UNIVERSIDADE
LUSÓFONA

Model for AI in MISRA-C

(Modelo Para AI Em MISRA-C)

Final Course Work

1st Half Progress Report

Student Name:Alexandre Gama Dos Santos

Name of supervisor:Daniel Silveira

Name of supervisor:Zuil Pirola

Final Course Work | LEI | 13/04/2024

www.ulusofona.pt

Direitos de cópia

Resumo

Este projeto visa explorar ferramentas de processamento de linguagem natural [28], Analise dinâmica[17], Análise Estática [8] e Inteligência Artificial(IA) [4] ,quando adotam as normas MISRA-C(Motor Industry Software Reliability Association) [2] na programação C/C++.

O relatório apresenta as soluções propostas do plugin que vai ser criado. Vamos usar as ferramentas CPPCHECK[7] e CHATGPT[14] e vamos associá-las ao plugin onde vai ser compatível em MISRA-C para a linguagem C, acrescentando às regras do MISRA-C[1].

Através de analise de confusion Matrix será possivel avaliar a eficácia do plugin na deteção de verdadeiros e falsos positivos sublinham a sua eficácia, prometendo a codificação ser legível.

Este TFC mostra a compatibilidade com as regras MISRA-C, como também mostra a criação de um plugin que resolve os erros de um código em linguagem C[10].

A primeira opção será primeiro o CPPCHECK vai analisar o código e ver se a alguma parte que não cumpre com as regras do MISRA-C,o CPPCHECK cria um ficheiro em formato html onde tem lá o código e mostra as zonas onde não cumpriu de seguida o CHATGPT usando a prompt[12] gera um código corrigido onde vamos testar a sua eficácia pela matriz[13] para saber se gerou um código que se pretendia.

A segunda opção será o CHATGPT usando a prompt[12] gera um código corrigido onde vamos testar a sua eficácia pela matriz[13] para saber se gerou um código que se pretendia.

Abstract

This project aims to explore natural language processing tools [28], Dynamic Analysis [8], Static Analysis [8] and Artificial Intelligence (AI) [4], when adopting the MISRA-C (Motor Industry Software Reliability Association) standards [2] in C/C++ programming.

The report presents the proposed solutions chosen for the plugin that will be created. We are going to use the CPPCHECK[7] and CHATGPT[14] tools and associate them with the plugin, which will be compatible with MISRA-C for the C language, adding the MISRA-C rules[1].

By analysing the Confusion Matrix, it will be possible to assess the plugin's effectiveness in detecting true and false positives, underlining its effectiveness in promising readable coding.

This TFC shows compatibility with the MISRA-C rules, as well as the creation of a plugin that resolves errors in C language code[10].

The first option will first be for CPPCHECK to analyse the code and see if there are any parts that don't comply with the MISRA-C rules, CPPCHECK will create a file in html format with the code in it and show the areas where it didn't comply, then CHATGPT using the prompt[12] will generate a corrected code where we'll test its effectiveness using the matrix[13] to see if it generated the code we wanted.

The second option will be for CHATGPT using prompt[12] to generate a corrected code where we will test its effectiveness using matrix[13] to see if it has generated a code that was intended.

Index

List of Figures

List of Tables

1 Identifying the Problem

1.1 Key Concepts

1.1.1 **What is a software and Language C**

Software is a set of computer programs [29] and associated documentation [30] and data [31]. This is in contrast to hardware [32], from which the system is built and which actually performs the work.

C is a general-purpose computer programming language. It was created in the 1970s by Dennis Ritchie, and remains very widely used and influential. By design, C's features cleanly reflect the capabilities of the targeted CPUs.[10]

1.1.2 MISRA C(Motor Industry Software Reliability Association)[2]:

The MISRA C and MISRA C++ standards are a set of coding guidelines for the C and C++ programming languages that promote safety, security, and reliability in embedded system software. These guidelines are published by the Motor Industry Software Reliability Association (MISRA), a collaboration between vehicle manufacturers, component suppliers, and engineering consultancies. MISRA C guidelines, which originally targeted only the automotive industry, are now widely accepted and used in other industries such as aerospace and defence, industrial automation, and medical devices.

Most engineering organizations originally applied MISRA C and C++ standards to their hand-written C and C++ code. The MISRA standards have since been refined to accommodate the use of code generated from tools such as MATLAB® and Simulink®.

1.2 **Artificial intelligence and automation**

Artificial intelligence (AI) is the intelligence of machines [34] or software [33], as opposed to the intelligence of humans or animals. It is a field of study [35] in computer science [36] which develops and studies intelligent machines. Such machines may be called AIs.[AI]

Intelligent automation (IA), sometimes also called cognitive automation, is the use of automation technologies – Artificial Intelligence (AI), Business Process Management (BPM), and Robotic Process Automation (RPA) – to streamline and scale decision-making across organizations. Intelligent automation simplifies processes, frees up resources and improves operational efficiencies, and it has a variety of applications. For example, an automotive manufacturer may use IA to speed up production or reduce the risk of human error, or a pharmaceutical or life sciences company may use intelligent automation to reduce costs and gain resource efficiencies where repetitive processes exist. An insurance provider can use intelligent automation to calculate payments, make predictions used to calculate rates, and address compliance needs.[11]

1.3 Artificial Intelligence and prompting engineering

An artificial intelligence (AI) prompt is a mode of interaction between a human and a large language model that lets the model generate the intended output. This interaction can be in the form of a question, text, code snippets or examples.[12]

Prompt engineering is a relatively new discipline that aims to develop and optimise prompts and efficiently use language models (LMs) for a wide variety of applications and research topics. Prompt engineering skills help to better understand the features and limitations of large language models (LLMs).[40]

Researchers use prompt engineering to improve the ability of LLMs in a wide range of common and complex tasks, such as question answering and arithmetic reasoning. Developers use prompt engineering to design robust and effective prompt techniques that interface with LLMs and other tools.[40]

Prompt engineering is not just about designing and developing prompts. It covers a wide range of skills and techniques that are useful for interacting and developing with LLMs. It's an important skill for interacting with, building and understanding the capabilities of LLMs. You can use prompt engineering to improve the security of LLMs and create new features, such as augmenting LLMs with domain knowledge and external tools.[40]

1.4 Confusion Matrix:

We will evaluate the plugin's ability to identify errors based on analysis of a confusion matrix.. Confusion matrix is a very popular measure used while solving classification problems. It can be applied to binary classification as well as for multiclass classification problems. An example of a confusion matrix for binary classification is shown in Table Confusion Matrix.[13]

1.5 AI assistants and CHATGPT

ChatGPT is a language model developed by OpenAI. It belongs to the GPT (Generative Pre-trained Transformer) family of models, specifically GPT-3.5 architecture. GPT-3, which stands for Generative Pre-trained Transformer 3, is one of the most advanced and powerful language models created by OpenAI. [14]

ChatGPT is designed to understand and generate human-like text based on the input it receives. It has been trained on a diverse range of internet text and is capable of performing various natural language processing tasks, such as answering questions, engaging in conversations, providing information, and more. [14]

The model operates by predicting the next word in a sentence given the context of the preceding words. It is trained on a massive amount of data, allowing it to learn grammar, context, and even some reasoning abilities. ChatGPT has been widely used for generating human-like responses in chat-based applications and has garnered attention for its versatility and ability to understand and generate coherent and contextually relevant text.[14]

1.6 The Problem

"In software development, ensuring compliance of written source code with MISRA-C standards entails a considerable cost. Nowadays, companies often rely on external tools to detect non-compliance with these rules.One of the most used tools is CPPCHECK is a static analysis tool [8] for C/C++ code. It provides unique code analysis [9] to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to have very few

false positives. CPPCHECK is designed to be able to analyze your C/C++ code even if it has non-standard syntax (common in embedded projects).[6]

Unfortunately, it has three major disadvantages:

1. It detects false positives.
2. Software developers need to manually review and correct the detected issues.
3. Customization and exception creation efforts are required to adapt/select the MISRA-C rules applicable to a software context. In other words, each company and software implementation uses MISRA-C differently.

This TFC solves the presented disadvantages by using an IA solution based on Open AI prompt to interpret our code and apply the rules via a prompt.

2 Feasibility and Relevance

2.1 Feasibility Assessment

This TFC report presents a comprehensive assessment of the feasibility and relevance of developing AI-driven tools for achieving MISRA-C compliance in safety-critical industries. The report delves into technical feasibility, skill and expertise, resource constraints.

2.1.1 Technical Feasibility

To be able to work we need to have access to a complete use case containing thousands of lines of C source code and a validation suite that confirm the applied changes have maintained the software correctness

Hence, we use as the use-case AIR[109] Air is an open source real-time hypervisor for safety-critical embedded space software implementing time and space partitioning (TSP). The complete suit of AIR is provided through an WSL[111] image running in Ubuntu[110] called AIR-UBUNTU or “doxigia.”Skill and Expertise

It's essential to know what MISRA-C and CHATGPT are and the rules of MISRA-C in order to be able to analyse and see if the answers generated from CHATGPT were correct or not.

2.1.2 Resource Constrains

Since the college does not provide a CHATGPT api key and the author had to put his own money into CHATGPT to be able to use prompts this is very critical due to lack of resources and limits our results due to financial reasons.

2.2 Relevance Assessment

2.2.1 Research Gap

See section 2.1.2 about resource constraints.

2.2.2 Stakeholder Perspectives

MISRA-C (Motor Industry Software Reliability Association – C) was created by experts in the automotive industry. Initiates by MISRA . MISRA-C is a set of rules for writing C code that is safe and reliable. It was created by a group of car companies in the early 1990s and is now widely used in the automotive industry. MISRA-C helps to make sure that software in cars is safe and reliable.

2.3 Limitations Considerations

2.3.1 Algorithmic Bias

AI applications, stressing the need to detect and mitigate bias to ensure fair compliance assessments using AI tools. It underscores the importance of specifying the domains of AI tool application.

The section provides examples and deeper insights into bias, as well as a brief explanation of how AI models learn, highlighting their training data, such as GitHub repositories, and the

potential biases therein. It advocates for ethical and unbiased AI practices for trustworthy outcomes.

The exact data set used to train ChatGPT is confidential, but there are some examples that have been used, Wikipedia,Stack Overflow,GitHub,Google Scholar,OpenAI.

3 Benchmarking

3.1 State of art in Development AI Tools

Code Review Automation:

Code review is a technique that helps engineering teams improve the quality of the codebase through collaboration. It's a process by which multiple developers work together to evaluate each other's code. The resulting peer review process naturally reduces mistakes.

When done effectively, code reviews improve code quality, improve developer performance, and save engineering teams time.

[25]

What Is AI Code Review?

Artificial Intelligence (AI) code review is an automated process that examines the code of a software application for potential problems and inefficiencies. It involves the use of machine learning models to identify and fix coding errors, optimize code performance, and make recommendations for improvements. [26]

What is AI in Software Development and How Does It Work?

Before we explore its technical intricacies, let's dissect what AI in software development entails. At its core, artificial intelligence [38]emulates human intelligence processes with algorithms and data. Some versions of AI automate human, manual tasks, others mimic cognitive functions associated with the human mind, and still others do both. When applied to software development, AI becomes a powerful tool.

AI software development automates a great deal of the repetitive and laborious aspects of building an application. Examples are code duplication, manual testing, creating and updating UI elements and layouts for multiple screens, and configuring build scripts. AI in software development also handles advanced problem-solving, data analysis, and intelligent decision-making, all of which are powered by the vast computational capabilities of machines.[27]

3.2 List of Development AI Tools and Features

1. ChatGPT

ChatGPT[39], an OpenAI creation, is a dynamic language model known for its remarkable ability to produce lifelike text. With a knack for crafting natural conversations, elucidating queries, and aiding creative writing, ChatGPT showcases exceptional versatility.

This innovation finds its stride in customer support bots, content creation, and author brainstorming, fostering success across multiple fields. Its forte in natural language is a boon, continually refined through updates.

Yet, challenges arise in its potential for misinformation and bias, coupled with limitations in comprehending intricate contextual nuances.

Key features:

- Generates text that closely resembles human language.
- Engages in conversations that feel natural and authentic.
- Provides detailed and insightful answers to a wide range of queries.
- Offers valuable support and suggestions for creative writing endeavors.
- Demonstrates the ability to understand and respond in contextually relevant ways.

Use cases:

- Revolutionized customer support by powering interactive chatbots for businesses.
- Enhances content creation, from articles to marketing materials, streamlining the writing process.
- Help writers in brainstorming sessions, overcoming creative blocks with innovative ideas.

2. Scribe

Scribe, the AI writing assistant, revolutionizes content creation. Its prowess in summarizing articles, crafting reports, and aiding academic writing is unparalleled. This tool empowers journalists, students, and professionals, streamlining research and writing. Scribe excels in tailored tasks, supercharging productivity, although intricate creative writing may demand human touch for precision.

Key features:

- Dedicated AI writing assistant.
- Generates content in diverse styles and formats.
- Summarizes articles, creates reports, and aids academic writing.
- Assists in training and documentation.
- Simplify documenting complex tasks.

Use cases:

- Utilized by journalists, students, and professionals in research and content creation.
- Useful for generating high-quality documentation and visuals.

- Creates onboarding materials for new hires, and makes onboarding easy for hiring managers.

Help companies in onboarding and training clients by creating step-by-step guides.

3. AlphaCode

AlphaCode, a revolutionary coding assistant, leverages generative AI to empower developers. It excels in writing code, bug resolution, and suggesting optimal programming solutions. Developers embrace AlphaCode for hastened workflows, task automation, and language acquisition.

The tool enhances efficiency, curbing errors and fostering coding proficiency. However, intricate situations might lead to suboptimal code generation due to its reliance on established programming patterns.

Key features:

- Utilizes advanced generative AI for coding assistance.
- Supports various programming languages and paradigms.
- Offers real-time code suggestions and bug fixes.
- Provides code optimization solutions.
- Assists in collaborative coding with shared suggestions.

Use cases:

- Streamline coding workflows to accelerate project development.
- Automate repetitive tasks for enhanced productivity.
- Facilitate rapid learning of new programming languages.
- Efficiently resolve coding errors and bugs.

4. GitHub Copilot

GitHub Copilot revolutionizes coding through collaborative features and integration with popular code editors. It offers code snippets, explanations, and context-based guidance, enhancing developer productivity and learning. It's a versatile tool, enabling faster coding, facilitating learning, and supporting various programming languages[49]. Nonetheless, vigilance is required as some generated code might need polishing, and it heavily depends on external APIs for suggestions.

Key features:

- Seamlessly integrates with popular code editors such as Visual Studio Code[1].
- Generates not only code snippets but also explanations and contextual information to aid developers.
- Provides instant and relevant suggestions for code completion, improving coding efficiency.
- Offers a diverse range of programming language support, accommodating various projects.
- Learns from usage patterns, adapting to individual developer preferences.

Use cases:

- Accelerates the coding process, enabling developers to meet deadlines more effectively.
- Serves as an educational tool, helping programmers grasp new programming concepts.
- Elevates code quality by promoting adherence to coding best practices and standards.
- Assists newcomers in understanding codebases and making meaningful contributions.
- Empowers developers to explore innovative solutions by suggesting alternative approaches.

5. GPT-4

GPT-4 represents a leap forward in AI language models, boasting enhanced text generation across various domains. Its prowess shines in content creation, aiding writers, marketers, and educators. Its versatile use in natural language processing tasks elevates data analysis.

Success stories reveal its aid in generating innovative narratives and personalized conversational experiences. While GPT-4[51] shows improved quality and versatility, vigilance is still essential due to the potential for inaccuracies and biases.

Key features:

- Generates high-quality text across diverse subjects.
- Produce more coherent and nuanced responses.
- Generates content that feels even more natural and human-like.
- Tackles intricate questions and tasks with greater accuracy.
- Empower users to steer the tone, style, and specifics of generated text.

Use cases:

Vital in content creation, GPT-4 aids writers, bloggers, and marketers with engaging material.

Empowers natural language processing tasks, enhancing sentiment analysis and language understanding.

Serves as a creative writing companion, sparking inspiration for authors and screenwriters.

6. Bard

Bard emerges as a cutting-edge chatbot and content-generation tool developed by Google. It leverages LaMDA, a transformer-based model, and stands as Google's response to ChatGPT. Presently in its experimental phase, Bard caters to a limited user audience in the US and UK.

Key features:

- Harnesses the power of LaMDA, a robust transformer-based model.
- Limited-access waitlist catering to select customers in the US and UK.
- Incorporates a user response rating mechanism.
- Accessible via individual Google accounts.

- Empowers users with assistance in software development and programming-related tasks.

Use cases:

- Aids in brainstorming ideas, generating code snippets and solving programming-related queries.
- Assists in drafting written content, articles, blog posts, and creative pieces.
- Help users in research and provide concise information on a wide range of topics.
- Serves as a study partner, helping users understand complex concepts and providing explanations.

7. Cohere Generate

Cohere Generate's versatile capabilities empower developers to craft dynamic dialogue systems, enhancing user engagement. Its strength lies in personalized content creation, benefiting marketing campaigns by generating custom emails. While its adaptability is praised, refining the fine-tuning process remains crucial to maintaining coherent context across interactions.

Key features:

- Tailored for diverse natural language generation tasks.
- Creates interactive conversational agents with ease.
- Proficient in crafting personalized email content.
- Delivers human-like responses for authentic interactions.

Use cases:

- Empowers companies to build interactive chatbots for enhanced user experiences.
- Streamlines customer communication through automated responses.
- Facilitates the creation of personalized messages, enhancing client engagement.
- Successful deployment in various sectors, including e-commerce and customer support.

8. Dall-E2

Dall-E2, a cutting-edge generative AI model, excels in image synthesis. By translating text into captivating visuals, it empowers artists and designers to explore new realms of creativity. Notable successes include producing unique artwork and custom images. While bridging the text-image gap, it fuels innovation, despite occasional inaccuracies and constraints within its trained concepts.

Key features:

- Generative AI model specializing in image synthesis.
- Transforms textual prompts into intricate visual content.
- Accommodates a wide range of image styles and genres.
- Offers control over image attributes like composition and lighting.
- Supports both high-level concept visualization and detailed imagery.

Use cases:

- Empowers artists to envision ideas.
- Designers can create novel visual concepts.
- Content creators can generate captivating custom images.
- Dall-E2 aids in creating stunning virtual worlds.

9. Claude

Claude stands as a state-of-the-art AI assistant crafted by Anthropic, embodying the result of dedicated research into creating AI systems that are not only helpful but also just and secure.

Key features:

- Proficiently handles extensive text data processing.
- Engages in natural and fluent conversations.
- Displays multilingual proficiency, spanning common languages and programming languages.
- Streamlines and automates complex workflows.
- Seamlessly adapts to user feedback for continuous improvement.

Use cases:

- Assists programmers in coding tasks across languages.
- Streamlines business processes for enhanced efficiency.
- Supports students with interactive educational experiences.

10. Synthesia

Synthesia AI revolutionizes content creation by enabling the generation of lifelike videos using text inputs. Through advanced deep learning techniques, it seamlessly merges text with realistic visuals, transforming concepts into engaging visual experiences. This technology finds applications in marketing, entertainment, and education, reshaping the way we communicate and visualize ideas.

