Software Design Document

Milestone 2

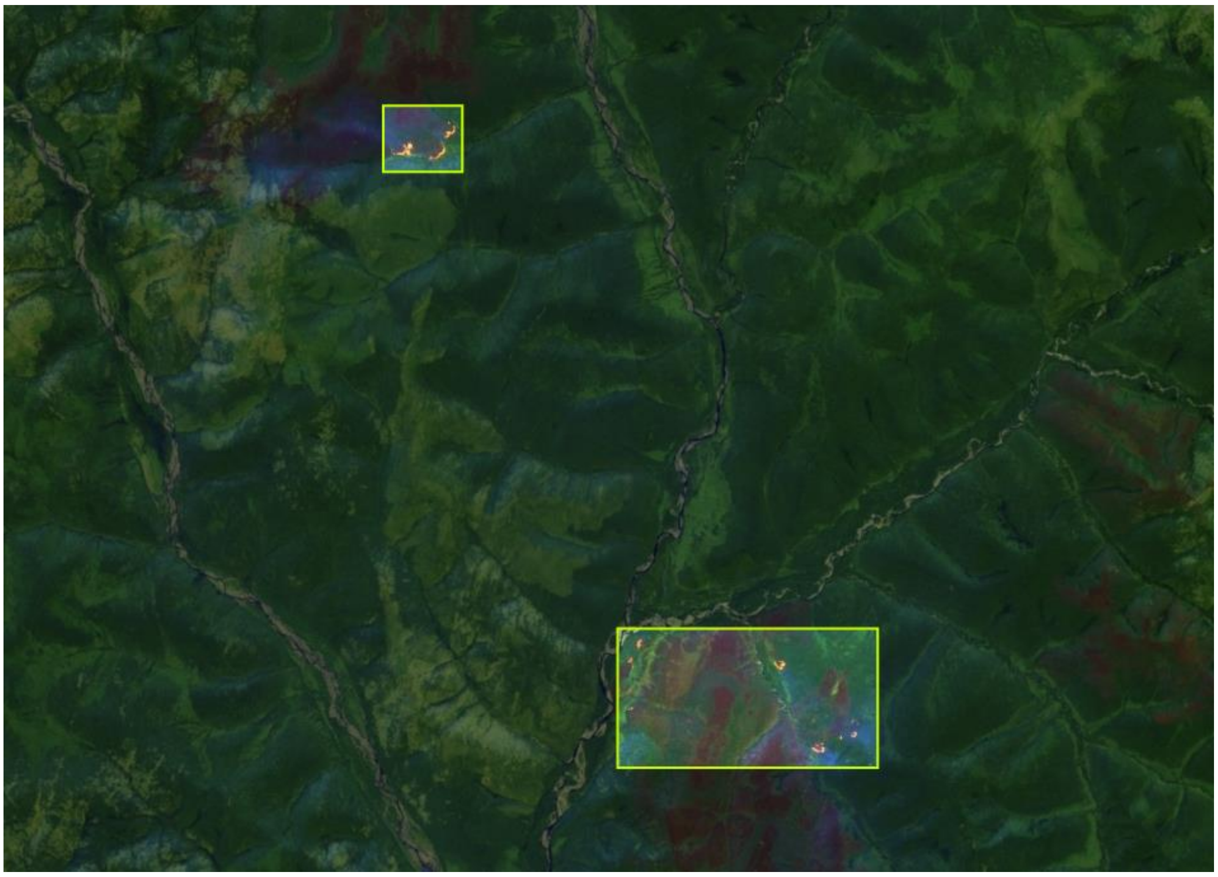


Figure 1 Marked satellite image

Group A

Moritz Lechner

Leon Sobotta

Konstantin Roßmann

Content

1. Introduction and Goals

Wildfires are a major threat to natural resources, human lives, and property. Early detection and rapid response can greatly reduce the impact of wildfires, but traditional methods for detecting fires, such as ground patrols and calls from the public, have limitations. In this project, we aim to develop a software program that utilizes satellite imagery to detect wildfires in near real-time. In a real-life scenario this program could be integrated with existing wildfire management systems.

Our goals are to:

- Develop a program in Rust that can process satellite imagery and detect the presence of wildfire with high accuracy and precision.
- Implement machine learning algorithms to improve the accuracy of wildfire detection.
- Implement the model into a software program that can process satellite images in near real-time.
- Integrate the software with existing wildfire management systems to improve response times and decision making.
- Continuously improve the model's accuracy through feedback and additional training data.
- Provide easy access to the software and its output for relevant authorities and organizations.

