Instituto Superior de Engenharia
Politécnico de Coimbra

DEPARTMENT OF SYSTEMS AND COMPUTER ENGINEERING

Artificial Intelligence on board Spacecraft

Internship Report to fulfill the Master's degree in Informatics Engineering

Specialization Area of Software Engineering

Author

Rafael Diogo Nóbrega da Fonseca

Supervisor

Simão Pedro Mendes Cruz Reis Paredes

Advisor in the organisation Critical Software, S.A.

Daniel Chichorro de Carvalho

Pedro Reis Nunes

Coimbra, dezembro de 2025

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Rafael Fonseca (DEIS)

RESUMO

Para concluir o meu mestrado em engenharia informática, durante o ano letivo de 2024/2025, realizei este estágio na Critical Software com o objetivo de pesquisar o tema «Utilização da Inteligência Artificial em software a bordo de naves espaciais».

A incorporação da IA nas operações espaciais está a impulsionar uma rápida evolução no panorama da exploração espacial. Ao utilizar métodos de ponta, como visão computacional, aprendizagem automática e processamento de linguagem natural, os sistemas de inteligência artificial (IA) melhoram a eficiência, a segurança e a autonomia das missões.

Neste contexto, foram analisados os princípios e métodos fundamentais subjacentes aos sistemas alimentados por IA, destacando a sua capacidade de transformar os setores da exploração espacial. O relatório aborda aplicações como o processamento de dados, a deteção de anomalias, a navegação autónoma e o planeamento de missões, bem como os desafios associados à implementação da IA em ambientes espaciais, caracterizados por restrições técnicas e operacionais significativas.

Vários exemplos da Agência Espacial Europeia (ESA) demonstram o uso da IA na resolução de desafios complexos da exploração espacial. Estes exemplos demonstram aplicações práticas da tecnologia para melhorar os resultados das missões. Este relatório pretende ser um recurso útil para investigadores, engenheiros e decisores políticos, oferecendo uma análise aprofundada do envolvimento da IA nas operações de naves espaciais.

No âmbito deste estágio, foi-me atribuída a responsabilidade de explorar e implementar abordagens inovadoras baseadas em inteligência artificial, alinhadas com os desafios identificados no contexto das operações espaciais. Nesse sentido, desenvolvi um chatbot privado assente na arquitetura Retrieval-Augmented Generation, concebido para permitir a consulta segura de documentação técnica interna, incluindo ficheiros PDF, e a criação de contextos conversacionais adaptados a projetos específicos. Paralelamente, projetei e implementei uma ferramenta de apoio à análise de código que recorre a Large Language Models para detetar, analisar e propor correções automáticas para violações identificadas por ferramentas de análise estática, integrando-se de forma transparente em pipelines de integração e entrega contínua (CI/CD).

De forma global, os objetivos inicialmente definidos para o estágio foram alcançados, embora o seu âmbito tenha evoluído ao longo do período de desenvolvimento, ajustando-se às necessidades e às recomendações resultantes do trabalho realizado.

Palavras-chave: Inteligência Artificial (IA), Naves Espaciais, Técnicas, Algoritmos, Machine Learning (ML)

ABSTRACT

To complete my master's degree in computer engineering, during the 2024/2025 academic year I did this internship at Critical Software with the aim of researching the topic "Use of Artificial Intelligence in software on board spacecraft".

AI's incorporation into spacecraft operations is driving a rapid evolution in the space exploration landscape. By utilizing cutting-edge methods like computer vision, machine learning, and natural language processing, artificial intelligence (AI) systems improve mission efficiency, safety, and autonomy. Space missions reach new heights thanks to this technological synergy, which guarantees more dependable and efficient results.

Within this context, the fundamental principles and methodologies underlying AI-driven systems were analyzed, highlighting their potential to transform various domains of space exploration. The report addresses applications such as data processing, anomaly detection, autonomous navigation, and mission planning, while also discussing the challenges associated with implementing AI in space environments characterized by strict technical and operational constraints.

Several examples from the European Space Agency (ESA) demonstrate the use of AI in solving complex space exploration challenges. These demonstrate practical applications of AI technology to improve mission results. This report aims to be a useful resource for researchers, engineers, and policymakers by offering a thorough examination of AI's involvement in spacecraft operations. It provides information on the present status of artificial intelligence technology in space and its prospective future advancements.

As part of this internship, I was tasked with exploring and implementing innovative AI-based solutions aligned with the challenges identified in space-related operations. In this regard, I developed a private chatbot based on the Retrieval-Augmented Generation (RAG) architecture, designed to enable secure querying of internal technical documentation, including user-uploaded PDF files, and to support project-specific conversational contexts. In parallel, I designed and implemented a real-time code analysis support tool leveraging Large Language Models (LLMs) to detect, analyze, and automatically propose corrections for static code analysis violations. This tool was integrated with existing static analysis platforms and embedded within CI/CD pipelines to ensure automated validation and verification processes.

Overall, the initial objectives of the internship were achieved; however, the scope of the work evolved progressively in response to the team's operational needs and the technical recommendations that emerged throughout the development process.

Keywords: Artificial Intelligence (AI), Spacecraft, Techniques, Algorithms, Machine Learning (ML)

DEDICATORY

This thesis is dedicated to the people whose support, encouragement, and presence shaped both my academic journey and the person I have become throughout it.

To my family, who have always believed in me, even during the moments when I doubted myself. Your patience, sacrifices, and unconditional support provided the foundation on which every success of mine has been built. The values you instilled, resilience, curiosity, and integrity, guided me through the challenges of this work and continued to guide me through life.

To my friends, who stood by me during late nights, difficult deadlines, and moments of uncertainty. Your conversations, humour, and reminders to take breaks allowed me to keep balance when everything seemed overwhelming. Thank you for being a constant source of motivation and companionship, both inside and outside the academic world.

To the colleagues and teammates, I met throughout this journey, whose collaborative spirit and shared passion made the work environment inspiring and motivating. The exchange of knowledge, ideas, and perspectives not only enriched this thesis but also helped me grow into a more capable and confident professional.

Finally, I dedicate this work to everyone who believes in continuous learning and the pursuit of knowledge. This thesis is a small reflection of that belief, and I hope it serves as a reminder that persistence, curiosity, and support from others can lead us farther than we imagine.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to all those who contributed, directly and indirectly, to the completion of this thesis and to my development throughout this internship.

First and foremost, I extend my deepest appreciation to my internship supervisors, whose guidance, expertise, and feedback were invaluable throughout the entire process. Their openness to new ideas and their willingness to challenge me academically and professionally played a significant role in shaping the work presented here. I am grateful for the trust placed in me and for the opportunity to contribute meaningfully to the ongoing projects within the team.

I would also like to acknowledge the SpaceForce team for welcoming me as part of their group and for creating an environment where learning, collaboration, and innovation were encouraged. Their technical insights, constructive discussions, and everyday support enriched my experience and allowed me to evolve not only as a student but also as an aspiring engineer.

A special thank you goes to my professors and academic mentors, whose dedication to teaching and research inspired me throughout my academic journey. Their encouragement and belief in my potential motivated me to pursue excellence and to approach engineering challenges with curiosity and determination.

I am profoundly grateful to my family and friends for their unwavering support, understanding, and encouragement during the most demanding stages of this work. Their presence, whether through words of motivation, moments of distraction, or simple acts of kindness played an essential role in helping me maintain focus and achieve this milestone.

To everyone who supported me along the way, thank you.

INDEX

INDEX TABLE

LIST OF ACRONYMS

AI	Artificial Intelligence
AIDA	AI-powered Digital Assistant
ANN	Artificial Neural Networks
CD	Continuous Deployment
CI	Continuous Integration
CNN	Convolutional Neural Network
CSW	Critical Software
DL	Deep Learning
ESA	European Space Agency
GA	Genetic Algorithms
GAN	Generative Adversarial Network
IOD	In-Orbit Demonstration
IOV	In-Orbit Validation
ISEC	Instituto Superior de Engenharia de Coimbra
KG	Knowledge Graph
kNN	k-Nearest Neighbors
LLM	Large Language Models
LSTM	Long Short-Term Memory
ML	Machine Learning
NASA	National Aeronautics and Space Administration
NLP	Natural Language Processing
NN	Neural Networks
OLS	Ordinary Least Squares
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAR	Synthetic Aperture Radar
SVM	Support Vector Machine
UC	Use Case
US	Use Story

1 INTRODUCTION

The increasing integration of artificial intelligence (AI) into onboard spacecraft systems represents a transformative shift in space exploration, the potential for enhancing mission efficiency, safety, and autonomy achieving performance and reliability levels previously unattainable is driving this evolution.

However, the rapid proliferation of AI technologies within spacecraft operations presents significant challenges. While AI systems incorporate technologies like computer vision, machine learning, and natural language processing demonstrably enhance mission efficiency, safety, and autonomy, the inherent complexity of spacecraft systems, coupled with the stringent requirements for reliability and safety, necessitates a robust and adaptable approach to AI development and integration. Furthermore, the traditionally siloed nature of software development processes for onboard systems often focused solely on individual components can impede the seamless incorporation of AI tools and techniques.

Recognizing this paradigm shift, Critical Software initiated a strategic focus on integrating artificial intelligence across multiple stages of spacecraft system development. In alignment with this organizational direction, my internship work contributed to expanding Critical Software's practical understanding of AI's applicability within space systems. This contribution materialized through research and development activities aimed at identifying concrete opportunities for AI integration within existing workflows.

More specifically, this effort resulted in the development of two complementary tools. The first consisted of a private chatbot based on a Retrieval-Augmented Generation (RAG) architecture, designed to enable secure and contextual access to internal technical documentation. The second was a real-time code analysis support tool leveraging Large Language Models (LLMs) to automatically detect, analyze, and propose corrections for static code violations, integrating seamlessly with existing development and CI/CD pipelines.

As the internship progressed, it became increasingly evident that isolated AI applications were not sufficient to fully unlock their potential within Critical Software's spacecraft system development processes. Instead, a more holistic and integrated approach was required, one capable of embedding AI tools coherently throughout the entire software development lifecycle. Consequently, the primary objective of my work evolved into identifying existing gaps and proposing structured solutions that would enhance efficiency, reliability, and innovation in the adoption of AI-driven methodologies.

Within this framework, this report examines several advanced AI techniques relevant to space systems, including natural language processing, computer vision, and machine learning. Machine learning algorithms enable spacecraft systems to analyze large volumes of data, supporting improved prediction and adaptive decision-

making. Computer vision facilitates the interpretation of visual information, enhancing tasks such as object recognition and navigation. Natural language processing contributes to more advanced communication and control systems by enabling the understanding and generation of human language.

In addition, specific AI methodologies such as reinforcement learning, convolutional neural networks, and recurrent neural networks are discussed in relation to their respective applications in autonomous decision-making, image analysis, and sequential data processing. The report also addresses the particular challenges associated with deploying AI in space environments, including real-time processing requirements, computational constraints, and the need for robustness under extreme and dynamically changing conditions.

1.1 Critical Software

In 1998 three engineers that met while pursuing their PhDs in computer engineering at Coimbra University, Diamantino Costa, Gonçalo Quadros, and João Carreira funded Critical Software. Today, with over two decades of experience, Critical Software has more than 1000 employees and 10 offices.

In addition to its headquarters in Coimbra, Critical Software has established a strong presence in several other locations, including Lisbon, Porto, the United Kingdom, and Germany. Working in multiple types of industries like space, automotive, railway, and many other Critical Software is committed to development of software.

Although Critical Software operates as a single entity, it is comprised of multiple subgroups or "divisions" that are organized based on the specific market they serve. These divisions provide tailored solutions to meet the unique needs of each market, there are three divisions:

- **Automotive, Railway and Medical Systems (ARMS):** This division focuses on working in Automotive, Railway and Medical Systems that cannot fail. Currently, significant contributions are being made towards the development of autonomous electric cars, interoperability of railway networks with integrated train maintenance and operation systems and connected medical devices for futuristic hospitals.
- **Digital Engineering Services (DES):** At DES a specialized digital transformation offer has been developed in response to the increasing trend in digital solutions, the company serves a diverse range of clients, including those in the financial services, eCommerce, health, and telecom industries.
- **Smart & Reliable Systems (SRS):** SRS division is solely responsible for Aviation, Defence, Space and Energy industries. In the aviation, space and defence industries, the focus is on developing spacecraft and aircraft with the

highest standards of safety, meanwhile, in the energy market, DES strives to be a strategic partner for operators and system vendors across Europe with their mission-critical software systems.

During this internship I integrated the *SpaceForce* platoon, which is part of the SRS division, where our main focus is the Space and Launch segments, it is also the only platoon to have the capacity to go and operate beyond Earth's limits.

1.2 Internship's Role

During my internship at Critical Software (CSW), I transitioned from a theoretical understanding of AI techniques, including natural language processing, computer vision, machine learning, reinforcement learning, and convolutional/recurrent neural networks, to a practical application of those methods within the context of spacecraft system development. Initially, I focused on expanding my understanding of how AI could enhance mission efficiency, safety, and autonomy within space exploration, drawing upon insights from research and established ESA initiatives.

Specifically, I gained hands-on experience in identifying and implementing AI-driven tools and techniques to enhance the efficiency and effectiveness of the software development lifecycle, including automating tasks, improving code analysis, and streamlining testing procedures. My role involved analyzing existing development processes, researching suitable AI solutions, prototyping and testing these solutions, and collaborating closely with experienced engineers to integrate them into the team's workflow.

Furthermore, through this practical experience, I developed a deeper appreciation for the unique challenges of applying AI in space, the stringent requirements for instant processing, limited computational resources, and the need for robust systems capable of operating in harsh and unpredictable environments. I became familiar with best practices for data management, algorithm design, and system validation, all within the framework of ensuring the reliability and safety of spacecraft systems.

Ultimately, this internship provided me with a valuable foundation for pursuing further studies and research in the field of intelligent systems and their application to space exploration.

1.3 Objectives and work plan

My primary objective, as detailed in the preceding sections, was to investigate the application of algorithms and AI techniques in tasks executed by onboard software, with the ultimate goal of developing and/or reusing a library/tool that might be integrated into CSW's software development process. This involved a structured approach, initially, a comprehensive literature review was conducted to assess the current state-of-the-art in AI for space applications, followed by hands-on experimentation with technologies such as RAG and LLMs.

I focused on exploring and applying artificial intelligence techniques to enhance onboard software development processes. The research was guided by the need to identify innovative solutions that could improve efficiency, support automation, and provide practical tools for real-world spacecraft operations. The following sections summarize the main contributions and innovative aspects of the work:

- Technology Exploration & Evaluation: I systematically evaluated several AI techniques natural language processing, computer vision, and machine learning to determine their suitability for specific onboard software tasks. This included detailed research into ESA's implementation of AI in spacecraft operations, providing a strong contextual understanding.
- RAG-Based Chatbot Prototyping and Validation: I designed, developed, and tested an internal chatbot based on a Retrieval-Augmented Generation (RAG) architecture to enable secure and efficient access to technical documentation. The prototype incorporated document indexing, contextual retrieval, and locally hosted Large Language Models to ensure confidentiality compliance. Its development demonstrated the feasibility of deploying AI-driven knowledge retrieval systems within a restricted aerospace software engineering environment, significantly reducing the time required to locate and interpret project documentation.
- LLM-Powered Code Analysis Automation Tool (FixFlow): In parallel, I developed and validated a proof-of-concept tool aimed at automating the remediation of static code analysis violations. The system integrates static analysis outputs with a Large Language Model to generate contextualized, standards-aware code corrections, which are then validated through a Git-based workflow and CI/CD pipelines. This prototype demonstrated the potential of AI-assisted development workflows to accelerate issue resolution while preserving developer oversight and code reliability.
- Identification of a Reusable Library Component: Through this exploratory work, I identified a core component – the LLM-powered code analysis tool – that possessed significant potential for reuse within CSW's software

development pipeline. This represents a tangible contribution to the team's ability to accelerate development and improve code quality.

This work demonstrates CSW's commitment to embracing innovation and leveraging cutting-edge technologies to enhance the capabilities of its software solutions for space exploration.

To achieve the objectives of the internship five tasks were assigned to me, as shown in the Figure 1.1, which were:

- T1: Research into AI techniques and algorithms and their applicability;
- T2: Research into ESA's work in the field of AI;
- T3: Research into the operations typically carried out by software on board spacecraft;
- T4: Implement and/or propose the use of existing AI algorithms or libraries and integrate them into CRITICAL's spacecraft control software platform;

T5: Writing the dissertation on the topics covered above, as well as preparing two attached presentations, one for the academic assessment and the other, more technical, for internal presentation at Critical Software.

Figure 1 - Tasks Scheduling

1.4 Report's Structure

This report is divided into six chapters, each serving a distinct purpose within the overall document.

- Chapter Two – AI's Background: This chapter presents the historical development and key concepts of artificial intelligence as applied to space systems. It covers the evolution of onboard AI from early ground-based intelligence to modern autonomous systems, discusses the fundamental AI concepts relevant to this work, and identifies the main challenges associated with deploying AI in space, including performance, cyber, legal, and ethical considerations.
- Chapter Three – Background Knowledge: This chapter provides a detailed overview of the algorithms, techniques, and tools employed in this work. It includes an examination of classical and modern machine learning methods, neural network architectures, and ESA's AI initiatives, with illustrative use cases such as AIDA, Φsat, OPS-SAT, Hera, and ClearSpace-1. This background establishes the foundation for understanding the practical implementation of AI-driven tools.
- Chapter Four – Implementation: This chapter presents the core contributions of the internship. It details the design, development, and evaluation of AI-driven tools created during the work, including SpaceSage, a private RAG-based chatbot, and FixFlow, an LLM-powered code analysis tool. It also discusses technical innovations, system architecture, workflows, performance evaluation, limitations, and potential future improvements.
- Chapter Five – Conclusions: The final chapter summarizes the main contributions and innovative aspects of the work. It reflects on the outcomes achieved, lessons learned, and the impact of AI integration in spacecraft software development. The chapter also identifies potential directions for future research and development in the field.

2 AI'S BACKGROUND

This section chronicles the emergence and progression of artificial intelligence (AI) within space exploration, tracing its transformation from early mission-support systems to increasingly autonomous onboard systems in satellites, spacecraft, and launch operations.

2.1 Evolution of AI in Space

In the 1970s and 1980s, AI in space began with supporting roles primarily on the ground rather than onboard spacecraft. Early efforts focused on automating mission control tasks such as fault diagnosis and scheduling observations, introducing expert systems into mission operations despite the hardware limitations of the era. [1]

A major leap occurred in 1999 with the Remote Agent aboard NASA's *Deep Space 1*, the first AI system to autonomously control a spacecraft. Over several days, it performed fault diagnosis, replanning, and execution without human intervention, earning NASA's Software of the Year award.[1], [2]

Building on that, the *Earth Observing-1* (EO-1) mission featured the Autonomous Sciencecraft Experiment (ASE) by 2004. ASE enabled the spacecraft to analyze imagery onboard and autonomously retask the satellite by redirecting its camera to capture volcanic eruptions detected in real-time. [1], [3]

AI capabilities expanded beyond orbiters. Mars rovers Spirit and Opportunity benefitted from early autonomous navigation, while Curiosity introduced the AEGIS system allowing autonomous selection of rock targets for its ChemCam instrument by 2015. [1]

On the launch side, JAXA's Epsilon rocket (2013) became the first launch vehicle to perform AI-based health checks during launch countdown, significantly reducing launch preparation effort and human overhead. [1]

In 2018, Airbus and IBM launched CIMON aboard the ISS a conversational AI assistant powered by IBM Watson, designed to assist astronauts using voice-controlled, interactive support. [1]

The ESA's Φ-sat-1 (launched 2020) marked the debut of dedicated AI hardware in orbit. Its onboard AI chip performed real-time cloud detection to avoid downlinking useless images, conserving bandwidth and ushering in "edge AI" for Earth observation. The follow-up Φ-sat-2 (2024) extended capabilities further with multiple AI applications onboard and remote model updates.[1]

NASA's Perseverance rover (landed 2021) introduced AutoNav, enabling navigation up to five times faster than Curiosity by processing visual input onboard to avoid hazards in real-time. It also features AI-guided sampling tools like PIXL to autonomously identify geologically interesting targets.[1]

Meanwhile, recent advances include reinforcement learning-based command agents tested aboard small satellites. A notable example is a CubeSat experiment (2025) using RL to adjust solar orientation autonomously with safety-validated policies, a significant step toward trusted onboard AI autonomy.[4]

Beyond autonomy, AI applications have expanded across various mission domains:

- AI-driven space situational awareness systems, such as those developed by IIIT-Delhi with ISRO now monitor orbital debris and rogue satellites using agentic AI.[5]
- Innovative infrastructure is emerging, China recently launched satellites capable of onboard supercomputing, forming the initial part of a planned AI satellite constellation for in-orbit data processing, mitigating downlink constraints.[6], [7]

From early ground-based expert systems to mission-critical autonomy, AI in space has evolved rapidly. Key milestones Remote Agent, EO-1's onboard science, rover autonomy, AI assistants like CIMON, and the drive toward edge AI with Φ-sat satellites underscore the journey from theoretical possibility to operational integration. AI now plays a central role in mission efficiency, autonomy, and scalability, laying the groundwork for next-generation autonomous space systems.

2.2 The Challenges of AI in Space

Although Artificial Intelligence has the potential to transform space exploration through autonomous navigation, immediate decision-making, and sophisticated data analysis, its effective application depends on addressing a complicated mix of technical, legal, and ethical hurdles.