Key features:

- The AI-powered platform converts text to videos efficiently.
- Creates dynamic visuals, avatars, and scenes.
- Automated voice synthesis for seamless audio.
- Users can tweak visuals, text, and voice style.
- Rapidly produce multiple videos simultaneously.

Use cases:

- Assist marketers in creating advertisements and marketing campaigns.
- Content creators can develop tutorials and personalized video messages.
- Educators can produce educational videos and presentations.

11. Duet AI

Duet AI for Google Workspace seamlessly integrates with Google applications. In the future, you'll discover features for content generation, summarization, and content rewriting within familiar tools such as Gmail, Google Docs, and Google Meet. It uses a robust large language model (LLM) as its foundation. If you've used AI chatbots like ChatGPT, you're acquainted with this concept.

Nevertheless, Duet takes it a step further by seamlessly integrating with well-known Google applications such as Gmail and Meet. Thanks to these seamless integrations, the need for manual copying, pasting, or exporting content between different programs is eliminated. The product is currently in beta and here are the features you can expect from Duet AI for Google Workspace:

Key features:

- Leverages generative AI capabilities to facilitate text generation
- Provides AI-driven code assistance for cloud users.
- Easy image generation based on prompts.
- Enables users to create intelligent business applications.
- Build workflows within Google Workspace using natural language commands.

Use cases:

- Help users with tasks such as writing, organizing, and visualizing data.
- Aids seasoned developers by swiftly suggesting optimized code for complex projects.
- Assist users in preparing for meetings, organizing their day, and visualizing solutions.

Conclusion

Generative AI has undoubtedly ushered in a new era of possibilities, where machines are not just mimicking human actions but actively contributing to creative processes. From generating code snippets to composing music and even crafting visual content, the tools, and platforms discussed in this blog exemplify the remarkable progress of technology.

As we move forward, it is essential to recognize both the potential and limitations of generative AI, embracing its power while understanding the need for human oversight and refinement. With these 11 cutting-edge generative AI tools, industries are well on their way to achieving unprecedented levels of efficiency, creativity, and innovation.

At Turing, companies can leverage our generative AI services[48] and the expertise of skilled AI engineers to turn innovative ideas into reality. Turing emphasizes the collaboration between human creativity and AI capabilities, ensuring refined and tailored outcomes that drive industries toward remarkable achievements. Talk to our expert today and join 900+ Fortune 500 companies and fast-scaling startups that have trusted Turing for their engineering needs.

[16]

3.3 Features for Code Correction (MISRA-C)

Static Code Analysis:

In computer science[36], **static program analysis** (or **static analysis**) is the analysis[41] of computer programs performed without executing them, in contrast with dynamic program[17] analysis, which is performed on programs during their execution.

The term is usually applied to analysis performed by an automated tool, with human analysis typically being called "program understanding", program comprehension[42], or code review[43]. In the last of these, software inspection[44] and software walkthroughs[45] are also used. In most cases the analysis is performed on some version of a program's source code[46], and, in other cases, on some form of its object code[47].

The kinds of deviations that you can find with static analysis are:

• Undefined behaviour

• Using dangerous code patterns

• Coding style There are many faults that you can not find with static analysis. Static analysis tools do not have human knowledge about what your program is intended to do. If the output from your program is valid but unexpected then in most cases this is not detected by static analysis tools. For instance, if your small program writes “Helo” on the screen instead of “Hello” it is unlikely that any tool will complain about that. Static analysis should be used as a complement in your quality assurance. It does not replace any of;

- Careful design
- Testing
- Dynamic analysis
- Fuzzing

[8]

Dynamic Code Analysis:

Dynamic program analysis is analysis of computer software[41] that involves executing the program in question (as opposed to static program analysis[8], which does not). Dynamic program analysis includes familiar techniques from software engineering[52] such as unit testing[53], debugging[54], and measuring code coverage[55], but also includes lesser-known techniques like program slicing and invariant inference. Dynamic program analysis is widely applied in security[56] in the form of runtime memory error detection[57], fuzzing[58], dynamic symbolic execution[59], and taint tracking.

For dynamic program analysis to be effective, the target program must be executed with sufficient test inputs to address the ranges of possible inputs and outputs. Software testing[60] measures, such as code coverage[55], and tools such as mutation testing[61], are used to try to identify cases where testing is inadequate.

For dynamic analysis, care must be taken to minimize the effect that instrumentation[62] has on the execution (including temporal properties) of the target program. Unit tests[64], integration tests[63], system tests[65] and acceptance tests[66] are forms of dynamic testing[67]. [17]

Types of dynamic analysis

Code coverage

Computing the code coverage [55] according to a test suite or a workload is a standard dynamic analysis technique. It identifies cases where code has not been executed during tests—distinguishing between code that is "covered" and code that is not.

This form of analysis identifies negatives (code that is not tested), but does not identify positives (it cannot identify if code has been adequately tested—even for code with "100% coverage"). Code can be caused to be executed by an automated test system, resulting in "code coverage", even if the "tests" do not actually test anything about what that code does.

Gcov[68] is the GNU[69] source code coverage program.

VB Watch[70] injects dynamic analysis code into Visual Basic programs to monitor code coverage[55], call stack, execution trace, instantiated objects and variables.

Dynamic testing

Main article: Dynamic testing[67]

Dynamic testing involves executing a program on a set of test cases.

Memory error detection

Address Sanitizer[71]: Memory error detection for Linux, macOS[72], Windows, and more. Part of LLVM[73].

Bounds Checker[74]: Memory error detection for Windows based applications. Part of Micro Focus[75] DevPartner[76].

Dmalloc[77]: Library for checking memory allocation and leaks. Software must be recompiled, and all files must include the special C header file dmalloc.h.

Intel Inspector[78]: Dynamic memory error debugger for C, C++, and Fortran applications that run on Windows[79] and Linux[80].

Purify[81]: Mainly memory corruption[82] detection and memory leak detection.

Valgrind[83]: Runs programs on a virtual processor and can detect memory errors (e.g., misuse of malloc[84] and free[84]) and race conditions[85] in multithread[86] programs.

Fuzzing

Main article: Fuzzing[58]

Fuzzing is a testing technique that involves executing a program on a wide variety of inputs; often these inputs are randomly generated (at least in part). Gray-box fuzzers[87] use code coverage to guide input generation.

Dynamic symbolic execution

Main article: Concolic testing [89]

Dynamic symbolic execution (also known as DSE or concolic execution) involves executing a test program on a concrete input, collecting the path constraints associated with the execution, and using a constraint solver[90] (generally, an SMT solver[91]) to generate new inputs that would cause the program to take a different control-flow path, thus increasing code coverage of the test suite. [17] DSE can considered a type of fuzzing[88] ("white-box" fuzzing).

Dynamic data-flow analysis

Dynamic data-flow analysis tracks the flow of information from sources to sinks. Forms of dynamic data-flow analysis include dynamic taint analysis and even dynamic symbolic execution[59].[17]

Invariant inference

Daikon[92] is an implementation of dynamic invariant detection. Daikon runs a program, observes the values that the program computes, and then reports properties that were true over the observed executions, and thus likely true over all executions.

Security analysis

Dynamic analysis can be used to detect security problems.

IBM Rational AppScan[93] is a suite of application security solutions targeted for different stages of the development lifecycle. The suite includes two main dynamic analysis products: IBM Rational AppScan Standard Edition, and IBM Rational AppScan Enterprise Edition. In addition, the suite includes IBM Rational AppScan Source Edition—a static analysis tool.

Concurrency errors

Parasoft[94] Jtest [95]uses runtime error detection to expose defects such as race conditions[85], exceptions, resource and memory leaks, and security attack vulnerabilities.

Intel Inspector[96] performs run-time threading and memory error analysis in Windows.

Parasoft[94] Insure++[97] is a runtime memory analysis and error detection tool. Its Inuse component provides a graphical view of memory allocations over time, with specific visibility of overall heap usage, block allocations, possible outstanding leaks, etc.

Google's[98] Thread Sanitizer is a data race detection tool. It instruments LLVM[73] IR to capture racy memory accesses.

Program slicing

Main article: Program slicing[99]

For a given subset of a program's behavior, program slicing consists of reducing the program to the minimum form that still produces the selected behavior. The reduced program is called a "slice" and is a faithful representation of the original program within the domain of the specified behavior subset. Generally, finding a slice is an unsolvable problem, but by specifying the target behavior subset by the values of a set of variables, it is possible to obtain approximate slices using a data-flow algorithm. These slices are usually used by developers during debugging to locate the source of errors.

Performance analysis

Most performance analysis tools[100] use dynamic program analysis techniques.

MISRA-C Compliance Checker:

In order for a claim of MISRA compliance to have meaning, it is necessary to establish:

- Use of a disciplined software development process;
- Exactly which guidelines are being applied;
- The effectiveness of the enforcement methods;
- The extent of any deviations from the guidelines;
- The status of any software components developed outside of the project.

The Guidelines recognize that, in some situations, it is unreasonable or even impossible to comply with a coding guideline and that it is necessary to deviate from its requirements. The freedom to deviate does not necessarily compromise claims of compliance, but it does carry with it great responsibility. In the absence of a disciplined development process, it is easy for that freedom to be abused. At best, that will undermine the credibility of any claims of MISRA compliance; at worst, it will compromise code quality, safety or security. It is therefore important to emphasize that a credible claim of compliance with The Guidelines can only be made when code is developed under a process which meets the principles laid out in this document. The guidance given in this document supersedes the compliance, deviation and process requirements published previously in the various MISRA Guidelines. Note: The various MISRA Guideline documents have been refined and revised over a number of years. This document uses examples and extracts from a number of them, but the issues discussed are equally relevant to all of The Guidelines.[18]

Integration with IDEs:

What is an IDE?

An IDE, or Integrated Development Environment, enables programmers to consolidate the different aspects of writing a computer program.

IDEs increase programmer productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging.

Editing Source Code

Writing code is an important part of programming. We start with a blank file, write a few lines of code, and a program is born! IDEs facilitate this process with features like syntax highlighting and autocomplete.

Syntax Highlighting

An IDE that knows the syntax of your language can provide visual cues. Keywords, words that have special meaning like class in Java, are highlighted with different colors.

Compare these two code samples:

// without syntax highlighting

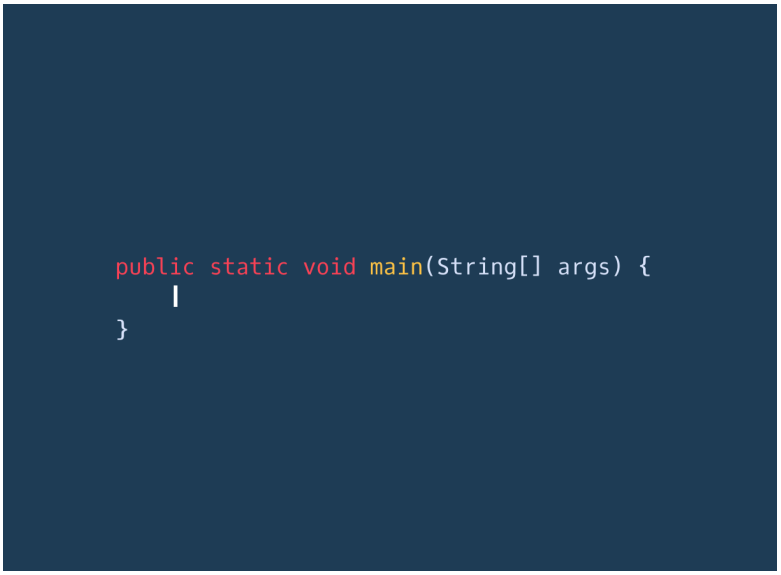
```
public class NiceDay {
  public static void main(String[] args) {
    System.out.println("It's a nice day out!");
  }
}
```

Syntax highlighting makes code easier to read by visually clarifying different elements of language syntax.

Autocomplete

When the IDE knows your programming language, it can anticipate what you're going to type next!

We've seen statements with System.out.println() quite a bit so far. In an IDE, we might see System as an autocomplete option after only typing Sy. This saves keystrokes so the programmer can focus on logic in their code.



Building Executables

Java is a compiled language. Before programs run, the source code of a .java file must be transformed into an executable .class by the compiler. Once compiled, the program can be run from the terminal.

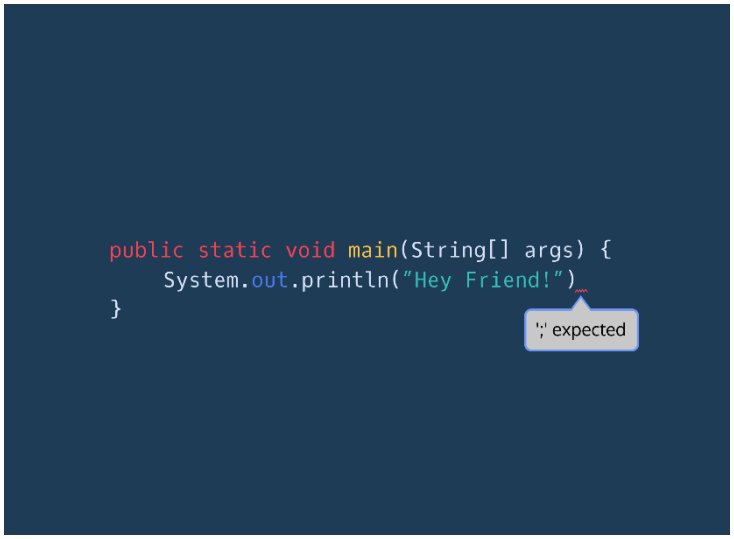
This compilation process is necessary for every program, so why not have the IDE do it for us? IDEs provide automated build processes for languages, so the act of compiling and executing code is abstracted away, like in Codecademy lessons.

Debugging

No programmer avoids writing bugs and programs with errors.

When a program does not run correctly, IDEs provide debugging tools that allow programmers to examine different variables and inspect their code in a deliberate way.

IDEs also provide hints while coding to prevent errors before compilation.



[19]

3.4 List of Code Correction Tools and Features

CPPCHECK:

Unique code analysis that detect various kinds of bugs in your code.

Both command line interface and graphical user interface are available.

Cppcheck has a strong focus on detecting undefined behaviour.[6]

SonarQube:

First what is SonarQube?

SonarQube[101] is a self-managed, automatic code review tool that systematically helps you deliver Clean Code. As a core element of our Sonar solution[102], SonarQube integrates into your existing workflow and detects issues in your code to help you perform continuous code inspections of your projects. The product analyses 30+ different programming

languages[103] and integrates into your Continuous Integration pipeline of DevOps platforms[104] to ensure that your code meets high-quality standards.

Features:

SonarLint[105] provides immediate feedback in your IDE as you write code so you can find and fix issues before a commit.

SonarQube's PR analysis[106] fits into your CI/CD workflows with SonarQube's PR analysis and use of quality gates.

Quality gates[107] keep code with issues from being released to production, a key tool in helping you incorporate the Clean as You Code methodology.

The Clean as You Code[108] approach helps you focus on submitting new, clean code for production, knowing that your existing code will be improved over time. [20]

Coverity (Synopsys):

Coverity® provides comprehensive static analysis that empowers developers and security teams to deliver high-quality software that complies with security, functional safety, and industry standards.[21]

PC-lint / FlexeLint (by Gimpel Software):

PC-lint Plus is a static analysis tool that finds defects in software by analyzing C and C++ source code.

Like a compiler, PC-lint Plus parses your source code files, performs semantic analysis, and builds an abstract syntax tree to represent your program.

Advantages:

Identify a wide range of defects and vulnerabilities

Deep analysis to find potential bugs and suspicious code

Quickly identify root cause and provide actionable fixes

Support for coding Standards such as MISRA, AUTOSAR, and CERT C

Certified for ISO 26262 and IEC 61508

[22]

3.5 About CPPCHECK

CPPCHECK is an analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to detect only real errors in the code, and generate as few false positives (wrongly reported warnings) as possible. CPPCHECK is designed to analyze your C/C++ code even if it has non-standard syntax, as is common in for example embedded projects. Supported code and platforms:

• Cppcheck checks non-standard code that contains various compiler extensions, inline assembly code, etc.

• Cppcheck should be compilable by any compiler that supports C++11 or later.

• Cppcheck is cross platform and is used in various posix/windows/etc environments. The checks in Cppcheck are not perfect. There are bugs that should be found, that Cppcheck fails to detect.

[7]

3.6 Contextualization

First what is a What is Software Engineering?

The Software Engineering branch of engineering focuses on developing software products utilizing various scientific principles, techniques, and procedures. Software engineering leads to a product that is efficient and reliable.

Software Engineering is the combination of two words, Software, and engineering. The Software comprises integrated programs designed carefully with organized instructions and codes. And engineering refers to inventing, designing, building, maintaining, and improving devices and processes using scientific and practical knowledge.[23]

Importance of Software Engineering

The importance of software engineering lies in the fact that a specific piece of Software is required in almost every industry, every business, and purpose. As time goes on, it becomes more important for the following reasons.

1. Reduces Complexity

Dealing with big Software is very complicated and challenging. Thus, to reduce the complications of projects, software engineering has great solutions. It simplifies complex problems and solves those issues one by one.

2. Handling Big Projects

Big projects need lots of patience, planning, and management, which you never get from any company. The company will invest its resources; therefore, it should be completed within the deadline. It is only possible if the company uses software engineering to deal with big projects without problems.

3. To Minimize Software Costs

Software engineers are paid highly as Software needs a lot of hard work and workforce development. These are developed with the help of a large number of codes. But programmers in software engineering project all things and reduce the things which are not needed. As a result of the production of Software, costs become less and more affordable for Software that does not use this method.

4. To Decrease Time

If things are not made according to the procedures, it becomes a huge loss of time. Accordingly, complex Software must run much code to get definitive running code. So, it takes lots of time if not handled properly. And if you follow the prescribed software engineering methods, it will save your precious time by decreasing it.

5. Effectiveness

Making standards decides the effectiveness of things. Therefore, a company always targets the software standard to make it more effective. And Software becomes more effective only with the help of software engineering.

6. Reliable Software

The Software will be reliable if software engineering, testing, and maintenance are given. As a software developer, you must ensure that the Software is secure and will work for the period or subscription you have agreed upon.

[23]

And where does the AI plays a vital role in software development?

The Role of AI in Software Development: Trends, Statistics, and Growth in 2023

The role of AI in software development has taken on a whole new direction in recent years. While AI (artificial intelligence) has yet to reach a point where it can completely function without human input in conceptualizing, creating, testing, and launching software products, it has made the overall turnaround time much faster than it once was.

As experts in software development, our team here at Aloa aims to streamline the software development process to make it as efficient and productive as possible. Having the ability to use various AI tools, applications, and practices makes that possible. [24]

3.7 Identification of Gaps

Context in Artificial Intelligence: A Critical Component

In the realm of artificial intelligence, context functions not merely as an added feature but as a fundamental component that dictates the effectiveness and reliability of AI systems. This section explores how AI processes context, its applications where context is vital, and the challenges it faces in understanding and utilizing context.