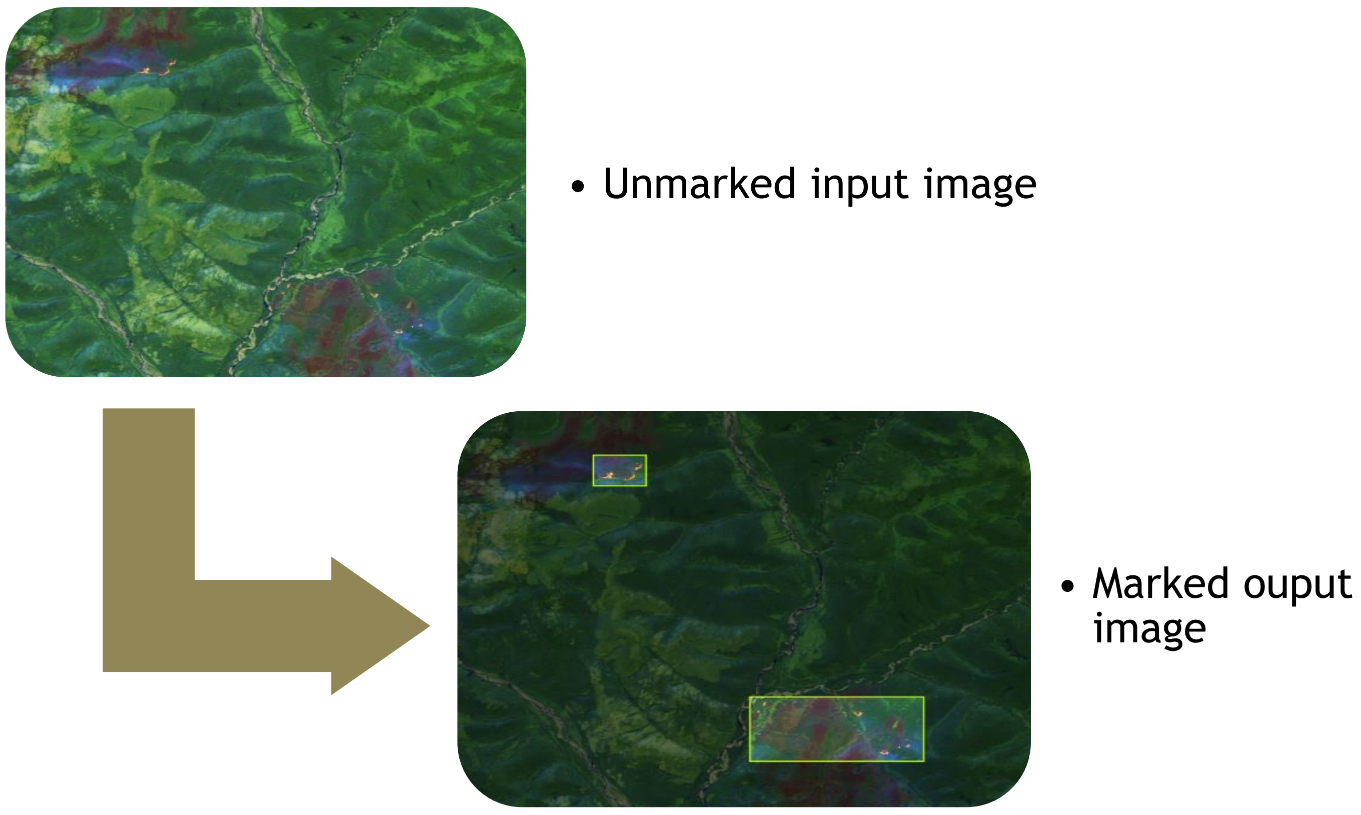


Figure 2. Input and output images of the software ([2], 2023)

Graphic 1. Input and output images of the program

2. Constraints

Constraints are an important part of any software development project as they help to define the scope and boundaries of the project and can have a significant impact on the final outcome. In this project, we will be focusing on several key constraints that have been identified as critical to the success of the project. These constraints have been established based on a thorough analysis of the project's requirements, as well as feedback from stakeholders.

2.1 Technical Constraints

Table 1. List of Technical Constraints

	Constraint	**Background and / or motivation**
Software and programming constraints		
TC1	Implementation in Rust	• The application shall run on the ScOSA Architecture • Requirement set by the stakeholders • Gain programming experience in Rust • Reduce risk of safety critical errors
TC2	As little third-party software and / or crates as possible	• Requirement set by the stakeholders
Operating System Constraints		
TC3	UNIX / UNIX-like OS	• The application should be compilable on UNIX or UNIX-like operating systems e.g., *Raspberry Pi OS*
Hardware Constraints		
TC4	Memory for the image is 1 MB	• Memory is very limited on the satellite • Requirement set by the stakeholder
TC5	Low-end hardware configuration	• Satellites run on robust and well tested hardware, therefore cannot use newest high-end components • The application shall run and be tested on a Raspberry Pi

2.2 Organizational Constraints

Table 2. List of Organizational Constraints

	Constraint	**Background and / or motivation**
OC1	Team	• Moritz Lechner • Leon Sobotta • Konstantin Roßmann
OC2	Time schedule	• Start at the end of 2022 with Python based prototypes running by the first two weeks of 2023. • 07.01.2023, the day access to the ScOSA API is given. The development in Rust can start after that day. • Application is Rust shall be done and running by the end of January of 2023
OC3	IDE independent project setup	• The project must be compilable on the command line via standard build

		tools. • However primary IDEs used are by JetBrains with a student licence: • PyCharm for the development of the prototype and creation of the tflite-model. • IntelliJ for the development in Rust
OC4	Configuration and version control / management	• Private git repository with a complete commit history and a public master branch pushed to GitLab.