2.2.1 Performance Challenges

A major technical obstacle is maintaining the dependability and efficiency of AI systems in the extreme conditions of space. Extreme temperatures, radiation, and microgravity pose significant challenges to electronic components in space missions.

These elements may cause hardware deterioration, software malfunctions, and eventually system breakdowns.

To reduce these risks, investigators are examining different options:

- **Radiation-resistant hardware:** Creating specialized hardware elements capable of enduring the severe radiation conditions of space is essential.
- **Redundancy and fault tolerance:** Creating AI systems with inherent redundancy and error correction features can assist in maintaining functionality even if certain parts malfunction.

Nonetheless, applying these solutions frequently results in greater size, weight, and energy usage – all valuable resources on spacecraft.

2.2.2 Cyber Vulnerabilities

AI systems in space are susceptible to various cyber threats, including hacking, data breaches, and malicious code injections. These vulnerabilities can compromise the integrity and security of space missions. The absence of standardized safety protocols and technical guidelines for AI in space poses significant risks. Without proper regulations, AI systems may make decisions with unintended consequences, leading to accidents or malfunctions.

One major technical hurdle lies in the inherent vulnerability of AI systems to cyberattacks, these systems are susceptible to various cyber threats. These include:

- **Unpatched software vulnerabilities:** Traditional software exploits can be equally effective against AI systems if they lack proper patching procedures. The report highlights the challenge of updating AI systems deployed in space due to communication delays and limited bandwidth.
- **Data poisoning attacks:** Malicious actors could manipulate the training data used to develop AI, leading the system to make biased or erroneous decisions. This could have disastrous consequences for critical space missions.
- **Physical attacks:** The report also raises concerns about potential physical attacks on satellites or rovers housing AI systems, potentially leading to data breaches or hardware malfunctions.

These vulnerabilities necessitate the development of robust cybersecurity measures specifically tailored for space applications. This includes employing encryption techniques to safeguard sensitive data, implementing intrusion detection systems to

identify and thwart cyberattacks, and establishing secure communication protocols for data transmission between Earth and space assets.

2.2.3 Legal Challenges

The swift advancement of AI in space requires a global framework to tackle legal and liability concerns. Such a pact would create definitive rules for the advancement, implementation, and utilization of AI in space.

The growing privatization of space endeavors prompts worries regarding the possible exploitation of AI for business or military objectives, because of that, there's a need for strong regulations to avert the militarization of AI and guarantee accountable space operations. Moreover, the legal structures regulating space operations must adjust to this changing technological environment. These frameworks need to tackle concerns like accountability, data ownership gathered by AI technologies, and possible disputes stemming from the growing commercialization of space.

2.2.4 Ethical Challenges

The integration of AI into space missions raises ethical questions about the level of autonomy granted to machines. While AI can enhance decision-making capabilities, it is crucial to maintain human oversight and accountability.

As AI becomes more sophisticated, it is essential to establish ethical principles to guide its development and deployment. These principles should address issues such as fairness, transparency, and accountability, ensuring that AI is used for the benefit of humanity and the environment.

Beyond technical challenges, AI in space raises a host of ethical and legal questions, as AI systems become more autonomous in space missions, concerns arise regarding:

- **Accountability:** In case of accidents or malfunctions caused by AI-driven decisions, who is held accountable, the programmers, the mission controllers, or the AI itself? Establishing clear lines of responsibility is paramount.
- **Transparency:** The decision-making processes employed by AI systems can be opaque, making it difficult to understand how they arrive at certain conclusions. Ensuring transparency in AI algorithms is crucial for building trust and mitigating potential biases.

- **Weaponization of AI:** The report emphasizes the need for robust international regulations to prevent the use of AI for military purposes in space, ensuring peaceful exploration and cooperation.

Addressing these ethical concerns necessitates the development of international guidelines on the responsible development and deployment of AI in space. These guidelines should emphasize human oversight, promote transparency in AI algorithms, and prioritize peaceful applications.

2.2.5 Additional Challenges

Space missions possess restricted power and computing capabilities, which may obstruct the deployment of intricate AI algorithms. Additionally, they produce substantial quantities of data; however, gathering and processing this information in real-time can be difficult because of bandwidth limitations and communication lags.

The severe environment of space, including extreme temperatures and radiation, can affect the functionality and dependability of AI systems. Significant communication delays between Earth and space missions can hinder real-time decision-making and feedback processes, reducing the efficiency of AI systems.

In summary, AI has significant potential to transform space exploration, but its effective implementation demands a comprehensive strategy. By tackling the technical hurdles of cybersecurity and space-based reliability, setting forth explicit ethical standards, and creating a strong legal structure, we can guarantee the responsible and advantageous application of AI to tap into the immense possibilities of space exploration.

As AI technology continues to advance, ESA is committed to exploring its full potential in space. By leveraging AI, ESA aims to push the boundaries of space exploration, gain deeper insights into our universe, and address global challenges from space. However, successful implementation requires addressing significant technical, ethical, and legal challenges. From a technical standpoint, ensuring the cybersecurity of AI systems in space is paramount to protect against potential cyberattacks. Additionally, designing AI systems that are resilient to the harsh conditions of space, such as extreme temperatures and radiation, is crucial for their reliability and longevity.

Ethical considerations are equally important, as AI systems become increasingly autonomous, it is essential to establish clear guidelines for their use, including transparency, accountability, and the avoidance of malicious applications.

Finally, the legal framework for AI in space is still evolving. International cooperation is needed to develop clear regulations that govern the use of AI in space, ensuring its peaceful and sustainable development.

2.2.6 Relevance of These Challenges to the Present Work

The challenges outlined in this chapter are not merely theoretical considerations, they directly informed the design decisions and development strategy adopted throughout this internship. The technical constraints of space systems, particularly computational limitations, reliability requirements, and strict cybersecurity concerns, were central factors in the conception of both AI-based tools developed during this work.

For instance, the private RAG-based chatbot was intentionally designed to operate entirely within a local and secure infrastructure, avoiding reliance on external cloud services and thereby addressing both confidentiality and cybersecurity risks. By ensuring that all document processing and inference occurred on-premise, the system aligned with the legal and operational constraints typically associated with aerospace environments. Similarly, the modular architecture of the FixFlow code analysis tool reflects the need for traceability, validation, and controlled automation, ensuring that AI-generated corrections remain subject to human oversight and continuous integration safeguards.

Furthermore, the emphasis on explainability, traceability, and validation within both tools directly responds to the ethical and reliability challenges discussed in this chapter. By incorporating source referencing in the chatbot and verification checkpoints in the automated code remediation pipeline, the developed solutions seek to balance AI-driven efficiency with accountability and robustness. In this sense, the challenges of AI in space do not represent abstract barriers, but rather guiding principles that shaped the practical implementation of AI within the scope of this thesis.

3 BACKGROUND KNOWLEDGE

During the first phase of the internship there was a need to investigate many AI approaches and strategies, such as natural language processing, machine learning and neural networks, to have base of knowledge. As part of my internship activities, I also explored the contributions of the European Space Agency (ESA) to the advancement of artificial intelligence in space applications. This exploration provided a comprehensive understanding of the current landscape of AI technology and its potential impact on diverse industries.

3.1 Algorithms and Techniques

A variety of AI and machine learning methods are applied in space to process data from satellites, rovers, and spacecraft. These methods generally fall into supervised learning (trained on labeled examples), unsupervised learning (finding patterns in unlabeled data), or reinforcement learning (learning by trial-and-error with reward signals). This chapter presents several key algorithms and techniques below, emphasizing **what they are as well as where they are used in space contexts**.

3.1.1 Linear Regression

Linear regression is a foundational model in machine learning used primarily for predictive analysis. It operates on the premises of a weighted sum of inputs plus a constant bias term. The fundamental assumption of this model is the existence of a linear relationship between the input variables (features) and the output variable. Distinctions are drawn based on the number of inputs: "Simple Linear Regression" utilizes a single input variable, whereas "Multiple Linear Regression" involves multiple inputs. This simple approach is widely used in astronomy and engineering to uncover trends and calibrate sensors[8].

An example of a linear equation:

$$y = \beta 0 + \beta 1x + \epsilon$$

y – Estimated output

X_n – represents the features

B_0 – represents the bias term

B_n – represents the features weights

n – represents the number of features

This solution minimizes the residual sum of squares between predictions and true values.

regression is used in many space contexts where relationships are approximately linear. For example, researchers have applied linear regression to **satellite communications management**: a NASA study used linear regression to predict future communication link suitability between satellites and ground stations, improving channel efficiency in a cognitive radio network[8]. In **astronomy and planetary science**, linear (and log-linear) regression is a common first step to fit trends in data (e.g. brightness versus distance). It also serves in spacecraft engineering for sensor calibration (e.g. calibrating radiometric sensors on Earth-observing satellites) and preliminary orbit determination where small corrections can be linearized.

3.1.2 Polynomial Regression

Polynomial regression is an extension of linear regression that models nonlinear relationships by including polynomial terms of the input variables. It can capture curvilinear trends that simple linear models cannot. Despite its name, polynomial regression remains a form of linear regression.

An example of a linear equation:

$$y = \beta 0 + \beta 1x + \beta 2x2 + \beta 3x3 + \cdots + \beta nxn + \epsilon$$

y – Estimated output

X_n – represents the features

B_0 – represents the bias term

B_n – represents the features weights

n – represents the number of features

Polynomial regression can model certain spacecraft or planetary data where the relationship is smooth but nonlinear. One concrete example is orbit determination: researchers have proposed a polynomial-based method to track a maneuvering spacecraft's orbit, fitting trajectory segments by polynomial curves[9]. In that work, a "polynomial representation" allowed accurate tracking of unknown maneuvers. Polynomial regression may also be used in spacecraft trajectory planning (e.g. approximating gravity-assist curves) or in remote sensing, to fit sensor response curves or Earth profiles (e.g. quadratic fits of atmospheric parameters).

3.1.3 Logistic Regression

Logistic Regression is a statistical method used for **classification problems** to estimate the probability that an instance belongs to a specific class. Unlike linear

regression, it maps any real number into a value between 0 and 1 using the **sigmoid function**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z$is a linear combination of the input features:

$$z = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

Thus, the predicted probability is:

$$p(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

There are three primary types of logistic regression:

Binary Logistic Regression: Used when the dependent variable is binary (0 or 1). The probability of $y = 1$is:

$$p(y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

The predicted class is then:

$$\hat{y} = \begin{cases} 1 & \text{if } p \geq 0.5 \\ 0 & \text{if } p < 0.5 \end{cases}$$

Multinomial Logistic Regression: Applied when predicting probabilities for a categorical dependent variable with more than two possible outcomes. Used when the dependent variable has **more than two classes**. For Kclasses, the probability of class $k$is:

$$p(y = k \mid x) = \frac{e^{\beta_{0k} + \beta_{1k} x_1 + \cdots + \beta_{nk} x_n}}{\sum_{j=1}^{K} e^{\beta_{0j} + \beta_{1j} x_1 + \cdots + \beta_{nj} x_n}}$$

This is also called the **softmax function**, a generalization of the sigmoid.

Ordinal Logistic Regression: Similar to multinomial, used when the dependent variable has **ordered categories** (e.g., low, medium, high). It models the **cumulative probabilities**:

$$\text{logit}[P(y \leq j \mid x)] = \ln \frac{P(y \leq j \mid x)}{1 - P(y \leq j \mid x)} = \theta_j - (\beta_1 x_1 + \cdots + \beta_n x_n)$$

Where θ_j are thresholds (cutpoints) for each category j.

Logistic regression is useful for binary classification tasks in aerospace. For instance, it has been applied to satellite anomaly detection: one study used a logistic regression–based anomaly detector to identify small reaction-wheel failures in a spacecraft system[10]. The model was trained on normal telemetry and learned to flag deviations. Logistic models are also well-suited for classification of space mission data when outputs are binary, such as predicting whether a communication link is viable (yes/no), or classifying a sensor reading as nominal vs. anomalous. In general, any onboard fault detection or binary decision (e.g. target detected or not) in space systems can employ logistic regression.

3.1.4 k-Nearest Neighbors (kNN)

The k-nearest neighbors (kNN) algorithm is a simple, non-parametric method for classification (or regression) based on the similarity of input instances. Given a query point, kNN finds the closest K training examples (by some distance metric) and for classification outputs the majority class among them[11]. It is called *lazy* learning because there is no explicit model fitting step; the algorithm stores all training data and defers computation to query time. To try and find the most similar instances, the most common way to calculate the distance to find these neighbors is using the Euclidian distance.

$$Euclidean\ Distance = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

p - coordinates of point one (new instance)

q - coordinates of point two (existing instance)

n - represents the number of features

kNN can be applied to various pattern-recognition tasks in remote sensing and anomaly detection. For example, in Earth observation, researchers have used kNN to **forecast sea ice concentration** from satellite imagery: they trained an "Ice-kNN" model on historical Arctic sea ice data and achieved improved summer concentration predictions[12]. More generally, kNN is used in spacecraft operations for classification of telemetry points (flagging an alarm state if most similar past cases were alarms) and in astrophysics for sorting objects by similarity. Its simplicity makes it attractive for onboard systems with limited modeling capacity but plenty of data.

3.1.5 Support Vector Machines (SVM)

Support Vector Machines are a class of supervised learning models for classification and regression. In the binary classification case, an SVM finds an optimal separating hyperplane between classes in the input space. It chooses the hyperplane that maximizes the margin (distance) between the nearest points of each class (the *support*

vectors). If data is not linearly separable in the original space, SVM uses kernel functions to operate in a higher-dimensional feature space where separation is possible. For the linear classification, we will have something like:

$$W^T x + B = \mathrm{B} + \mathrm{w}_1\mathrm{x} + \mathrm{w}_2\mathrm{x} + \cdots + \mathrm{w}_\mathrm{n} x_n$$

Where $\boldsymbol{W^T x + B}$ will give us the decision boundary, and the output model is:

$$y = \begin{cases} 1 & if \quad W^T x + B > 0 \\ -1 & if \quad W^T x + B < 0 \end{cases}$$

$\mathrm{w_n}$ – represents the features weights

$\mathrm{x_n}$ – represents the features

B – represents the bias term

y – Estimated output

n – represents the number of features

All the instances above the Dividing Hyperplane will have label 1, as for the ones below the line will have label -1. For the values close to the Dividing Hyperplane it will return a value close to zero and the instance may be difficult to classify.

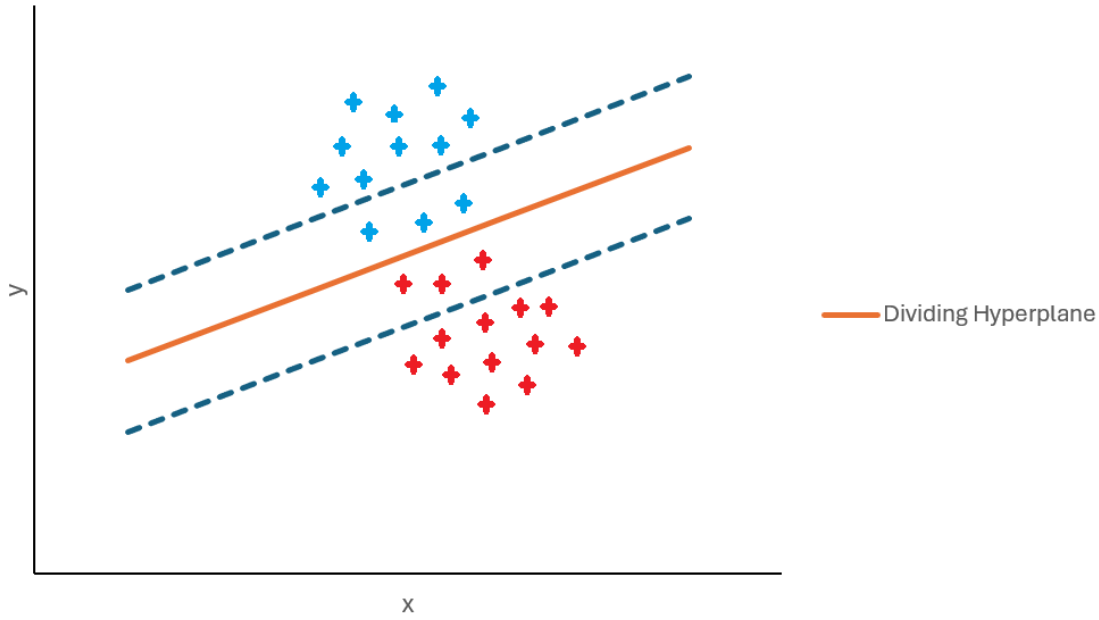

Figure 2 - Hyperplane Representation

SVMs have seen wide use in **remote sensing and Earth observation** due to their strong performance with limited training data[13]. For example, on the Earth Observing One (EO-1) mission, SVMs were trained on the ground and then uploaded to onboard systems for real-time classification of imagery, such as detecting cryosphere events (sea ice, snow) from Hyperion sensor data[14]. Similarly,

SVMs have been used to classify atmospheric phenomena from satellite sensors. In planetary science, SVMs can classify geological terrain from orbiter images. More generally, SVM's ability to handle high-dimensional inputs with small samples makes it valuable for many space applications where labeled data are scarce[13].

3.1.6 Decision Trees

A decision tree is a hierarchical, rule-based model for classification or regression. It recursively partitions the input space using tests on feature values, with each internal node representing a test and each leaf node assigning a class label or value. Trees are interpretable: they correspond to a set of if-then rules that segment the data by feature thresholds. Decision Trees are a Binary tree that recursively splits the dataset until we only have data with only one type of class, we can call them "pure" leaf nodes.

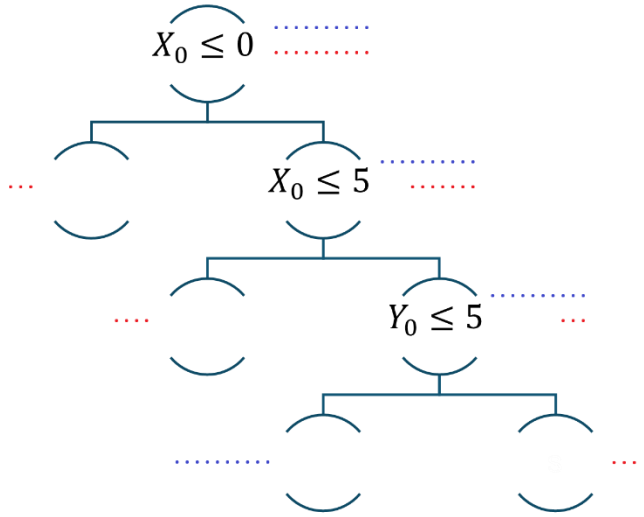

Figure 3 - Representation of a Decision Tree

As seen in the picture above the "pure" leaf nodes are the ones that only have a single type of class. Now imagine that we have an instance with the coordinates (3,4) that we want to classify, we would feed the algorithm with this new instance, and we would check for the conditions, until we reach the node that met all the conditions.

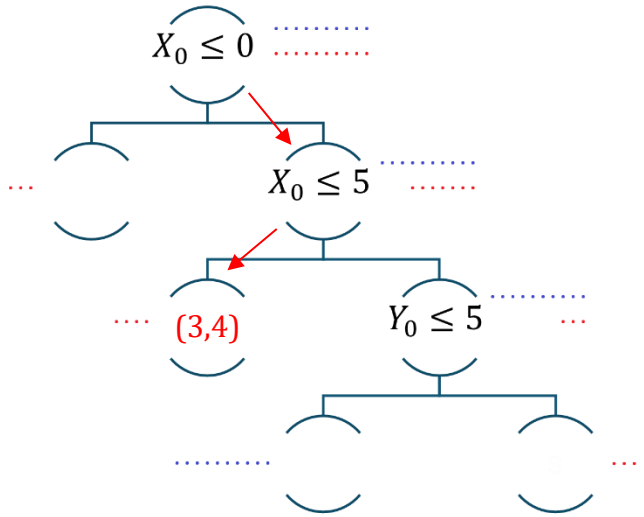

Figure 4 - Decision Tree's Process

In this simple case it's easy to choose the right conditions to perform the classification, but in more complex cases we can use Entropy or the Gini Index to select the best conditions/feature to split the data.

Decision trees are often used for rapid decision-making tasks in space systems. For example, they can classify instrument readings or orbital events by learning threshold rules on telemetry channels. In Earth imagery, decision-tree classifiers (and ensembles thereof) have been used for land cover classification and disaster detection because of their interpretability. On spacecraft, a decision tree could quickly categorize a sensor state as "normal", or "fault" based on input flags. When combined into ensembles (see Random Forest below), trees become even more powerful in remote sensing classification and anomaly detection.

3.1.7 Random Forest

Random Forest is an ensemble learning method based on constructing a multitude of decision trees and aggregating their predictions. Each tree in the forest is trained on a random subset of the data and a random subset of features (bagging and feature randomness), which makes the ensemble robust and reduces overfitting. In classification, the forest's output is the mode of the individual trees' votes; in regression, it is the average prediction[15].