What is a Confusion Matrix:

Confusion matrix is a very popular measure used while solving classification problems. It can be applied to binary classification as well as for multiclass classification problems. An example of a confusion matrix for binary classification is shown in Table Confusion Matrix.[13]

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Confusion matrices represent counts from predicted and actual values. The output “TN” stands for True Negative which shows the number of negative examples classified accurately. Similarly, “TP” stands for True Positive which indicates the number of positive examples classified accurately. The term “FP” shows False Positive value, i.e., the number of actual negative examples classified as positive; and “FN” means a False Negative value which is the number of actual positive examples classified as negative. One of the most commonly used metrics while performing classification is accuracy. The accuracy of a model (through a confusion matrix) is calculated using the given formula below. [13]

3.8 About MISRARAIVOSO

MISRAivoso: Automated MISRA-C Code Correction Tool

MISRAivoso is a powerful Python script designed to facilitate adherence to MISRA-C coding standards by automating code corrections. This tool utilizes the cppcheck static analysis tool to generate an XML report highlighting violations of MISRA-C guidelines. MISRAivoso then rectifies specific rule violations, ensuring code accuracy and maintainability.

4. Engineering

4.1. Requirement Gathering and Analysis

First solution:

Id	MISRA-PLUGIN-00040
Title	Generate the html with CPPCHECK
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
We run the CPPCHECK with the code that the user gave. The CPPCHECK run and generate the Html to be used on the other requisites.	

Id	MISRA-PLUGIN-0001
Title	Install Air-Ubuntu(doxigia)

Type	Non-Functional
Component	System
Estimated time	2 month
SIZE SCALE	Must have
Acceptance criteria	• Have 25 gb free in the disk • Have a user login
Description:	
We need to installed Air-Ubuntu(doxigia) for that we need available space and be a user login that have access.	

Id	MISRA-PLUGIN-0002
Title	Create a code that does the prompt
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	• Have Air-Ubuntu (doxigia) installed on the pc • Have a python file created
Description:	
We need to have a python file already created and Air-Ubunto (doxigia) already installed.	

Id	MISRA-PLUGIN-0003
Title	Test on prompt that include MISRA-C rule 1.1
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0003.py ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 1.1 .	

Id	MISRA-PLUGIN-0004
Title	Test on prompt that include MISRA-C rule 2.1-2.4
Type	Functional
Component	Plugin Interface
Estimated time	4 days

SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0004.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 2.1,2.2,2.3,2.4.	

Id	MISRA-PLUGIN-0005
Title	Test on prompt that include MISRA-C rule 1 and 2
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0003.py

	● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0004.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0005.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0003 and MISRA-PLUGIN-0004 Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains that will join the two rules done in MISRA-PLUGIN-0003 and MISRA-PLUGIN-0004	

Id	MISRA-PLUGIN-0006
Title	Test on prompt that include MISRA-C rule 4
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0006.py

	• Have a code in c that we want to test • See if the output is what we pretended • Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 4.1,4.2.	

Id	MISRA-PLUGIN-0007
Title	Test on prompt that include MISRA-C rule 5
Type	Functional
Component	Plugin Interface
Estimated time	4 days

SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0007.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 5.1,5.2,5.3,5.4,5.5,5.6,5.7.	

Id	MISRA-PLUGIN-0008
Title	Test on prompt that include MISRA-C rule 4 and 5
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0006.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0007.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0008.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0006 and MISRA-PLUGIN-0007	

Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains that will join the two rules done in MISRA-PLUGIN-0006 and MISRA-PLUGIN-0007

Id	MISRA-PLUGIN-0009
Title	Test on prompt that include MISRA-C rule 6
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0009.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 6.1,6.2,6.3,6.4,6.5.	

Id	MISRA-PLUGIN-0010
Title	Test on prompt that include MISRA-C rule 7
Type	Functional

Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0010.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
First we create a prompt that contains MISRA-C rule 7.1.	

Id	MISRA-PLUGIN-00011
Title	Test on prompt that include MISRA-C rule 6 and 7
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0009.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0010.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0011.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	

First we need to have these 2 requisites already done MISRA-PLUGIN-0009 and MISRA-PLUGIN-0010 Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains that will join the two rules done in MISRA-PLUGIN-0009 and MISRA-PLUGIN-0010

Id	MISRA-PLUGIN-0012
Title	Test on prompt that include MISRA-C rule 8.1-8.12
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0012.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 8.1,8.2,8.3,8.4,8.5,8.6,8.7,8.8,8.9,8.10,8.11,8.12.	

Id	MISRA-PLUGIN-0013
Title	Test on prompt that include MISRA-C rule 9.1-9.3

Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0013.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 9.1,.9.2,9.3.	

Id	MISRA-PLUGIN-0014
Title	Test on prompt that include MISRA-C rule 8.1-8.12 and 9.1-9.3
Type	Functional
Component	Plugin Interface
Estimated time	4 days

SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0012.py working with good efficiency ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0013.py working with good efficiency ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0014.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0012 and MISRA-PLUGIN-0013 Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains that will join the two rules done in MISRA-PLUGIN-0012 and MISRA-PLUGIN-0013	

Id	MISRA-PLUGIN-0015
Title	Test on prompt that include MISRA-C rule 10
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc

	● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0015.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 10.1,10.2,10.3,10.4,10.5,10.6.	

Id	MISRA-PLUGIN-0016
Title	Test on prompt that include MISRA-C rule 11
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0016.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains MISRA-C rule 11.1,11.2,11.3,11.4,11.5.	

Id	MISRA-PLUGIN-0017
Title	Test on prompt that include MISRA-C rules 10 and 11
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0015.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0016.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0017.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0015 and MISRA-PLUGIN-0016 Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains that will join the two rules done in MISRA-PLUGIN-0015 and MISRA-PLUGIN-0016	

Id	MISRA-PLUGIN-0018
Title	Create and Test on prompt that includes all 11 rules of MISRA-C
Type	Functional
Component	Plugin Interface
Estimated time	10 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0017.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0014.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0011.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0008.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0005.py ● Have a code in c that we want to test ● See if the output is what we pretended ● Have CPPCHECK.py
Description:	
First we need to have these 5 requisites already done MISRA-PLUGIN-0005, MISRA-PLUGIN-0008, MISRA-PLUGIN-00011, MISRA-PLUGIN-0014, MISRA-PLUGIN-0017,	

Complete this requirement first MISRA-PLUGIN-0040 then we create a prompt that contains that will join all the rule done on the requisites mentioned .

Id	MISRA-PLUGIN-0030
Title	Testing and fine-tune the prompt that includes 11 rules of MISRA-C
Type	Non-Functional
Component	Plugin Interface
Estimated time	10 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (doxigia) installed on the pc ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0017.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0014.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0011.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0008.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0005.py ● Have the file with the prompt we pretend in this case MISRA-PUGLIN-0018.py ● Have a code in c that we want to test ● See if the output is what we pretended

Description:	
First we need to have these 5 requisites already done MISRA-PLUGIN-0005, MISRA-PLUGIN-0008, MISRA-PLUGIN-00011, MISRA-PLUGIN-0014, MISRA-PLUGIN-0017, MISRA-PLUGIN-0018. We are gonna enter a Testing fase and fine-tune the prompt that includes 11 rules of MISRA-C .	

Second solution :

Id	MISRA-PLUGIN-0001
Title	Install Air-Ubuntu(MISRAPLUGIN)
Type	Non-Functional
Component	System
Estimated time	2 months
SIZE SCALE	Must have
Acceptance criteria	• Have 25 gb free in the disk • Have a user login
Description:	
We need to installed Air-Ubuntu(MISRAPLUGIN) for that we need available space and be a user login that have access.	

Id	MISRA-PLUGIN-0002
Title	Create a code that does the prompt
Type	Functional
Component	Plugin Interface
Estimated time	2 months
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a python file created
Description:	
We need to have a python file already created and Air-Ubunto (MISRAPLUGIN) already installed.	

Id	MISRA-PLUGIN-0003
Title	Create and Test on prompt that include MISRA-C rule 1.1
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubunto (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	

First we create a prompt . Second we use the prompt to test with the code if the prompt does the MISRA-C rule 1.1

Id	MISRA-PLUGIN-0004
Title	Create and Test on prompt that include MISRA-C rule 2.1-2.4
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 2.1,2.2,2.3,2.4.	

Id	MISRA-PLUGIN-0005
Title	Create and Test on prompt that include MISRA-C rule 1 and 2
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	

First we need to have these 2 requisites already done MISRA-PLUGIN-0003 and MISRA-PLUGIN-0004

Second we use the prompt to test with the code if the prompt does both rules in MISRA-PLUGIN-0003 and MISRA-PLUGIN-0004

Id	MISRA-PLUGIN-0006
Title	Create and Test on prompt that include MISRA-C rule 4

Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	• Have Air-Ubuntu (MISRAPLUGIN) installed on the pc • Have a code in c that we want to test • See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 4.1,4.2.	

Id	MISRA-PLUGIN-0007
Title	Create and Test on prompt that include MISRA-C rule 5
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	• Have Air-Ubuntu (MISRAPLUGIN) installed on the pc • Have a code in c that we want to test • See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 5.1,5.2,5.3,5.4,5.5,5.6,5.7.	

Id	MISRA-PLUGIN-0008
Title	Create and Test on prompt that include MISRA-C rule 4 and 5
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0006 and MISRA-PLUGIN-0007 Second we use the prompt to test with the code if the prompt does both rules in MISRA-PLUGIN-0006 and MISRA-PLUGIN-0007	

Id	MISRA-PLUGIN-0009
Title	Create and Test on prompt that include MISRA-C rule 6
Type	Functional
Component	Plugin Interface

Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	• Have Air-Ubuntu (MISRAPLUGIN) installed on the pc • Have a code in c that we want to test • See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 6.1,6.2,6.3,6.4,6.5.	

Id	MISRA-PLUGIN-0010
Title	Create and Test on prompt that include MISRA-C rule 7
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	• Have Air-Ubuntu (MISRAPLUGIN) installed on the pc • Have a code in c that we want to test • See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 7.1.	

Id	MISRA-PLUGIN-00011
Title	Create and Test on prompt that include MISRA-C rule 6 and 7
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0009 and MISRA-PLUGIN-0010 Second we use the prompt to test with the code if the prompt does both rules in MISRA-PLUGIN-0009 and MISRA-PLUGIN-0010	

Id	MISRA-PLUGIN-0012
Title	Create and Test on prompt that include MISRA-C rule 8.1-8.12
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended

Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 8.1,8.2,8.3,8.4,8.5,8.6,8.7,8.8,8.9,8.10,8.11,8.12.	

Id	MISRA-PLUGIN-0013
Title	Create and Test on prompt that include MISRA-C rule 9.1-9.3
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 9.1,.9.2,9.3.	

Id	MISRA-PLUGIN-0014
Title	Create and Test on prompt that include MISRA-C rule 8.1-8.12 and 9.1-9.3
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0012 and MISRA-PLUGIN-0013 Second we use the prompt to test with the code if the prompt does both rules in MISRA-PLUGIN-0012 and MISRA-PLUGIN-0013	

Id	MISRA-PLUGIN-0015
Title	Create and Test on prompt that include MISRA-C rule 10
Type	Functional

Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we use the prompt to test with the code if the prompt does the MISRA-C rule 10.1,10.2,10.3,10.4,10.5,10.6.	

Id	MISRA-PLUGIN-0016
Title	Create and Test on prompt that include MISRA-C rule 11
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	

First we use the prompt to test with the code if the prompt does the MISRA-C rule 11.1,11.2,11.3,11.4,11.5.

Id	MISRA-PLUGIN-0017
Title	Create and Test on prompt that include MISRA-C rule 10 and 11
Type	Functional
Component	Plugin Interface
Estimated time	4 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we need to have these 2 requisites already done MISRA-PLUGIN-0015 and MISRA-PLUGIN-0016 Second we use the prompt to test with the code if the prompt does both rules in MISRA-PLUGIN-0015 and MISRA-PLUGIN-0016	

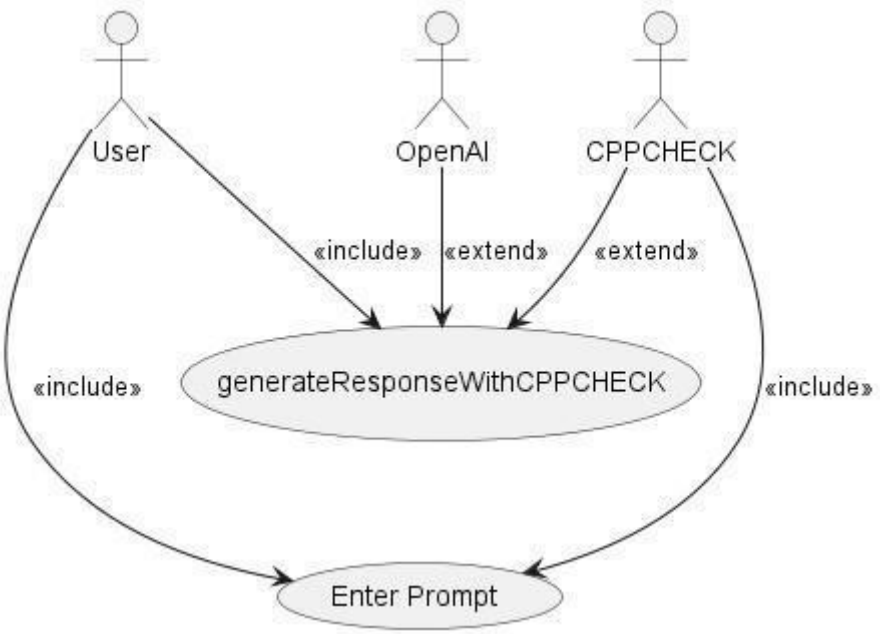
Id	MISRA-PLUGIN-0018

Title	Create and Test on prompt that include 11 rules of MISRA-C
Type	Functional
Component	Plugin Interface
Estimated time	10 days
SIZE SCALE	Must have
Acceptance criteria	● Have Air-Ubuntu (MISRAPLUGIN) installed on the pc ● Have a code in c that we want to test ● See if the output is what we pretended
Description:	
First we need to have these 5 requisites already done MISRA-PLUGIN-0005, MISRA-PLUGIN-0008, MISRA-PLUGIN-00011, MISRA-PLUGIN-0014, MISRA-PLUGIN-0017, Second we use the prompt to test with the code that will join all the rule done on the requisites mentioned .	

Id	MISRA-PLUGIN-0030
Title	Testing and fine-tune the prompt that includes 11 rules of MISRA-C
Type	Non-Functional
Component	Plugin Interface
Estimated time	8 days
SIZE SCALE	Must have

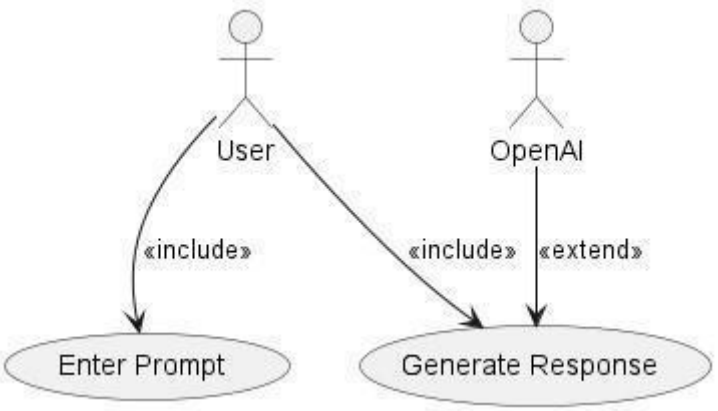
Acceptance criteria	• Have Air-Ubuntu (MISRAPLUGIN) installed on the pc • Have a code in c that we want to test • See if the output is what we pretended
Description:	
First we need to have these 5 requisites already done MISRA-PLUGIN-0005, MISRA-PLUGIN-0008, MISRA-PLUGIN-00011, MISRA-PLUGIN-0014, MISRA-PLUGIN-0017, MISRA-PLUGIN-0018. Enter a Testing fase and fine-tune the prompt that includes 11 rules of MISRA-C .	

4.2. Use Case Diagrams or Description of Application Scenarios



First proposed solution :

-User puts submit is code then the code will be run by CPPCHECK that will generate a html file with instructions about what rules he didn't do right then we send the prompt to CHATGPT and the user will have the final response with the code.

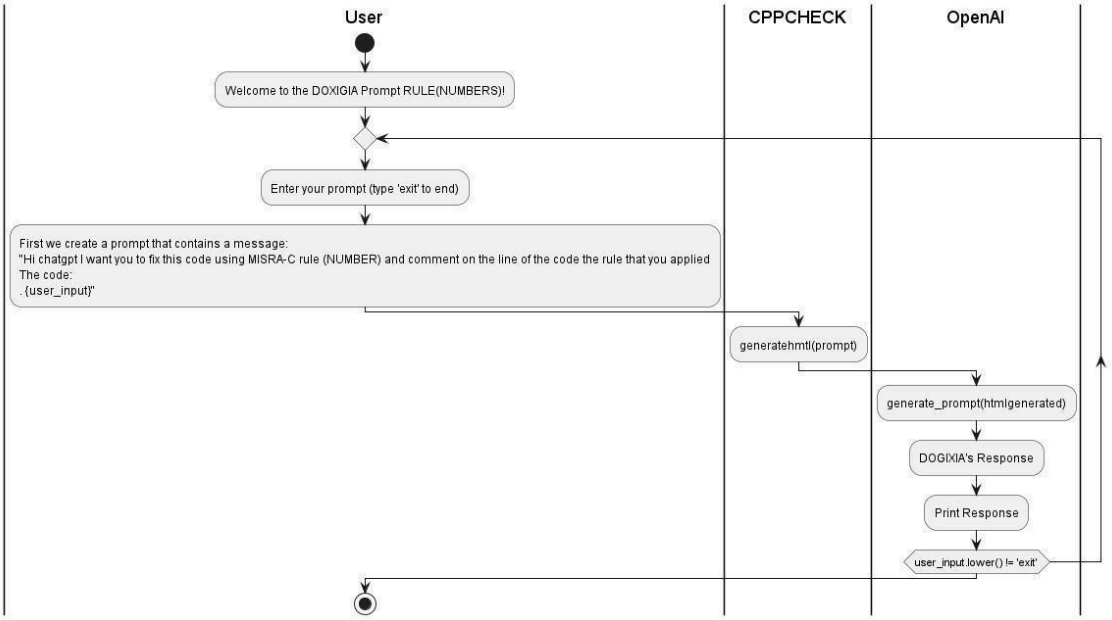


Second proposed solution :

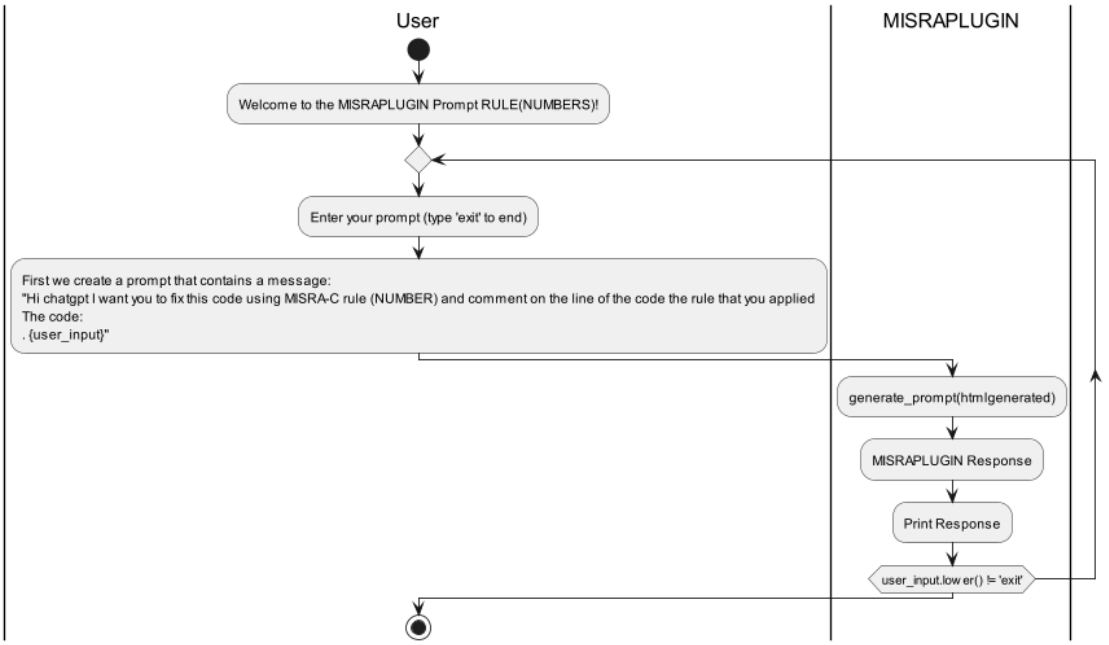
-User puts submit is code then the code send the prompt to CHATGPT then the user will have the final response with the code.