2.3 Conventions

Table 3. List of Conventions

	Conventions	**Background and / or motivation**
C1	Architecture documentation	• Structure based on the english arc42-Template
C2	Coding conventions	• The project uses Doxygen-style comments
C3	Language	• German • English

3. Context and Scope

This chapter describes the environment and context of this project: Who uses the system and on which other system does this project depend.

3.1 Business Context

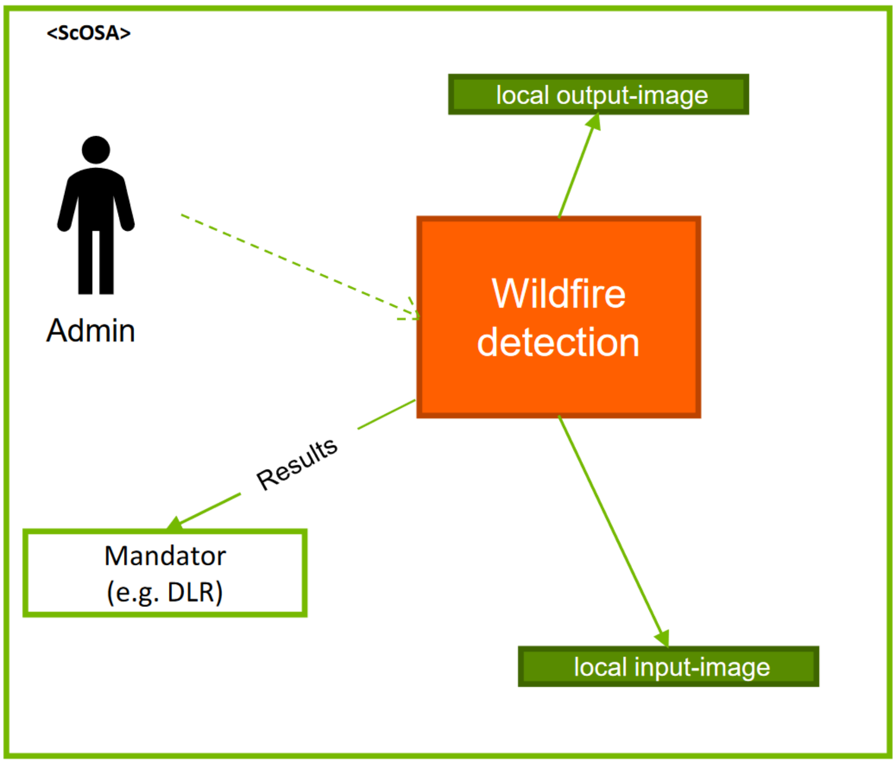


Figure 3. Business context visualization

Admin:

The engineer sitting at a ground station of the satellite executing the program while setting the satellite up to fly over areas where a wildfire is confirmed and areas where there should not be any.

Mandator:

The organization that has ordered the program to be run on the satellite. They receive the results so that in a real-life scenario they could communicate these to the corresponding fire station to trigger a fast response and put out the fire as fast as possible.

Local input-image:

For the program to work, it needs an image to check for a possible wildfire. This image is taken on the satellite by a 1 MP RGB camera (1000x1000x3). This image is saved in the memory of the satellite and then read by the program.

Local output-image:

The program will in any case output an image. If a fire had been detected, the output image will have the marked in the image itself. If no fire could be found the output image is the same image as the input image. The output image will be saved at the same address as the input image and therefore overwrite it. The memory size for the image amounts to 1 megabyte.