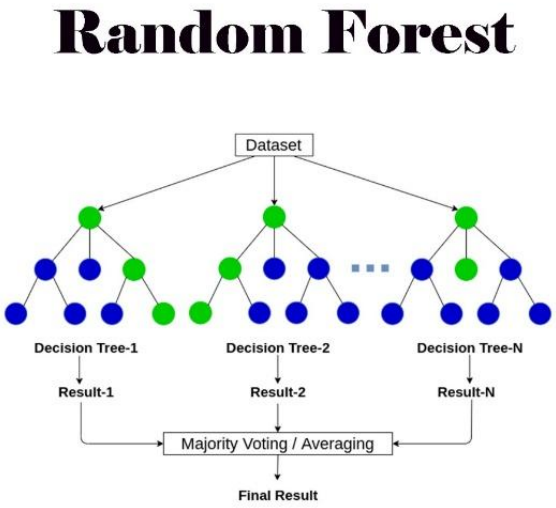

Figure 5 - Visual representation of a Random Forest

Random forests are widely used in remote sensing and aerospace data analysis. For instance, NASA provides tutorials using random forests for **land cover classification** from satellite imagery[16]. In that training material, a random forest model is built on optical remote sensing data to distinguish classes like water, vegetation, and urban. In practice, random forests have been applied to classify clouds, vegetation health, and geological features in Earth and planetary images. They are also used for **regression** tasks, such as estimating atmospheric variables from sensor data, because of their accuracy and ability to handle mixed data types. The ensemble's resilience to overfitting makes it suitable for the noisy, high-dimensional data common in space applications[15], [16].

3.1.8 Naïve Bayes

Naïve Bayes is a probabilistic classification technique based on Bayes' theorem, with the simplifying assumption that all input features are conditionally independent given the class[17]. The Bayes' theorem is:

$$P(C \mid X) = \frac{P(X \mid C)\, P(C)}{P(X)}$$

C= class variable

$X = (x_1, x_2, \ldots, x_n)$= feature vector

$P(C)$= prior probability of class C

$P(X \mid C)$= likelihood of features given class

$P(C \mid X)$= posterior probability of class given features

The **naïve assumption** of feature independence simplifies $\boldsymbol{P(X \mid C)}$as:

$$P(X \mid C) = \prod_{i=1}^{n} P(x_i \mid C)$$

Thus, the **predicted class** is:

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{n} P(x_i \mid C)$$

Even though the independence assumption is rarely strictly true, Naïve Bayes often performs very well in practice.

Naïve Bayes classifiers can be used in any classification task where probabilistic outputs are useful and computational simplicity is required. For example, they have been applied to **astronomy data analysis**, classifying stars, galaxies, or transit signals, because they can quickly yield probabilities and handle high-dimensional data. On spacecraft, a Naïve Bayes model could perform **fast onboard classification of telemetry vectors** (e.g. "anomaly" vs "normal") or even simple object recognition tasks, owing to its low computational overhead. Because Naïve Bayes can provide class likelihoods directly, it is also used in decision-support systems for prioritizing warnings under uncertainty.

3.1.9 Bayesian Networks

A Bayesian Network (BN) is a graphical model representing a joint probability distribution over a set of variables via a directed acyclic graph (DAG). Each node in the graph is a random variable, and edges indicate probabilistic dependencies. The joint probability distribution of all variables can be factorized as:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$

$X_1, \ldots, X_n$= variables in the network

Parents(X_i)= immediate parent nodes of $X_i$in the DAG

$P(X_i \mid \text{Parents}(X_i))$= conditional probability of $X_i$given its parents

Bayesian networks are especially useful for **diagnostic and decision support systems** in aerospace, where there are many interacting subsystems. A prominent

example is spacecraft fault diagnosis: NASA developed a BN model for electrical power system faults, encoding the causal relationships between component failures and observed symptoms[18]. This BN was compiled into an efficient form for onboard diagnosis (called ADAPT) and could quickly infer likely fault causes from sensor readings. More generally, BNs can model **uncertain reasoning in operations**, such as estimating the probability of success of a maneuver given uncertain input. They also apply to **mission planning** under uncertainty: planners can use BNs to represent risks and dependencies among tasks and make informed choices.

3.1.10 Neural Networks

Neural networks (NNs) are a family of machine learning models inspired by the brain, consisting of layers of interconnected "neurons" (nodes) with weighted connections. Each neuron computes a weighted sum of its inputs, applies a nonlinear activation function, and passes the result to the next layer. By stacking multiple layers (an input layer, hidden layers, and an output layer), a neural network can learn complex nonlinear functions mapping inputs to outputs[19].

Neural networks have numerous applications in space. For instance, NASA's *SIANN* project trained neural networks to recognize scenes of interest in satellite imagery archives, allowing automated search for features like weather patterns[20]. Neural nets are also used for spacecraft **orbital predictions** (learning complex perturbations) and **anomaly detection** in telemetry by modeling normal behavior patterns. In Earth science, they forecast weather and climate variables from historical data. Additionally, neural networks can process raw sensor data, for example, neural nets have been trained to classify star types from telescope spectra. The flexibility of neural networks makes them a general-purpose tool for pattern recognition, control, and prediction in aerospace engineering[20].

3.1.11 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a specialized type of deep neural network designed to process grid-structured data like images. CNNs use convolutional layers that apply learned spatial filters (kernels) across the input, capturing local patterns such as edges and textures. They also include pooling layers to downsample and extract multi-scale features. CNNs automatically learn hierarchical representations of image data[21].

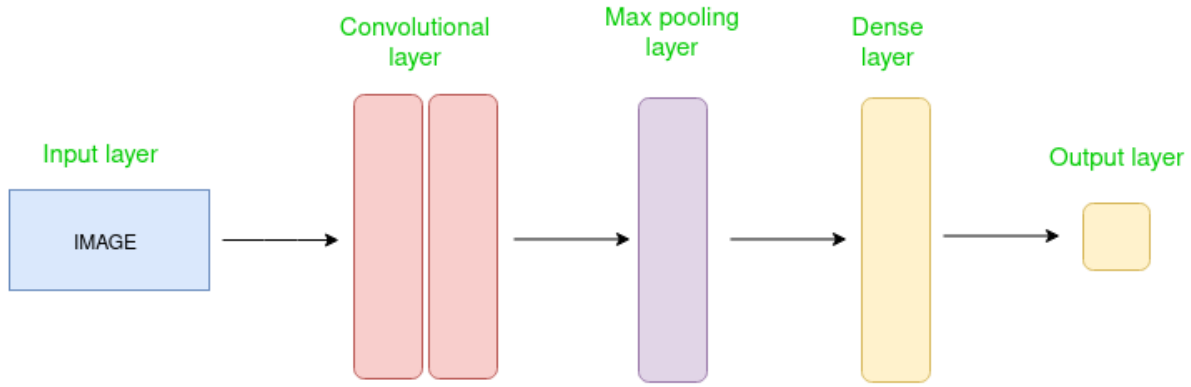

Figure 6 - Visual representation of a CNN

CNNs have become the standard for image-based tasks in space. For example, NASA and JPL have developed CNNs for spacecraft **pose estimation** given an image of a target spacecraft, a CNN can infer its relative position and orientation, aiding rendezvous and docking operations. On the Earth observation side, CNNs are used extensively for **scene classification** (identifying urban, rural, water, ice scenes) and **object detection** in satellite imagery (see below). They are also applied in astronomy for classifying galaxy morphologies or detecting transient events. In short, any vision task involving space imagery – from planetary surface feature recognition to space debris identification – benefits from CNNs' ability to learn visual patterns.

3.1.12 Recurrent Neural Networks (RNN)

Recurrent Neural Networks are a family of neural networks designed to process sequential or time-series data. Unlike feedforward networks, RNNs have *recurrent connections* that feed a hidden state from one time step back into the network at the next step, giving the network memory of past inputs. This makes RNNs suitable for data where order matters (text, signals, telemetry over time)[22]. Architectures include simple RNNs, LSTMs, and GRUs, which can capture longer-term dependencies.

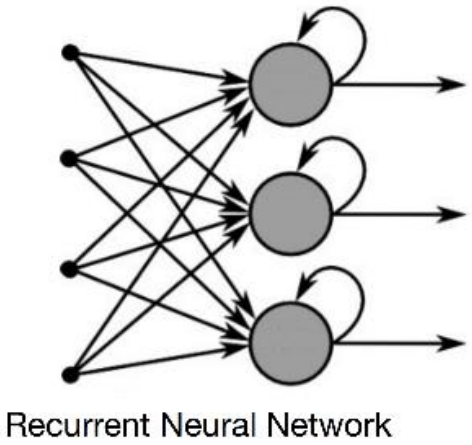

Figure 7 - Visual representation of an RNN

RNNs are ideal for modeling **temporal data in space missions**. One use is in processing spacecraft telemetry or health monitoring over time: an RNN can learn to predict future sensor readings or detect temporal anomalies by understanding the sequence of past observations. Another application is **signal and communication decoding**: RNN-based decoders (e.g. recurrent decoders for error-correcting codes) can process sequential channel data. In robotics and control, RNNs (especially LSTMs) can be used for **trajectory prediction** or adaptive control of spacecraft. For example, RNNs can learn the dynamics of a free-flying robot arm and improve path planning. In general, any space problem involving time series – from analyzing time-varying climate data from satellites to recognizing spoken commands in astronaut interfaces – can leverage RNNs[22].

3.1.13 Generative Adversarial Networks (GANs)

Generative Adversarial Network is a framework for training generative models via a game between two neural networks. GANs consist of two main models:

- Generator: This model generates new data instances.
- Discriminator: This model evaluates the generated data and classifies it as real (from the original dataset) or fake (generated by the generator).

The generator and discriminator are trained together in a zero-sum game. The generator aims to produce data that can fool the discriminator, while the discriminator tries to distinguish between real and fake data. This adversarial process continues until the discriminator is fooled about half the time, indicating that the generator is producing realistic data.

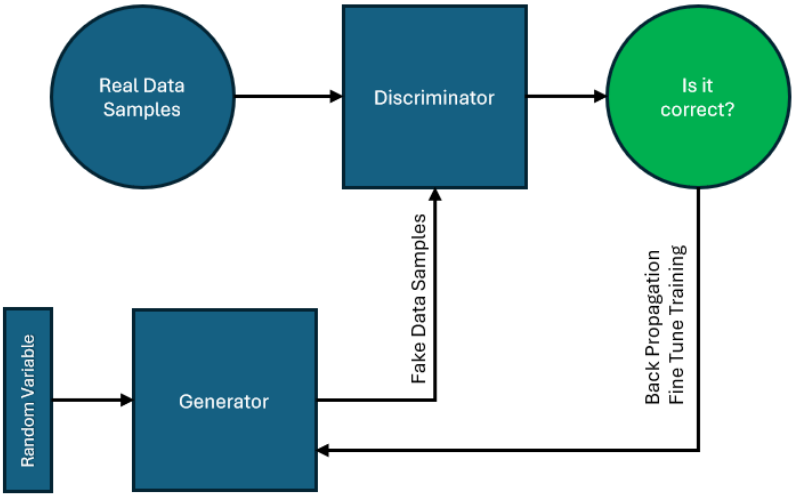

Figure 8 - Visual representation of a GAN

GANs have emerged as powerful tools for generating and enhancing space imagery. A notable use is in **satellite image super-resolution**: researchers have adapted GANs to increase the resolution of satellite photos while preserving radiometric accuracy[23]. In that work, a GAN was trained to sharpen small-scale features in diverse satellite images, yielding more detailed Earth observations. GANs can also generate synthetic training data for space systems, for example, creating realistic-looking star fields or space object images for training detection algorithms. In astronomy, GANs can help simulate galaxy images or predict missing parts of data. Any domain requiring high-fidelity synthetic data (planetary surfaces, telescopic images) can benefit from GANs. Moreover, GANs can be used in data augmentation for rare-event detection (e.g. generating examples of rare astronomical phenomena).

3.1.14 Genetic Algorithms

Genetic Algorithms (GAs) are population-based optimization techniques inspired by natural evolution. A GA maintains a population of candidate solutions (encoded as chromosomes, often bit-strings or parameter vectors). Through iterative application of genetic operators, **selection** (choosing fitter individuals), **crossover** (recombining parts of two parents), and **mutation** (randomly flipping bits), the population evolves toward better solutions to a given fitness function[24]. Over generations, good solutions propagate and overall fitness improves.

GAs have a long history in aerospace for solving design and optimization problems. A famous example is the **evolved antenna** for NASA's Space Technology 5 mission: researchers used a genetic algorithm (and a genetic-programming variant) to automatically design an X-band spacecraft antenna with wide beamwidth and bandwidth. The GA-produced design matched the performance of a hand-crafted

antenna, demonstrating "human-competitive" results[24]. Other uses include **trajectory optimization** for interplanetary missions: GAs can search complex, multi-modal solution spaces to find fuel-efficient transfer orbits. They have also been applied to optimize spacecraft control parameters, sensor placement, and scheduling tasks. Anywhere a complex engineering design or schedule is sought (often with discrete choices), GAs offers a flexible search strategy.

3.1.15 Graph Neural Networks (GNN)

Graph Neural Networks extend neural network models to data represented as graphs. In a GNN, data are given as a graph with nodes and edges, and each node and/or edge may have feature vectors. GNNs learn to propagate and transform information across the graph: each node's representation is updated by aggregating features from its neighbors through neural layers. This allows the network to learn patterns on irregular structures[25].

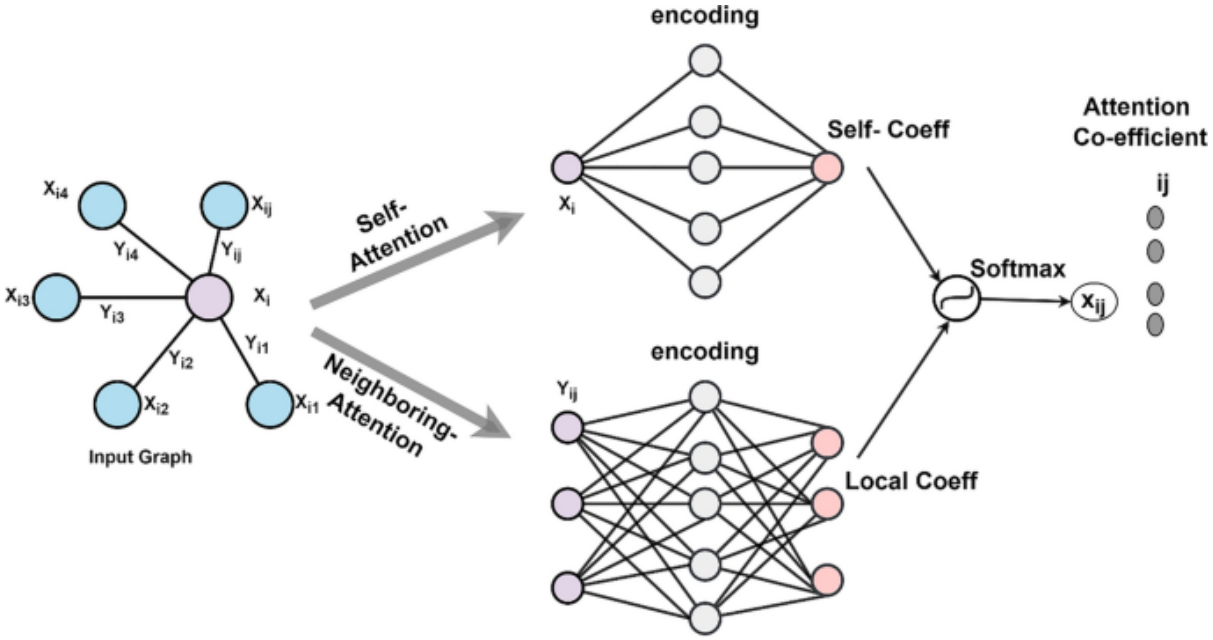

Figure 9 - Visual representation of a GNN

GNNs are well-suited to any space problem with relational data. For example, a network of satellites or sensors can be modeled as a graph and optimized using GNN-based methods. In astronomy, researchers have begun to represent cosmic structures (like galaxies and their halos) as graphs to apply GNNs for tasks such as inferring galaxy properties. A GNN could also model an **orbit determination network**: each node is an object in space (satellite, debris) and edges encode interactions or proximity; the GNN could learn to predict collision risks. More generally, GNNs could analyze communication networks among spacecraft (edges = links) to optimize routing. In summary, whenever data have an inherent graph

structure (networks, relations, molecules), GNNs allow those structures to be directly exploited[25].

3.2 ESA's work

European Space Agency (ESA) missions increasingly incorporate onboard artificial intelligence (AI) to enhance autonomy, data processing, and decision-making. In particular, ESA satellites now run neural networks and other machine learning algorithms in flight, enabling real-time analysis of sensor data and autonomous control. For example, ESA emphasizes that in-mission AI can make Earth observation "smarter and more efficient," by processing imagery onboard and downlinking only essential information[26], [27]. The following subsections describe key ESA initiatives that apply AI onboard spacecraft, including the Φsat cubesats, the OPS-SAT technology demonstrator, the Hera asteroid mission, and the ClearSpace-1 debris-removal mission. Each subsection covers mission purpose, AI methods, technical implementation, mission goals, and results.

3.2.1 Φsat (PhiSat-1 / PhiSat-2)

The Φsat (PhiSat) missions are small cubesat Earth-observation demonstrators explicitly built to validate onboard AI processing. Φsat-1, flown in 2020 as part of the FSSCat mission, carried a hyperspectral imager and an AI processor. Its goal was to use on-board inference to filter cloud-covered images and reduce data downlink[28]. Φsat-2 (launched August 2024) is a 6U cubesat that further demonstrates AI for Earth monitoring[27]. Both missions aim to show that embedding AI near the sensor ("at the edge") yields more efficient, timely data products, aiding applications such as disaster response and environmental monitoring.

The Φsat platforms use deep learning methods (convolutional neural networks and generative models) implemented on specialized vision processors. For example, cloud-detection on Φsat-1 and Φsat-2 is performed by a CNN classifier running on an Intel Movidius Myriad VPU[28], [29]. Φsat-2's software architecture (NanoSat MO Framework) supports multiple AI "apps" uploaded in orbit. Demonstrated algorithms include: a CNN to mask clouds, a CycleGAN to transform multispectral images into street maps (the Sat2Map app)[30], and deep neural networks for maritime vessel detection[30]. Other planned apps perform tasks such as image compression, wildfire detection, and detection of anomalies in ocean color. In all cases, inference is performed onboard in near-real time.

Technical Implementation: Φsat satellites carry a multispectral camera (7 bands from visible to near-IR) coupled with an onboard computer equipped with an Intel Myriad VPU[31]. On Φsat-1, the Myriad II VPU ran a CNN that segmented clouds in hyperspectral frames, discarding cloudy images before downlink[28]. Φsat-2 uses

an updated Myriad 2 VPU and a more powerful on-board CPU. AI apps are managed through the NanoSat MO framework, allowing ground operators to upload or update models remotely. In practice, each app is a small neural network (often quantized to 8 bits) that processes incoming images. For example, KP Labs implemented a CNN model with 8-bit quantization on Φsat-2 to automatically detect and drop cloud-covered images[29]. Other groups prepared models (e.g. adversarial-networks for map generation) and validated them on simulated or precursor data before uploading to the flight unit. The architecture is optimized for low-power, high-throughput inference: the Myriad chip can deliver thousands of billions of operations per second for vision workloads, enabling CNNs to run on the small satellite[31].

Φsat-1 and -2 are part of ESA's Φ-Lab R&D program to seed new applications of AI in space. Φsat-1 flew on the FSSCat constellation in 2020 and proved the concept of onboard cloud filtering[28]. Φsat-2, developed by ESA with Open Cosmos as prime contractor, carried six AI apps at launch (cloud detection, map generation, vessel detection, image compression, etc.), with more to be uploaded after commissioning[32]. The objectives are to validate that these apps improve data utility: for example, the cloud app ensures only clear-sky images are downlinked, while Sat2Map can deliver navigation maps in near–real time during disasters[30], [32]. By fusing on-board processing with a compact satellite design, Φsat enables a "smarter" Earth-observation approach, reducing bandwidth needs and latency in obtaining insight[26], [33].

Outcomes and Expected Benefits: Φsat-1's AI cloud-filtering succeeded in reducing downlink load and proving the Myriad chip's space readiness[28]. The ongoing Φsat-2 mission is expected to further quantify benefits: by running AI in orbit it can send only extracted information (cloud masks, vessel locations, map vectors, etc.), rather than raw images[27]. This yields faster turnaround for critical products (e.g. disaster maps), and conserves power and bandwidth. Preliminary reports confirm that Φsat-2 has activated its AI apps and is performing as designed. In summary, the Φsat series demonstrates that onboard AI – using methods from CNNs to generative nets – can make small satellites far more "intelligent", paving the way for future operational systems with real-time image interpretation[26], [31].

3.2.2 OPS-SAT

OPS-SAT is an ESA experimental CubeSat (3U) launched in 2019 as an "open on-orbit laboratory" for flight software and control experiments[34]. Its primary goal is to validate novel algorithms and architectures in space. While not designed for a specific scientific mission, OPS-SAT has become a platform for many AI demonstrations, leveraging its unusually powerful on-board computer and flexible software environment[34]. In essence, OPS-SAT enables developers to try new autonomy, image-processing, and control techniques in actual orbit conditions, which would be too risky on an operational spacecraft.

A wide range of AI techniques have been tested on OPS-SAT. These include supervised deep learning (e.g. convolutional neural networks for vision tasks), reinforcement learning (for control), recurrent networks for policy adaptation, and even binary networks optimized for space deployment. Example use cases include on-board image enhancement (super-resolution), object segmentation, autonomous feature tracking, and adaptive attitude control. In several projects, well-known architectures (YOLO for object detection, DQN for RL, CycleGAN for image-to-image mapping) have been implemented on OPS-SAT's hardware.

Technical Implementation: OPS-SAT's on-board computer is an Altera Cyclone V SoC, which combines an ARM Cortex-A9 dual-core CPU with an FPGA fabric[34]. It runs Linux and allows experimenters to upload code in C/C++ or Java. For AI workloads, two hardware elements are commonly used: the ARM CPU (for general-purpose tasks and running ML frameworks) and the FPGA (for accelerating or hosting neural nets). For instance, the "DeepCube" experiment ported a binary CNN (a convolutional network with binary weights/activations) onto the Cyclone-V FPGA, achieving real-time forest segmentation in orbit[35][36]. In another example, the "multi-frame super-resolution" project had OPS-SAT capture burst images and run a deep neural network to reconstruct higher-resolution scenes, improving image detail without ground processing[37].