4.3. Activity Diagrams or Equivalent

First solution :



Second solution :



4.4. Structure

First solution :

Windows Operating System:

Hosts the entire system.

Manages the installation of WSL (Windows Subsystem for Linux).

WSL (Air-Ubuntu with Doxigia):

Installed within Windows, providing a Linux-like environment.

Allows running Linux-based programs and tools.

Python Programs within Air-Ubuntu:

a. CPPCHECK:

A program for static code analysis in C and C++.

Runs within the Air-Ubuntu environment.

Utilizes Python for scripting.

b. PromptFile:

A Python program designed for handling file-related operations.

Interacts with CPPCHECK and facilitates communication with CHATGPT.

Operates within the Air-Ubuntu environment.

CHATGPT Integration:

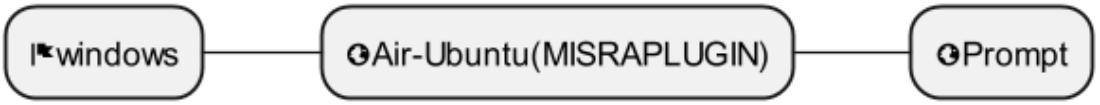
Used as an intelligent assistant for programming-related queries.

Interacts with both PromptFile.

Provides insights, suggestions, and code snippets based on the queries.

Enhances the development and debugging process.

Second solution :



Windows Operating System:

Hosts the entire system.

Manages the installation of WSL (Windows Subsystem for Linux).

WSL (Air-Ubuntu with MISRAPLUGIN):

Installed within Windows, providing a Linux-like environment.

Allows running Linux-based programs and tools.

MISRAPLUGIN within Air-Ubuntu:

A Python program designed for handling file-related operations.

Using the part of typescript the commands and design.

The javascript part pick the prompt from the file using typescript and sends it to CHATGPT.

Interacts with CHATGPT and facilitates communication.

Operates within the Air-Ubuntu environment.

CHATGPT Integration:

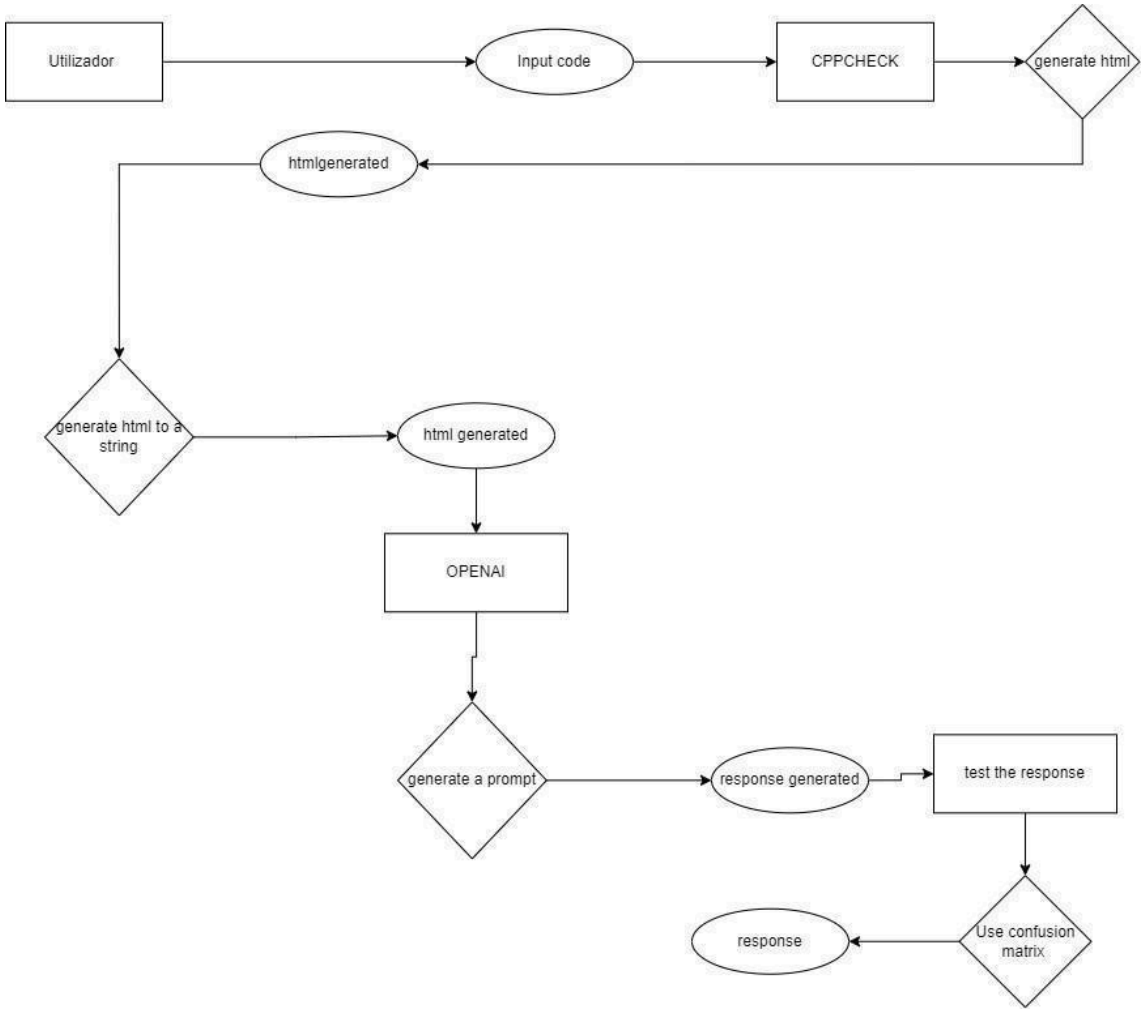
Used as an intelligent assistant for programming-related queries.

Interacts with Prompt.

Provides insights, suggestions, and code snippets based on the queries.

Enhances the development and debugging process.

4.5. Mockup



User Interaction:

Users provide code and input to the system through the Windows environment.

PromptFile:

Calls CPPCHECK :

Gathers the user-provided code and input.

Generates an hmtl that will guide CHATGPT.

PromptFile:

Receives the html and generates it to a string

Calls CHATGPT:

Generates Query:

Constructs a query based on the user's code and input.

Ensures the query is clear and concise for effective communication with CHATGPT.

Communicates with CHATGPT:

Sends the query to CHATGPT for analysis and suggestions.

Retrieves responses from CHATGPT.

Testing CHATGPT Efficiency:

Response Evaluation:

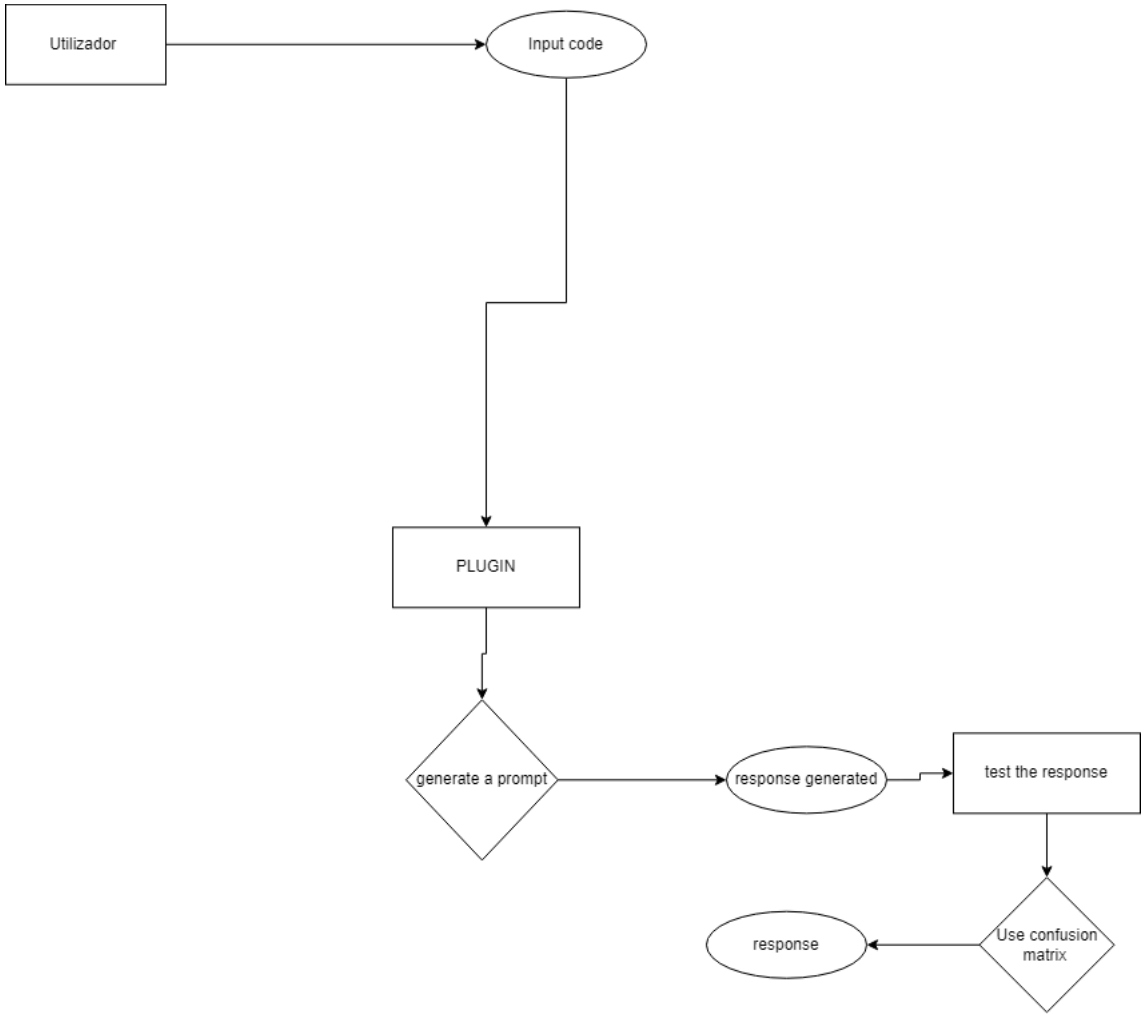
Assesses the efficiency and relevance of CHATGPT's responses.

Evaluates the generated responses against expected outcomes.

Feedback Loop:

Provides feedback on the performance of CHATGPT to refine future interactions.

Iteratively improves the system based on testing outcomes.



User Interaction:

Users provide code and input to the system through the Windows environment.

Prompt:

Receives User Input:

Gathers the user-provided code and input.

Handles File Operations:

Manages the storage and retrieval of code and input files.

Ensures seamless communication between user-provided data and other components.

Calls CHATGPT:

Generates Query:

Constructs a query based on the user's code and input.

Ensures the query is clear and concise for effective communication with CHATGPT.

Communicates with CHATGPT:

Sends the query to CHATGPT for analysis and suggestions.

Retrieves responses from CHATGPT.

Testing CHATGPT Efficiency:

Response Evaluation:

Assesses the efficiency and relevance of CHATGPT's responses.

Evaluates the generated responses against expected outcomes.

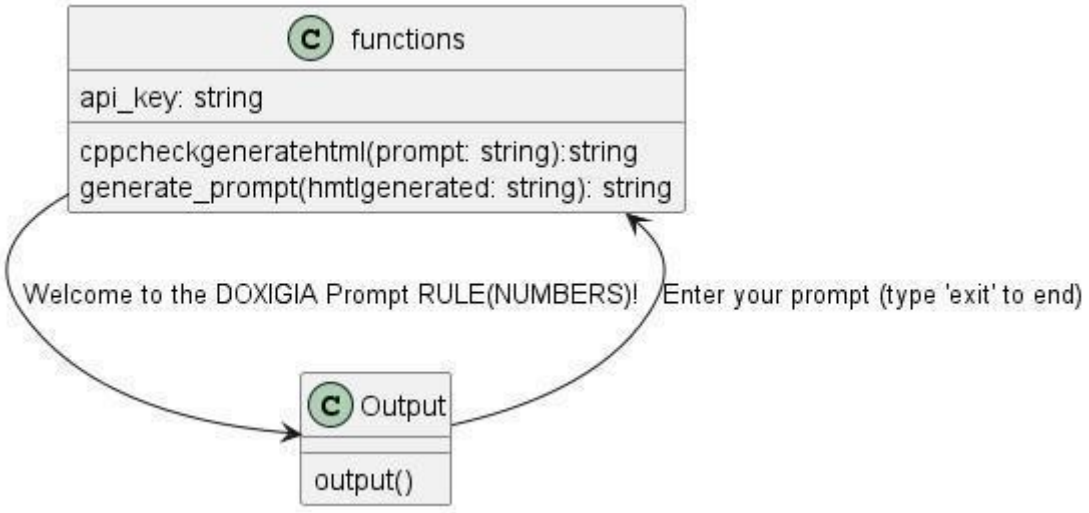
Feedback Loop:

Provides feedback on the performance of CHATGPT to refine future interactions.

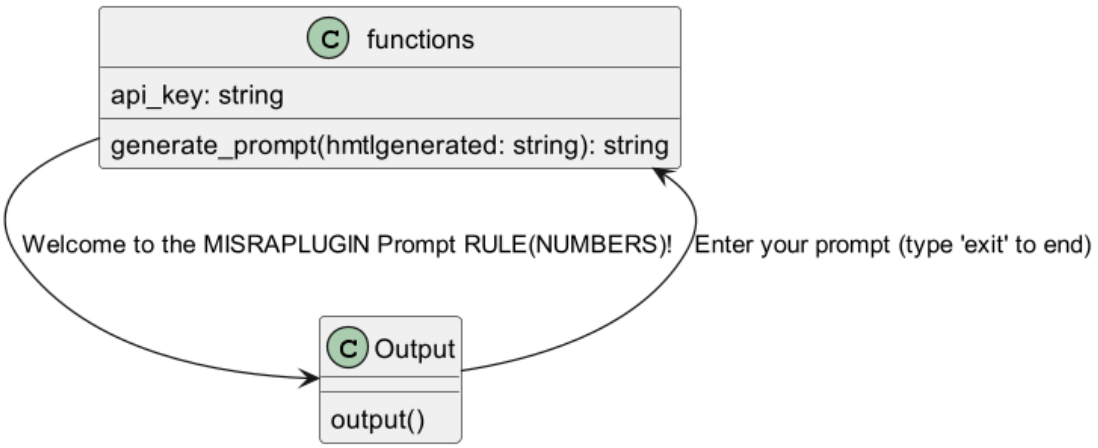
Iteratively improves the system based on testing outcomes.

4.6. Relevant Models

First solution :



Second solution :



5 Proposed solutions

5.1 Introduction

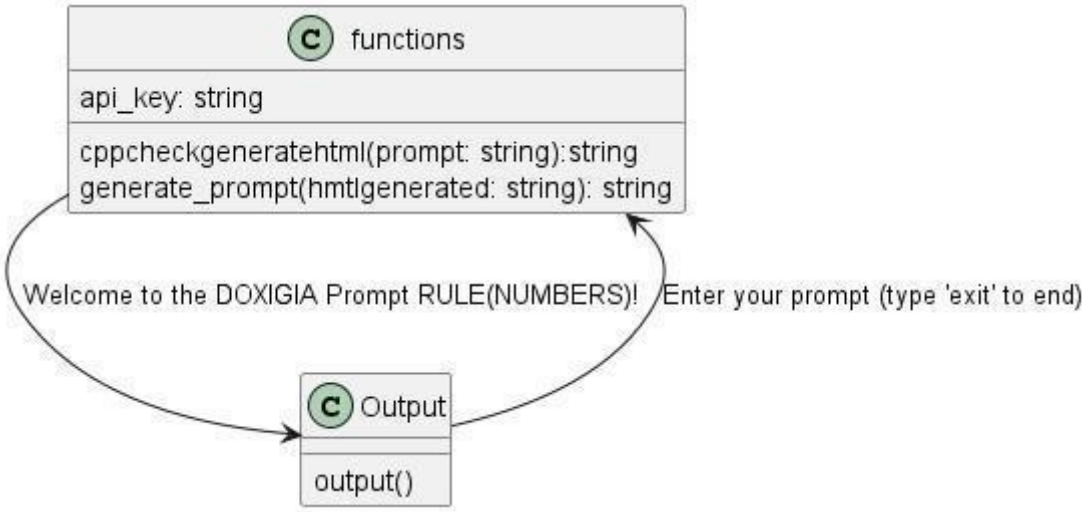
The solution is a software plugin (in this case installed in VSCode[112]) that uses ChatGPT with a prompt that will generate code corrections that align with MISRA-C rules. This step automates the code correction process, ensuring adherence to coding standards then we going to test the corrected code is then rigorously tested to determine whether it now complies with MISRA-C standards. The testing phase evaluates whether the code passes compliance assessments, distinguishing between true positives, false positives, and false negatives. This helps ensure that the code remains both safe and compliant.

youtube : https://youtu.be/ZYhbyd_xHB4

github : https://github.com/DEISI-ULHT-TFC-2023-24/TFC-DEISI2-Modelo-IA-para-MISRA-C

5.2 Architecture

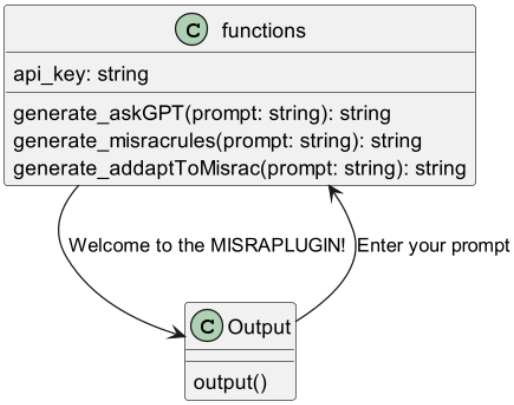
First solution:



Cppcheckgeneratehmtl: is gonna run cppcheck and return string on is html generated

Generate prompt will include all rules like in requisite MISRA-PLUGIN-0018 and the user only is gonna run the return string of cppcheckgeneratehtml.

