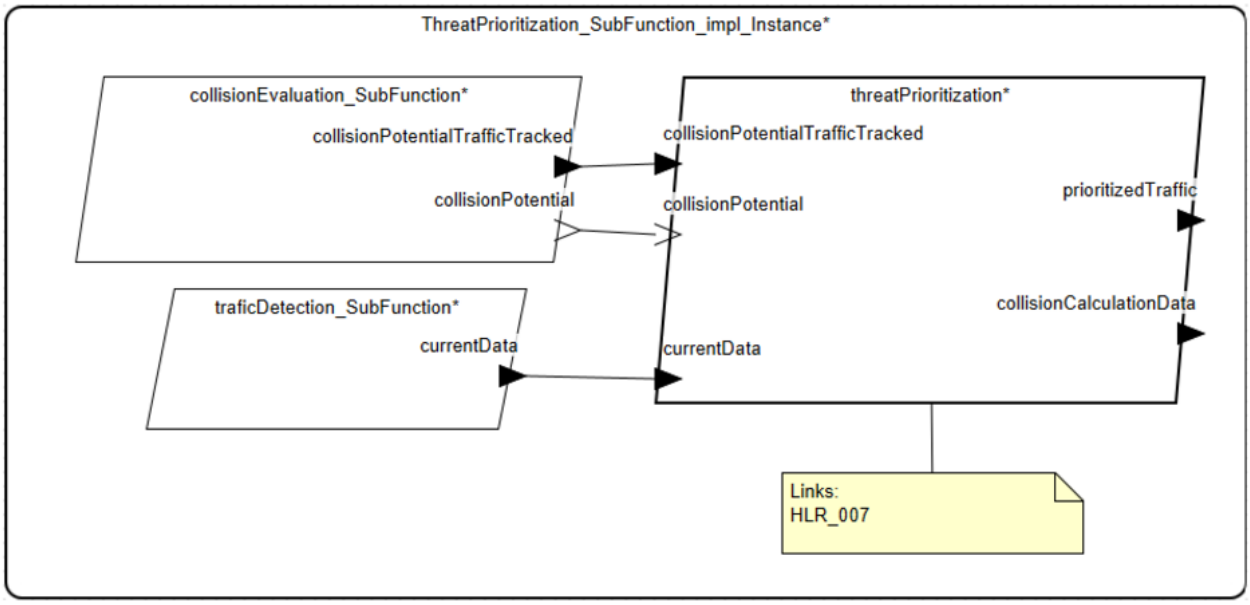


Figure A.4 AADL model of the threat prioritization subsystem

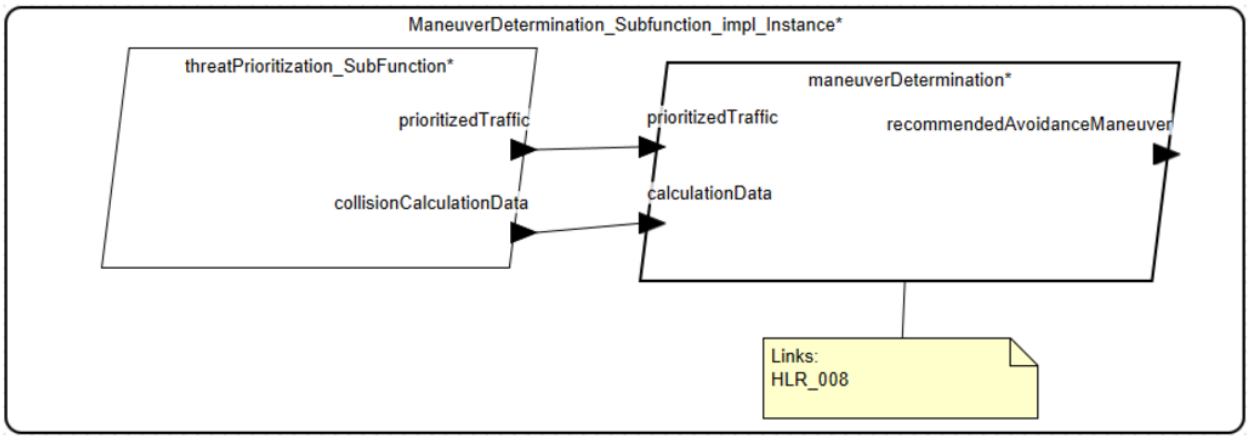


Figure A.5 AADL model of the maneuver determination subsystem

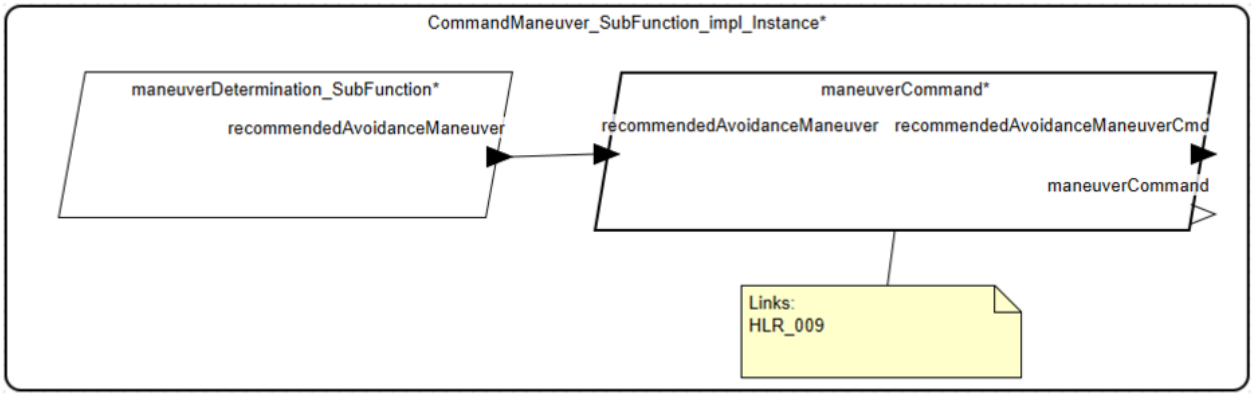


Figure A.6 AADL model of the maneuver command subsystem

APPENDIX B FMEA REPORT FOR CAS

Software Element	Failure Mode	Local Effect	Potential Severity	Recommendation
Traffic Detection	Sensor Failure	No traffic detected	Critical (10)	Implement dual-sensing technology with automatic failover. Ensure periodic testing of failover capability.
Traffic Detection	Software Error	Incorrect traffic data processing	High (9)	Conduct extensive integration testing and add error-checking algorithms to validate data before processing.
Traffic Detection	Communication Failure	Delayed or no data transmission	High (8)	Implement redundant communication channels and improve error-handling procedures to detect and reroute data transmission automatically.
Traffic Detection	Configuration Error	Incorrect detection parameters set	Medium (7)	Standardize configuration procedures and introduce periodic configuration audits and validation tests.
Traffic Detection	Physical Obstruction	Sensor input blocked or degraded	Medium (6)	Schedule regular maintenance checks and install environmental monitoring systems to alert staff to obstructions.
Traffic Tracking	Loss of tracking data	Inability to continue monitoring detected traffic	High (8)	Implement data recovery protocols and redundant tracking mechanisms with automatic switching to backup systems.