To streamline experimentation, ESA supported frameworks like SaasyML on OPS-SAT. SaasyML provides "Machine Learning as a Service" on the satellite: it reuses an open-source Java ML library (JSAT) so that multiple users can run training and inference onboard without re-writing low-level code[38]. Over 100 algorithms (classification, regression, clustering) are made available via this service architecture. For vision tasks, developers have also implemented standard networks. For example, an autonomous tracking experiment used a YOLO-based detector on the ARM core to find islands in OPS-SAT's images, then a Deep Q-Network (DQN) agent (an RL approach) to command the satellite's reaction wheels and keep the target centered[39]. In sum, OPS-SAT's hardware and OS enable flexible deployment of AI code: kernels can run on CPU or be synthesized into the FPGA, and developers can tap Linux libraries and frameworks as on a terrestrial computer.

OPS-SAT is fundamentally a technology demonstration platform. Its 'mission' is to accept user-proposed experiments (through ESA's Open Space Innovation Platform) and validate them in orbit[34]. Projects range from optical image processing to autonomous control to cyber-security. In the AI domain, the goals have been to quantify how much machine learning can improve payload performance or spacecraft autonomy. For example, engineers have used OPS-SAT to test whether adaptive AI controllers can hold more accurate pointing than traditional fixed-control laws, and whether on-board super-resolution can overcome the physical limits of a small camera.

Outcomes and Results: OPS-SAT has demonstrated numerous successful AI proofs-of-concept. The super-resolution experiment (by OHB Hellas/FORTH)

confirmed that a neural network running in flight can enhance spatial resolution by a significant margin[40]. The forest-segmentation (DeepCube) test proved that a carefully quantized CNN can run on the FPGA at low power and produce accurate semantic maps[35]. The active-tracking experiment showed that a vision-based RL controller could continuously recenter a target in the frame without human intervention[39], [41]. Similarly, the HOPAS study (Airbus) demonstrated in simulation that an RL-enhanced controller could improve pointing accuracy by an order of magnitude[42]; this algorithm is being readied for flight tests on OPS-SAT. In general, OPS-SAT has raised the technology readiness of on-board AI: many techniques have moved from lab experiments to in-orbit validation. These outcomes build confidence that future operational satellites can safely run similar AI workloads, enabling smarter sensors and more autonomous spacecraft.

3.2.3 Hera (Asteroid Planetary Defense Mission)

ESA's Hera mission (launched October 2024) is a planetary defense probe targeting the binary asteroid system Didymos/Dimorphos. Hera's primary science goal is to survey the impact crater on Dimorphos made by NASA's DART mission. Uniquely, Hera is also a technology demonstration of deep-space autonomy: it must navigate very close to small bodies without relying on Earth control. Autonomous navigation is critical for Hera's proximity operations, so ESA equipped it with advanced on-board vision systems and AI-like processing to steer the spacecraft and acquire science data.

Hera's autonomy centers on computer-vision and feature-tracking algorithms. Prior to the asteroids, Hera performed a gravity-assist flyby of Mars (March 2025) to test its navigation system[43]. During this flyby, Hera's on-board software autonomously identified dozens of new surface landmarks (craters, albedo features) in successive images and tracked them as Mars moved out of view[43]. Although not explicitly deep learning, these algorithms use pattern-recognition and optical navigation techniques that could be augmented with neural networks. In effect, Hera's system solves a kind of simultaneous localization: by imaging multiple landmarks and computing their relative motion, the spacecraft derives its position and velocity without ground intervention.

Technical Implementation: Hera carries the Asteroid Framing Camera and a dedicated Image Processing Unit (IPU) developed by GMV. The IPU is a hardware accelerator (two custom FPGAs) that performs onboard vision tasks[43]. The spacecraft's main flight computer includes a separate "co-processor" (akin to a GPU) for these calculations. The feature-tracking software running on the IPU uses algorithms developed by GMV: it can acquire up to 100 surface features per image, select a subset for tracking, and update navigation solutions in real time. These computations run continuously during approach, entirely autonomously. The system does not require prior maps; it can latch onto unknown landmarks on the fly. For example, in the Mars flyby test, Hera acquired a new image every 48 seconds and

immediately matched features with its internal model, all without human commands[43].

Hera's autonomy is needed because Didymos and Dimorphos are small, irregular bodies with unknown surface features. As Hera approaches within ~30 km, it must keep Didymos in frame for broad navigation, then switch to "centroid" and eventually feature-relative navigation as it nears Dimorphos. Ultimately (below ~2 km) the spacecraft will rely entirely on onboard image processing to gauge altitude and attitude by comparing successive images of the asteroid's surface[24]. The Mars flyby (March 2025) served as a rehearsal: Hera passed within ~5700 km of Mars, autonomously acquired landmarks it had never seen, and successfully tracked them[43]. This proved the "self-driving" navigation system under real deep-space conditions, increasing confidence for the critical asteroid approach. ESA notes that this approach is unprecedented: unlike NASA's OSIRIS-REx (which needed prior maps for autonomous moves), Hera's system starts from scratch without preloaded landmark databases[43].

Outcomes and Expected Benefits: The in-flight tests on Hera have already validated the vision-based autonomy. During the Mars flyby, the system "immediately managed to acquire features … and continue tracking them". All tested features remained in view, confirming robustness. ESA reported that this success "gives us high confidence" for Hera's asteroid operations. On final approach, Hera will produce a real-time navigation solution entirely on board, enabling unprecedentedly precise and safe maneuvers around Dimorphos. In addition to spacecraft guidance, the IPU can accelerate data processing of the hyperspectral payload, potentially applying on-board AI to flag interesting materials. Overall, Hera demonstrates that vision-based AI-like processing can enable fully autonomous exploration and defense missions in deep space.

3.2.4 ClearSpace-1 (Debris Removal)

ClearSpace-1 (planned launch 2029) is ESA's first active debris-removal mission. Its goal is to rendezvous with ESA's defunct 95-kg PROBA-1 satellite in LEO and deorbit it, demonstrating the technology needed for removing space junk. A key challenge is that PROBA-1 is uncooperative (no rendezvous beacon or grappling fixture). Thus, the servicer must autonomously detect, approach, and capture the target using onboard sensors and control algorithms.

The mission relies on vision-based guidance and navigation algorithms. Though detailed algorithms are proprietary, the servicer is equipped with multiple cameras and radar to perceive the target[44]. These sensors feed image-processing software that can recognize and localize the target's body in real time. In principle, this could use convolutional neural networks for object detection and pose estimation, although formal publications describe it as "vision-based navigation." The capture maneuver uses a combination of control laws and possibly AI planners to position the spacecraft such that its tentacles can grab the target.

Technical Implementation: According to ESA sources, ClearSpace-1's servicer carries several vision sensors (wide-field and narrow-field cameras) and a radar for close-range navigation[44]. Processing is done on a dedicated avionics suite with high-performance computing elements. The spacecraft also has four articulated robotic arms ("tentacles") designed to embrace and secure the target from multiple sides. The navigation software must fuse the sensor inputs and command the servicer's thrusters to match velocity and orientation with PROBA-1. This likely involves an onboard image-matching pipeline; for example, algorithms may identify edges or features on PROBA-1's body and compute relative pose. Development of these algorithms was informed by simulations (e.g. ESA's PANGU tool for simulating visual scenes) and hardware-in-the-loop testing, but the flight software will operate independently once deployed.

ClearSpace-1 is a commercial-influenced mission (led by startup ClearSpace in partnership with ESA and OHB) under the Clean Space initiative. Its objective is twofold: to demonstrate that a previously launched, uncontrolled satellite can be autonomously captured and removed, and to kick-start an industrial sector for debris removal. The mission profile involves a commissioning phase, orbit-raising to PROBA-1's altitude, a far-range rendezvous, and a final capture sequence[44], [45]. All of these phases will be conducted with minimal ground intervention; the onboard autonomy must handle uncertainties in target motion and environment. The use of multiple cameras and AI-like algorithms allows the servicer to adjust its approach dynamically, ensuring safety.

Outcomes and Expected Benefits: ClearSpace-1 is still in development, but its technology stack is largely defined. ESA reports that the mission will remove PROBA-1 "as the first-ever mission to remove an unprepared and uncooperative object from orbit"[45]. By employing vision-based autonomous guidance and a novel four-arm capture mechanism, the mission will validate that a spacecraft can autonomously rendezvous with and grasp debris. Success will prove the feasibility of on-orbit servicing and debris removal, potentially using similar onboard AI techniques for future missions. In addition, the project drives innovation in autonomy, since the algorithms developed for ClearSpace-1 could be re-used for capturing other objects or for robotic servicing tasks.

3.2.5 Relevance to the Objectives of This Work

The initiatives described above illustrate a clear strategic trend: artificial intelligence is no longer a peripheral research topic within ESA, but a core enabler of autonomy, efficiency, and mission resilience. From onboard image processing in Φsat to autonomous navigation in Hera and debris capture in ClearSpace-1, AI systems are increasingly embedded directly into critical spacecraft operations. This evolution reinforces the importance of developing robust, secure, and operationally viable AI solutions within the space sector.

Within this context, the objectives of the present internship gain additional relevance. While the ESA missions discussed focus primarily on onboard autonomy and vision-based processing, they share common challenges with the tools developed in this work: reliability under constraints, secure deployment, computational efficiency, and controlled integration into mission-critical workflows. The development of a secure RAG-based chatbot and an LLM-assisted code analysis tool addresses complementary dimensions of AI integration, not at the spacecraft hardware level, but within the software engineering lifecycle that supports such missions.

Therefore, this chapter serves not only as an overview of ESA's current AI applications, but also as justification for the broader objective of this thesis: contributing to the structured, secure, and efficient integration of AI technologies into space system development processes. By understanding how AI is already transforming spacecraft operations, it becomes evident that equivalent innovation must also occur in the supporting software engineering environment, which formed the primary focus of this internship.

3.3 AIDA (AI-powered Digital Assistant)

Following the analysis of ESA's broader AI initiatives, it became relevant to examine how artificial intelligence could also support earlier phases of spacecraft system development, particularly within systems engineering activities. In this context, the AI-powered Digital Assistant (AIDA) emerged as a relevant case study. Unlike onboard AI applications focused on autonomy or perception, AIDA targets model-based systems engineering (MBSE) processes, aiming to enhance requirement management, traceability, and design validation through AI-assisted analysis.

The exploration of AIDA was therefore relevant to the objectives of this work, as it provided insight into how AI can be integrated into structured engineering workflows rather than purely operational spacecraft functions. Given that one of the core goals of this internship was to investigate practical mechanisms for embedding AI within the software and systems engineering lifecycle, AIDA represented a concrete example of AI supporting high-level engineering decision-making. AIDA can perform three main functions:

- **Requirement flow/down (UC1):** this Use Case involves users seeking to identify automatic links between lower-level requirements and components, as well as detecting potentially problematic requirements during requirements flow down. Additionally, during requirement traceability review, users may propose additional requirements in either an upward or downward flow and suggest verification methods based on similar missions when necessary.

- **Design drivers for components and functions based on previous projects (UC2):** This Use Case allows users to select a partial or complete specification and design of their system, including requirements, functions, logical components, physical components, and traceability between these elements. The tool proposes elements to be added or updated to the user's design, and the user can browse and accept the proposals while maintaining traceability to the source of the suggestion. It also allows users to select a particular type of artifact for which they want recommendations and suggests elements with attached information and browse suggestions for added or updated properties on their design elements.
- **Automatic model verification based on associated requirements (UC3):** This Use Case involves a tool that enables users to select a set of specifications or a single specification and a design of their system or part of the design. The user should be able to browse the requirements that imply model elements not found by the tool's assistant. The overall aim of this Use Case is to provide users with a means to detect and address any inconsistencies or missing elements in their system or design, which can improve the overall quality and effectiveness of the product.

3.3.1 Use-case 1: Requirement Flow/down

a) UC1-US1: Proposing new traceability links from a given requirement to a given "container"

This User Story describes the requirement for a traceability tool that would allow users to select a requirement and one or various containers and then browse proposed trace links between the two. The proposed trace links should be ordered from most to least probable, and the user should be able to accept or reject a proposal.

In this US1 a Deep Neural Network will be used, to do a binary classification on requirements. This means the requirements will be sorted into two groups: traced and not traced. Traced means there's a logical link between two requirements, where one requirement provides more context or detail about the other. To do this, a pre-trained language encoder from the BERT-like language model family will be used to encode the concatenated requirements, then this will be used to perform the classification.

Another important step for these user stories was to create a balanced dataset that includes negative samples. Therefore, heuristics were used to curate negative examples. For every positive couple (req1, req2), one negative example involving req1 and one negative example involving req2 were added to the dataset to balance it. The negative examples were chosen from the same specification level as the positive ones because they could not be traced together in theory. As for the pre-

processing of the dataset, the dataset was shuffled and then split into training and test (80/20)

b) UC1-US2: Proposing new traceability links from a given "container" to a given "container"

Just like US1 this User Story describes the requirement for a traceability tool that would allow users to select a pair of containers from their project and browse pairs of requirements proposed by the assistant. The proposed pairs of requirements should be ordered from most to least probable, and the user should be able to accept or reject a proposal.

As for its methodology, it is the same for US1.

c) UC1-US3: Detecting suspicious trace link for a given "container"

This User Story describes the requirement for a traceability tool that would allow users to browse suspicious traces found by the assistant in their project, ordered by probability. The user should be able to filter these traces by container and export the list to a TBC (To Be Confirmed) format. Additionally, the user should be able to discard false positive trace problems, which means that the assistant will not propose them again in the future. Finally, the user should be able to review or amend the list of discarded false positive trace problems.

To detect if a requirement has a suspicion verification method, there was a need to develop an automatic system based on leveraging the model trained in US1 and 2 and US6. The steps performed by the prototype are:

1. Check if input is well posed.
2. Load trained NLP (Natural Language Processing) model.
3. Pass the input and target requirements to the NLP model and predict the probability of them being traced.
4. Output the trace probability

d) UC1-US4: From previous projects, propose requirements that are close from the current one in previous projects (at the same specification level)

This User Story would allow users to select a requirement from their project and browse requirements from previous projects that may be complementary to it. The user should be able to filter proposals by originating project or container and accept or reject a proposal.

To evaluate the similarity between requirements three types of similarity scores are evaluated:

- The **semantic similarity** is evaluated performing the cosine similarity between the embeddings of two requirements. The embeddings are numerical arrays obtained by encoding the requirement text using the 'all-mpnet-base-v2' Sentence BERT model.

- The **text and concept similarities** are evaluated using the elasticsearch score obtained after performing the concept recognition on the requirement's text.

The difference between the text and concept similarities lies in what aspect of the recognized concepts is used. A word is recognized by the concepts recognition model by creating a dictionary for the word containing the word itself and a label, which states to which broader category the word belongs. For example, the word antenna is seen as {word: 'antenna', label: Telecom}. To evaluate the text similarity, the words are used as concepts, while for the concept similarity, the labels are used as concepts.

The concept recognition is performed with the '**spacescibert_CR**' BERT model developed by the University of Strathclyde. The scoring function is defined as (with **q** being a specific concept and **d** the set of requirements)

$$score(q,d) = queryNorm(q) * coord(q,d) * SUM\ [tf(t\ in\ d) * idf(t)^2 * t.getBoost() * norm(t,d)]\ (t\ in\ q)$$

- **queryNorm:** depends only on the weights applied to the terms in the query, can be turned off
- **Coord:** counts the number of concepts from the baseline requirement that appear in the other requirement
- **tf:** term frequency. Is the square root of the number of times the term appears in the field of a document. The more times a term appears in a document, the higher its relevance should be.
- **Idf:** inverse document frequency. Is one plus the natural log of the documents in the index divided by the number of documents that contain the term. If many documents in the index have the term, then the term is less important than another term would be where few documents include the term.
- **t.getBoost():** weights applied to the different terms – NOT USED (set to value 1 to not weight differently any of the engineering concepts
- **Norm:** used to consider the number of concepts in the other compared document

e) UC1-US5: From previous projects, propose requirements that has already been traced to a similar given requirement (at lower specification level)

This User Story would allow users to select a requirement from their project and browse requirements from previous projects that are generally traced to it. The user should be able to filter proposals by originating project or container and accept or reject a proposal. When accepting a proposal, the user should be able to select a targeting container for the "new" requirements.

The methodology for US5 is the same for US4.

f) UC1-US6: Propose verification method

This User Story allows users to select a requirement from their project and consult the verification method proposed by the assistant. The user should be able to accept or reject a proposal, and the exact outcome of accepting or rejecting a proposal is yet to be determined.

The methodology is to train a Deep Neural Network model called BERT + classification layer for multi-label classification. The classification problem includes three labels: TEST, REVIEW, and ANALYSIS. After performing the preprocessing steps to prepare the training set and perform the classification it was necessary to remove all the requirements that didn't have a labelled verification method from the dataset.

g) UC1-US7: Detecting suspicious verification method

This User Story describes a tool that would allow users to browse suspicious verification methods found by the assistant in their project, ordered by probability. The user should be able to filter these verification methods by container and export the list to a TBC format. Additionally, the user should be able to discard false positive verification methods, which means that the assistant will not propose them again in the future. Finally, the user should be able to review or amend the list of discarded false positive verification methods.

Just like US3 to detect if a requirement has a suspicion verification method, there was a need to develop an automatic system based by leveraging the model trained in US1 and 2 and US6. The steps performed by the prototype are:

1. Check if input is well posed.
2. Retrieve all requirements from knowledge graph.
3. Load trained NLP (Natural Language Processing) model.
4. Predict verification method for each of the requirements in the dataset.
5. Output n requirements with the most suspicious verification method. A verification method is considered suspicious if is present in the data but is predicted by the NLP with a low probability.

3.3.2 Use Case 2: Design drivers for components and functions based on previous projects

a) UC2-US1: Update of the topology

This User Story would allow users to select a partial specification and design of their system, including requirements, functions, logical components, physical components, and trace between these elements. The user should then be able to select a particular kind of artifact for which they want recommendations, such as completing the functional design or completing the physical design. The tool should then propose elements to be added to the user's designs, along with attached

information such as the project from which the component is coming from. The user should be able to accept the proposal and push it into their current design while keeping the traceability to the source of the suggestion.

The methodology for this user story involves identifying similar elements within a knowledge graph (KG) based on a given textual input and element type. The element types for this project are "PhysicalComponent", "LogicalComponent", "PhysicalFunction", and "LogicalFunction". Sentence embeddings were computed for these types and saved in the KG during data insertion. To find similar elements, the input text is encoded using the same sentence embedding model and compared to the embeddings of elements in the KG using cosine similarity. Once the most similar entity is identified, a breadth-first search algorithm is used to find neighboring nodes with specific element types. This approach efficiently retrieves relevant information from the KG and expands the architectural topology of a new mission.

b) UC2-US2: Update/Create new properties

This User Story allows users to select a specification and design of their system, including requirements, functions, logical components, physical components, and trace between these elements. The user should be able to browse suggestions for added or updated properties on their design elements. The user should be able to accept the proposal and push it into their current design.

The second user story aims to identify the most similar "PhysicalComponent" in the KG and derive design parameters from it. This process starts by encoding a given textual input using a sentence transformer to generate an embedding. The embedding is then compared to the embeddings of other "PhysicalComponent" names saved in the KG using cosine similarity to find the most similar entity. Currently, only the attributes associated with the most similar entity are output.

3.3.3 Use Case 3: Automatic model verification based on associated requirements

a) UC3-US1: The requirements and the model are contradictory.

This User Story describes the requirement for a tool that would allow users to select a set of specifications or a single specification and a design of their system or part of the design. The user should be able to browse the couples (requirement and design element/set of design element) that the assistant qualifies as contradictory.

b) UC3-US2: The specification implies a design element that is not found in the current design.

This User Story describes the requirement for a tool that would allow users to select a set of specifications or a single specification and a design of their system or part of the design. The user should be able to browse the requirements implying model elements that are not found.

c) UC3-US3: The design implies some model elements that are not found in the current specification set.

This User Story describes the requirement for a tool that would allow users to select a set of specifications or a single specification and a design of their system or part of the design. The user should be able to browse the model elements implying requirements that are not found by the assistant

3.3.4 AIDA's testing phase

During the testing phase, however, the tool consistently failed to operate as intended. The application repeatedly produced errors, particularly related to the database component on which its internal reasoning mechanisms depend. This persistent malfunction prevented the execution of any of the three use cases in a complete or meaningful manner. As a result, the tool could not be evaluated in real engineering scenarios, nor could its AI-based analytical capabilities be validated in practice.

To address these issues, a meeting was held with ESA's chief officer responsible for the AIDA project. Despite his extensive knowledge of the system, he was also unable to resolve the technical problems encountered during testing. During the meeting, he clarified that AIDA was never intended to function as a fully operational engineering tool in its current form. Instead, it had been developed as a research prototype, primarily to assess and compare AI algorithms and techniques for their potential applicability in systems engineering workflows. Its purpose was therefore exploratory rather than operational, serving as a proof of concept rather than a deployable solution.

In light of these observations, it is concluded that AIDA cannot yet be employed as a functional assistant for systems engineers. Although conceptually promising and demonstrating ESA's commitment to exploring AI-enabled engineering methodologies, the tool remains immature and not technically ready for practical use. Its present limitations, particularly the unstable database layer and incomplete implementation of the intended functionalities, restrict its applicability to experimental and research contexts only. A fully operational version would require substantial development, testing, and refinement before it could be integrated into real mission design processes.