Second option:



Generate prompt will include all rules like in requisite MISRA-PLUGIN-0018 and the user only is gonna need to put the code.

5.3 Technologies and used tools

API KEY - An API key is a special code that grants access to specific services or data through an API. To use it, you include the key in your requests, either in the URL or headers. This key acts as a secure passcode, identifying and authorizing your application to access the API's resources. It's essential to keep API keys confidential for security reasons.

MISRAPLUGIN WITH VSCODE – the vscode receive the code and sends the prompt to CHATGPT throw an APIKEY that make the connection then a code supposed to be fix will be generated and sent back throw MISRA-C compliance.

MISRAPLUGIN – languages used JavaScript, TypeScript,JSON .

5.4 Implementation

Lets start with the file extension.ts

```
import * as vscode from 'vscode';
import { ChatGPTAPI } from 'chatgpt';
import * as cp from 'child_process';


type AuthInfo = {apiKey?: string};
type Settings = {selectedInsideCodeblock?: boolean, codeblockWithLanguageId?: false, pasteOnClick?: boolean, keepCo


const BASE_URL = 'https://api.openai.com/v1';
```

The first and import is for VS code module to develop an extension.

The second import is for the ChatGPT module API.

The third is the import to be able to execute the commands.

In the type part and to define types for information and authentication settings.

The BASE_URL is the URL of the ChatGPT API.

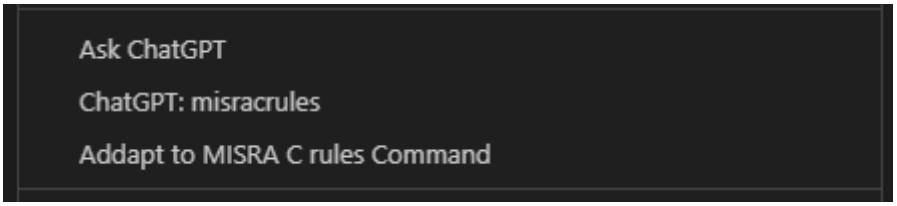
```
12  export function activate(context: vscode.ExtensionContext) {
13
14      console.log('activating extension "chatgpt"');
15
16      const config = vscode.workspace.getConfiguration('chatgpt');
17
18
19      const provider = new ChatGPTViewProvider(context.extensionUri);
20
21
22      provider.setAuthenticationInfo({
23          apiKey: config.get('apiKey')
24      });
25      provider.setSettings({
26          selectedInsideCodeblock: config.get('selectedInsideCodeblock') || false,
27          codeblockWithLanguageId: config.get('codeblockWithLanguageId') || false,
28          pasteOnClick: config.get('pasteOnClick') || false,
29          keepConversation: config.get('keepConversation') || false,
30          timeoutLength: config.get('timeoutLength') || 60,
31          apiUrl: config.get('apiUrl') || BASE_URL,
32          model: config.get('model') || 'gpt-3.5-turbo'
33      });
34
35
36      context.subscriptions.push(
37          vscode.window.registerWebviewViewProvider(ChatGPTViewProvider.viewType, provider,  {
38              webviewOptions: { retainContextWhenHidden: true }
39          })
40      );
41
42
43      const commandHandler = (command:string) => {
44          const config = vscode.workspace.getConfiguration('chatgpt');
45          const prompt = config.get(command) as string;
46          provider.search(prompt);
47      };
```

This is the function that makes the entry point for the extension.

It first fetches the extension settings, creates a ChatGPT instance and registers the extension in that instance (lines 12 to 19).

On lines 22 to 33, it places the settings in the provider's settings, and on lines 35 to 40 it registers with the extension's context.

```
50  context.subscriptions.push(
51      vscode.commands.registerCommand('chatgpt.ask', () =>
52          vscode.window.showInputBox({ prompt: 'What do you want to do?' })
53              .then((value) => provider.search(value))
54      ),
55      vscode.commands.registerCommand('chatgpt.misracrules', () => commandHandler('promptPrefix.misracrules')),
56      vscode.commands.registerCommand('chatgpt.resetConversation', () => provider.resetConversation()),
57      vscode.commands.registerCommand('chatgpt.addapmisrac', () => {
58          const misraCValue = 'Addapt to MISRA C rules and comment every line the rule number';
59          provider.search(misraCValue);
60      })
61  );
```



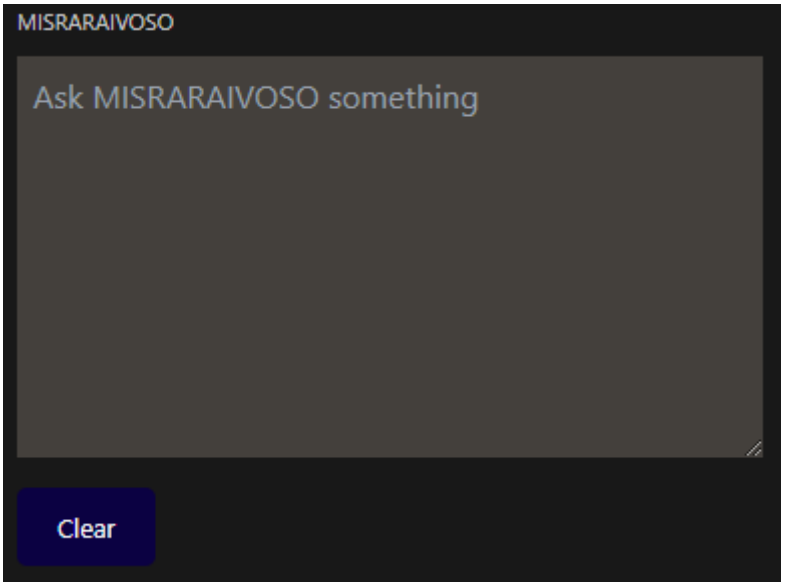
On the lines 55 to 60 registers the commands that can be called from the extension when we press the right button shows the options that we can see in the image above.

```
63    // Change the extension's session token or settings when configuration is changed
64    vscode.workspace.onDidChangeConfiguration((event: vscode.ConfigurationChangeEvent) => {
65        if (event.affectsConfiguration('chatgpt.apiKey')) {
66            const config = vscode.workspace.getConfiguration('chatgpt');
67            provider.setAuthenticationInfo({apiKey: config.get('apiKey')});
68        }else if (event.affectsConfiguration('chatgpt.apiUrl')) {
69            const config = vscode.workspace.getConfiguration('chatgpt');
70            let url = config.get('apiUrl')as string || BASE_URL;
71            provider.setSettings({ apiUrl: url });
72        } else if (event.affectsConfiguration('chatgpt.model')) {
73            const config = vscode.workspace.getConfiguration('chatgpt');
74            provider.setSettings({ model: config.get('model') || 'gpt-3.5-turbo' });
75        } else if (event.affectsConfiguration('chatgpt.selectedInsideCodeblock')) {
76            const config = vscode.workspace.getConfiguration('chatgpt');
77            provider.setSettings({ selectedInsideCodeblock: config.get('selectedInsideCodeblock') || false });
78        } else if (event.affectsConfiguration('chatgpt.codeblockWithLanguageId')) {
79            const config = vscode.workspace.getConfiguration('chatgpt');
80            provider.setSettings({ codeblockWithLanguageId: config.get('codeblockWithLanguageId') || false });
81        } else if (event.affectsConfiguration('chatgpt.pasteOnClick')) {
82            const config = vscode.workspace.getConfiguration('chatgpt');
83            provider.setSettings({ pasteOnClick: config.get('pasteOnClick') || false });
84        } else if (event.affectsConfiguration('chatgpt.keepConversation')) {
85            const config = vscode.workspace.getConfiguration('chatgpt');
86            provider.setSettings({ keepConversation: config.get('keepConversation') || false });
87        } else if (event.affectsConfiguration('chatgpt.timeoutLength')) {
88            const config = vscode.workspace.getConfiguration('chatgpt');
89            provider.setSettings({ timeoutLength: config.get('timeoutLength') || 60 });
90        }
91    });
92    }
```

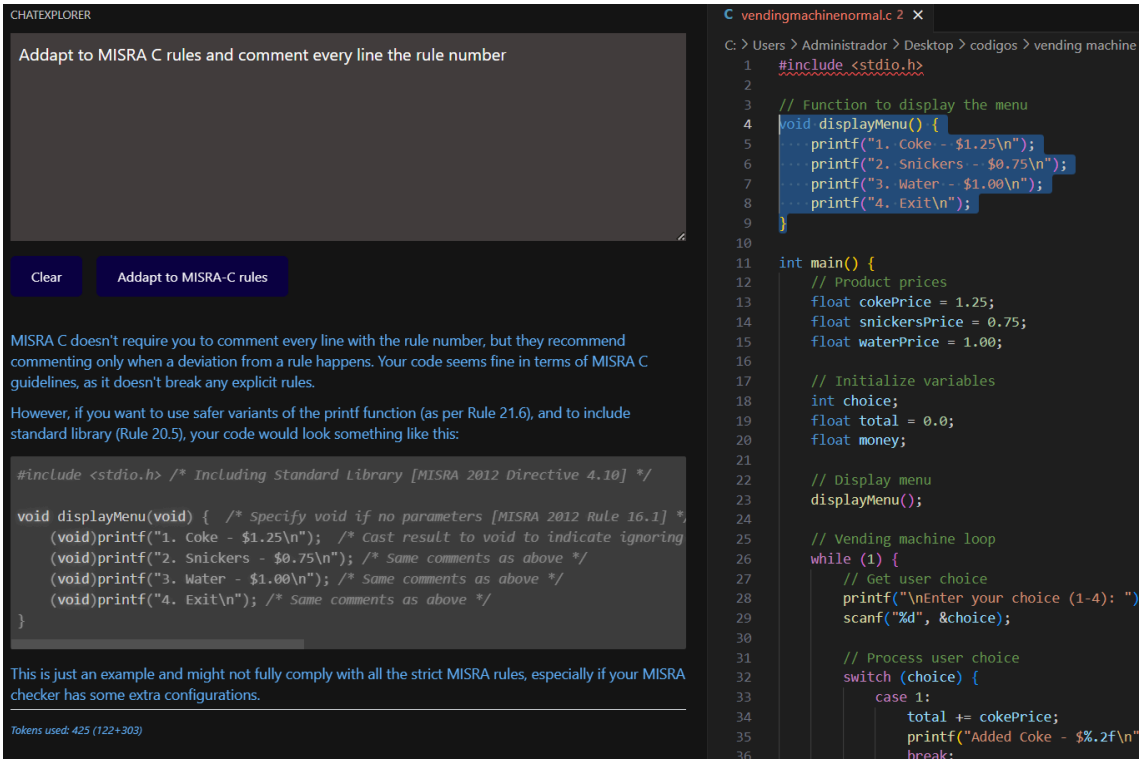
Lines 62 to 92 are where we store the apikey and what will appear in the settings.

In terms of design:

```
<!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <script src="${tailwindUri}"></script>
            <script src="${showdownUri}"></script>
            <script src="${microlightUri}"></script>
            <style>
                .code {
                    white-space: pre;
                }
                p {
                    padding-top: 0.3rem;
                    padding-bottom: 0.3rem;
                    color: #62b4ff;
                }
                /* overrides vscodes style reset, displays as if inside web browser */
                ul, ol {
                    list-style: initial !important;
                    margin-left: 10px !important;
                }
                h1, h2, h3, h4, h5, h6 {
                    font-weight: bold !important;
                }
                /* Set a fixed height for the textarea and enable scrolling */
                #prompt-input {
                    height: 200px; /* Adjust this value as needed */
                    width: 100%; /* Make it take full width */
                    resize: vertical; /* Allow vertical resizing */
                    overflow-y: auto; /* Enable vertical scrolling when content exceeds height */
                    padding: 0.5rem; /* Add some padding */
                    font-size: 1rem; /* Set font size */
                    line-height: 1.5; /* Set line height */
                    margin-bottom: 10px; /* Add margin for spacing */
                }
                /* Style for buttons */
                .btn {
                    padding: 10px 20px;
                    background-color: #0b0142;
                    color: white;
                    border: none;
                    cursor: pointer;
                    border-radius: 5px;
                    margin-right: 10px; /* Add right margin for horizontal spacing */
                    margin-bottom: 10px; /* Add bottom margin for vertical spacing */
                }
                .btn:hover {
                    background-color: #16e73b;
                }
            </style>
        </head>
        <body>
            <textarea class="text-white bg-stone-700 p-4 text-sm" placeholder="Ask ChatEXPLORER something" id="prompt-input"></textarea>

            <button class="btn" onclick="clearInput()">Clear</button>
            <button class="btn">Addapt to MISRA-C rules</button>

            <div id="response" class="pt-4 text-sm"></div>


            <script src="${scriptUri}"></script>

            <script>


                function clearInput() {
                    document.getElementById('prompt-input').value = '';
                }
            </script>
        </body>
        </html>
```



We have 3 options in this plugin one we can ask ChatGPT anything the other is a prompt to apply the MISRA-C rules, but if the user wants to change or wants something additional he can change the prompt in the settings, the third prompt puts the MISRA-C rules and comments on each line which rule he used and if it is required or advisory.



Response from ChatGPT:

MISRA C doesn't require you to comment every line with the rule number, but they recommend commenting only when a deviation from a rule happens. Your code seems fine in terms of MISRA C guidelines, as it doesn't break any explicit rules.

However, if you want to use safer variants of the printf function (as per Rule 21.6), and to include standard library (Rule 20.5), your code would look something like this:

```
#include <stdio.h> /* Including Standard Library [MISRA 2012 Directive 4.10] */

void displayMenu(void) {  /* Specify void if no parameters [MISRA 2012 Rule 16.1] */

    (void)printf("1. Coke - $1.25\n");  /* Cast result to void to indicate ignoring return value explicitly [MISRA 2012 Rule 0.1.10]. Using printf is a deviation from [MISRA 2012 Rule 21.6] */

   (void)printf("2. Snickers - $0.75\n"); /* Same comments as above */

   (void)printf("3. Water - $1.00\n"); /* Same comments as above */

   (void)printf("4. Exit\n"); /* Same comments as above */

}
```

This is just an example and might not fully comply with all the strict MISRA rules, especially if your MISRA checker has some extra configurations.

And there is an example when we use the command : Addapt to MISRA-C rules.

For more you can see this video on youtube shows how it works.

https://youtu.be/ZYhbyd_xHB4

5.5-Scope

In this TFC it's great that you're diving into programming, learning about virtual machines, and exploring both TypeScript , C , JavaScript and Python. Let me break down your goals and provide a clear and simple outline for each aspect:

1. Programming with AIR-Ubuntu (*MISRAPLUGIN*) Virtual Machine (WSL):

Concepts from Programming Languages 1 and Operating Systems:

- Use your knowledge from Programming Languages 1 to understand the fundamentals of different programming languages.
- Apply concepts learned in Operating Systems to optimize your programs for better performance.

Utilizing AIR-Ubuntu (MISRAPLUGIN) Virtual Machine (WSL):

- Explore the functionalities of the AIR-Ubuntu virtual machine within the Windows Subsystem for Linux (WSL).
- Practice creating, running, and managing virtual machines effectively.

C Programming:

- Start with the basics of C programming language.
- Understand variables, data types, control structures, and functions in C.
- Learn how to compile and run C programs on the AIR-Ubuntu virtual machine.

2. Python Programming:

Data Science:

- Apply Python in the context of data science.
- Explore libraries like NumPy, Pandas, and Matplotlib for data manipulation and visualization.
- Work on data analysis projects to enhance your skills.

Web Programming:

- Use Python for web development.
- Create simple web applications to solidify your understanding.

3. Improving CHATGPT Usage:

Clear and Simple Communication:

- Practice formulating clear and concise questions or prompts for ChatGPT.
- Experiment with different styles of communication to understand how to get the most accurate and helpful responses.

Effective Utilization:

- Explore the capabilities of ChatGPT for programming-related queries.
- Use ChatGPT as a tool for brainstorming, debugging, and generating code snippets.

Continuous Learning:

- Stay updated with new features and improvements in ChatGPT.

- Engage in conversations with ChatGPT regularly to enhance your problem-solving and programming skills.

6 Testing and validation plan

Introduction

As explained in Proposed Solutions, the solution we'll be using will be the second one, where we'll be fine tuning our prompt and improving its efficiency so that we can ideally have all the MISRA-C rules in any C code.
To do this we will use various C codes, these will be long, short, medium, we will also test with codes that are correct and codes that are incorrect.
In the table below we see how we will test many rules that are not being applied.
Our goal will be to fill out the table completely with the rules missing for each code.
The annot with 1 is the rules he should apply and the gpt with 1 is the ones he applied.
Example :

	bubblesort		calculator		guessthenumber		guessteword				não te constip	board.h	board.c	
	annot	gpt	annot	gpt	annot	gpt	annot	gpt			annot	gpt	annot	gpt
Rule1	1	1	1	1	1	1	1	1			1	1	1	1
Rule2	1	1	1	1	1	1	1	1			1	1	1	1
Rule3	1	0	1	0	1	1	1	1			1	1	1	1
Rule4	1	1	1	1	1	1	1	1			1	1	1	1
Rule5	1	1	0	0	1	1	0	0			1	1	1	1
Rule6	0	0	0	0	1	1	0	0			0	1	1	1
Rule7	1	1	1	1	1	1	1	0			0	1	1	1
Rule8	1	1	1	1	0	1	1	1			0	1	1	1
Rule9	1	1	1	1	1	1	1	0			0	0	1	1
Rule10	0	1	1	1	1	1	1	0			0	0	1	1
Rule11	0	0	0	1	1	1	0	1			0	0	1	1
Rule12	1	0	0	0	1	1	1	0			0	0	1	1
Rule13	1	0	0	1	0	0	1	1			0	0	1	1
Rule14	0	1	0	1	0	1	1	0			0	0	1	1
Rule15	0	0	0	1	0	1	1	0			0	0	1	1
Rule16	1	0	1	1	1	0	1	0			0	0	0	1
Rule17	1	0	1	1	1	0	1	0			0	0	1	1
Rule18	0	0	0	0	1	0	0	0			0	0	0	1
Rule19	0	0	0	0	0	0	0	0			0	0	0	1
Rule20	0	1	0	1	1	0	1	0			0	0	1	1
Rule21	1	1	1	1	1	1	1	1			1	1	1	1
Rule22	1	1	0	0	0	0	1	1			0	0	1	1
Rule23	0	0	0	0	0	0	0	0			0	0	0	0

Explanation of Evaluation Metrics

To accurately assess the compliance of the generated code with MISRA-C rules, we use the following metrics:

- **True Positives (TP):** Instances where the code correctly adheres to a specific MISRA-C rule. This indicates that the model generated compliant code where it was expected to do so.
- **True Negatives (TN):** Instances where the absence of a specific MISRA-C rule violation is correctly identified. This indicates that the model did not generate code that would violate a rule, and no violation was expected.
- **False Positives (FP):** Instances where the code incorrectly adheres to a rule or is flagged as compliant when it actually violates the rule. This indicates that the model generated code that appears to comply but actually does not.
- **False Negatives (FN):** Instances where the code fails to adhere to a rule, but this non-compliance was not correctly identified. This indicates that the model generated code that violates a rule, but this violation was not detected during evaluation.