Traffic Tracking	Erroneous velocity vector computation	Incorrect assessment of traffic movement	Medium (7)	Enhance algorithm accuracy with machine learning techniques and introduce additional validation checks at each computation stage.
Traffic Tracking	Sensor misalignment	Degraded tracking accuracy	Medium (6)	Implement regular calibration protocols and real-time monitoring of sensor alignment with alerts for deviations.
Traffic Tracking	Communication delay	Late update of traffic data, potential for outdated tracking info	High (8)	Upgrade to faster data transfer technology and establish a secondary communication protocol for redundancy.
Traffic Tracking	Software crash	System stops tracking traffic altogether	High (9)	Develop a robust error handling framework and system recovery processes including auto-restart features.
Collision Evaluation	Incorrect threat assessment	Incorrect collision potential evaluation	High (8)	Integrate comprehensive machine learning algorithms to refine threat assessment based on real-world data continuously.
Collision Evaluation	Delayed data processing	Late collision threat response	High (8)	Optimize processing algorithms for efficiency and upgrade hardware for faster data handling capabilities.
Collision Evaluation	Sensor data corruption	Inaccurate threat data used for evaluations	High (8)	Implement stringent data integrity checks and use error-correcting codes for all incoming data streams.

Collision Evaluation	Software malfunction	System fails to evaluate collision threats	Critical (10)	Establish a routine for periodic software audits and develop robust error handling and recovery procedures.
Collision Evaluation	Communication failure with tracking component	No updated data received for evaluation	High (8)	Ensure reliable communication protocols are in place and implement fallback mechanisms for immediate switch-over when failures are detected.
Threat Prioritization	Incorrect ranking logic	Misordered threat prioritization	High (8)	Audit and optimize ranking algorithms annually and incorporate adaptive machine learning to enhance decision-making.
Threat Prioritization	Delay in processing updates	Outdated threat prioritization	High (8)	Invest in faster processing hardware and optimize real-time data handling capabilities to process updates instantaneously.
Threat Prioritization	Data synchronization errors	Inconsistent threat data used	High (8)	Implement a robust protocol for data verification and synchronization across all system components.
Threat Prioritization	Software glitches	Interruption in threat prioritization process	Critical (10)	Conduct frequent software testing and ensure updates are deployed; establish fail-safe modes for glitch detection.
Threat Prioritization	Communication breakdown with collision evaluation system	Lack of input for prioritization decisions	High (8)	Design a multi-channel communication strategy with automatic failover to backup channels in case of failure.

Maneuver Determination	Inaccurate threat assessment	Incorrect maneuver determination	High (9)	Continuously update and refine threat assessment algorithms based on latest data and simulation outcomes.
Maneuver Determination	Delay in maneuver computation	Delayed response to collision threats	High (9)	Streamline computation algorithms and invest in advanced computing hardware to minimize response times.
Maneuver Determination	Software bugs or glitches	Incorrect or no maneuver generated	Critical (10)	Implement comprehensive testing protocols, regular software audits, and updates to detect and rectify bugs.
Maneuver Determination	Communication latency with sensors	Outdated data used for maneuver planning	High (8)	Enhance the real-time data transmission system and implement a checking mechanism to confirm data freshness.
Maneuver Determination	System overload	System unresponsive or slow to determine maneuvers	Critical (10)	Design the system for scalability, implement load balancing, and prioritize critical data processing.
Maneuver Command	Command signal failure	No maneuver initiated or incorrect maneuver initiated	Critical (10)	Implement comprehensive signal integrity checks and establish multiple fallback communication protocols.
Maneuver Command	Delay in command execution	Delayed response in executing maneuvers	High (9)	Optimize system response time with real-time processing enhancements and streamline command execution paths.
Maneuver Command	Incorrect command data	Inappropriate maneuver based on incorrect or corrupted data	Critical (10)	Strengthen data validation processes at each stage and employ redundant data verification systems.

Maneuver Command	Software malfunction	Incorrect maneuver execution, potential system crash	Critical (10)	Establish a protocol for regular software testing, updates, and robust error handling systems.
Maneuver Command	Hardware failure	Inability to execute commands due to hardware malfunction	High (9)	Utilize high-reliability hardware components, conduct regular maintenance, and implement hardware health monitoring.