4 IMPLEMENTATION

The implementation phase of my internship focused on developing two complementary software solutions aimed at improving efficiency, knowledge accessibility, and code quality within the aerospace software engineering team. Both tools address distinct challenges encountered in the development lifecycle, yet they share a common objective: enhancing productivity through automation and intelligent assistance while respecting the strict confidentiality and reliability requirements inherent to aerospace projects.

The first tool, SpaceSage, is a secure, Retrieval-Augmented Generation (RAG)–based knowledge assistant designed to streamline the process of locating and understanding project documentation. Operating entirely on-premises and powered by locally hosted large language models, SpaceSage enables team members to query internal documents, such as requirements, architecture descriptions, and verification procedures in a unified and traceable manner. Its purpose is to reduce the time spent manually searching repositories and to provide consistent, grounded answers that comply with confidentiality constraints.

The second tool, FixFlow, addresses a different but equally significant challenge, addressing static analysis violations in source code. FixFlow introduces an automated, AI-assisted workflow that integrates SonarQube defect reports with Azure OpenAI to generate contextualized and standards-aware code fixes. These proposed corrections are then applied and verified through a Git-based cycle supported by Jenkins pipelines, which handle compilation, testing, and validation. By accelerating the correction process while maintaining developer oversight, FixFlow enhances code quality and reliability within modern DevOps practices.

Together, SpaceSage and FixFlow represent the practical outcomes of the implementation work carried out during the internship. The following subsections describe their motivations, design principles, architecture, and evaluation results in detail.

4.1 SpaceSage

SpaceSage represents one of the principal outcomes of the internship and was conceived as a direct response to the challenges of managing and retrieving knowledge within an aerospace software engineering environment. As projects grow in scale and technical complexity, the corresponding volume of documentation, covering architecture, verification, requirements, and domain-specific standards, expands significantly. In this context, efficiently navigating and cross-referencing this material becomes critical not only to sustain engineering productivity, but also to ensure traceability and compliance with the stringent quality and certification

standards that characterize aerospace software development. Yet, despite its importance, accessing the right information at the right moment is often slow and fragmented, particularly when strict confidentiality rules prevent the use of cloud-based AI assistants capable of accelerating such tasks.

The subsections that follow present the rationale behind the system's development, the architectural and methodological choices involved, and the evaluation process used to identify the most suitable model configurations. Together, they demonstrate how SpaceSage serves as a scalable and secure solution aligned with the documentation-intensive nature of aerospace software engineering.

4.1.1 Background, Motivation and Problem Statement

SpaceSage was developed within an aerospace software engineering setting where teams routinely rely on extensive, multilayered technical documentation. These materials, including system architecture descriptions, verification procedures, and requirement specifications, play a central role in guiding daily development activities. Their volume and complexity, however, present significant challenges. Documentation is frequently updated, spans large page counts, and is subject to stringent confidentiality constraints that prohibit the use of external, cloud-based LLM tools such as ChatGPT, Gemini, or Copilot. As a result, although modern AI tools could theoretically accelerate information retrieval, they cannot be adopted in environments where sensitive technical details must remain fully isolated from external infrastructures.

During the internship, the impact of these constraints was evident. Team members, ranging from developers to interns, often spent substantial time manually searching internal repositories, version-controlled documents, and technical manuals to resolve relatively routine questions. These inefficiencies were amplified by the nature of aerospace documentation, in which critical information is frequently distributed across multiple standards, requirement sets, or subsystem descriptions. Even when the required information existed, locating it quickly and reliably remained a persistent challenge.

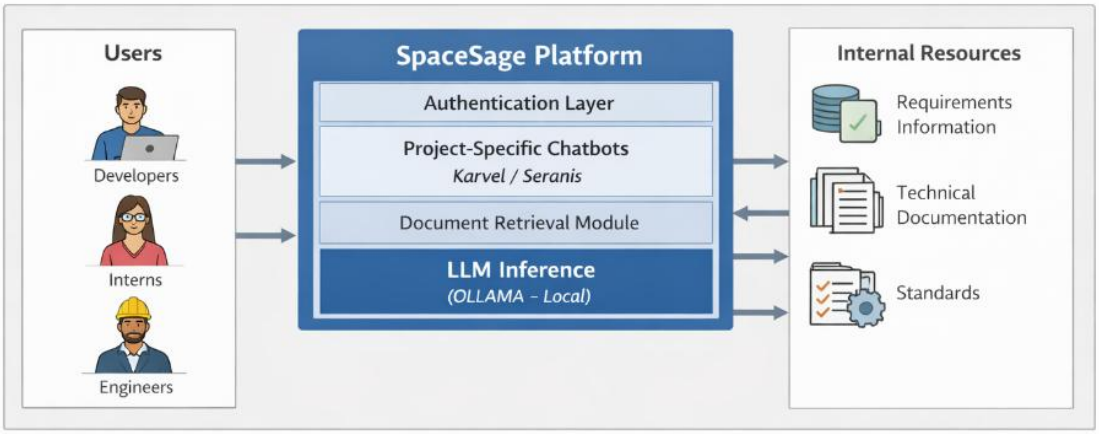

Figure 10 - Spacesage's Overview

SpaceSage was conceived to address these limitations by providing a centralized, confidential, and intelligent system capable of retrieving information directly from internal resources. By combining document retrieval with locally deployed LLM inference via OLLAMA, the system ensures that no data leaves the controlled environment, fully respecting security requirements while delivering AI-enhanced search capabilities. The solution also needed to accommodate multiple parallel aerospace projects, each with its own access-restricted documentation. To address this, SpaceSage includes dedicated chatbots for Karvel and Seranis, protected by authentication mechanisms that enforce project-specific confidentiality boundaries. In summary, the problem SpaceSage addresses emerges from the intersection of three organizational realities:

1. **strict confidentiality requirements**
2. **high documentation volume**
3. **the need for fast and accurate information retrieval**.

SpaceSage positions itself as a solution that respects all three constraints, improving team efficiency without compromising security.

4.1.2 Model Evaluation and Experiments

After implementing an initial functional version of SpaceSage, a comprehensive evaluation phase was conducted to determine the optimal combination of language model, embedding model, and text splitting strategy. To support this process, synthetic question–answer (QA) pairs were generated using a scripted approach aligned with established RAG evaluation practices. From the larger dataset produced, a curated set of eighty QA pairs, representing realistic technical queries encountered by the aerospace team, these pairs were selected to serve as the standardized evaluation benchmark.

Four key metrics were used to evaluate the system's responses:

- **LLMContextRecall:** This metric evaluates the extent to which the language model successfully incorporates the relevant retrieved context into its generated response. In RAG systems, the retrieval component provides contextual documents that should ground the answer. A high context recall score indicates that the model effectively utilizes the retrieved information rather than ignoring it or generating responses based solely on prior training knowledge. This metric is particularly important for confidentiality-sensitive environments, as it ensures that answers are derived from internal documentation rather than external pre-trained knowledge.[46]
- **Faithfulness:** Faithfulness measures the degree to which the generated answer remains consistent with the retrieved source content. In other words, it evaluates whether the model introduces unsupported claims or hallucinates information beyond what is present in the provided documents. For enterprise and aerospace applications, this metric is critical, as hallucinated technical information could lead to incorrect engineering decisions.[46]
- **FactualCorrectness:** This metric assesses the factual accuracy of the generated response when compared against authoritative documentation. While faithfulness ensures alignment with retrieved passages, factual correctness verifies whether the overall answer accurately reflects the ground truth within the broader document corpus. This distinction is important because a model may faithfully reflect a retrieved chunk that is incomplete or partially relevant yet still fail to provide a fully correct answer.[46]
- **ResponseRelevancy:** Response relevancy evaluates how effectively the answer addresses the user's query. Even when an answer is factually correct and grounded in retrieved context, it may fail to directly respond to the user's intent. This metric therefore captures the alignment between the query semantics and the generated output, ensuring that responses remain concise, targeted, and useful in practical engineering scenarios. [46]

Together, these metrics provided a multi-dimensional assessment of grounding, precision, and reliability.

A major part of the evaluation focused on assessing different text segmentation strategies. The way text is split can greatly influence the retrieval of relevant context and the overall quality of the model's responses, for this study, I explored two main splitting methods provided by Langchain:

1. **Semantic Chunker:** This method uses statistical techniques to determine optimal chunk boundaries, ensuring that the resulting segments are semantically coherent. I tested several techniques within this approach:
    - **Percentile**
    - **Standard Deviation**

- **Interquartile**
- **Gradient**

2. **Recursive Character Text Splitter:** This method recursively splits the text based on character count, with configurable parameters to balance context retention and chunk size. The configurations tested were:
   - **chunk_size=2048 with chunk_overlap=512**
   - **chunk_size=1024 with chunk_overlap=512**
   - **chunk_size=512 with chunk_overlap=150**

These experiments demonstrated that chunking strategy directly influences retrieval quality, while larger chunks preserve context, they also increase noise, while smaller chunks improve precision but risk fragmenting information.

Once these segmentation strategies were defined, they were combined with a broad set of model and embedding configurations to perform a comprehensive benchmarking phase. Each splitter served as the foundation for evaluating multiple LLMs and embedding models, an approach reflected in the detailed results contained in the Model_Embeddings_Results.xlsx in the attachments.

The performance of every combination was assessed using the standardized set of eighty QA pairs, allowing for direct comparison across architectures, embedding techniques, and segmentation granularities. Across all these experiments, one configuration consistently outperformed the rest, the **Qwen2.5 7B model**, as seen in the Figure 11 - Qwen2.5 benchmark results, paired with the nomic-embed-text embedding model combined with the use of interquartile statistical breakpoints.

Embeddings: nomic-embed-text Modelo: qwen2.5-7b	context_recall	faithfulness	factual_correctness	answer_relevancy	Total
Gradient	0,7562	0,8625	0,711	0,8376	3,1673
Standard Deviation	0,825	0,8352	0,6262	0,8865	3,1729
Interquartile	0,825	0,896	0,6996	0,8854	3,306
Percentile	0,725	0,8554	0,6044	0,8363	3,0211
RecursiveCharacterTextSplitter(2048, 512)	0,7125	0,8079	0,5246	0,8294	2,8744
RecursiveCharacterTextSplitter(1024, 512)	0,6875	0,7971	0,5005	0,864	2,8491
RecursiveCharacterTextSplitter(512, 150)	0,6875	0,6853	0,5165	0,7989	2,6882

Figure 11 - Qwen2.5 benchmark results

This combination achieved the highest overall score of 3.306 out of 4, demonstrating superior grounding, factual accuracy, and contextual retrieval, while remaining computationally feasible on OLLAMA's CPU-only environment. Its consistent performance across different chunking strategies ultimately established it as the most robust and reliable choice for the SpaceSage system.

Overall, the evaluation process not only optimized SpaceSage's performance but also yielded critical insights into the interactions between model architecture, retrieval strategies, and document structure. These findings form a robust foundation for guiding future improvements to the system.

4.1.3 System Overview and Key Features

SpaceSage was implemented as a secure, modular RAG-based system designed to serve as a central information hub for the aerospace software development team. Its operation begins with the ingestion of internal documentation, primarily in PDF format, though requirement files and URLs can also be processed, ensuring that diverse project knowledge is captured within a unified retrieval framework. After ingestion, documents are converted into text, semantically segmented, vectorized, and embedded into a ChromaDB instance, which serves as the system's vector database.

To comply fully with confidentiality requirements, all inference is performed locally using OLLAMA-hosted large language models, this avoids reliance on cloud-based LLMs and ensures that sensitive documentation remains entirely within the organization's internal infrastructure.

SpaceSage's user-facing interface was developed using Streamlit, providing a simple and intuitive front end accessible to all team members. Features include:

- persistent chat history,
- Downloadable source documents
- URL ingestion for the general chatbot
- Project-specific chatbots

By consolidating multiple document repositories, restricting access through project-level separation, and providing grounded answers with explicit citations, SpaceSage offers a powerful and transparent knowledge retrieval platform for the team.

4.1.4 Architectural Overview

The architecture of SpaceSage integrates document ingestion, text segmentation, embedding generation, context retrieval, and response generation into a cohesive Retrieval-Augmented Generation (RAG) workflow, explicitly designed for confidentiality-sensitive environments. The overall system is structured to balance

accuracy, performance, and data security, ensuring that all processing occurs on-premise without exposing sensitive documentation to external services.

SpaceSage begins by ingesting technical documentation, this ingestion module performs text extraction and preprocessing, this step ensures that downstream modules receive clean and structured textual content suitable for semantic processing.

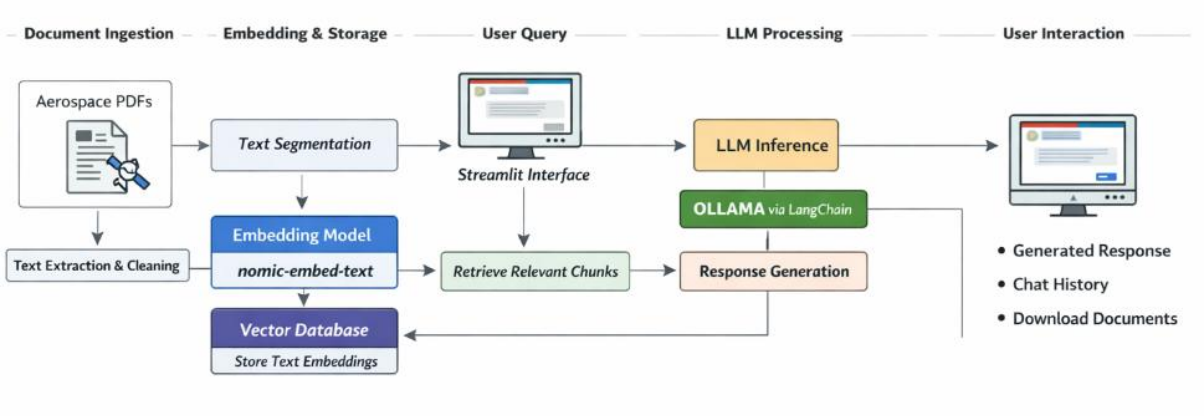

Figure 12 - SpaceSage's Processing Flow

Then the extracted text is split into smaller, semantically coherent chunks. The system supports both recursive character-based splitting and semantic chunking, allowing flexibility depending on document size and structure. Segmentation is critical in preserving context within each chunk, ensuring that the language model can interpret and generate accurate responses. Chunk size and overlap parameters are carefully tuned to balance information density with model input limits.

Each chunk is then converted into a vector representation using the nomic-embed-text embedding model. This embedding model was chosen after preliminary benchmarking for its superior performance in capturing semantic relationships compared to alternatives such as Azure's text-embedding-ada-002. Generated embeddings are stored in ChromaDB, a vector database optimized for high-dimensional similarity search. This enables fast retrieval of the most relevant document chunks in response to user queries.

When a user submits a query, SpaceSage first generates an embedding of the query, then retrieves the most semantically similar chunks from ChromaDB. Both the retrieved context and the query are fed into an OLLAMA-hosted LLM via LangChain. The LLM combines its pre-trained knowledge with the retrieved context to generate a grounded, contextually accurate answer.

The Streamlit interface provides a user-friendly front-end, supporting real-time interactions, session tracking, persistent chat history, and the ability to download referenced documents. Users can also pass URLs or additional files to expand the searchable knowledge base.

4.1.5 Limitations and Future Work

Despite its success, the development of SpaceSage highlighted several limitations, most notably, **the reliance on CPU-only hardware** affected inference speed and scalability. Modern LLMs perform significantly better when supported by GPU acceleration, however, the restricted hardware environment limited the size and speed of models that could be deployed. This occasionally resulted in latency and slowdowns under simultaneous usage.

While the architecture is robust and flexible, deployment on a CPU-only Lenovo machine revealed limitations under multi-user load. High concurrency resulted in increased latency, highlighting potential benefits of GPU acceleration or distributed deployment for larger teams or more complex queries.

Another limitation involved **citation accuracy**, although SpaceSage consistently produced well-grounded answers, the citation mechanism occasionally pointed to adjacent or semantically similar document sections instead of the exact originating chunk. These inconsistencies stem from chunking granularity and retrieval overlap, enhancing reranking algorithms or integrating hybrid retrieval methods may improve precision.

Scalability and automation also present opportunities for improvement, as of right now, new documents must be manually ingested and re-embedded. A continuous ingestion pipeline would streamline the process and ensure that the knowledge base remains up to date as project documentation evolves.

Future iterations of SpaceSage should focus on deploying the system on GPU-enabled hardware, improving retrieval ranking, supporting additional file formats, and integrating feedback loops that allow users to rate responses. Integration with internal communication platforms such as Teams or BitBucket could further enhance usability and adoption across the organization.

4.2 FixFlow

Maintaining high-quality source code is a critical aspect of software engineering, especially in safety-critical or compliance-heavy domains such as automotive, aerospace, and industrial systems. Traditional static analysis tools like SonarQube and compliance checkers based on MISRA or CERT standards provide insights into code quality, but the resolution of these issues often remains manual, time-consuming, and error prone.

This report introduces an AI-powered automation tool that bridges this gap by integrating static analysis findings with a Large Language Model (LLM), specifically Azure OpenAI, to propose contextualized and standards-aware code fixes. These suggestions are processed and applied to the source code via a Git-based cycle that

includes commit and verification checkpoints. Verification is conducted automatically using Jenkins pipelines that assess build integrity and execute unit tests.

The goal is to accelerate remediation of static analysis violations while preserving developer oversight and code correctness.

4.2.1 Background, Motivation and Problem Statement

Software maintenance accounts for approximately 60–80% of the total software lifecycle cost, with code quality issues contributing significantly to long-term technical debt. Static analysis tools such as SonarQube are widely adopted to detect code smells, bugs, and security vulnerabilities. However, they typically rely on manual developer intervention to resolve the identified issues.

Manual refactoring is often time-consuming, error-prone, and inconsistently applied, especially in large teams or across multiple repositories. This leads to several challenges in modern software development workflows:

- Accumulation of unresolved technical debt
- Inconsistent adherence to coding standards
- Reduced developer productivity
- Delays in software delivery cycles

The recent advancements in LLMs present an opportunity to automate code repair tasks that have traditionally required human oversight. By integrating static analysis with LLM-powered code transformation, we can streamline the resolution process, reduce human effort, and improve consistency across codebases.

To address these challenges, we developed a fully automated tool that integrates SonarQube analysis with LLM-based code repair. The tool is designed with the following principles:

- **Portability:** Implemented entirely in Python, with a single .env configuration file and CLI flags (--use-sonarqube, --no-sonarqube, --use-standards, etc.) for flexibility.
- **Configurability:** Supports toggling of SonarQube and compliance rule integration, customizable verification steps, and targeting of specific Git branches.
- **Non-destructive:** Includes backup and rollback mechanisms to ensure only validated changes are merged.

- **Extensibility:** Utilizes Tree-sitter grammars (with on-demand regeneration) and regex-based fallbacks to support multiple languages.
- **CI/CD Compatibility:** Offers seamless Jenkins integration, enabling immediate inclusion in existing pipelines without modifying existing Jenkinsfiles.

4.2.2 System Overview and Key Features

FixFlow is a modular, extensible system designed to automate the identification, correction, and validation of static code analysis violations. It integrates industry-standard tools such as SonarQube for defect detection, Azure OpenAI for code fixing generation, and Jenkins for continuous integration and validation. The system is highly configurable, language-agnostic, and designed to operate seamlessly within modern DevOps pipelines.

a) Architectural Overview

FixFlow integrates industry-standard tools such as SonarQube for defect detection, Azure OpenAI for code fixing generation, and Jenkins for continuous integration and validation. The tool is organized into six core subsystems that interact to form a full remediation pipeline:

1. **Static Analysis Engine**
2. **Input Parsing and Conversion**
3. **AI Fix Generation Engine**
4. **Fix Application and Version Control**
5. **Verification and CI Integration**
6. **Reporting and Metrics**

b) Key Features

FixFlow provides a wide array of capabilities that enhance automation, configurability, and safety:

- **LLM-Powered Code Fixing**
  Utilizes GPT-4 and GPT-3.5-turbo to generate rule-based code fixes. Each prompt is structured using a Jinja2 template that enforces strict input isolation and syntactic integrity.
- **Modular Verification Pipeline**
  Integrates directly with Jenkins to automate compilation and unit testing

after every applied patch. The tool supports multiple verification stages configured via .env.

- **Granular Error Recovery System**
  Supports **multi-level rollback** strategies including:
  - Line-level reversion
  - Function-level rollback
  - Complete undo of all changes
    This enables safe, non-destructive experimentation within production-grade repositories.
- **Dynamic Line Offset Tracking**
  Automatically adjusts line numbers during multi-line edits to maintain positional accuracy across sequential fixes.
- **Tree-Sitter Based Parsing**
  Allows robust, syntax-aware context extraction across multiple programming languages. Custom grammars can be rebuilt on demand to extend support.
- **Jenkins-Friendly CLI Interface**
  The tool can be executed via simple shell scripts (fixflow.sh) and accepts CLI flags such as:
  - --use-sonarqube
  - --no-sonarqube
  - --use-standards

  This enables seamless CI/CD integration with no need for Jenkinsfile modifications.
- **Comprehensive Logging and Reporting**
  Every AI interaction, fix outcome, and rollback events are logged. Output includes side-by-side diffs, HTML reports, rule-based metrics, and success statistics.