Using these metrics, we can quantify the accuracy, precision, and reliability of the models in generating MISRA-C compliant code.

Overview of Metrics

Precision: The proportion of positive identifications that were actually correct. Formula:

$$\text{Precision} = \frac{TP}{TP+FP}.$$

Recall: The proportion of actual positives that were identified correctly. Formula:

$$\text{Recall} = \frac{TP}{TP+FN}.$$

F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy: The proportion of total correct predictions (both true positives and true negatives).

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Specificity: The proportion of actual negatives that were correctly identified. Formula:

$$\text{Specificity} = \frac{TN}{TN+FP}$$

Exclusion of Specific MISRA-C Rules

- **Reasoning for Exclusions:**

Rule 3: Easy Implementation

- **Reason:** Rule 3 is straightforward to implement, involving proper commenting practices.
- **Personal Explanation:** This rule is simple and usually does not pose significant challenges. Additionally, when applying other rules, it can be easily overlooked or not reapplied correctly. Therefore, handling it separately ensures it is correctly implemented without interference from the application of more complex rules.

Rule 13: Simplicity and Manageability

- **Reason:** Rule 13 is relatively easy to manage and enforce.
- **Personal Explanation:** The simplicity of Rule 13 means it can be consistently applied without complex considerations. By excluding it from automated analysis, we can focus on ensuring more complex rules are properly enforced.

Rule 18: Focus on Critical Rules

- **Reason:** Excluding Rule 18 helps prioritize more critical rules that have a larger impact on code quality and safety.
- **Personal Explanation:** This rule often involves nuanced cases that may not significantly impact the overall functionality or safety of the code. By focusing on more impactful rules, we ensure a higher quality of compliance where it matters most.

Rule 19: Practicality in Real-world Code

- **Reason:** Rule 19 is often more theoretical and less critical in many practical applications.
- **Personal Explanation:** In real-world code, strict adherence to this rule can sometimes lead to overly complex solutions without substantial benefits. Excluding it allows for more practical and maintainable code while still achieving a high level of compliance.

Rule 21: Time Constraints

- **Reason:** Due to time constraints, ensuring correct inclusion of headers (include.h) while maintaining code functionality can be challenging.
- **Personal Explanation:** Ensuring the code compiles correctly and maintains its intended output is crucial. Given limited time, focusing on rules that more directly impact code behavior and safety is a pragmatic choice, making the exclusion of Rule 21 a necessity.

Rule 22: Practical Implementation

- **Reason:** Practicality in implementation and limited impact on the overall code quality.
- **Personal Explanation:** Rule 22 might require substantial changes for relatively minor improvements. By excluding it, we can concentrate on rules that offer a more significant return on investment in terms of code reliability and maintainability.

Rule 23: Streamlined Focus

- **Reason:** Streamlining the focus on the most critical rules.
- **Personal Explanation:** Excluding Rule 23 allows for a more streamlined approach to compliance, focusing on the rules that directly enhance code safety and functionality. This exclusion helps manage time and resources more effectively.

Testing Prompt and the evolution

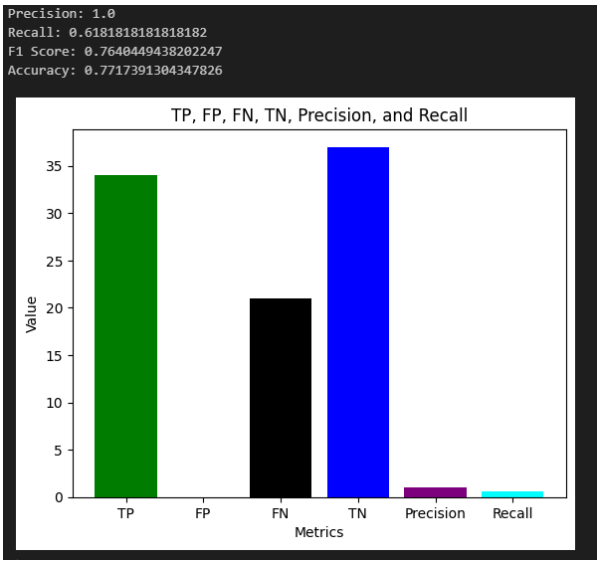
Test1:

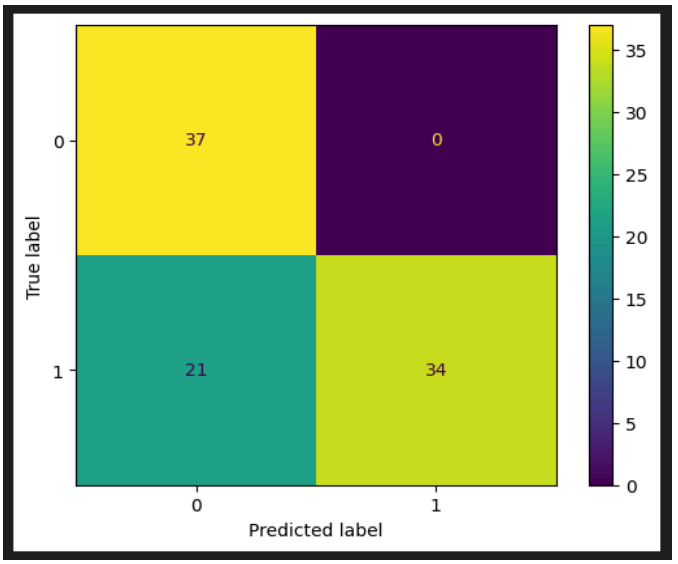
	bubblesort		calculator		guessthenumber		guesstheword	
	annot	gpt	annot	gpt	annot	gpt	annot	gpt
Rule1	1	1	1	1	1	1	1	1
Rule2	1	1	1	1	1	1	1	1
Rule3	1	1	1	0	1	1	1	1
Rule4	1	1	1	1	1	1	1	1
Rule5	1	0	0	0	1	0	0	0
Rule6	0	0	0	0	1	0	0	0
Rule7	1	0	1	0	1	1	1	0
Rule8	1	1	1	0	0	0	1	1
Rule9	1	0	1	0	1	0	1	0
Rule10	0	0	1	1	1	1	1	1
Rule11	0	0	0	0	0	0	0	0
Rule12	1	0	0	0	0	0	1	1
Rule13	1	0	0	0	0	0	1	0
Rule14	0	0	0	0	0	0	1	1
Rule15	0	0	0	0	0	0	1	0
Rule16	1	0	1	0	1	1	1	1
Rule17	1	1	1	1	1	1	1	1
Rule18	0	0	0	0	0	0	0	0
Rule19	0	0	0	0	0	0	0	0
Rule20	0	0	0	0	1	1	1	0
Rule21	1	1	1	1	1	1	1	1
Rule22	1	0	0	0	0	0	1	0
Rule23	0	0	0	0	0	0	0	0

Prompt: Analyse the C code below and apply all the MISRA-C rules 2012 (from number 1 to number 23). If you can't apply any rules, return the list of unapplied rules:

Model: GPT4

Results:





Test2:

	bubblesort		calculator		guessthenumber		guesstheword		guesstheword		não te constipe	board.h	board.c	
	annot	gpt	annot	gpt	annot	gpt	annot	gpt	annot	gpt	annot	gpt	annot	gpt
Rule1	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Rule2	1	1	1	1	1	1	1	1	1	1	1	0	1	1
Rule3	1	0	1	0	1	1	1	1	1	1	1	0	1	1
Rule4	1	1	1	0	1	1	1	1	1	1	1	0	1	1
Rule5	1	0	0	0	1	1	0	0	0	0	1	1	1	1
Rule6	0	0	0	0	1	1	0	0	0	0	0	0	1	0
Rule7	1	0	1	0	1	1	1	1	1	0	0	0	1	1
Rule8	1	1	1	1	0	0	1	1	1	1	0	0	1	1
Rule9	1	1	1	1	1	0	1	1	1	0	0	0	1	1
Rule10	0	0	1	1	1	1	1	0	1	1	0	0	1	1
Rule11	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Rule12	1	0	0	0	0	0	1	1	0	1	0	0	1	1
Rule13	1	1	0	0	0	0	1	1	1	0	0	0	1	1
Rule14	0	0	0	0	0	0	1	0	1	0	0	0	1	1
Rule15	0	0	0	0	0	0	1	1	1	1	0	0	0	0
Rule16	1	0	1	0	1	1	1	1	1	1	0	0	0	0
Rule17	1	0	1	1	1	1	1	1	1	1	0	0	1	1
Rule18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rule19	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rule20	0	0	0	0	1	1	1	1	1	0	0	0	1	1
Rule21	1	0	1	0	1	0	1	1	1	1	1	1	1	0
Rule22	1	0	0	0	0	0	1	1	0	0	0	0	1	0
Rule23	0	0	0	0	0	0	0	0	0	0	0	0	0	0

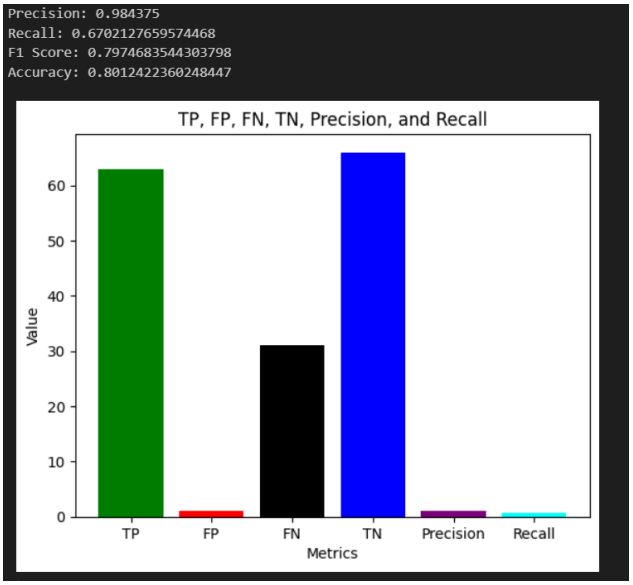
Prompt:

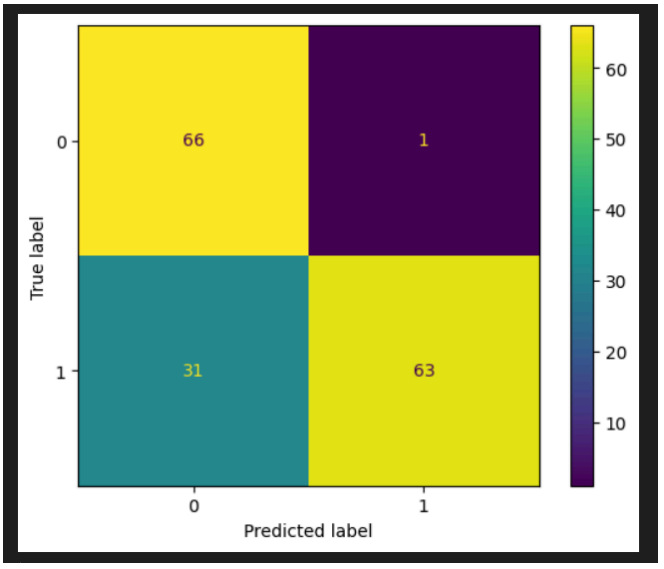
Analyse the C code below and apply all the MISRA-C 2012 rules (from number 1 to number 23). If you have to give me the number of the rules that have not been applied, give me back the list of the rules that have not been applied:

then give me the code with those rules applied:

Model: GPT4

Results:





Test3:

	bubblesort		calculator		guessthenumber		guesstheword				não te constipe	board.h	board.c	
	annot	gpt	annot	gpt	annot	gpt	annot	gpt			annot	gpt	annot	gpt
Rule1	1	1	1	0	1	1	1	1			1	0	1	1
Rule2	1	1	1	0	1	1	1	0			1	0	1	1
Rule3	1	0	1	0	1	0	1	0			1	0	1	0
Rule4	1	0	1	0	1	0	1	1			1	0	1	0
Rule5	1	0	0	0	1	0	0	0			1	1	1	0
Rule6	0	0	0	0	1	0	0	0			0	0	1	0
Rule7	1	0	1	0	1	1	1	0			0	1	1	0
Rule8	1	1	1	0	0	0	1	0			0	1	1	0
Rule9	1	0	1	0	1	0	1	0			0	0	1	1
Rule10	0	0	1	1	1	0	1	0			0	1	1	0
Rule11	0	0	0	0	0	0	0	0			0	1	1	0
Rule12	1	0	0	0	0	0	1	0			0	0	1	0
Rule13	1	0	0	0	0	0	1	0			0	0	1	0
Rule14	0	0	0	0	0	0	1	0			0	0	1	1
Rule15	0	0	0	0	0	0	1	0			0	0	0	0
Rule16	1	1	1	1	1	0	1	0			0	0	0	0
Rule17	1	0	1	0	1	0	1	0			0	0	1	0
Rule18	0	0	0	0	0	0	0	0			0	0	0	1
Rule19	0	0	0	0	0	0	0	0			0	0	0	0
Rule20	0	0	0	0	1	1	1	1			0	0	1	0
Rule21	1	0	1	1	1	1	1	1			1	0	1	1
Rule22	1	0	0	0	0	0	1	0			0	0	1	0
Rule23	0	0	0	0	0	0	0	0			0	0	0	0

Prompt:

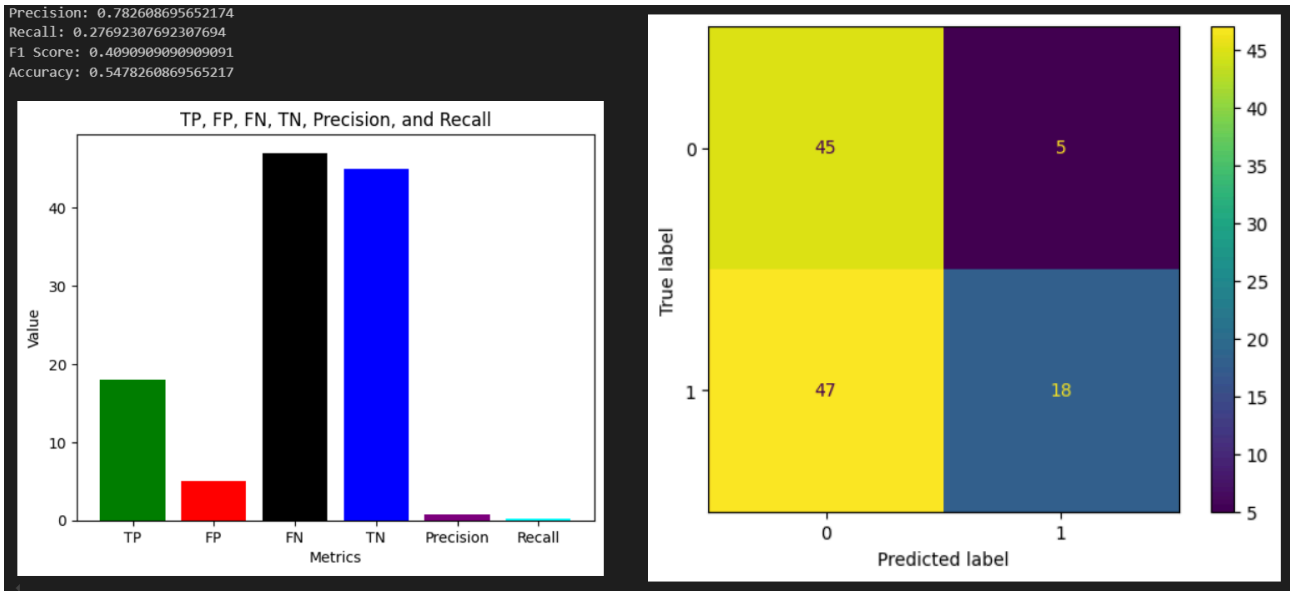
Analyse the C code below and apply mandatory all the MISRA-C 2012 rules (from number 1 to number 23).