3.2 Technical Context

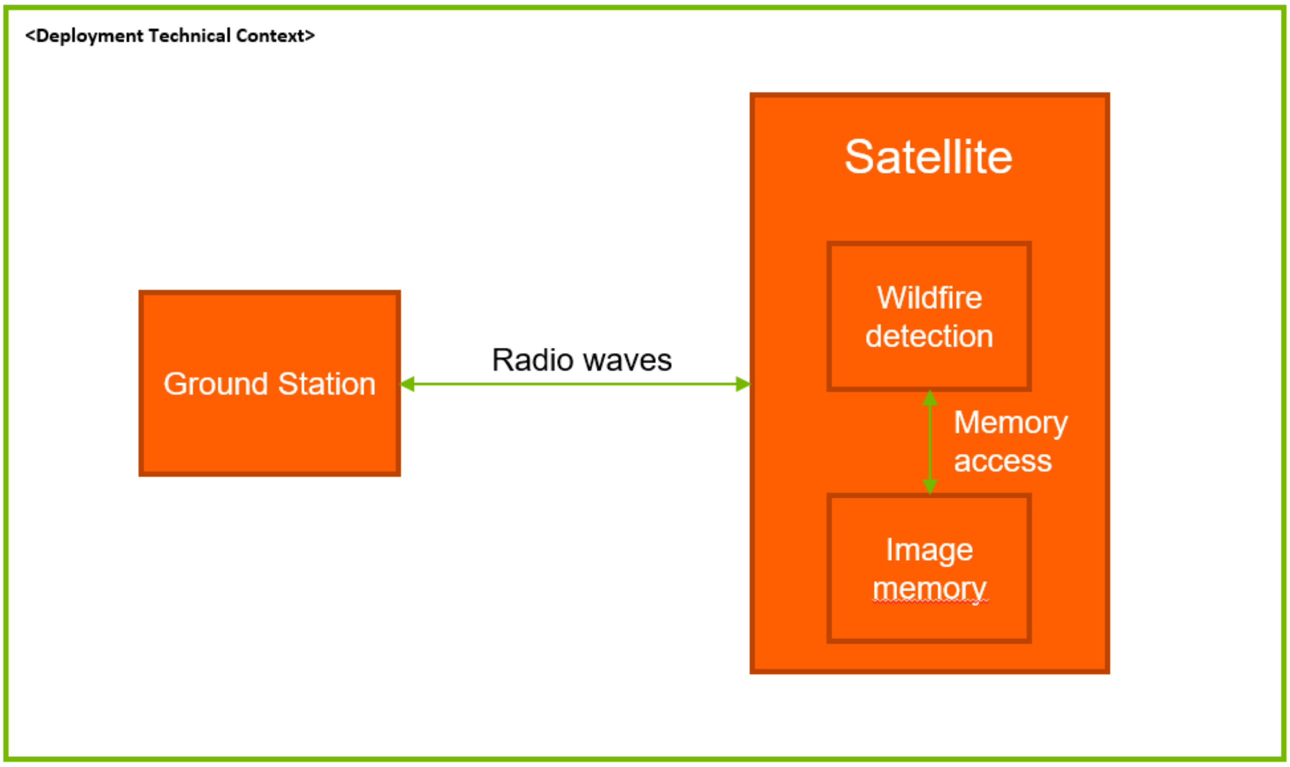


Figure 4. Technical context visualization

Ground Station:

A facility that is used for communicating with and controlling satellites. It typically includes equipment such as antennas, radio transceivers, and computers that are used to send and receive signals to and from the satellite. The ground station can also be used to run programs on the satellite. Additionally, a ground station can be used to track the satellite's location and status, as well as to perform maintenance and repairs on the satellite as needed. In this case the ground station would initiate the execution of the wildfire detection software.

Satellite:

A satellite is an object that is sent into orbit around the Earth or other planets. It can be used for a variety of purposes, such as communication, navigation, and remote sensing. In the case of wildfire detection, the satellite is equipped with a camera that is used to capture images of the Earth's surface. These images are then processed using the software for wildfire detection. This software accesses the memory on the satellite where the images are stored and analyses them for signs of wildfire activity. If a wildfire is detected, the satellite can then transmit this information to the ground station where it can be further analysed and used to respond to the wildfire.

4. Solution Strategy

The first step in the development process would be to build a Python prototype of the software. This would include processing and analysing satellite images with libraries like OpenCV and NumPy. The prototype should be capable of taking an image as input and determine whether it contains a wildfire or not. Creating a prototype allows for early concept testing and validation, as well as identifying any issues or challenges that may arise during development.

However even before the prototype is being developed, we still must manually generate a dataset for the artificial neural network. This collection is made up of several satellite photos, some of which would contain wildfires and others would not. This step's purpose is to develop a dataset that is indicative of the real-world scenarios that the software will face.

After creating the dataset, the next step is to use it to train a TensorFlow model. This is possible using Python's TensorFlow framework. It is critical to use a well-established and effective model architecture, such as a convolutional neural network, that has been shown to perform well on similar tasks. After training, the model is being exported as a TensorFlow Lite (tflite) model for usage in the final application. The pretrained model will then be tested with images provided by the satellite’s camera using the TensorFlow Lite API in Rust.

Following this phase, the program shall return its result as a float value. This will assist in determining whether a fire is detected, and which portion of the picture is affected by the wildfire.

If a fire is found, the software's final output will be the processed image with the fire indicated. This will assist the user in determining the location of the fire on the image and making appropriate judgments. It is critical to have a solid testing and validation procedure in place to improve the performance and reliability of the program. This should include a wide range of tests, such as unit tests, integration tests, and end-to-end tests. It should also be tested in various circumstances, under various lighting conditions, and with various sorts of wildfires.

Finally, a well-defined maintenance and future enhancement procedure is essential, since the software will require regular upgrades to increase its performance and react to new wildfire circumstances. This should contain a procedure for collecting feedback, reporting bugs, and analysing the software's performance in real-world circumstances. This will ensure that the software is always up to date and provides accurate and dependable wildfire detection. Furthermore, it is critical to monitor the software's performance and adjust as needed, such as fine-tuning the model or upgrading the dataset.

In summary, a coherent strategy for a software project that detects wildfires using satellite images should include creating a prototype in Python, manually creating a dataset for the artificial neural network, training a TensorFlow model with the dataset, and exporting the tflite model, using the TensorFlow Lite API in Rust to test the pretrained model with processed image, returning results and processed image with the fire marked, a robust testing and validation process, and a robust testing and validation process.