4.2.3 Architecture and Component Design

The internal architecture of the AI Code Repair Tool is organized around a set of cooperating components, each responsible for a specific stage in the automated repair pipeline. These components form a loosely coupled system that enables scalability, robustness, and clear separation of concerns. The architecture emphasizes traceability, verification, and safe rollback at every stage of operation, from static analysis parsing to patch generation, application, and CI-based validation.

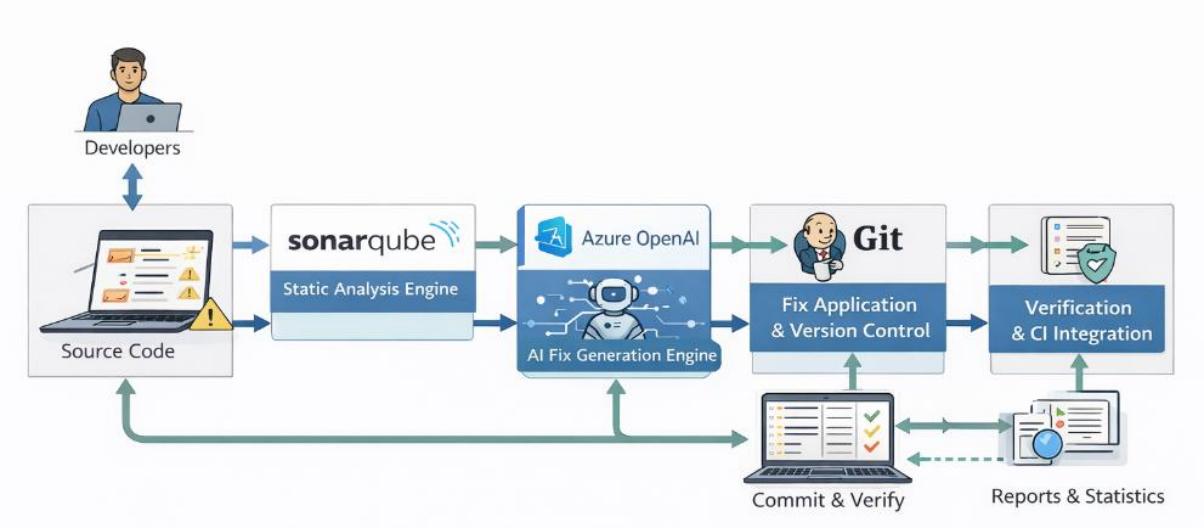

Figure 13 - FixFlow's Architecture

As for the architecture, which comprises, of the previously mentioned, six core subsystems: Static Analysis Engine, Input Parsing and Conversion, AI Fix Generation Engine, Fix Application and Version Control, Verification and CI Integration and Reporting and Metrics. Each of these modules is implemented as an independent, scriptable component that integrates through Git and Jenkins-based workflows.

a) Static Analysis Engine

The tool leverages **SonarQube** as its primary static analysis engine, which scans the target codebase and exports rule violations in a structured format (JSON or CSV). These reports can also include compliance findings from rule sets such as **MISRA**. Support for additional static analyzers (e.g., Cppcheck, Clang Static Analyzer) is also available through extensible input normalization.

- **Primary Tool:** SonarQube (Community, Developer, Enterprise)
- **Integration Mode:** RESTful API or file-based export
- **Output Format:** JSON/CSV violation reports

b) Input Parsing and Conversion

The sonarqube_converter.py module processes static analysis reports and merges them into a unified data structure (merged_violations.json). Optional compliance

references (e.g., MISRA lookups) are added using standards_lookup.py. The system also supports environment configuration through variables_configurator.py, allowing dynamic setup of compilation, unit testing, and analysis commands.

- **Key Modules:** sonarqube_converter.py, standards_lookup.py, variables_configurator.py and the .env file.
- **Inputs:** Static analysis output, compliance rule sets, environment variables
- **Outputs:** Normalized violation database

c) AI Fix Generation Engine

This is the core intelligence layer of the system, where violations are grouped by rule ID, and a Tree-sitter-based parser extracts relevant code contexts around each issue. The suggest_fixes.py module then orchestrates prompt generation, invoking Azure OpenAI or Ollama models to generate code fixes.

A strict context isolation policy is enforced using Jinja2-based prompt templates to prevent cross-contamination between unrelated fixes. Prompts include syntax balance checks, rule explanations, and structural constraints.

- **Models supported:** GPT-4, GPT-4-turbo, GPT-3.5-turbo, CodeLlama, Mistral
- **Prompt Features:** Two-phase analysis (solution + quality refinement), explicit context isolation
- **Key innovation:** Dynamic prompt generation with strict variable scope enforcement

To identify the most suitable large language model (LLM) for the AI Fix Generation Engine, a controlled benchmark was conducted using representative codebases from the target domain. Each candidate model was integrated into the repair pipeline and evaluated using SonarQube quality metrics after the generated fixes were applied and verified.

For each model, performance was assessed based on:

- **Bugs**: Remaining defects likely to cause incorrect program behavior
- **Code Smells**: Remaining maintainability-related issues
- **Security Vulnerabilities**: Confirmed exploitable weaknesses
- **Security Hotspots**: Code sections requiring manual review for security impact
- **Coverage (%)**: Percentage of code covered by automated tests after repairs

- **Duplications (%)**: Proportion of duplicated code in the codebase
- **Processing Time**: Total time required to execute the full repair workflow

Models	Reliability	Maintainability	Security	Security Review	Coverage	Duplications	Quality Gate	Time Needed (sec)	Time Needed (min)
Pre-Fix	33 BUGS	242 CS	0	49SH	96%	5.3%	PASSED	N/A	N/A
GPT4	8 BUGS	153 CS	0	31 SH	89%	5.3%	PASSED	1099	18,31666667
GPT3.5	19 BUGS	275 CS	0	65 SH	91,60%	5.7%	FAILED	317	5,283333333
GPT-4O-MINI	8 BUGS	153 CS	0	31 SH	89%	5.3%	PASSED	681	11,35
O3-MINI HIGH	15 BUGS	267 CS	0	65 SH	91.5%	5.7%	FAILED	1829	30,48333333
deepseek-r1:7b	16 BUGS	271 CS	0	65 SH	91,60%	5.7%	FAILED	17578	292,9666667
starcoder2:7b	16 BUGS	268 CS	0	65 SH	91,60%	5.7%	FAILED	2530	42,16666667
qwen2.5-coder:7b	16 BUGS	267 CS	0	65 SH	91,60%	5.7%	FAILED	3007	50,11666667
qwen2.5:7b	17 BUGS	273 CS	0	65 SH	91,60%	5.7%	FAILED	5787	96,45
codestral:22b	15 BUGS	271 CS	0	65 SH	91.6%	5.7%	FAILED	18092	301,5333333
codegemma:7b	1 BUGS	78 CS	0	1 SH	92%	5.7%	FAILED	6089	101,4833333
codellama:7b	18 BUGS	271 CS	0	65 SH	91.6%	5.7%	FAILED	7505	125,0833333
malibu-poolside	4 BUGS	87 CS	0	11 SH	91.6%	5.4%	FAILED	1912	31,86666667
malibu-Karvel-v1	4 BUGS	87 CS	0	11 SH	91.6%	5.4%	FAILED	1880	31,33333333

Figure 14 - Models's Benchmarking

As seen in the image above, Figure 14 - Models's Benchmarking, the analysis revealed that GPT-4O-MINI provided the most balanced performance across all metrics, it consistently achieved the largest reductions in bugs, code smells, and security hotspots while maintaining coverage and avoiding an increase in code duplication. It also demonstrated competitive processing times compared to other cloud-hosted models.

While GPT-4 produced similarly high-quality fixes, its higher processing cost and latency made it less practical for large-scale continuous integration.

A notable anomaly was observed with models malibu-poolside and malibu-Karvel-v1, which showed extremely favorable SonarQube post-fix metrics including dramatic reductions in bugs and code smells but also generated a high frequency of compilation errors during verification. Upon inspection, many of its "fixes" consisted of empty or truncated code segments, effectively removing the problematic code entirely rather than resolving it. While this caused issues to disappear from the SonarQube report, it severely degraded code integrity and functional behavior. This highlights the importance of combining static analysis results with compilation and runtime verification to prevent misleading quality gains.

Based on these findings, GPT-4O-MINI was selected as the default production model, with configuration options in .env and CLI flags allowing substitution with alternative LLMs.

d) Fix Application and Version Control

Fixes returned from the LLM are saved in JSON files and applied using apply_fixes.py. The system tracks line offsets dynamically to preserve positional accuracy across multiple edits, especially for multi-line fixes. All changes are

committed using Git, with reset checkpoints created before each fix group. A global backup is maintained to ensure full recoverability.

- **Line Tracking:** Dynamic offset management for positional accuracy
- **Backup tool:** global_backup.py creates snapshots before applying changes
- **Rollback Strategy:** Git-based reset points, full-file backup with global_backup.py

e) Verification and CI Integration

The verification and continuous integration (CI) module constitutes one of the central components of FixFlow, ensuring that all automatically generated code modifications undergo a rigorous validation workflow before being accepted. This process is orchestrated by a dedicated shell script that coordinates the CI pipeline and communicates directly with Jenkins, which serves as the primary automation server for compilation, unit testing, and optional static re-analysis. The verification workflow is structured into configurable stages that include build validation, compilation and execution of unit tests, and, when enabled, static analysis of the modified source code. Upon successful completion of all stages, the system updates the reset checkpoint used for safe rollback. Conversely, any failure in the CI process triggers the recovery module.

FixFlow is designed to operate in environments where development activity continuously generates new commits or pushes to remote branches. These events act as CI triggers, allowing the system to automatically evaluate each incremental change. The results produced by Jenkins or, with minor configuration adjustments, GitLab CI, are parsed into human-readable HTML or Markdown reports. These reports include diffs, metrics, and diagnostic information that support engineers in reviewing the outcomes of FixFlow's automated interventions.

A key element of this module is the error recovery engine, which implements a multi-tier rollback strategy to maintain system consistency. The first level of recovery performs a line-level undo, reverting only the specific modification responsible for a compilation or test failure. If this localized approach is insufficient, the system escalates to a function-level rollback, restoring the broader code block containing the problematic change. As a final safety mechanism, FixFlow can perform a global rollback to the last validated reset commit, effectively restoring the system to a known stable state. Before initiating any rollback, the engine attempts to syntax-aware auto-corrections based on bracket matching and expression structure analysis, aiming to resolve minor syntax errors without discarding fixes. These procedures are implemented across several internal modules, including revert_failed_fixes.py,

revert_entire_function.py, and undo_all_fixes.py, and have demonstrated a rollback reliability exceeding 95% during internal evaluations.

All fixes generated by FixFlow are versioned and fully traceable through Git commits. Each modification is preceded by the creation of a backup, ensuring that changes remain reversible at every stage of the verification process. Commit checkpoints serve as reference points for orchestrating the rollback strategy, enabling the system to maintain robustness even in the presence of complex or cascading errors.

f) Reporting and Visualization

Post-verification, the tool runs a final SonarQube scan and compares the results using compare_report.py. Reports include:

- Rule-specific success rates
- Verification outcomes
- HTML diff visualizations
- Token usage statistics and fix latency metrics
- Reports: diff_output.html, metrics_to_html.py
- Data tracked**:** Per-rule fix count, API response time, compilation/test results

4.2.4 Workflow and Process Flow

The tool operates as a multi-stage automation pipeline that integrates static analysis, AI-based code repair, version-controlled patch management, and CI/CD-based verification. This section outlines the complete process flow, as seen in the Figure 15 - FixFlow's Process Flow, from initial setup to final reporting, emphasizing traceability, rollback safety, and standards compliance at each step.

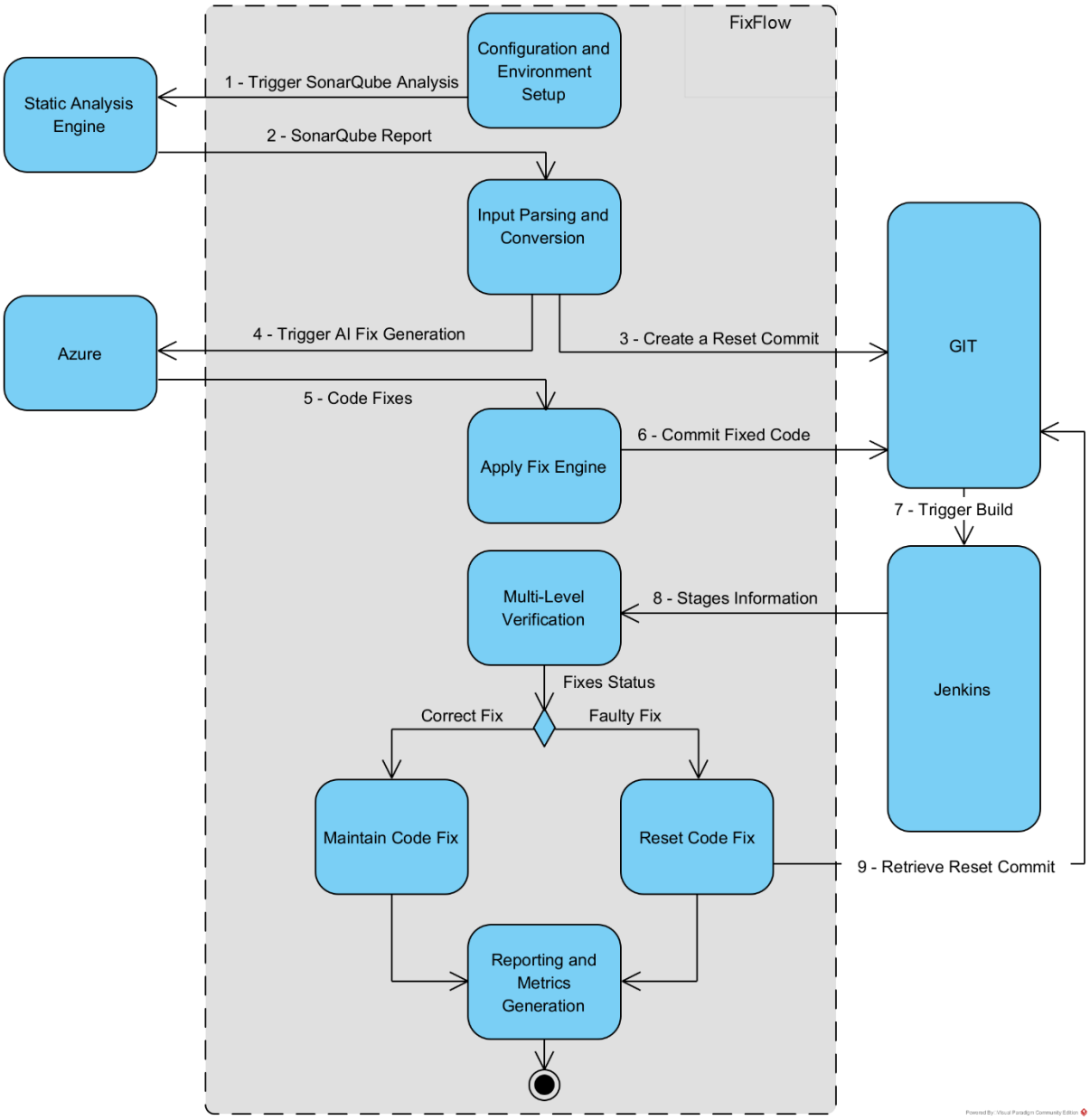

Figure 15 - FixFlow's Process Flow

4.2.5 Technical Innovations

The AI Code Repair Tool incorporates several technical innovations that distinguish it from conventional static analysis remediation workflows and existing AI-assisted repair systems. These innovations improve the safety, accuracy, consistency, and robustness of the automated code fixing process. The following subsections highlight the key innovations implemented in the tool's architecture and workflow.

a) Prompt-Level Context Isolation

A common failure mode in AI-based code repair is cross-request contamination, where a language model unintentionally carries contextual information from

previous prompts into subsequent ones. To mitigate this, the tool enforces strict prompt-level context isolation using:

- Explicit prompt headers
- Hard boundaries that prohibit referencing variables or functions outside the provided code context
- Embedded validation instructions to ensure variable declarations and syntax consistency

This isolation significantly reduces semantic hallucinations and unintended dependencies across fixes, ensuring high precision in rule-specific corrections.

b) Intelligent Line Offset Management

Applying multiple AI-generated code fixes sequentially can disrupt line numbering, leading to incorrect patch applications in later stages. To address this, the tool implements dynamic interval-based line offset tracking, which:

- Adjusts line positions after each fix is applied
- Maintains a real-time offset map across the codebase
- Supports multi-line replacements without misaligning subsequent patches

This approach preserves positional accuracy and prevents cascading errors in fix application.

c) Multi-Level Error Recovery

Unlike systems that rely on binary rollback mechanisms, the tool employs a multi-strategy error recovery framework that increases fault tolerance without discarding all progress. The recovery strategies include:

1. Line-level rollback for granular reversion
2. Function-level rollback if localized recovery fails
3. Global undo using Git checkpoints as a last resort

In cases where syntax issues are minor, the system also attempts auto-correction based on common patterns (e.g., unbalanced parentheses, missing semicolons) before reverting the patch.

d) Syntax-Aware Fix Application

To ensure structural validity, the system integrates Tree-sitter-based parsing with AI-generated fixes. Before applying a patch, the tool:

- Validates that the suggested code maintains balanced syntax (e.g., braces, brackets, parentheses)
- Performs AST-level matching to verify code placement within a correct structural scope
- Flags and defers any fixes that appear syntactically ambiguous

This tight coupling between AI suggestions and syntactic constraints improves both fix quality and downstream build success rates.

e) Two-Phase Prompt Engineering Strategy

The prompt design used in the tool separates reasoning and refinement into two logical phases:

- **Phase 1**: Code context understanding, variable consistency checks, and primary fix generation
- **Phase 2**: Fix validation, structural preservation analysis, and final quality assurance

This layered strategy ensures not only that the AI generates syntactically valid code, but also that it adheres to quality metrics such as rule compliance, clarity, and maintainability.

f) Verification-Aware Patch Grouping

The system groups violations by rule ID before initiating AI fix generation. This batch-oriented design enables:

- Reduced API usage by reusing similar prompt structures
- More consistent fix patterns within the same rule
- Parallel processing opportunities for scaling

Patch groups are processed and verified independently, enabling partial recovery if a specific group fails verification without halting the entire pipeline.

These innovations collectively improve the tool's performance, stability, and reliability in enterprise-grade software maintenance environments. By addressing the limitations of both traditional static analysis workflows and naive LLM integration, the tool establishes a practical foundation for safe, standards-compliant automated code repair.

g) Version Control and Rollback Safety

The tool is Git-native and designed to operate within clean working directories. It creates structured commits per patch group and includes:

- Reset commit tagging
- Global backups via global_backup.py
- Safe reversion through Git or script-based rollback layers

This ensures non-destructive operation and enables full traceability of all AI-driven changes.

4.2.6 Performance Evaluation

This section presents an explanation of how FixFlow's performance was measured across four real-world software projects. The evaluation assesses the tool's impact on overall code quality by comparing pre-fix and post-fix results obtained from SonarQube scans. The following quality metrics were used as the basis for evaluation:

- Bugs: Defects likely to cause incorrect program behavior
- Code Smells: Maintainability-related issues
- Vulnerabilities: Security risks that could be exploited
- Security Hotspots: Code sections requiring security review
- Coverage (%): Percentage of code covered by automated tests
- Duplicated Lines (%): Percentage of duplicated code across the project

For each project, the tool was executed using its standard verification pipeline. The process began with a baseline SonarQube scan to establish initial quality metrics. After automated repair, a post-fix scan was performed using the same analysis configuration to measure changes in these metrics.

The subsequent subsections present detailed results for each project, highlighting both quantitative improvements and qualitative observations regarding fix quality, verification stability, and rollback frequency.

a) Karvel

This case study investigates the application of FixFlow to enhance the quality and maintainability of the KARVEL project, it is a modular software structure adaptable to specific mission requirements, built upon the architecture of the NASA Core Flight System (cFS) – also created by CSW – designed to describe the high-level architecture of spaceflight systems.

KARVEL, a large-sized C project, is currently being developed at CSW and is a Software-on-Board (SoB) platform for spacecraft. It's being developed with the aim of one day being utilized as a framework for an OBSW (On-Board Software) – offering the flexibility for use across various types of space missions.