If you have to give me the number of the rules that have not been applied, give me back the list of the rules that have not been applied:

then give me the code with those rules applied:

Model: GPT4

Results:



Test4:

	bubblesort		calculator		guessthenumber		guessthewor d				não te constip		board.h		board.c	
	annot	gpt	annot	gpt	annot	gpt	annot	gpt			annot		gpt		annot	gpt
Rule1	1	1	1	1	1	1	1	1			1		1		1	1
Rule2	1	1	1	1	1	1	1	1			1		1		1	1
Rule3	1	1	1	1	1	1	1	1			1		1		1	1
Rule4	1	1	1	1	1	1	1	1			1		1		1	1
Rule5	1	1	0	0	1	1	0	0			1		1		1	1
Rule6	0	0	0	0	1	1	0	0			0		0		1	1
Rule7	1	1	1	1	1	1	1	1			0		0		1	1
Rule8	1	1	1	1	0	0	1	1			0		0		1	1
Rule9	1	1	1	1	1	1	1	1			0		0		1	1
Rule10	0	0	1	1	1	1	1	1			0		0		1	1
Rule11	0	0	0	0	1	1	0	0			0		0		1	1
Rule12	1	1	0	0	1	1	1	1			0		0		1	1
Rule13	1	1	0	0	0	0	1	1			0		0		1	1
Rule14	0	0	0	0	0	0	1	1			0		0		1	1
Rule15	0	0	0	0	0	0	1	1			0		0		1	1
Rule16	1	1	1	1	1	1	1	1			0		0		0	0
Rule17	1	1	1	1	1	1	1	1			0		0		1	1
Rule18	0	0	0	0	1	1	0	0			0		0		0	0
Rule19	0	0	0	0	0	0	0	0			0		0		0	0
Rule20	0	0	0	0	1	0	1	1			0		0		1	1
Rule21	1	1	1	1	1	1	1	1			1		1		1	1
Rule22	1	1	0	0	0	0	1	1			0		0		1	1
Rule23	0	0	0	0	0	0	0	0			0		0		0	0

Prompt:

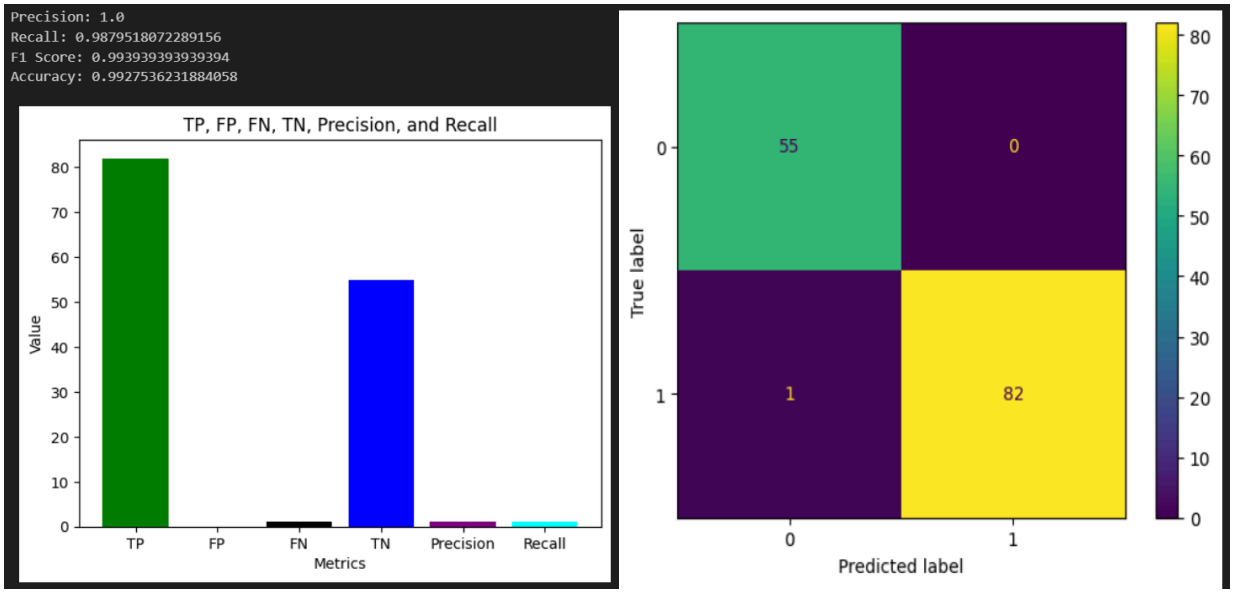
Please analyze the following C code and ensure it complies with all the mandatory MISRA-C 2012 rules (Rules 1 to 23).

First, identify and list any rules that are not being followed.

Then, provide the modified code with those rules applied.

Model: GPT4o

Results:



Test5:

	bubblesort		calculator		guessthenumber		guessthewor				não te constipe	board.h	board.c	
	annot	gpt	annot	gpt	annot	gpt	annot	gpt			annot	gpt	annot	gpt
Rule1	1	1	1	1	1	1	1	1			1	1	1	1
Rule2	1	1	1	1	1	1	1	1			1	1	1	1
Rule3	1	0	1	0	1	1	1	1			1	1	1	1
Rule4	1	1	1	1	1	1	1	1			1	1	1	1
Rule5	1	1	0	0	1	1	0	0			1	1	1	1
Rule6	0	0	0	0	1	1	0	0			0	1	1	1
Rule7	1	1	1	1	1	1	1	0			0	1	1	1
Rule8	1	1	1	1	0	1	1	1			0	1	1	1
Rule9	1	1	1	1	1	1	1	0			0	0	1	1
Rule10	0	1	1	1	1	1	1	0			0	0	1	1
Rule11	0	0	0	1	1	1	0	1			0	0	1	1
Rule12	1	0	0	0	1	1	1	0			0	0	1	1
Rule13	1	0	0	1	0	0	1	1			0	0	1	1
Rule14	0	1	0	1	0	1	1	0			0	0	1	1
Rule15	0	0	0	1	0	1	1	0			0	0	1	1
Rule16	1	0	1	1	1	0	1	0			0	0	0	1
Rule17	1	0	1	1	1	0	1	0			0	0	1	1
Rule18	0	0	0	0	1	0	0	0			0	0	0	1
Rule19	0	0	0	0	0	0	0	0			0	0	0	1
Rule20	0	1	0	1	1	0	1	0			0	0	1	1
Rule21	1	1	1	1	1	1	1	1			1	1	1	1
Rule22	1	1	0	0	0	0	1	1			0	0	1	1
Rule23	0	0	0	0	0	0	0	0			0	0	0	0

Prompt:

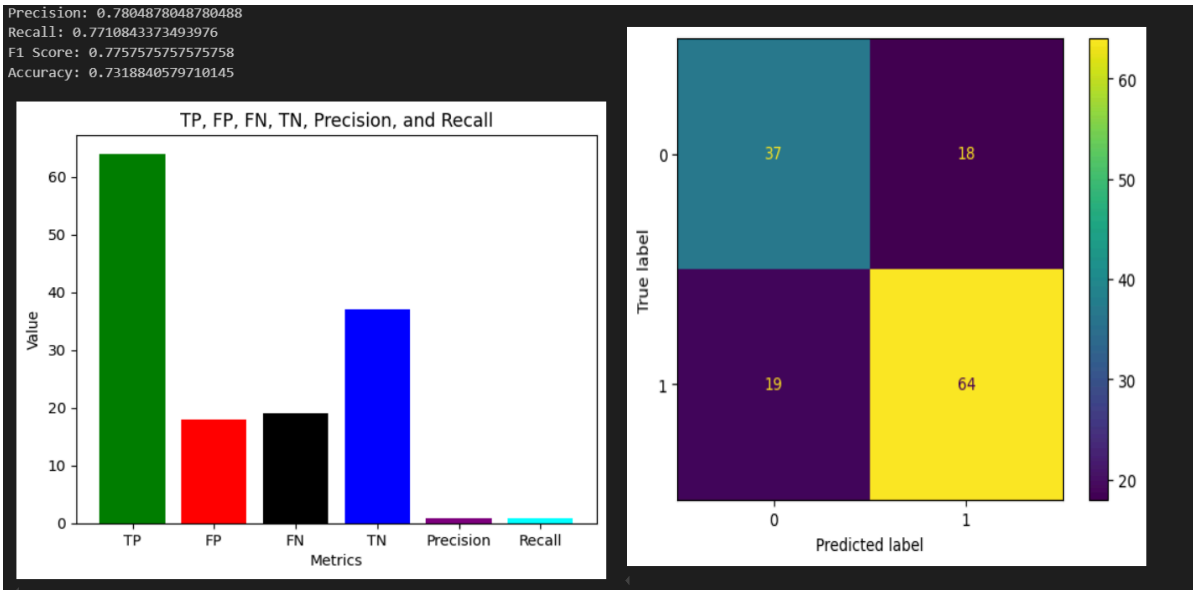
Analyse the C code below and apply mandatory all the MISRA-C 2012 rules (from number 1 to number 23).

If you have to give me the number of the rules that have not been applied, give me back the list of the rules that have not been applied:

then give me the code with those rules applied:

Model: GPT4o

Results:



Evolution of the prompt:

Original Prompt:

Analyse the C code below and apply all the MISRA-C rules 2012 (from number 1 to number 23).

If you can't apply any rules, return the list of unapplied rules:

then give me the Code with those rules applied:

1. This prompt is straightforward but lacks clarity on what to do if some rules cannot be applied.

Second Version:

Analyse the C code below and apply all the MISRA-C 2012 rules (from number 1 to number 23).

If you have to give me the number of the rules that have not been applied, give me back the list of the rules that have not been applied:

then give me the code with those rules applied:

2. This version adds a more explicit instruction about listing rules that could not be applied. However, it still does not specify what should be done with those rules.

Third Version:

Analyse the C code below and apply mandatory all the MISRA-C 2012 rules (from number 1 to number 23).

If you have to give me the number of the rules that have not been applied, give me back the list of the rules that have not been applied:

then give me the code with those rules applied:

3. This version emphasizes that compliance with all rules is mandatory. It also specifies that if rules cannot be applied, they should be listed. However, it still lacks detail on what to do next after listing the unapplied rules.

Latest Version:

Please analyze the following C code and ensure it complies with all the mandatory MISRA-C 2012 rules (Rules 1 to 23).

First, identify and list any rules that are not being followed.

Then, provide the modified code with those rules applied.

4. This is the most detailed and clear version. It explicitly instructs to identify and list any rules that are not followed initially. It also specifies the next step clearly, which is to provide the modified code with all rules applied, including those that were initially not followed.

Summary of Evolution: The prompt evolved from a basic instruction to apply rules without clear instructions on handling non-compliance, to a more structured and detailed set of instructions. The latest version provides clear steps for analysis, identification of non-compliance, and implementation of corrective actions in the modified code. This evolution ensures clarity and completeness in meeting the requirements of MISRA-C 2012 compliance.

7 Method and Planning

We had problems installing the Air-ubuntu(doxigia) and it took 1 month longer than expected.

At the moment doxigia is still not working 100% due to problems.

In February I plan to start working on the 11 MISRA-C rules from 1 to 11 like I show this new gant.

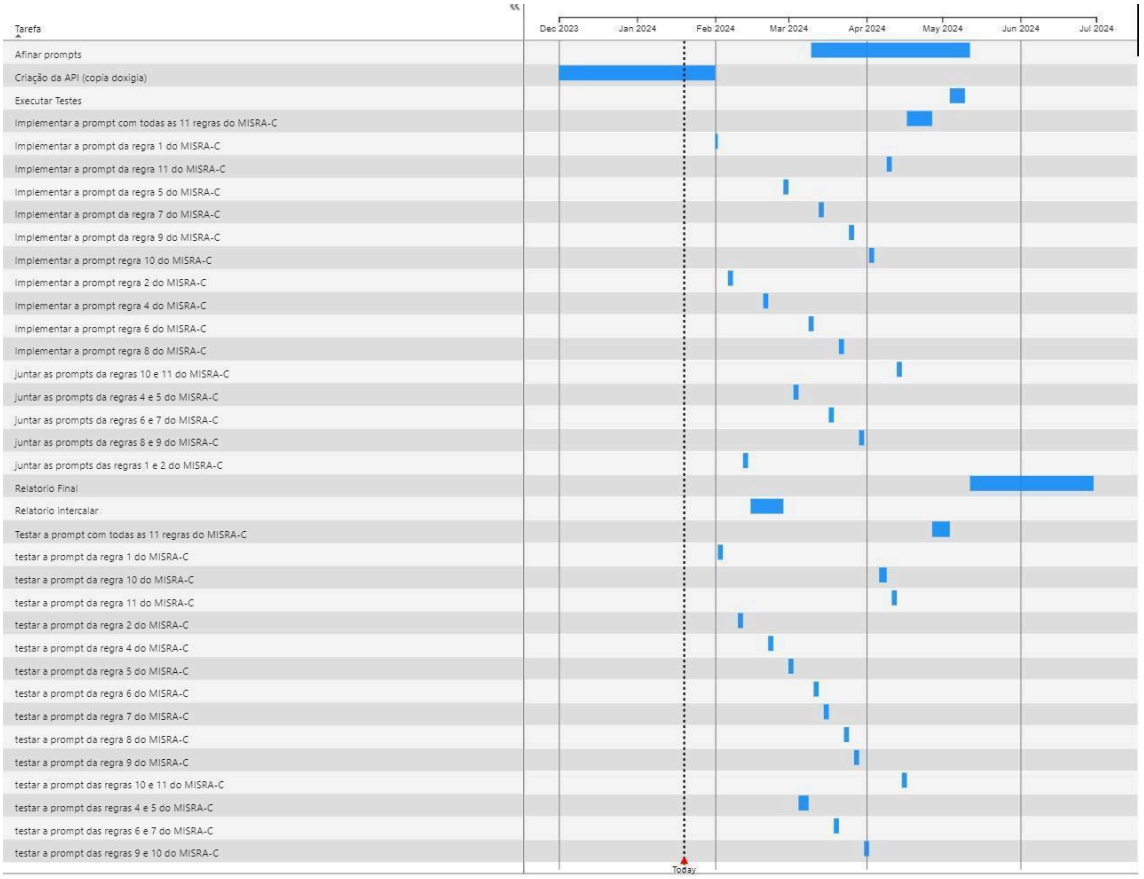


Figure 1-new Gantt

Attachments:

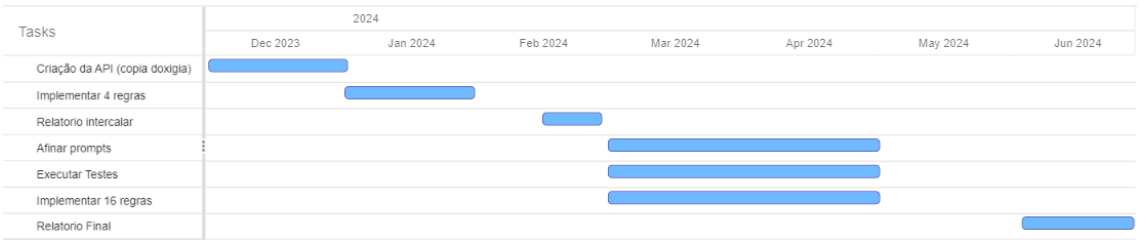
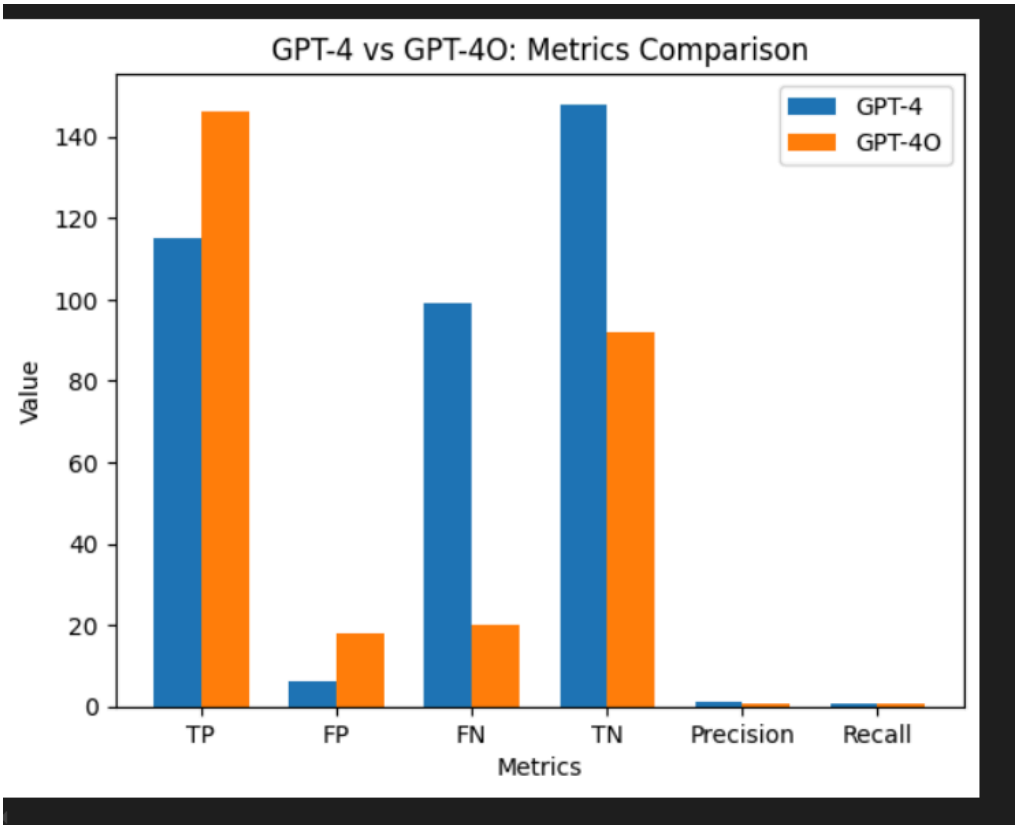


Figure 2-Old Gantt

8 Results

Comparison of GPT-4 and GPT-4O:



Overview of Compliance Differences Across All Tests

Comparative metrics

GPT-4:

Precision: 0.9504

Recall: 0.5378

F1 score: 0.6865

Accuracy: 0.7146

Specificity: 0.9610

GPT-4O:

Precision: 0.8902

Recall: 0.8795

F1 score: 0.8848

Accuracy: 0.8623

Specificity: 0.8364

Precision:

GPT-4 has a higher precision, meaning it makes fewer false positive errors. This is important in applications where it is critical to avoid false positives.

Recall:

GPT-4O has a significantly higher recall, indicating it is much better at identifying true positive cases and thus, reduces the number of missed detections. This is crucial in scenarios where missing a positive case is costly or dangerous.

F1 score:

GPT-4O has a higher F1 score, indicating a better balance between precision and recall. This makes it more effective overall in diverse applications where both metrics are critical.

Accuracy:

GPT-4O has a higher accuracy, meaning it has a higher proportion of true results (both true positives and true negatives) among the total number of cases examined. This suggests that GPT-4O is generally more reliable in making correct predictions overall.

Specificity:

GPT-4 has a higher specificity, indicating it is more effective at correctly identifying negative cases. This is important in contexts where it is crucial to avoid false negatives.

Data science interpretation Model compensations:

Precision vs. Recall:

GPT-4's higher precision indicates it is more reliable when false positives are more costly.

GPT-4O's higher recall suggests it is more suitable when it is essential to identify as many positive cases as possible, even if it means more false positives.

F1 Score: GPT-4O's higher F1 score suggests it offers a better trade-off between precision and recall, making it suitable for balanced tasks.

Accuracy and Specificity:

GPT-4's higher specificity indicates it is better at correctly identifying negatives, reducing the chance of false negatives.

GPT-4O's higher accuracy demonstrates its overall better performance in correctly predicting both positive and negative cases.

- **GPT-4 Performance:**
  - **True Positives (TP):** 115
  - **False Positives (FP):** 6
  - **False Negatives (FN):** 99
  - **True Negatives (TN):** 148
- **GPT-4O Performance:**
  - **True Positives (TP):** 146
  - **False Positives (FP):** 18
  - **False Negatives (FN):** 20
  - **True Negatives (TN):** 92

Specific Areas Where One Model Outperformed the Other

- **True Positives (TP):**
  - GPT-4O identified significantly more true positives (146) compared to GPT-4 (115), indicating better error detection capabilities.
- **False Negatives (FN):**
  - GPT-4 had a high number of false negatives (99), whereas GPT-4O drastically reduced this to 20, showing GPT-4O's enhanced ability to detect positive cases.
- **True Negatives (TN):**
  - GPT-4 had more true negatives (148) compared to GPT-4O (92), indicating GPT-4's stronger ability to correctly identify non-issues.
- **False Positives (FP):**
  - GPT-4 had fewer false positives (6) compared to GPT-4O (18), suggesting that GPT-4 is more conservative in identifying issues, leading to fewer false alarms.