5. Building Block View

5.1 Whitebox Wildfire detection

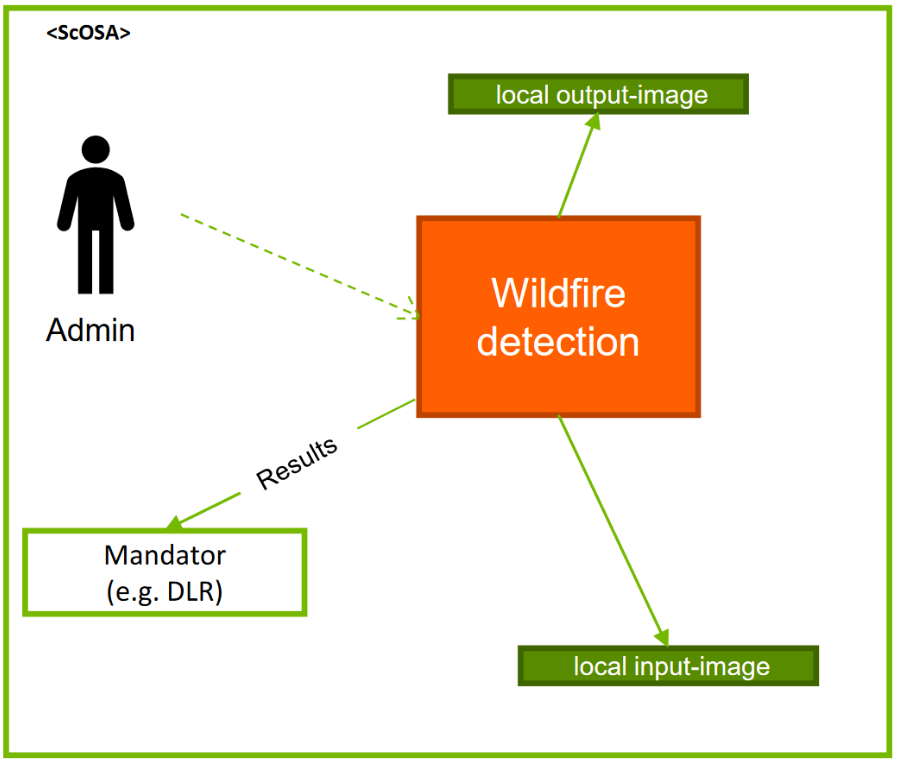


Figure 5. Context and scope

Graphic 4. Context and Scope

Contained Blackboxes:

Building block	Description
Wildfire detection	Program that is run on the satellite and detects wildfires using the images the onboard camera of the satellite provides

5.2 Building Blocks – Level 2

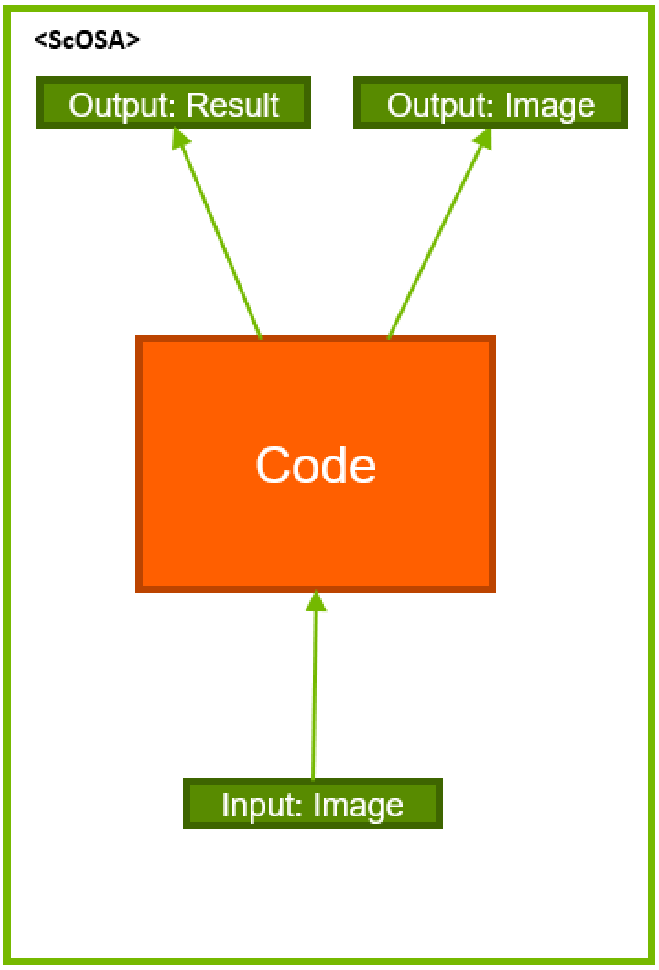


Figure 6. Building block view of level 2 visualized

Contained Blackboxes:

Building block	Description
Code	Takes an input image and uses a pretrained TensorFlow Lite Model to detect if there is a wildfire present in the image. If a wildfire is detected, it returns the results and an output image where the wildfire has been marked with a bounding box. If no wildfire is detected, it also returns the results and an output image, however the output image will be the same

	as the input image since no fire has been found.