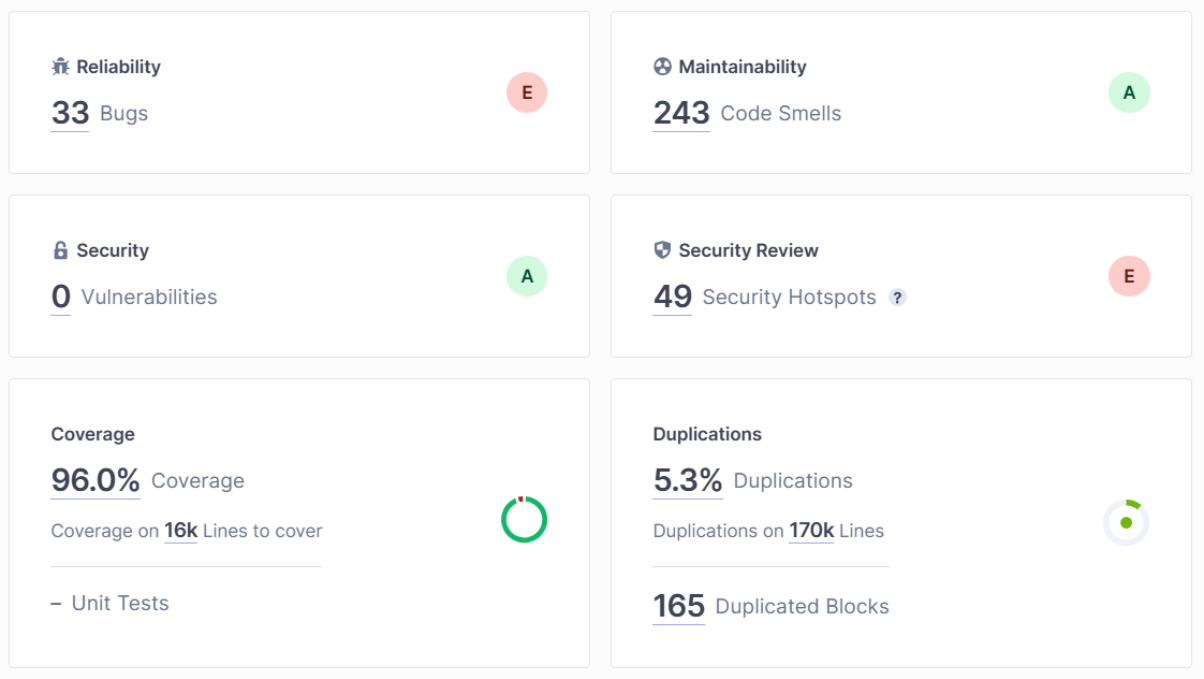

Figure 16 - Baseline Analysis Karvel

Figure 16 - Baseline Analysis Karvel presents the baseline analysis revealed a relatively high number of maintainability-related issues (243 code smells) and a notable presence of 33 bugs flagged by SonarQube's detection engine. While no vulnerabilities were reported, the 49 security hotspots indicate sections of code that may warrant manual security review. The project exhibited excellent unit test coverage (96%), suggesting that functional correctness is well-verified; however, the 5.3% duplicated code metric reflects moderate redundancy in implementation.

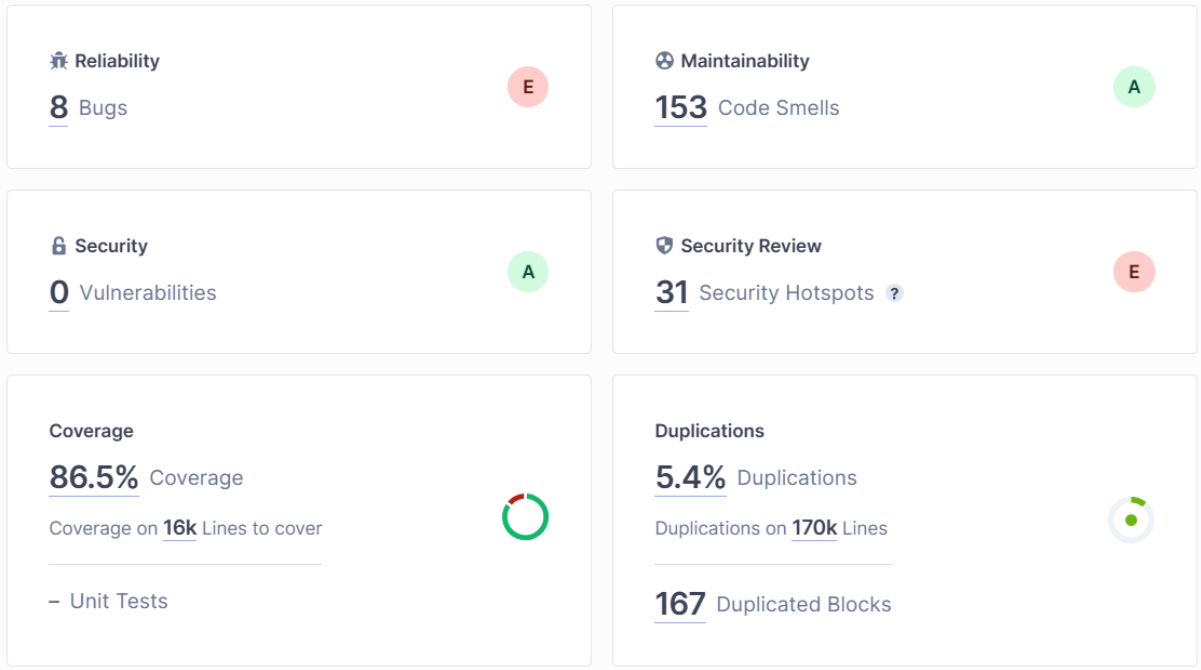

Figure 17 - Post Analysis Karvel

Following automated remediation with FixFlow, the number of bugs was reduced by approximately 76% (from 33 to 8) as Figure 17 - Post Analysis Karvel shows, and code smells were reduced by 37% (from 243 to 153). Security hotspots saw a 36% reduction (from 49 to 31), indicating improvements in code areas requiring manual review. No vulnerabilities were introduced or removed. Coverage decreased from 96% to 86.5%, likely due to changes in code structure that impacted test instrumentation or excluding certain files from coverage calculation. Duplicated code remained essentially unchanged, indicating that the repair process did not target redundancy-related issues.

b) Seranis

The SeRANIS AI-OBC is a core module within the AI-OBC, which resides on an NVIDIA Jetson System-on-Module (SoM) and operates within a real-time Linux (RT Linux) system. This module is responsible for managing and executing over ten complex experiments concurrently, utilizing technologies including sixth generation (6G) mobile communications, laser communication, and the Internet of Things (IoT).

The codebase, implemented on top of the Karvel execution platform in C, focuses on Push Telemetry, Telemetry Control (PTMTC) handling, and interaction with the Docker Engine API for managing experiments within containers, leveraging GPU-accelerated NVIDIA container runtime. The project scope encompassed the analysis and refinement of the AI-OBC software development process.

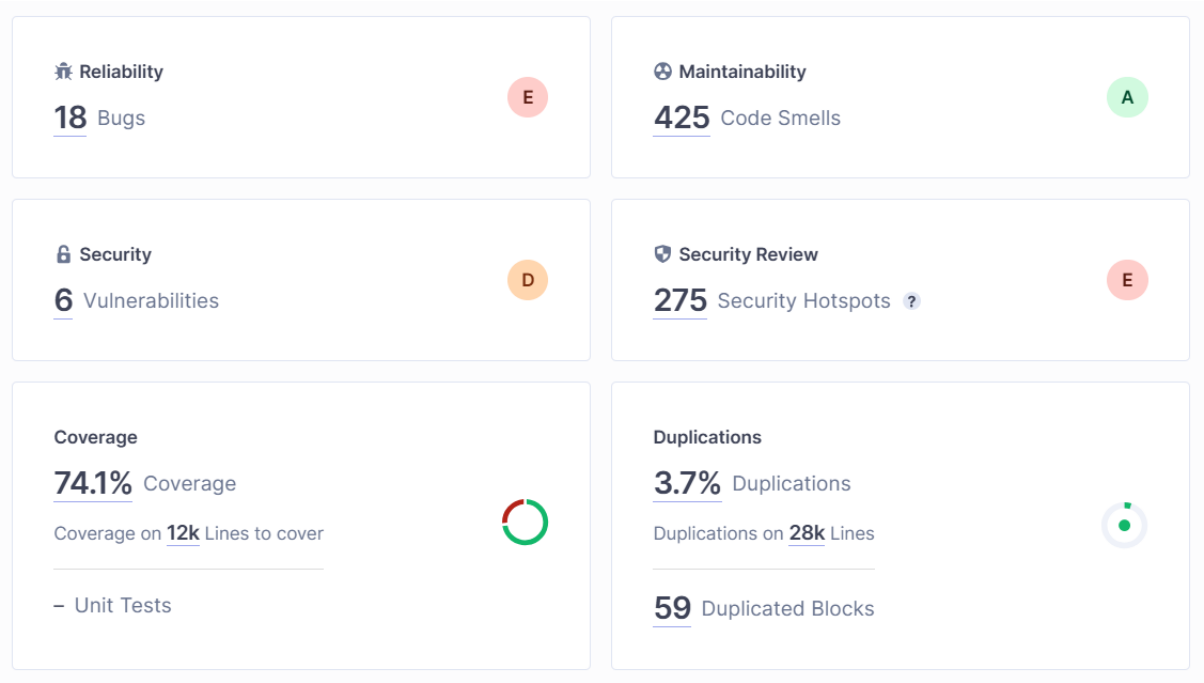

Figure 18 - Baseline Analysis Seranis

Figure 18 - Baseline Analysis Seranisshows the initial analysis revealed a substantial volume of code smells (425), indicating numerous maintainability challenges. The 18 bugs highlight functional defects with potential runtime impact, while the 6 vulnerabilities represent confirmed security flaws requiring remediation. The notably high number of security hotspots (275) suggests extensive code segments that warrant further security scrutiny. Coverage was measured at 74.1%, reflecting a reasonably broad but not exhaustive unit test suite. Code duplication was relatively low at 3.7%, suggesting minimal redundancy in this codebase.

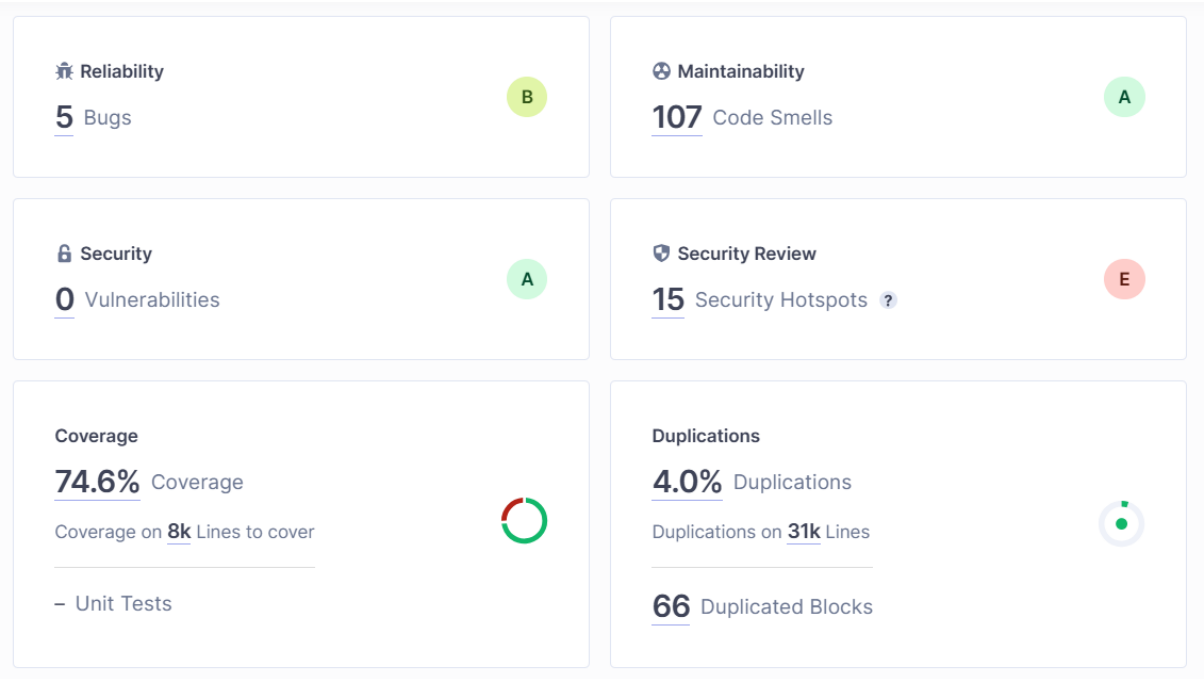

Figure 19 - Post Analysis Seranis

After using FixFlow, according to the Figure 19 - Post Analysis Seranis,the number of bugs decreased by 72% (from 18 to 5), while code smells dropped sharply by 75% (from 425 to 107), indicating a significant improvement in maintainability. All vulnerabilities were eliminated, representing a complete resolution of confirmed security defects. Security hotspots were reduced by 94% (from 275 to 15), dramatically lowering the volume of code requiring manual review. Test coverage remained essentially stable, and duplication saw only a marginal increase of 0.3%, suggesting that the repair process did not introduce notable redundancy. These results reflect a strong improvement in code health without sacrificing test coverage or structural clarity.

c) Updater Verticalla

The Verticalla project, developed by Critical, provides Sauter with software for the Sauter Vision Center (SVC). The project is a long-term active project, currently 11 years old, comprised of a team of approximately 30 members. The project consists of three main web applications: the Vision Center Portal, the Vision Center Manager, and the Vision Center Application Programming Interface (API). The Updater module, integrated within the Vision Center Manager application, serves as the core of the software, enabling users to specify settings and project configurations of an SVC installation. Users can utilize it to create and update projects and install their respective licenses, and to receive notifications regarding license expirations.

While relatively small compared to the other evaluated projects, it contains sections of code with exceptionally high cognitive complexity, making automated remediation challenging. This project provided an opportunity to evaluate FixFlow's handling of intricate logic, as well as its ability to avoid introducing regressions when suitable automated fixes could not be confidently generated.

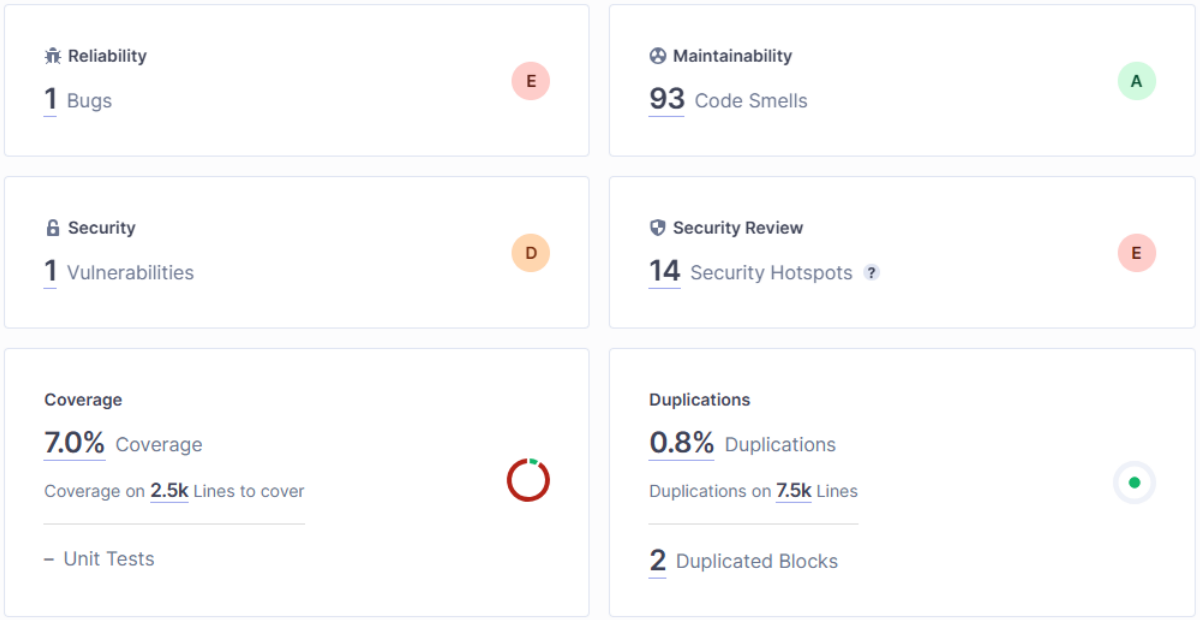

Figure 20 - Baseline Analysis Updater

The initial scan showed in Figure 20 - Baseline Analysis Updater, demonstrated a relatively small number of detected issues, but their nature made them particularly difficult to address through automated means. Most of the code smells were related to high cognitive complexity, indicating functions or methods that are long, deeply nested, or involve intricate branching logic. The single reported bug and vulnerability add to the safety and security concerns, while low coverage (7%) limits the effectiveness of automated verification.

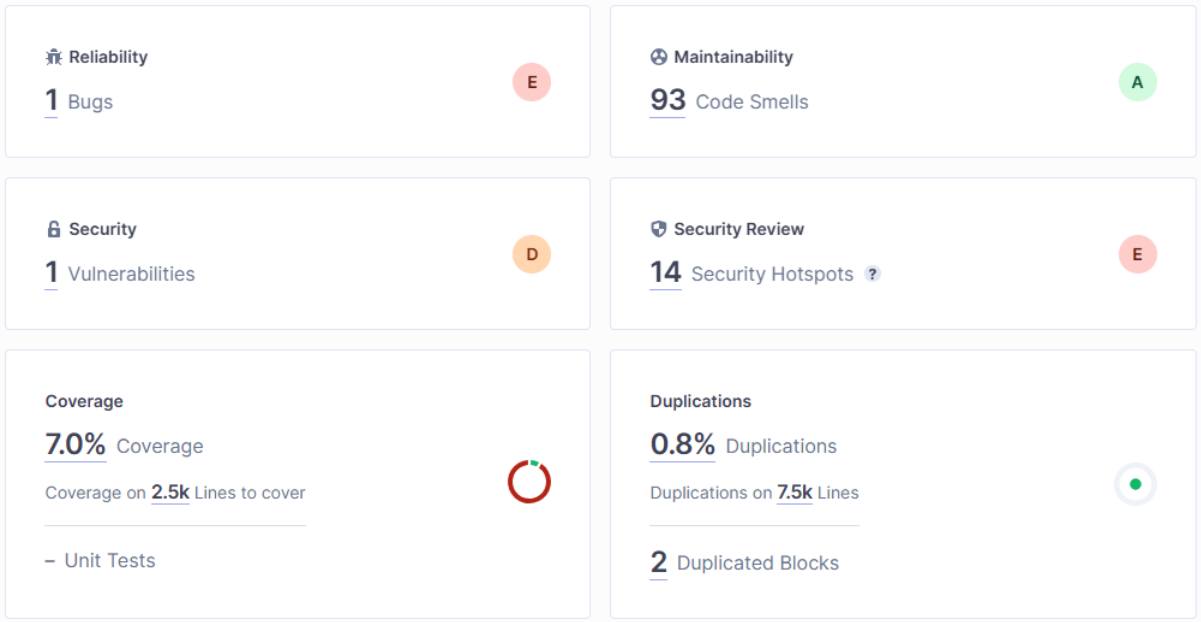

Figure 21 - Post Analysis Updater

At first glance looking at Figure 21 - Post Analysis Updater, the metrics appear unchanged however, this outcome reflects a positive technical result. FixFlow successfully recognized that the violations primarily stemming from cognitive complexity required structural redesign beyond the scope of safe, automated patching. Instead of producing speculative or incorrect modifications, the system refrained from altering the code, thereby avoiding the risk of introducing defects.

This behavior demonstrates two important qualities of the tool:

1. **Verification robustness:** The verification pipeline correctly prevented the merging of fixes that could not be validated within the project's limited test coverage.
2. **Multi-Level Error Recovery effectiveness:** The fallback mechanisms ensured that attempted fixes which failed verification were reverted cleanly, preserving the integrity of the codebase.

In this case, maintaining the status quo was preferable to introducing potentially unsafe changes, highlighting the system's capacity to prioritize correctness over aggressive remediation.

d) Portal Client Verticalla

Just like the previous project this case study was performed on the Verticalla project, but this time in the Portal Client module. The Portal Client module, integrated within

the Vision Center Portal application, enables users to visualize the data provided by the objects defined in the project. It offers a wide array of viewing functionalities, from being able to show the state of all objects within a room to being able to focus on a single one, obtaining detailed information on its current status.

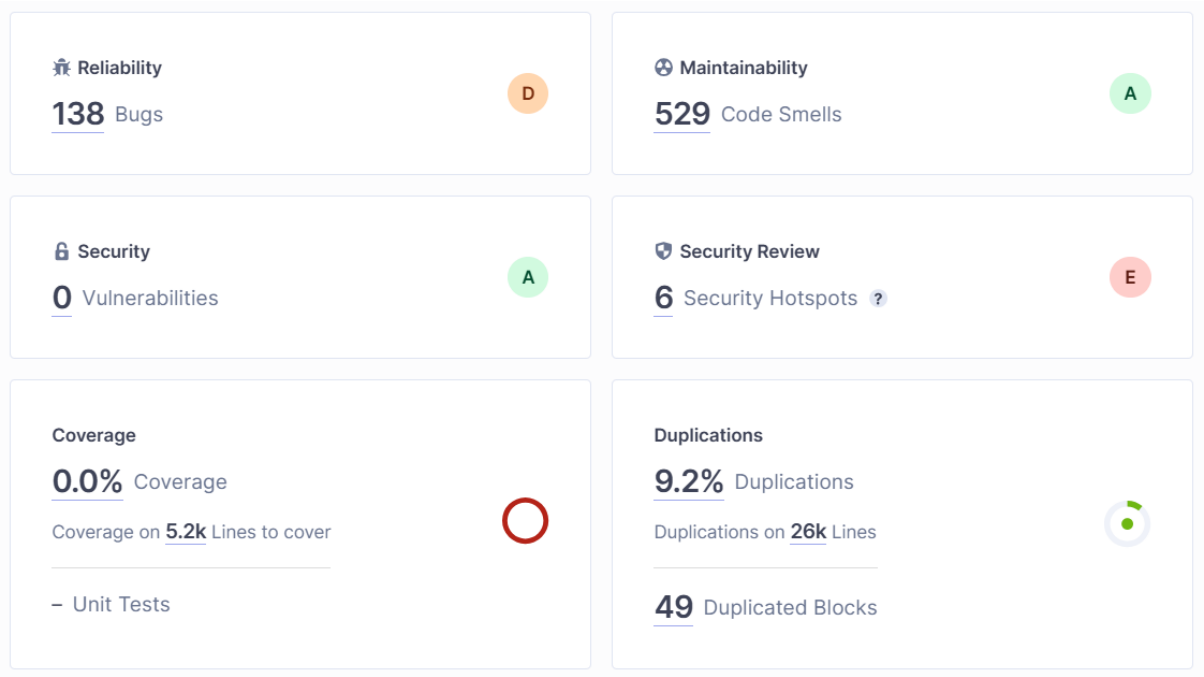

Figure 22 - Baseline Analysis Portal Client

The baseline, showed in Figure 22 - Baseline Analysis Portal Client, scan revealed a significant accumulation of quality issues, with 138 bugs and 529 code smells indicating extensive functional and maintainability concerns. Although no vulnerabilities were detected, the presence of 6 security hotspots still flagged certain areas as requiring manual inspection. The complete absence of test coverage (0%) is a critical limitation, as it significantly restricts the ability to automatically validate functional correctness. Additionally, the 9.2% code duplication suggests notable redundancy, which can contribute to maintainability challenges and inconsistency across modules.