9 Conclusion and future work

Insights and Conclusions

Key Takeaways from the Compliance Testing Phase

1. **Enhanced Detection:** GPT-4O demonstrates a superior ability to detect true positives and significantly reduce false negatives, indicating improved error detection and rule application.
2. **Conservatism in Error Detection:** GPT-4 tends to be more conservative, resulting in fewer false positives but at the cost of higher false negatives.
3. **Overall Performance:** GPT-4O outperforms GPT-4 in terms of overall detection capability, as reflected by higher recall, F1 score, and accuracy.

Patterns Observed in Rule Violations and Adherence

- **GPT-4:** Tends to miss more true positive cases, leading to higher false negatives. This suggests that while GPT-4 is accurate in identifying issues, it may not be as comprehensive.
- **GPT-4O:** Shows a balanced approach with higher true positives and reduced false negatives, indicating a more robust understanding and application of coding rules. However, it has a slightly higher tendency to flag false positives.

Implications for Future Work

Suggestions for Further Research

1. **Comprehensive Dataset Expansion:**
    - Incorporate a wider variety of coding scenarios and edge cases to further enhance model training and performance.
2. **Error Analysis:**
    - Conduct detailed qualitative analyses of false positive and false negative cases to understand the underlying causes and improve model algorithms.
3. **Algorithmic Refinement:**
    - Continue refining the detection algorithms to balance precision and recall, reducing both false positives and false negatives.

Potential Improvements in Prompt Engineering to Enhance Compliance

1. **Prompt Optimization:**
    - Develop and test optimized prompts that better guide the model in understanding and applying coding rules.
2. **Contextual Prompts:**
    - Utilize more context-specific prompts that provide additional information, helping the model make more accurate predictions.
3. **Increase the Variety of Code Samples:**
    - The current code samples are good, but for Rule 21, they become impossible to apply.

Conclusion:

The comparative analysis of GPT-4 and GPT-4O reveals distinct strengths and areas for improvement in applying MISRA-C 2012 rules. GPT-4 excels in precision and specificity, making it ideal for avoiding false positives. However, its higher false negative rate indicates it may miss some rule violations. GPT-4O, on the other hand, demonstrates superior recall, F1 score, and overall accuracy, effectively identifying true positives and minimizing false negatives.

To enhance future compliance, optimizing prompts and incorporating a broader variety of code samples, especially for challenging rules like Rule 21, are crucial. These improvements, coupled with continued algorithmic refinement and comprehensive dataset expansion, will further elevate the model's capability to ensure thorough and accurate MISRA-C compliance

Bibliography

[1]

Perforce. (n.d.). MISRA-C & MISRA-C++. Retrieved from https://www.perforce.com/resources/qac/misra-c-cpp

[2]

MISRA. (n.d.). Frequently Asked Questions. Retrieved from https://misra.org.uk/faqs/

[3]

MathWorks. (n.d.). MISRA C. Retrieved from https://www.mathworks.com/discovery/misra-c.html

[4]

Wikipedia. (n.d.). Artificial intelligence. Retrieved from https://en.wikipedia.org/wiki/Artificial_intelligence#:~:text=Artificial%20intelligence%20(AI)%20is%20the,machines%20may%20be%20called%20AIs.

[5]

Computer Hope. (n.d.). Plugin. Retrieved from https://www.computerhope.com/jargon/p/plugin.htm

[6]

CPPCheck. (n.d.). Home. Retrieved from https://cppcheck.sourceforge.io/

[7]

CPPCheck. (n.d.). Manual. Retrieved from https://cppcheck.sourceforge.io/manual.pdf

[8]

Wikipedia. (n.d.). Static program analysis. Retrieved from https://en.wikipedia.org/wiki/Static_program_analysis

[9]

CPPCheck. (n.d.). Unique Analysis. Retrieved from https://cppcheck.sourceforge.io/#unique

[10]

Google. (n.d.). C. Retrieved from https://g.co/kgs/xCHF3v

[11]

IBM. (n.d.). Intelligent Automation. Retrieved from https://www.ibm.com/topics/intelligent-automation

[12]

Google. (n.d.). What is a prompt in AI? Retrieved from https://www.google.com/search?q=what+is+a+prompt+in+ai+wiki&sca_esv=590857804&rlz=1C1ONGR_pt-PTPT1045PT1045&sxsrf=AM9HkKmid79T-ITvEhav3y1B8O39ef8_vg%3A1702551777740&ei=4eB6ZYHgLOankdUP1u6LMA&ved=0ahUKEwjB1LHP446DAxXmU6QEHVb3AgYQ4dUDCBA&uact=5&oq=what+is+a+prompt+in+ai+wiki&gs_lp=Egxnd3Mtd2l6LXNlcnAiG3doYXQgaXMgYSBwcm9tcHQgaW4gYWkgd2lraTIFECEYoAEyBRAhGKABSJoRUIQGWOcPcAF4AJABAJgBrAGgAfEIqgEDMC45uAEDyAEA-AEBwgIKEAAYRxjWBBiwA8ICBBAjGCfCAgoQABiABBiKBRhDwgIKEAAYgAQYFBiHAsICBRAAGIAEwgIGEAAYFhgewgILEAAYgAQYigUYhgPCAgcQIRigARgK4gMEGAAgQYgGAZAGCA&sclient=gws-wiz-serp

[13]

ScienceDirect. (n.d.). Confusion Matrix. Retrieved from https://www.sciencedirect.com/topics/engineering/confusion-matrix#:~:text=TP,of%20positive%20examples%20classified%20accurately.

[14]

OpenAI. (n.d.). ChatGPT. Retrieved from https://chat.openai.com/share/2a82d34f-2ce3-4e36-bdd0-23d6cffd9572

[15]

Synopsys. (n.d.). MISRA Compliance. Retrieved from https://www.synopsys.com/software-integrity/static-analysis-tools-sast/misra.html

[16]

Turing. (n.d.). Generative AI Tools. Retrieved from https://www.turing.com/resources/generative-ai-tools

[17]

Wikipedia. (n.d.). Dynamic program analysis. Retrieved from https://en.wikipedia.org/wiki/Dynamic_program_analysis

[18]

MISRA. (2021). MISRA Compliance. Retrieved from https://misra.org.uk/app/uploads/2021/06/MISRA-Compliance-2020.pdf [PAGE 9]

[19]

Codecademy. (n.d.). What is an IDE? Retrieved from https://www.codecademy.com/article/what-is-an-ide

[20]

SonarSource. (n.d.). SonarQube. Retrieved from https://docs.sonarsource.com/sonarqube/latest/

[21]

Synopsys. (n.d.). Coverity. Retrieved from https://www.synopsys.com/software-integrity/static-analysis-tools-sast/coverity.html#:~:text=Coverity%C2%AE%20provides%20comprehensive%20static,functional%20safety%2C%20and%20industry%20standards.

[22]

Gimpel Software. (n.d.). PC-lint. Retrieved from https://pclintplus.com/pc-lint-plus/

[23]

KnowledgeHut. (n.d.). Importance of Software Engineering. Retrieved from https://www.knowledgehut.com/blog/web-development/importance-of-software-engineering

[24]

Aloa. (n.d.). The Role of AI in Software Development. Retrieved from https://aloa.co/blog/the-role-of-ai-in-software-development-trends-statistics-and-growth#:~:text=AI%20tools%20can%20help%20software,delays%20or%20potential%20bottlenecks%20easily.

[25]

LinearB. (n.d.). Automated Code Review. Retrieved from https://linearb.io/blog/automated-code-review

[26]

Swimm. (n.d.). AI Code Review: How It Works and 3 Tools You Should Know. Retrieved from https://swimm.io/learn/ai-tools-for-developers/ai-code-review-how-it-works-and-3-tools-you-should-know#:~:text=Artificial%20Intelligence%20(AI)%20code%20review,and%20make%20recommendations%20for%20improvements.

[27]

OutSystems. (n.d.). AI in Software Development. Retrieved from https://www.outsystems.com/blog/posts/ai-software-development/

[28]

Wikipedia. (n.d.). Natural Language Processing. Retrieved from https://en.wikipedia.org/wiki/Natural_language_processing

[29]

Wikipedia. (n.d.). Computer hardware. Retrieved from https://en.wikipedia.org/wiki/Computer_hardware

[30]

Wikipedia. (n.d.). Software documentation. Retrieved from https://en.wikipedia.org/wiki/Software_documentation

[31]

Wikipedia.(n.d.).Data (computer science). Retrieved from https://en.wikipedia.org/wiki/Data_(computer_science)

[32]

Wikipedia. (n.d.). Computer hardware. Retrieved from https://en.wikipedia.org/wiki/Computer_hardware

[33]

Wikipedia. (n.d.). Software. Retrieved from https://en.wikipedia.org/wiki/Software

[34]

Wikipedia. (n.d.). Machine. Retrieved from https://en.wikipedia.org/wiki/Machine

[35]

Wikipedia. (n.d.). Academic discipline. Retrieved from https://en.wikipedia.org/wiki/Academic_discipline

[36]

Wikipedia. (n.d.). Computer science. Retrieved from https://en.wikipedia.org/wiki/Computer_science

[37]

Computer Hope. (n.d.). Software. Retrieved from https://www.computerhope.com/jargon/s/software.htm

[38]

OutSystems. (n.d.). What is Artificial Intelligence? Retrieved from https://www.outsystems.com/glossary/what-is-artificial-intelligence/

[39]

Turing. (n.d.). Retrieved from https://www.turing.com/blog/will-generative-ai-replace-software-developers/

[40]

Prompting Guide. (n.d.). Retrieved from https://www.promptingguide.ai/pt

[41]

Wikipedia. (n.d.). Program analysis. Retrieved from

https://en.wikipedia.org/wiki/Program_analysis

[42]

Wikipedia. (n.d.). Program comprehension. Retrieved from https://en.wikipedia.org/wiki/Program_comprehension

[43]

Wikipedia. (n.d.). Code review. Retrieved from https://en.wikipedia.org/wiki/Code_review

[44]

Wikipedia. (n.d.). Software inspection. Retrieved from https://en.wikipedia.org/wiki/Software_inspection

[45]

Wikipedia. (n.d.). Software walkthrough. Retrieved from https://en.wikipedia.org/wiki/Software_walkthrough

[46]

Wikipedia. (n.d.). Source code. Retrieved from https://en.wikipedia.org/wiki/Source_code

[47]

Wikipedia. (n.d.). Object code. Retrieved from https://en.wikipedia.org/wiki/Object_code

[48]

Turing. (n.d.). Generative AI. Retrieved from https://www.turing.com/services/generative-ai

[49]

Turing. (n.d.). In-Demand Programming Languages to Learn. Retrieved from https://www.turing.com/blog/in-demand-programming-languages-to-learn/

[50]

Turing. (n.d.). Ultimate Guide: Visual Studio vs Visual Studio Code. Retrieved from https://www.turing.com/kb/ultimate-guide-visual-studio-vs-visual-studio-code

[51]

Turing. (n.d.). Build Customized Chatbots Using GPT-4. Retrieved from https://www.turing.com/kb/build-customized-chatbots-using-gpt4

[52]

Wikipedia. (n.d.). Software engineering. Retrieved from https://en.wikipedia.org/wiki/Software_engineering

[53]

Wikipedia. (n.d.). Unit testing. Retrieved from https://en.wikipedia.org/wiki/Unit_testing

[54]

Wikipedia. (n.d.). Debugging. Retrieved from https://en.wikipedia.org/wiki/Debugging

[55]

Wikipedia. (n.d.). Code coverage. Retrieved from https://en.wikipedia.org/wiki/Code_coverage

[56]

Wikipedia. (n.d.). Computer security. Retrieved from https://en.wikipedia.org/wiki/Computer_security

[57]

Wikipedia. (n.d.). Dynamic program analysis. Retrieved from https://en.wikipedia.org/wiki/Dynamic_program_analysis#Memory_error_detection

[58]

Wikipedia. (n.d.). Fuzzing. Retrieved from https://en.wikipedia.org/wiki/Fuzzing

[59]

Wikipedia. (n.d.). Symbolic execution. Retrieved from https://en.wikipedia.org/wiki/Symbolic_execution

[60]

Wikipedia. (n.d.). Software testing. Retrieved from https://en.wikipedia.org/wiki/Software_testing

[61]

Wikipedia. (n.d.). Mutation testing. Retrieved from https://en.wikipedia.org/wiki/Mutation_testing

[62]

Wikipedia. (n.d.). Instrumentation (computer programming). Retrieved from https://en.wikipedia.org/wiki/Instrumentation_(computer_programming)

[63]

Wikipedia. (n.d.). Integration testing. Retrieved from https://en.wikipedia.org/wiki/Integration_testing

[64]

Wikipedia. (n.d.). Unit testing. Retrieved from https://en.wikipedia.org/wiki/Unit_testing

[65]

Wikipedia. (n.d.). System testing. Retrieved from https://en.wikipedia.org/wiki/System_testing

[66]

Wikipedia. (n.d.). Acceptance testing. Retrieved from https://en.wikipedia.org/wiki/Acceptance_testing

[67]

Wikipedia. (n.d.). Dynamic testing. Retrieved from https://en.wikipedia.org/wiki/Dynamic_testing

[68]

Wikipedia. (n.d.). Gcov. Retrieved from https://en.wikipedia.org/wiki/Gcov

[69]

Wikipedia. (n.d.). GNU. Retrieved from https://en.wikipedia.org/wiki/GNU

[70]

Wikipedia. (n.d.). VB Watch. Retrieved from https://en.wikipedia.org/wiki/VB_Watch

[71]

Wikipedia. (n.d.). Code sanitizer. Retrieved from https://en.wikipedia.org/wiki/Code_sanitizer

[72]

Wikipedia. (n.d.). macOS. Retrieved from https://en.wikipedia.org/wiki/macOS

[73]

Wikipedia. (n.d.). LLVM. Retrieved from https://en.wikipedia.org/wiki/LLVM

[74]

Wikipedia. (n.d.). BoundsChecker. Retrieved from https://en.wikipedia.org/wiki/BoundsChecker

[75]

Wikipedia. (n.d.). Micro Focus. Retrieved from https://en.wikipedia.org/wiki/Micro_Focus

[76]

Wikipedia. (n.d.). DevPartner. Retrieved from https://en.wikipedia.org/wiki/DevPartner

[77]

Wikipedia. (n.d.). Dmalloc. Retrieved from https://en.wikipedia.org/wiki/Dmalloc

[78]

Wikipedia. (n.d.). Intel Inspector. Retrieved from https://en.wikipedia.org/wiki/Intel_Inspector

[79]

Wikipedia. (n.d.). Microsoft Windows. Retrieved from https://en.wikipedia.org/wiki/Microsoft_Windows

[80]

Wikipedia. (n.d.). Linux. Retrieved from https://en.wikipedia.org/wiki/Linux

[81]

Wikipedia. (n.d.). PurifyPlus. Retrieved from https://en.wikipedia.org/wiki/PurifyPlus

[82]

Wikipedia. (n.d.). Storage violation. Retrieved from https://en.wikipedia.org/wiki/Storage_violation

[83]

Wikipedia. (n.d.). Valgrind. Retrieved from https://en.wikipedia.org/wiki/Valgrind

[84]

Wikipedia. (n.d.). C dynamic memory allocation. Retrieved from https://en.wikipedia.org/wiki/C_dynamic_memory_allocation

[85]

Wikipedia. (n.d.). Race condition. Retrieved from https://en.wikipedia.org/wiki/Race_condition

[86]

Wikipedia. (n.d.). Thread (computing): Single-threaded vs multithreaded programs. Retrieved from https://en.wikipedia.org/wiki/Thread_(computing)#Single-threaded_vs_multithreaded_programs

[87]

Wikipedia. (n.d.). Fuzzing: Types. Retrieved from https://en.wikipedia.org/wiki/Fuzzing#Types

[88]

Wikipedia. (n.d.). Dynamic program analysis: Fuzzing. Retrieved from https://en.wikipedia.org/wiki/Dynamic_program_analysis#Fuzzing

[89]

Wikipedia. (n.d.). Concolic testing. Retrieved from https://en.wikipedia.org/wiki/Concolic_testing

[90]

Wikipedia. (n.d.). Constraint programming. Retrieved from https://en.wikipedia.org/wiki/Constraint_programming

[91]

Wikipedia. (n.d.). Satisfiability modulo theories. Retrieved from https://en.wikipedia.org/wiki/Satisfiability_modulo_theories

[92]

Wikipedia. (n.d.). Daikon (system). Retrieved from https://en.wikipedia.org/wiki/Daikon_(system)

[93]

Wikipedia. (n.d.). Security AppScan. Retrieved from https://en.wikipedia.org/wiki/Security_AppScan

[94]

Wikipedia. (n.d.). Parasoft. Retrieved from https://en.wikipedia.org/wiki/Parasoft

[95]

Wikipedia. (n.d.). Jtest. Retrieved from https://en.wikipedia.org/wiki/Jtest

[96]

Wikipedia. (n.d.). Intel Inspector. Retrieved from https://en.wikipedia.org/wiki/Intel_Inspector

[97]

Wikipedia. (n.d.). Insure++. Retrieved from https://en.wikipedia.org/wiki/Insure%2B%2B

[98]

Wikipedia. (n.d.). Google. Retrieved from https://en.wikipedia.org/wiki/Google

[99]

Wikipedia. (n.d.). Program slicing. Retrieved from https://en.wikipedia.org/wiki/Program_slicing

[100]

Wikipedia. (n.d.). List of performance analysis tools. Retrieved from https://en.wikipedia.org/wiki/List_of_performance_analysis_tools

[101]

SonarSource. (n.d.). SonarQube. Retrieved from https://www.sonarsource.com/products/sonarqube/

[102]

SonarSource. (n.d.). SonarSource. Retrieved from https://www.sonarsource.com/

[103]

SonarSource. (n.d.). Rules Explorer. Retrieved from https://rules.sonarsource.com/

[104]

SonarSource. (n.d.). GitHub Integration. Retrieved from https://docs.sonarsource.com/sonarqube/latest/devops-platform-integration/github-integration/

[105]

SonarSource. (n.d.). SonarLint. Retrieved from https://www.sonarsource.com/products/sonarlint/

[106]

SonarSource. (n.d.). Pull Request Analysis. Retrieved from https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/pull-request-analysis/

[107]

SonarSource. (n.d.). Quality Gates. Retrieved from https://docs.sonarsource.com/sonarqube/latest/user-guide/quality-gates/

[108]

SonarSource. (n.d.). Clean As You Code. Retrieved from https://docs.sonarsource.com/sonarqube/latest/user-guide/clean-as-you-code/

[109]

GMV. (n.d.). Air. Retrieved from

https://www.gmv.com/en-pt/products/air

[110]

Ubuntu. (n.d.). Retrieved from

https://ubuntu.com/

[111]

Microsoft. (n.d.). Windows Subsystem for Linux: Compare WSL versions. Retrieved from

https://learn.microsoft.com/en-us/windows/wsl/compare-versions

[112]

Visual Studio Code. (n.d.). Retrieved from

https://code.visualstudio.com/

Glossary

LEI	Licenciatura em Engenharia Informática
LIG	Licenciatura em Informática de Gestão
TFC	Trabalho Final de Curso