5.3 Building Blocks – Level 3

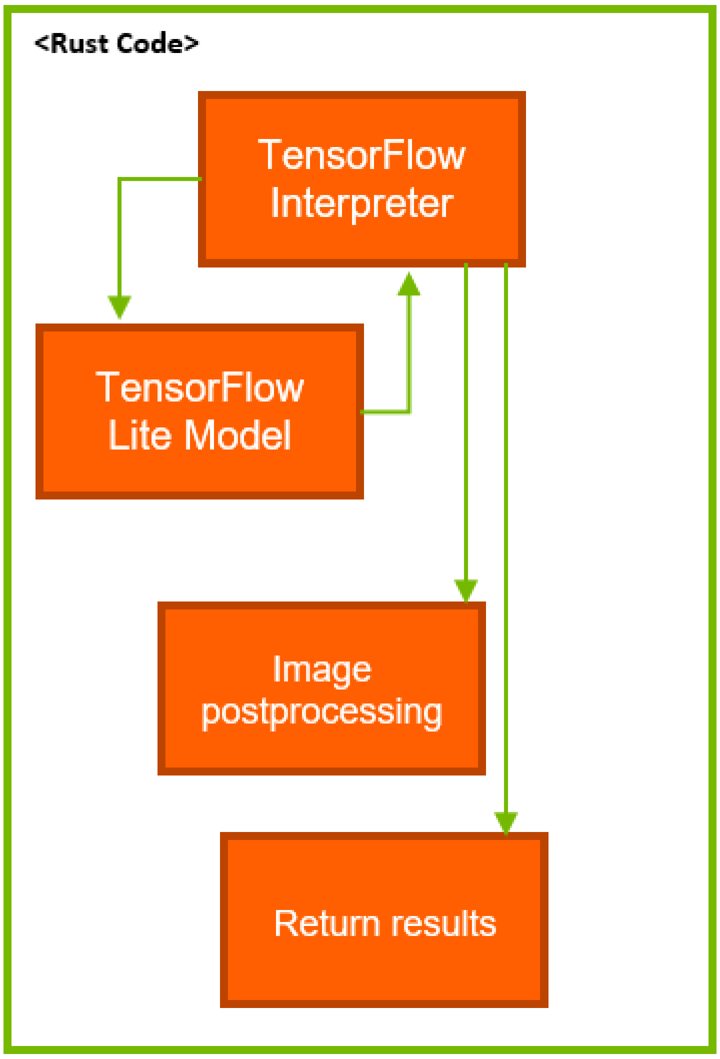


Figure 7. Building block view of level 3 visualized

Contained Blackboxes:

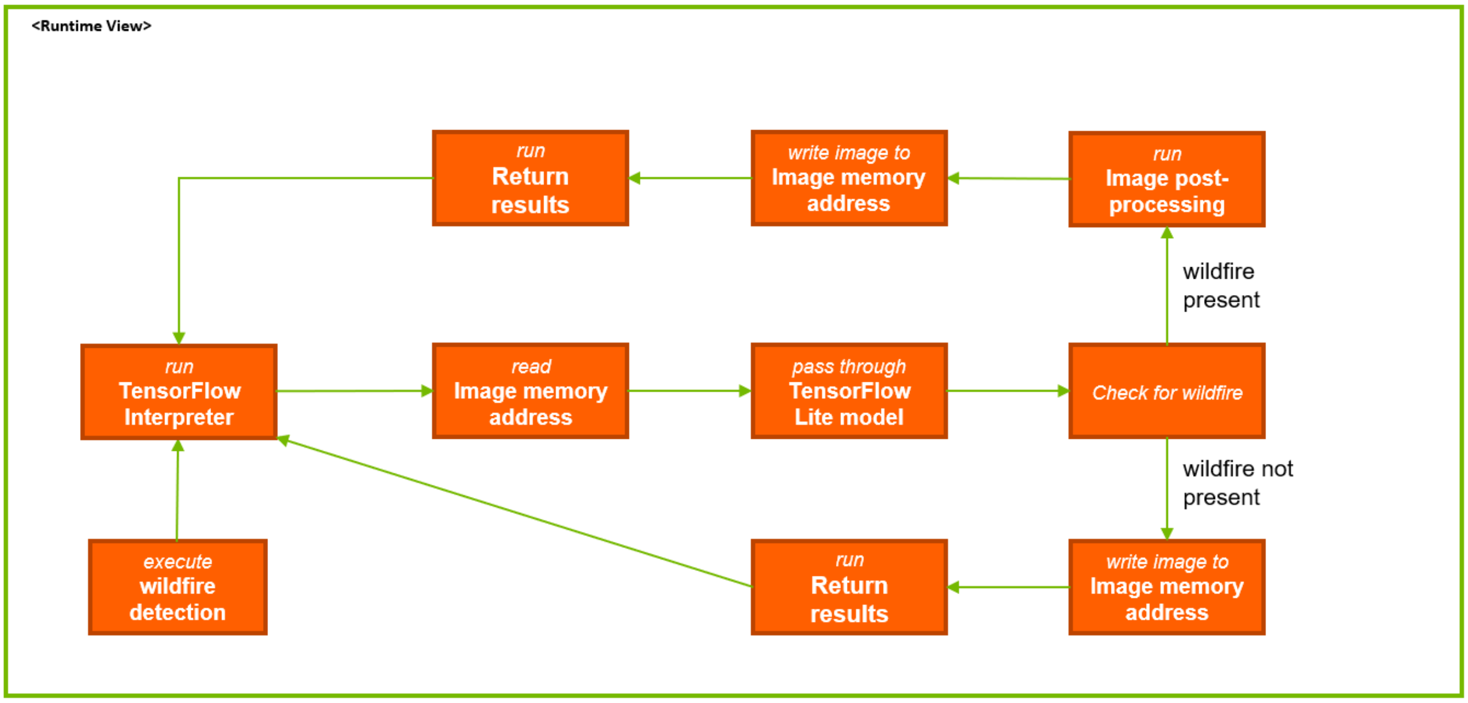
Building block	**Description**
TensorFlow Interpreter	The TensorFlow Interpreter is a tool that allows us to run TensorFlow models in the Rust programming language. In the context of object detection, in this case wildfires, it can be used to take an image as input, pass it through TensorFlow Lite model, and return the locations and labels of any objects detected in the image. The TensorFlow Interpreter in Rust allows for efficient and optimized execution of the object detection model, making it useful for real-time or resource-constrained applications.