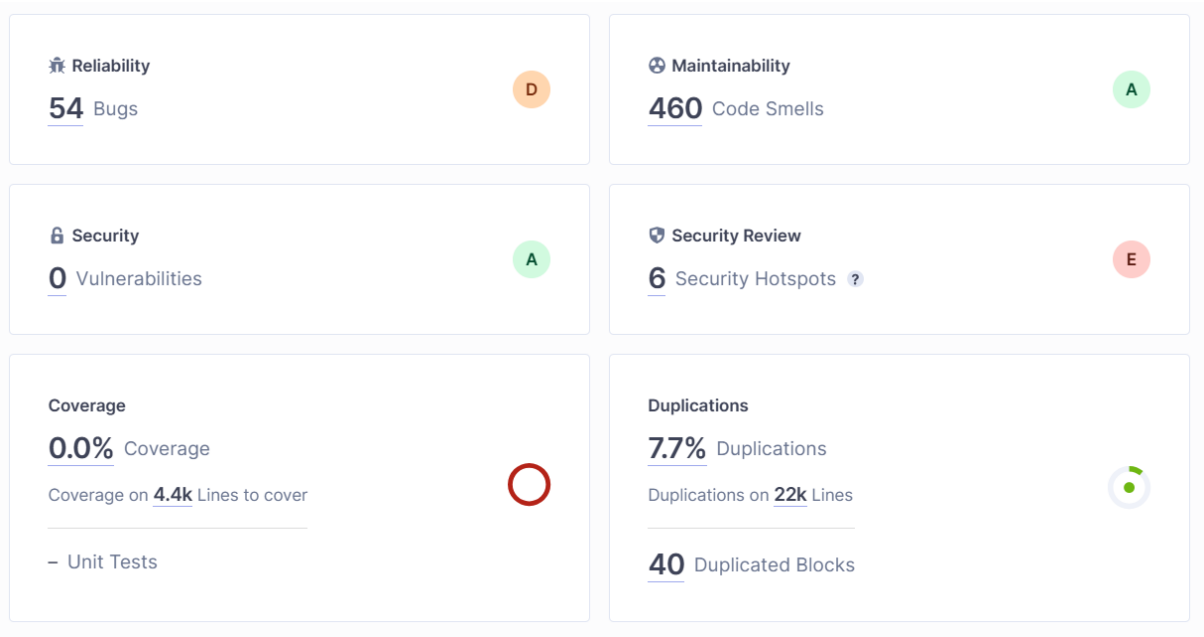

Figure 23 - Post Analysis Portal Client

Following automated repair, the number of bugs decreased by approximately 61% (from 138 to 54), as seen in the Figure 22 - Baseline Analysis Portal Client, reflecting a substantial improvement in functional correctness indicators. Code smells were reduced by around 13% (from 529 to 460), indicating partial progress in maintainability but leaving room for further refactoring. The counts for vulnerabilities and security hotspots remained unchanged, which is expected given that no new security rules were applied during this evaluation.

One unexpected benefit was a modest reduction in duplicated code (from 9.2% to 7.7%), suggesting that some of the AI-generated patches naturally consolidated redundant logic. However, the lack of test coverage meant that verification had to rely solely on compilation success and static analysis re-checks, increasing the conservatism of the applied fixes. This conservative approach likely prevented more aggressive code smell reduction but ensured that the integrity of the system was preserved in the absence of runtime validation.

4.2.7 Conclusion

This chapter presented the design, implementation, and evaluation of FixFlow, an AI-powered system designed to automate the detection, repair, and verification of code quality issues. Motivated by the high cost of software maintenance and the limitations of manual refactoring, the system was developed to reduce technical debt, improve maintainability, and enhance security in a consistent and automated manner. Based in three main components, SonarQube for violation detection, Tree-sitter for structural context extraction, and large language models (LLMs) for fix generation, FixFlow provides a portable and configurable solution aligned with modern DevOps practices. Key innovations include strict context isolation, multi-level error recovery mechanisms, and seamless integration with Jenkins pipelines to ensure robustness in real-world development workflows.

Overall, FixFlow consistently delivered measurable improvements across bug density, code smells, and security hotspots, while ensuring that unsafe or unverifiable changes were not introduced. The relevance and technical contribution of this work were further validated through its acceptance and presentation at two conferences, Software Product Assurance Conference 2025 by ESA and RECPAD 2025, highlighting its potential impact and applicability within the broader software engineering and AI research communities.

4.2.8 Limitations and Future Work

While FixFlow demonstrates strong potential for automating static analysis remediation in safety-critical environments, several limitations constrain its generalizability, robustness, and scalability. These limitations inform a roadmap for future development aimed at enhancing language coverage, contextual understanding, verification safety, and deployment flexibility.

a) Scope Limitation: Rule Coverage and Validation Context

The tool is currently tailored to **MISRA rules for C/C++**, targeting embedded systems and safety-critical software. While MISRA enforcement is a high-value use case, the system does not yet support broader rule sets such as **CERT** (a set of secure coding guidelines for preventing vulnerabilities in software) or **OWASP** (a framework for identifying and mitigating security risks in web applications), nor project-specific guidelines.

Moreover, the tool has been evaluated exclusively on **private, internal codebases**. While these real-world repositories provide valuable validation, they limit the ability to benchmark performance and accuracy on open-source, community-driven

projects. Extending support to additional rule sets and validating performance across public repositories will be essential to generalize the tool's applicability.

b) Fix Safety and Semantic Preservation

One of the most significant challenges in automated code repair is ensuring that patches do not introduce **semantic regressions**. Although the tool verifies patches using existing compilation and unit tests, this process inherently relies on the completeness and quality of the test suite. As a result:

- Patches may pass tests but still alter intended behavior
- Incomplete test coverage can allow logic-altering or security-impacting fixes to slip through

AI-generated patches may also "overfit" to test expectations, particularly if tests only validate shallow properties (e.g., output formatting). To address this, future work should explore:

- **Formal verification methods** (e.g., symbolic execution or model checking)
- **Automated test generation** for uncovered code paths
- **Differential testing** or behavioral equivalence checks across versions

c) Context Limitations and LLM Behavior

Although the system uses Tree-sitter to extract relevant context windows, certain fixes may require broader program-level information (e.g., global variables, cross-file dependencies) that exceed current prompt limits. This can result in:

- Omitted dependencies or incomplete fix suggestions
- Unstable or invalid code if assumptions are made incorrectly by the LLM

Additionally, large language models are prone to **hallucinations**, occasionally introducing syntax errors (e.g., unclosed blocks or missing semicolons) or misapplying coding conventions. These issues, while often caught during verification, introduce risk and highlight the need for:

- Post-fix syntactic analysis
- Optional human-in-the-loop review workflows
- LLM filtering based on confidence thresholds or fix risk classification

d) Language and Rule Set Expansion

The system currently supports a fixed set of languages (C, C++, Java, Python, JavaScript, and HTML) and requires custom Tree-sitter grammars for syntax-aware

processing. Although adding support for new grammar is technically straightforward, it remains a manual process. Expanding language support to cover **Rust**, **Go**, **C#**, and **TypeScript**, among others, is necessary for broader adoption in polyglot environments.

Likewise, generalized support for arbitrary rule sets will require a more abstract rule-mapping layer that can handle diverse violation formats beyond SonarQube or MISRA.

Addressing these limitations will be key to the next phase of development.

5 CONCLUSIONS

The work presented throughout this thesis reflects a comprehensive examination of the role of artificial intelligence in the context of space systems engineering and software development. Motivated by the growing reliance on autonomous systems, data-driven decision-making, and the increasing complexity of spacecraft software, this internship provided the opportunity to investigate AI from both a theoretical and a practical perspective. The research combined a thorough study of AI foundations, an analysis of ESA's ongoing initiatives, and the development of two AI-driven tools, SpaceSage and FixFlow, designed to address concrete challenges within Critical Software.

From a conceptual standpoint, this thesis first established the historical evolution and technological principles underlying modern AI, with a particular emphasis on algorithms relevant to space applications. Classical methods such as regression techniques, decision trees, and clustering were discussed alongside more advanced neural architectures, including CNNs, RNNs, GANs, reinforcement learning agents, and graph neural networks. This background not only contextualized the rapid progress of AI capabilities but also highlighted the challenges associated with deploying these technologies in space, ranging from computational constraints and safety requirements to ethical and legal considerations.

Building on this foundation, an investigation into ESA's ongoing efforts revealed substantial engagement with AI-driven approaches across missions such as Φsat, OPS-SAT, Hera, and ClearSpace-1. These initiatives demonstrate the agency's strategic commitment to embedding machine intelligence into spacecraft autonomy, Earth observation pipelines, and mission design workflows. The evaluation of AIDA, ESA's prototype tool for automated systems engineering model analysis, further illustrated both the innovation potential and the practical limitations of early-stage AI systems. Although AIDA was unable to operate reliably during testing due to persistent database errors, discussions with the project's chief officer clarified that the tool had primarily served as a research vehicle for algorithmic exploration rather than as a ready-to-use engineering product. This outcome helped to frame the challenges of transitioning AI prototypes into operational environments.

In parallel with these investigations, the internship enabled the end-to-end development and evaluation of two complementary AI-based tools designed to support aerospace software engineering activities. **SpaceSage**, a private Retrieval-Augmented Generation (RAG) system, was developed to address the need for confidential, efficient access to internal documentation and **FixFlow**, the second major contribution, tackled the problem of automating the remediation of static code analysis violations.

Taken together, these tools demonstrate that AI, when carefully engineered and responsibly integrated, can significantly enhance workflow efficiency, knowledge

accessibility, and software quality in aerospace environments. The internship provided not only a deeper understanding of the technical capabilities and limitations of AI but also practical insights into real-world engineering considerations, including confidentiality constraints, computational limitations, legacy documentation structures, and the importance of auditability and trust.

In conclusion, this thesis demonstrates the feasibility, relevance, and value of applying artificial intelligence to real engineering challenges in the space sector. Through academic investigation, hands-on experimentation, and the creation of operational tools, the work completed during this internship contributes meaningfully to ongoing efforts to harness AI for safer, more efficient, and more autonomous spacecraft systems and its development. The results achieved, supported by both research and implementation, affirm that AI can play a transformative role in the development process of aerospace software engineering and will continue to shape the future of mission design, operations, and onboard intelligence.

6 REFERENCES

[1] "Artificial Intelligence in Satellite and Space Systems." Accessed: Aug. 20, 2025. [Online]. Available: https://ts2.tech/en/artificial-intelligence-in-satellite-and-space-systems/?utm_source=chatgpt.com

[2] "Deep Space 1 - Wikipedia." Accessed: Aug. 20, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Deep_Space_1?utm_source=chatgpt.com

[3] "The Journey of NASA's Smartest Satellite Finally Comes to an End | WIRED." Accessed: Aug. 20, 2025. [Online]. Available: https://www.wired.com/2017/03/say-farewell-eo-1-nasas-smartest-satellite/?utm_source=chatgpt.com

[4] C. Whitney and J. Melville, "[SSC25-P1-05] Toward Trusted Onboard Artificial Intelligence (AI): Advancing Small Satellite Operations using Reinforcement Learning".

[5] "Eyes beyond the skies: How IIIT-Delhi and ISRO are training AI to watch over space debris, rogue satellites - The Economic Times." Accessed: Aug. 20, 2025. [Online]. Available: https://economictimes.indiatimes.com/news/science/eyes-beyond-the-skies-how-iiit-delhi-and-isro-are-training-ai-to-watch-over-space-debris-rogue-satellites/articleshow/121565100.cms?utm_source=chatgpt.com

[6] "China begins assembling AI supercomputer in SPACE made of thousands of satellites circling Earth that talk using lasers | The US Sun." Accessed: Aug. 20, 2025. [Online]. Available: https://www.the-sun.com/tech/14273133/china-ai-supercomputer-space-satellites/?utm_source=chatgpt.com

[7] "China is building a constellation of AI supercomputers in space — and just launched the first pieces | Live Science." Accessed: Aug. 20, 2025. [Online]. Available: https://www.livescience.com/technology/computing/china-is-building-a-constellation-of-ai-supercomputers-in-space-and-just-launched-the-first-pieces

[8] T. Aghayev, S. Diamond, S. Kumar, R. Dudukovich, and J. Briones, "Linear Regression Model for Predictive Service Provider Selection," 2023.

[9] X. Zhou, T. Qin, and L. Meng, "Maneuvering Spacecraft Orbit Determination Using Polynomial Representation," *Aerospace 2022, Vol. 9, Page 257*, vol. 9, no. 5, p. 257, May 2022, doi: 10.3390/AEROSPACE9050257.

[10] I. Siddique, "Detection and Analysis of Anomalous Behavior in On-Orbit Satellites Using AI Algorithms," *Journal of Firewall Software and Networking*, vol. 2, no. 2, pp. 6–17, Jul. 2024, doi: 10.48001/JOFSN.2024.226-17.

[11] "k-nearest neighbors algorithm - Wikipedia." Accessed: Dec. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[12] Y. Lin *et al.*, "Optimization of the k-nearest-neighbors model for summer Arctic Sea ice prediction," *Front Mar Sci*, vol. 10, p. 1260047, Oct. 2023, doi: 10.3389/FMARS.2023.1260047/BIBTEX.

[13] G. Mountrakis, J. Im, and C. Ogole, "ISPRS Journal of Photogrammetry and Remote Sensing Support vector machines in remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, pp. 247–259, 2011, doi: 10.1016/j.isprsjprs.2010.11.001.

[14] J. Doubleday, D. Mclaren, S. Chien, and Y. Lou, "Using Support Vector Machine Learning to Automatically Interpret MODIS, ALI, and L-Band SAR Remotely Sensed Imagery for Hydrology, Land Cover, and Cryosphere Applications," 2011.

[15] "Random forest - Wikipedia." Accessed: Dec. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Random_forest

[16] "Fundamentals of Machine Learning for Earth Science | NASA Earthdata." Accessed: Dec. 12, 2025. [Online]. Available: https://www.earthdata.nasa.gov/learn/trainings/fundamentals-machine-learning-earth-science

[17] "Naive Bayes classifier - Wikipedia." Accessed: Dec. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[18] O. J. Mengshoel *et al.*, "Diagnosing Faults in Electrical Power Systems of Spacecraft and Aircraft", Accessed: Dec. 12, 2025. [Online]. Available: http://ti.arc.nasa.gov/adapt/

[19] "Neural network (machine learning) - Wikipedia." Accessed: Dec. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Neural_network_(machine_learning)

[20] R. A. Sheldon, "Satellite image analysis using neural networks," *NASA, Goddard Space Flight Center, The 1990 Goddard Conference on Space Applications of Artificial Intelligence*, 1990.

[21] "Convolutional neural network - Wikipedia." Accessed: Dec. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network

[22] "Recurrent neural network - Wikipedia." Accessed: Dec. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network

[23] M. Greza, I. Bhattacharya, L. Hoegner, and B. Jutzi, "GAN-Based Dual Image Super Resolution for Satellite Imagery Decreasing Radiometric Uncertainty," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. X-3–2024, no. 3, pp. 155–162, Nov. 2024, doi: 10.5194/ISPRS-ANNALS-X-3-2024-155-2024.

[24] J. D. Lohn, G. S. Hornby, and D. S. Linden, "An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission," 2004.

[25] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A Gentle Introduction to Graph Neural Networks," *Distill*, vol. 6, no. 9, p. e33, Sep. 2021, doi: 10.23915/DISTILL.00033.

[26] "Open Cosmos and ESA set new standard for space AI with the launch of AI app-powered Phisat-2 Earth Observation satellite." Accessed: Dec. 07, 2025. [Online]. Available: https://www.open-cosmos.com/news/phisat-2-launch

[27] "ESA Launched Φsat-2: Harnessing AI for Advanced Earth Observation | UN-SPIDER Knowledge Portal." Accessed: Dec. 07, 2025. [Online]. Available: https://www.un-spider.org/news-and-events/news/esa-launched-%CF%86sat-2-harnessing-ai-advanced-earth-observation

[28] "Phi-Sat-1 - Wikipedia." Accessed: Dec. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Phi-Sat-1

[29] "Φsat-2 – KP Labs Innovative Missions and Projects." Accessed: Dec. 07, 2025. [Online]. Available: https://www.kplabs.space/projects-and-missions/phi-sat-2

[30] "Φ-Lab Challenges." Accessed: Dec. 07, 2025. [Online]. Available: https://platform.ai4eo.eu/orbitalai-phisat-2

[31] "Phi-Sat-2 - Wikipedia." Accessed: Dec. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Phi-Sat-2

[32] "Open Cosmos and ESA set new standard for space AI with the launch of AI app-powered Phisat-2 Earth Observation satellite." Accessed: Dec. 07, 2025. [Online]. Available: https://www.open-cosmos.com/news/phisat-2-launch

[33] "ESA Launched Φsat-2: Harnessing AI for Advanced Earth Observation | UN-SPIDER Knowledge Portal." Accessed: Dec. 07, 2025. [Online]. Available: https://www.un-spider.org/news-and-events/news/esa-launched-%CF%86sat-2-harnessing-ai-advanced-earth-observation

[34] "ESA - OPS-SAT." Accessed: Dec. 07, 2025. [Online]. Available: https://www.esa.int/Enabling_Support/Operations/OPS-SAT

[35] "AI AT THE EDGE : DEEP CUBE SERVICE IOD/IOV | Nebula Public Library." Accessed: Dec. 07, 2025. [Online]. Available: https://nebula.esa.int/content/ai-edge-deep-cube-service-iodiov

[36] "Learn more on www.esa.int/discovery Deliverables published on https://nebula.esa.int AI at the edge: DeepCube Service IOD/IOV Executive Summary Study", Accessed: Dec. 07, 2025. [Online]. Available: www.esa.int/discovery

[37] "ONBOARD MULTI-FRAME SUPER RESOLUTION IMAGE | Nebula Public Library." Accessed: Dec. 07, 2025. [Online]. Available: https://nebula.esa.int/content/onboard-multi-frame-super-resolution-image

[38] "SaasyML: Onboard Machine Learning Software As A Service For Experimenters | Nebula Public Library." Accessed: Dec. 07, 2025. [Online]. Available: https://nebula.esa.int/content/saasyml-onboard-machine-learning-software-service-experimenters

[39] "AN AI-BASED SYSTEM FOR AN ACTIVE TRACKING OF EARTH FEATURES FROM OPS-SAT | Nebula Public Library." Accessed: Dec. 07, 2025. [Online]. Available: https://nebula.esa.int/content/ai-based-system-active-tracking-earth-features-ops-sat

[40] "ONBOARD MULTI-FRAME SUPER RESOLUTION IMAGE | Nebula Public Library." Accessed: Dec. 07, 2025. [Online]. Available: https://nebula.esa.int/content/onboard-multi-frame-super-resolution-image

[41] "Learn more on www.esa.int/discovery Deliverables published on https://nebula.esa.int ➔ DISCOVERY Deep Active Tracking: an AI-based system for an active tracking of Earth features from OPS-SAT", Accessed: Dec. 07, 2025. [Online]. Available: www.esa.int/discovery

[42] "HOPAS (Hybrid online policy adaptations strategy) | Activities Portal." Accessed: Dec. 07, 2025. [Online]. Available: https://activities.esa.int/4000137215

[43] "ESA - Hera asteroid mission tested self-driving technique at Mars." Accessed: Dec. 07, 2025. [Online]. Available: https://www.esa.int/Space_Safety/Hera/Hera_asteroid_mission_tested_self-driving_technique_at_Mars

[44] R. Biesbroek, S. Aziz, A. Wolahan, S. Cipolla, M. Richard-Noca, and L. Piguet, "THE CLEARSPACE-1 MISSION: ESA AND CLEARSPACE TEAM UP

TO REMOVE DEBRIS", Accessed: Dec. 07, 2025. [Online]. Available: https://clearspace.today/

[45] "ESA - ClearSpace-1." Accessed: Dec. 07, 2025. [Online]. Available: https://www.esa.int/Space_Safety/ClearSpace-1

[46] "List of available metrics - Ragas." Accessed: Dec. 12, 2025. [Online]. Available: https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/

7 APPENDICES

7.1 Appendix A – Internship Proposal

CSW-2024-DOC-021
83-proposta-estagic

7.2 Appendix B – RECPAD 2025 Report

RECPAD2025.pdf

7.3 Appendix C – Software Product Assurance Conference 2025 by ESA Report

Automated-AI-Drive
n-Code-Repair-pres

7.4 Appendix D – SpaceSage Benchmarking Results (Excel Dataset)

Model_Embeddings
_Results_SS.pdf

Instituto Superior de Engenharia
Politécnico de Coimbra