TensorFlow Lite Model	TensorFlow Lite is a lightweight version of TensorFlow that is specifically designed for mobile and embedded devices. TensorFlow Lite models are smaller and more efficient than regular TensorFlow models, making them suitable for use on devices with limited resources such as smartphones, IoT devices, and in this case satellites. They can be used for a variety of tasks such as image classification, object detection, and text-to-speech conversion. Our TensorFlow Lite model will be trained to execute object detection.
Image post-processing	Once the fire has been detected, the image post-processing algorithm will mark these regions with a bounding to make them more visible and easier to analyse. This can help emergency responders and other stakeholders to quickly and accurately identify the location of wildfires in satellite images.
Return results	In addition to the output image the program will also return the results of its object detection. This will be done as a float containing the confidence of the neural network, whether it has detected a wildfire or not.

Additional information:

At the current state of the development, it is not possible to go another level deeper as these details have not been worked out yet. However, upon the resolution of these details, they will be added into the document for a more in-depth examination.

6. Runtime View


<Runtime View>
run Return results
write image to Image memory address
run Image post-processing
wildfire present
run TensorFlow Interpreter
read Image memory address
pass through TensorFlow Lite model
Check for wildfire
wildfire not present
execute wildfire detection
run Return results
write image to Image memory address

Graphic 7. Runtime view visualized

Figure 8. Runtime view visualized

1. The wildfire detection software is being executed

2. The TensorFlow Interpreter starts to run and calls the next step
3. The memory address of the image is being read
4. The image is being passed through the TensorFlow Lite model
5. If the model detects a wildfire:
    a. The image post-processing starts, where the fire is being marked on the image
    b. This image is then written into the same image memory address where the input image has been stored
    c. The results of the wildfire detection are then being returned

6. If the model detects no wildfire:
    a. The current image is being written into the same image memory address where the input image has been stored
    b. The results of the wildfire detection are being returned
7. The TensorFlow Interpreter is starting again

7. Architectural Decisions

7.1 Use pretrained model and TensorFlow Lite API

The program uses a pretrained TensorFlow Lite model to check the images for wildfires. In order to use the TensorFlow Lite model we had to use the TensorFlow Lite API, luckily there already existed a Rust crate that accelerated the development of our program. We had decided to use a pretrained model to save memory and CPU power. If we hadn't, we would have to train the model every time we wanted to test an image. That would not only use up a lot of CPU power but also need a lot of memory to store the dataset.

7.2 Autonomous program

The program is supposed to run autonomously. That means the *Admin* only has to start the program once and the program runs continuously and checks images. We decided to make the program autonomous to save communication between the satellite and the ground station. Since the satellite is constantly circling the earth, it doesn't always have a connection to a ground station, which means the time to communicate between each other is very limited. We try to maximize the efficiency of that time window and therefore reduce communication to a bare minimum.

8. Quality Requirements

8.1 75% of wildfires shall be detected

In the current version we try to detect at least 75% of wildfires where one has been confirmed. This number is completely dependent on the size and quality of the dataset.

With increasingly better quality and quantity of the dataset, the model will be detecting more wildfires with a higher confidence.

8.2 Confidence threshold

To determine if a fire on the image is present or not the model needs a threshold. Although we return the confidence of the model as a float, we still need a threshold, so that if the confidence is above the threshold the model can say that is has detected a fire or vice versa.

8.3 Always return an output

The program is supposed to always return an output. This means that even if no fire has been detected the program should still produce an output. In this case the output would be the confidence of no present fire and an output image, which would be the same as the input image.

8.4 Image size limit

Given the limited amount of memory the system has, the output image that the program produces shall not be bigger that 1 MB, which is the same as the input image. Even after postprocessing the image and marking the fire, the size limit cannot be exceeded.

9. Glossary

Word	**Description**
Artificial neural net ([3], 2022)	An artificial neural net is a computational model inspired by the structure and function of biological neural networks that process and transmit information through layers of interconnected "neurons" and can be trained to perform tasks such as classification, prediction and pattern recognition using a set of labelled examples.
Wildfire ([4], 2023)	A wildfire is an uncontrolled and rapidly spreading fire that occurs in wildland areas. Wildfires can be caused by natural events or human activities and can have devastating effects on communities and ecosystems, destroying homes, property, wildlife, vegetation, and releasing pollutants into the air.
TensorFlow ([5], 2022)	TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as

	neural networks. TensorFlow was developed by the Google Brain team and is now maintained by the TensorFlow team at Google.
ScOSA ([6], 2022)	The Scalable on-board Computing for Space Avionics (ScOSA) project by the DLR aims to create a distributed, versatile, and dependable on-board computer that surpasses the capabilities of traditional on-board computers.
Stakeholder ([7], 2023)	A stakeholder in a software project is an individual or group who has an interest or concern in the project and its outcome. They can be internal or external to the organization and can be affected by the project's success or failure.
Rust ([8], 2022)	Rust is a systems programming language that runs extremely fast, prevents segmentation faults, and guarantees thread safety. It aims to provide a concurrent, safe, and practical language for systems programming, with a strong focus on safety and performance, and it has a growing and dedicated community of developers.
Python ([9], 2023)	Python is a high-level, interpreted, programming language that is widely used for web development, data analysis, artificial intelligence and scientific computing. Its simplicity and versatility make it a popular choice among developers, researchers, and data scientists.
NumPy ([10], 2022)	NumPy is a powerful open-source library for the Python programming language. It provides support for large multi-dimensional arrays and matrices of numerical data, allowing for efficient mathematical operations such as linear algebra and Fourier transform, and it is widely used in scientific computing and data science.
OpenCV ([11], 2023)	OpenCV is a powerful open-source library for computer vision and image processing. It provides a wide range of functionality for tasks such as object detection, image recognition, video analysis, and more, enabling developers to build applications that can understand and interpret visual data.
IDE ([12], 2023)	An IDE (Integrated Development Environment) is a software application that provides comprehensive facilities to computer programmers for software development. It typically includes a source code editor, a compiler or interpreter and a debugger that are bound together by a

	graphical user interface (GUI) and designed to ease the process of writing and testing software.
UNIX ([13], 2023)	UNIX is an operating system that was first developed in the 1960s and is known for its stability, scalability and versatility. It's a multi-user and multi-tasking system, widely used in servers, workstations, and mobile devices and its open-source nature has led to the development of many variants such as Linux, macOS and BSD.
Raspberry Pi ([14], 2023)	A Raspberry Pi is a small, low-cost, single-board computer developed by the Raspberry Pi Foundation in the UK. It is designed to promote the teaching of basic computer science in schools and developing countries, and it can be used for a variety of projects such as media centres, game consoles, and home automation systems.
API ([15], 2023)	An API (Application Programming Interface) is a set of protocols and routines that allow different software programs to interact with one another. It acts as a bridge between different software systems, allowing them to communicate and share data, and it enables developers to access the functionality of a platform, operating system, library or service in a consistent and predictable way.

Table of figures

Sources

[1]. (2022, December 2). https://docs.arc42.org/home/

[2]. (2023, January 3). Retrieved from https://universe.roboflow.com/bairock/wildfire-satellite

[3]. (2022, December 6). Retrieved from https://www.ibm.com/topics/neural-networks

[4]. (2023, January 6). Retrieved from https://education.nationalgeographic.org/resource/wildfires

[5]. (2022, December 12). Retrieved from https://www.tensorflow.org/

[6]. (2022, December 13). Retrieved from https://www.dlr.de/sc/desktopdefault.aspx/tabid-11139/19481_read-45210/

[7]. (2023, January 12). Retrieved from https://www.investopedia.com/terms/s/stakeholder.asp

[8]. (2022, November 11). Retrieved from https://www.rust-lang.org/

[9]. (2023, January 4). Retrieved from https://www.python.org/

[10]. (2022, December 27). Retrieved from https://numpy.org/

[11]. (2023, January 10). Retrieved from https://opencv.org/

[12]. (2023, January 10). Retrieved from https://www.redhat.com/en/topics/middleware/what-is-ide

[13]. (2023, January 10). Retrieved from https://www.hpc.iastate.edu/guides/unix-introduction

[14]. (2023, January 10). Retrieved from https://opensource.com/resources/raspberry-pi

[15]. (2023, January 10). Retrieved from https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces