1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects Problem, context, and related work Throughout the last couple of years, the research in the field of artificial intelligence (AI) evolved rapidly. That happens thanks to machine learning (ML) and deep learning (DL), which are sub-categories of AI, and concrete applications of AI. In ML, the target is, instead of programming with if-conditions, loops, and instructions as it is done in traditional programming approaches, to let the system learn how to behave. For this purpose, a tremendous amount of data is necessary for training similar to how we, as humans, are learning. Before a newborn is able to recognize a car and distinguish all different kinds of brands and forms, it needs to see hundreds of vehicles repetitively, getting explained by its parents that this is a car. That is why engineers try to apply the way the human brain is working to the software. This approach is referred to as deep learning, where the main idea is to create neural networks with the target to identify specific criteria of a given input in order to recognize it accordingly. The considerable benefit of this approach is its flexibility. Let us consider an example of distinguishing cars according to their brand. The creation of a system with a traditional software engineering approach that is able to identify different brands with its various models, shapes, and colors in any weather and lighting condition is almost impossible. But with AI, this is a solvable task. By feeding the neural network with immense data of all kinds of vehicles in any situation, it is possible to create a system that able to recognize these. Hence, tasks that could not be solved in the past might be solvable with the help of AI. And this is applicable for every primary industry – the consumer market, healthcare, automotive, transportation, retail, logistics, education, agriculture, and more. Some companies are already applying AI to their products and business philosophy to create added values that are enabling them to come up with new business models and new customer target groups. Although there is a significant level of mistrust in AI, particularly from the perspective of losing jobs, existing experiences prove the development of technology does not necessarily lead to higher unemployment. "The countries with the highest robot density – which is Germany, Japan, South Korea, and Singapore - all of them have more than 300 robots per 10,000 workers and have among the lowest unemployment rates. So what we need to learn is, technology and humans combined in the right way will drive prosperity and growth, will drive employment, will drive affordability of products, and will drive demand. And that means, altogether, that if we embrace technology in a responsible way, we can really get there." - Ulrich Spiesshofer, President, and CEO, ABB Ltd. Nowadays, the majority of companies still do not use AI for their products and services. Although these companies consider AI and are excited about its possibilities (both realistic and unrealistic), they are still failing to adopt AI because of a lack of understanding about the technology itself or due to missing experience with its implementation. Furthermore, the existing architecting approaches so far have ignored or not appropriately dealt with the important question of the adequacy of AI for the existing software systems and for those systems that are to be developed. The entry point of any technology for companies is a system architecture of their software systems. The purpose of the thesis is to compensate for the shortcomings of the state of the art. We aim to address companies and decision-makers with less experience in the field of AI and present an approach that supports them, on an architectural level, in designing an AI powered system while taking the most relevant AI-specific properties under consideration. AI is a disrupting technology with the potential to solve problems that traditional software engineering cannot manage. It is not anymore a simple framework or a plugin. It requires a complete reorganization of the system and introduces too many changes (development, process, system, and software and hardware components). Therefore, it cannot only be adopted by development teams. Still, its effects (benefits and drawbacks, perspectives, and shortcomings) must be discussed on a higher level – hence system

and software architecture. Actual integration and deployment of AI for small- and medium-sized embedded systems companies is still a widely unestablished process. The main reasons for this are: 1. The lack of the understanding of the AI technology itself, which leads to the lack of understanding about 2. potential application areas, and 3. the uncertainty if a migration towards an AI-based solution will be of benefit. The main problem in applying AI in the industry is a lack of a systematic approach for mapping the concerns of stakeholders to potential AI-enabled solutions, which, from the architectural point of view, also discusses consequences in terms of benefits and drawbacks. The domain of AI is very complex and exists in several variations and implementations. It addresses various vertical markets such as medical, consumer, automotive, and the industrial market. Also, its realization can be cloud, fog, or edge-based. Because of the lag of experience of industrial companies that mainly develop embedded hard- and software and due to my personal, professional background, the scope of the thesis will mainly cover AI on the edge and its hardware/software architecture. The problem for companies that are mainly operating in the industrial and embedded domain is a missing basis for decision-making when it comes to the implementation of an AI-approach to their products. For them, the question "Is AI suitable for the existing or new system?" is just barely, if not impossible, to answer confidently. That is where the thesis will provide its contribution. The objective is to support the decision-making process for embedded system companies when adopting AI. This challenge includes getting a profound impression and an increased awareness about technical and business-relevant aspects that need to be under consideration during an AI-implementation. Therefore, engineering techniques are required to get an understanding of the possible impact of implementing AI. Although the SWOT analysis seems to be one of the most common and widely used methods for strategic planning, it has its disadvantages when it comes to addressing particular domain related questions. Its advantage leys in its flexible applicability in different domains and its straightforward usability. But in our case, where we want to evaluate a specific technology, it does not provide any kind of support structure that allows us to incorporate an extension to overcome domain-specific deficiencies. This disadvantage is reflected in the fact that SWOT only describes a vague process of procuring information that is, most of the time, based on subjective impressions (Ashish B. Sasankar and Dr Vinay Chavan. Swot analysis of software development process models.) (Hanieh Kashfi. Adopting cloud computing for enterprise architecture: Swot analysis. International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE), 6(5), 2017.) In contrast, the Solution Adequacy Check is a research area that can be applied to many different use cases. Fraunhofer RATE, and especially the adequacy check, already provides a solid base when it comes to architecture evaluation. It describes a detailed process of how to acquire data necessary to get a rating for the underlying architecture and thereby enhance the awareness of possible pros, cons, and shortcomings of the architecture. The downside here is that it is not designed for specific domains and technologies that require a more detailed or domain related check as the implantation of AI would do. Therefore it is the aim of the thesis to enhance the adequacy check in such a way that it addresses AI-domain-related circumstances, and that allows evaluating AI-based architectures. Requirements for the solution Requirement Artifacts • A01: Support Decision Makers who want to adopt AI • A02: Getting an overview of the state of the art ML implementation in edge computing • A03: Identify AI-specific architectural properties • A04: Identify engineering techniques to evaluate the existing system concerning the applicability of AI • A05: Provide a design decision process based on adequacy check or SWOT analysis that maps requirements to potential solutions • A06: Enable a discussion about benefits, constraints, and risks of adopting AI towards technical and business-relevant concerns Requirements • R01: Support companies with existing traditional embedded systems to evaluate the adoption of AI into their current portfolio • R02: Getting an overview of the state of the art ML implementation in edge computing • R03: Identify

AI-specific architectural properties that influence business, system, and hardware architectures. • R04: Identify engineering techniques to evaluate the existing system concerning the applicability of AI • R05: Provide a design decision process based on the Solution Adequacy Check or SWOT analysis that maps requirements to potential solutions • R06: Enable a discussion (analysis and evaluation) about technical and business relevant concerns, such as added-values, benefits, constraints, and risks of adopting AI.

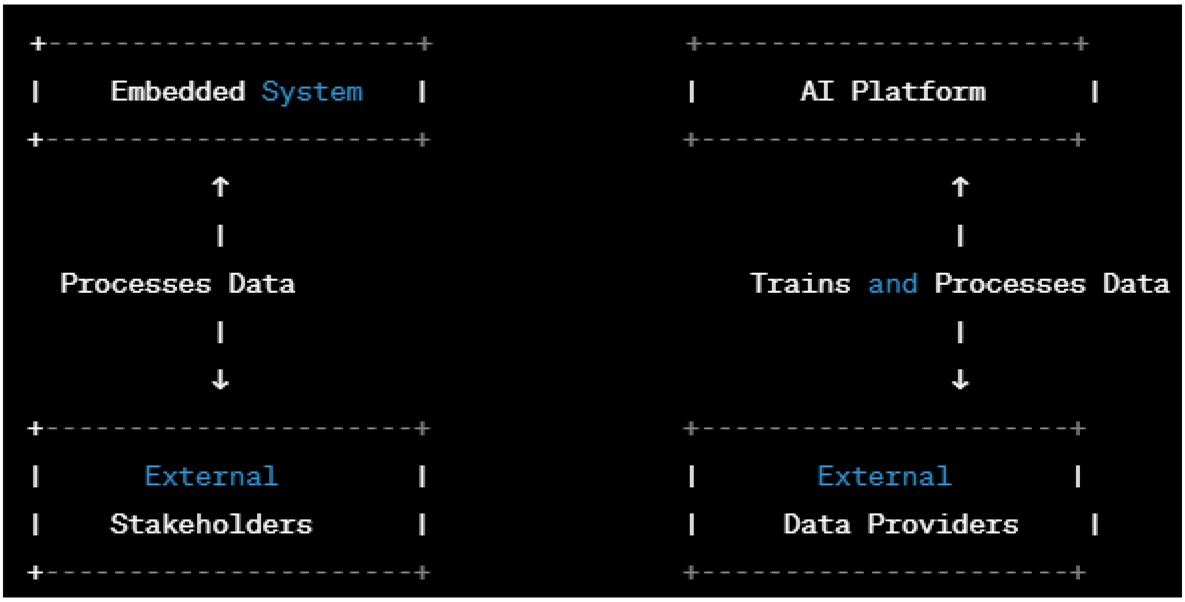
Problem: Many companies are failing to adopt AI due to a lack of understanding about the technology and its implementation. The existing approaches to software architecture have not adequately addressed the question of the adequacy of AI for existing systems and those that are to be developed.

Solution: The thesis aims to provide a systematic approach to help embedded system companies evaluate the adoption of AI into their current portfolio. The solution will support decision-makers, identify AI-specific architectural properties, engineering techniques to evaluate the existing system, and enable a discussion about the benefits, constraints, and risks of adopting AI. The Solution Adequacy Check will be enhanced to address AI-specific circumstances and evaluate AI-based architectures. The design decision process will map requirements to potential solutions using either the Solution Adequacy Check or SWOT analysis. The solution will focus on AI on the edge and its hardware/software architecture, which is applicable to various vertical markets such as medical, consumer, automotive, and industrial.

2. Suggest architecture of a conceptual solution for this problem. We need a C4 system context diagram (https://c4model.com/#SystemContextDiagram). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Identify relevant external systems (if any) and add them to the diagram. - Identify relevant external stakeholders (if any) and add them to the diagram. - Identify high level abstract data types transferred between components and stakeholders. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

Unfortunately, as an AI language model, I cannot create a visual diagram with arrows, dots, and different shapes. However, I can provide a text-based description of the C4 system context diagram for the conceptual solution:

System Context Diagram:



```
+----------------------+                +----------------------+
|    Embedded System   |                |       AI Platform     |
+----------------------+                +----------------------+
           ↑                                       ↑
           |                                       |
   Processes Data                       Trains and Processes Data
           |                                       |
           ↓                                       ↓
+----------------------+                +----------------------+
|       External       |                |       External       |
|     Stakeholders     |                |     Data Providers    |
+----------------------+                +----------------------+
```

The above C4 system context diagram shows the conceptual solution architecture for the adoption of AI in embedded systems. The solution comprises two main components - the embedded system and the AI platform. The embedded system processes data and sends it to the AI platform for training and processing. The AI platform also receives data from external data providers to enhance the training of the system.

The external stakeholders, which include customers, users, and other relevant parties, interact with the embedded system. The embedded system also interacts with external data providers to receive relevant data for processing.

The high-level abstract data types transferred between the embedded system and the AI platform are the raw data for processing, the processed data for storage, and the trained models for deployment. The external data providers transmit relevant data to the AI platform for training and processing.

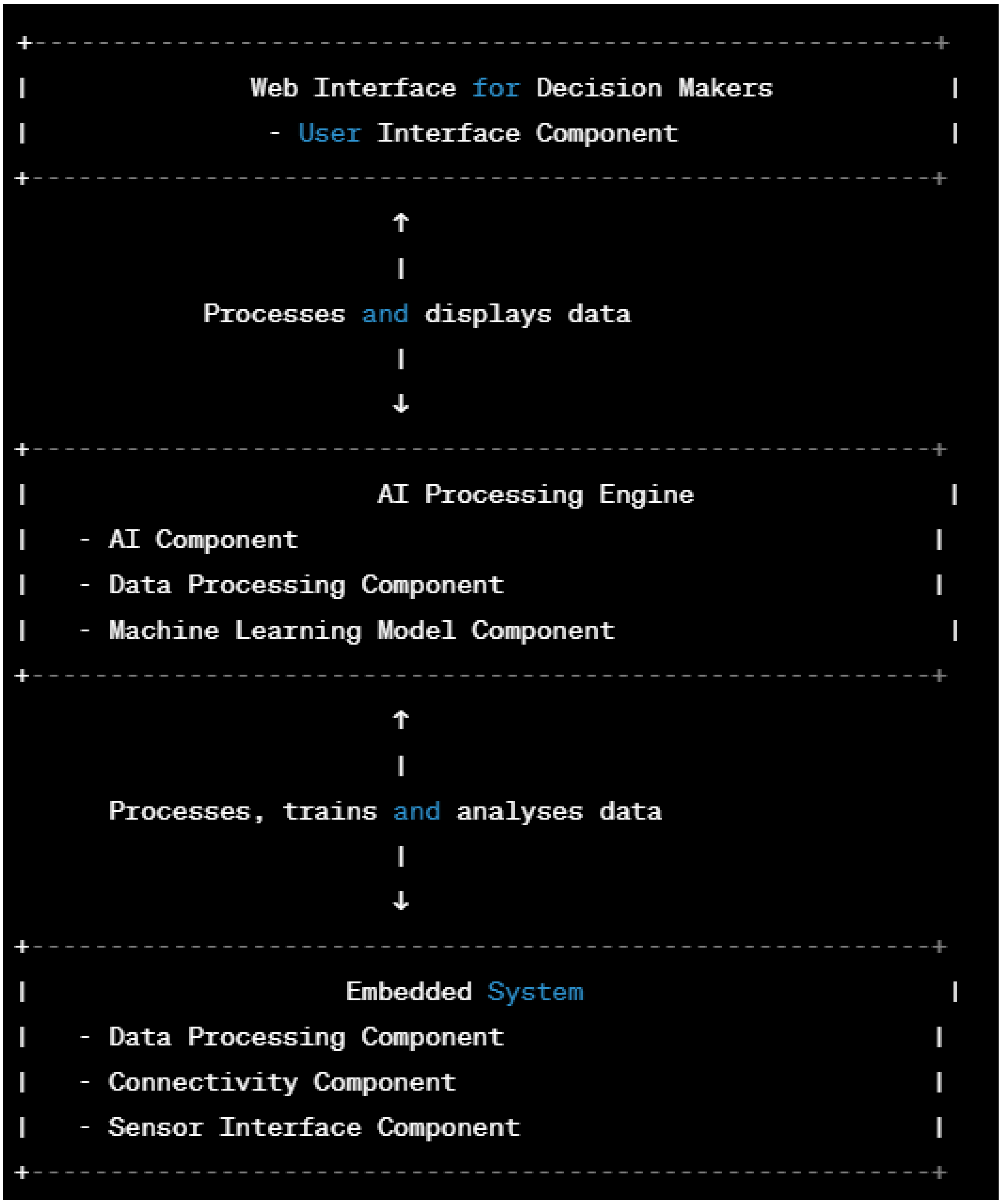
In summary, the embedded system processes data and sends it to the AI platform for training and processing. The AI platform receives data from both the embedded system and external data providers for training and processing. Relevant external stakeholders interact with the embedded system, and external data providers transmit data to the AI platform for training and processing. The high-level abstract data types transmitted between components are raw data, processed data, and trained models.

3. Suggest architectural components that we need to solve this problem. We need a C4 container solution diagram (https://c4model.com/#ContainerDiagram). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Define how the main stakeholder will use the solution. Add it to the diagram.. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required

and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

Unfortunately, as an AI language model, I cannot create a visual diagram with arrows, dots, and different shapes. However, I can provide a text-based description of the C4 container solution diagram for the architectural components:

Container Diagram:



```
+-----------------------------------------------------------+
|               Web Interface for Decision Makers            |
|                 - User Interface Component                 |
+-----------------------------------------------------------+
                        ↑
                        |
            Processes and displays data
                        |
                        ↓
+-----------------------------------------------------------+
|                    AI Processing Engine                    |
|   - AI Component                                          |
|   - Data Processing Component                             |
|   - Machine Learning Model Component                       |
+-----------------------------------------------------------+
                        ↑
                        |
      Processes, trains and analyses data
                        |
                        ↓
+-----------------------------------------------------------+
|                      Embedded System                       |
|   - Data Processing Component                             |
|   - Connectivity Component                                |
|   - Sensor Interface Component                            |
+-----------------------------------------------------------+
```

The above C4 container diagram shows the architectural components for the adoption of AI in embedded systems. The solution comprises three main components - the web interface for decision-makers, the AI processing engine, and the embedded system.

The main stakeholder, decision-makers, will use the web interface component to interact with the system. The web interface will provide the user interface for decision-makers to view and manage the data processed by the system.

The important quality properties for the system are reliability, performance, and scalability. To address these, the AI processing engine will use a microservices architecture style and the observer design pattern to handle data processing and machine learning models. The data processing component will use the publish-subscribe design pattern to handle incoming data from the embedded system. The connectivity component will use the client-server architecture style to handle communication with the embedded system.

The data types transferred between components are raw data, processed data, and trained machine learning models.

The interfaces between components include:

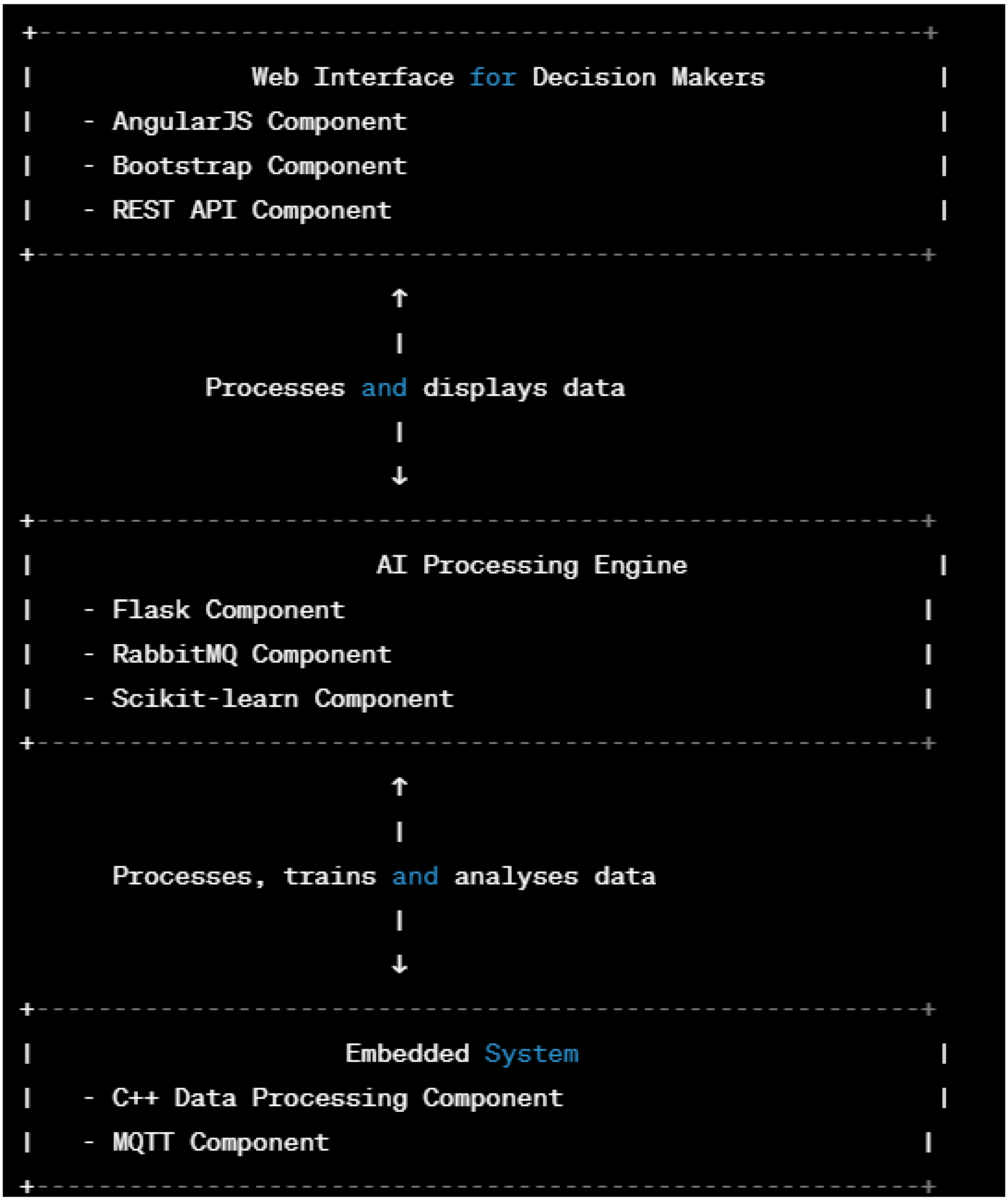
- The web interface component requires access to the processed data from the AI processing engine. The interface data types are processed data.
- The AI processing engine requires access to the raw data from the embedded system and sends the processed data and trained machine learning models back to the embedded system. The interface data types are raw data, processed data, and trained machine learning models.
- The embedded system requires access to the processed data and trained machine learning models from the AI processing engine. The interface data types are processed data and trained machine learning models.

Anti-patterns to avoid when implementing this solution include tight coupling, over-engineering, and the God object anti-pattern. These can be avoided by using appropriate architectural patterns and designing components with clear responsibilities and interfaces.

4. Suggest architectural components that we need to solve this problem. We need a C4 component solution diagram (https://c4model.com/#ComponentDiagram). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider following: - Define how the main stakeholder will use the solution. Add it to the diagram. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

Unfortunately, as an AI language model, I cannot create a visual diagram with arrows, dots, and different shapes. However, I can provide a text-based description of the C4 component solution diagram for the architectural components:

Component Diagram:



```
+----------------------------------------------------------+
|              Web Interface for Decision Makers            |
|   - AngularJS Component                                   |
|   - Bootstrap Component                                   |
|   - REST API Component                                    |
+----------------------------------------------------------+
                        ↑
                        |
            Processes and displays data
                        |
                        ↓
+----------------------------------------------------------+
|                      AI Processing Engine                 |
|   - Flask Component                                      |
|   - RabbitMQ Component                                   |
|   - Scikit-learn Component                               |
+----------------------------------------------------------+
                        ↑
                        |
      Processes, trains and analyses data
                        |
                        ↓
+----------------------------------------------------------+
|                    Embedded System                        |
|   - C++ Data Processing Component                         |
|   - MQTT Component                                       |
+----------------------------------------------------------+
```

The above C4 component diagram shows the architectural components for the adoption of AI in embedded systems. The solution comprises three main components - the web interface for decision-makers, the AI processing engine, and the embedded system.

The main stakeholder, decision-makers, will use the web interface component to interact with the system. The web interface component will consist of an AngularJS component, a Bootstrap component, and a REST API component.

The important quality properties for the system are reliability, performance, and scalability. To address these, the AI processing engine will use a microservices architecture style with a Flask component for the web framework, a RabbitMQ component for messaging, and a Scikit-learn component for machine learning models. The embedded system will use a C++ data processing component and an MQTT component for messaging.

The data types transferred between components are raw data, processed data, and trained machine learning models.

The interfaces between components include:

- The web interface component requires access to the processed data from the AI processing engine. The interface data type is processed data.
- The AI processing engine requires access to the raw data from the embedded system and sends the processed data and trained machine learning models back to the embedded system. The interface data types are raw data, processed data, and trained machine learning models.
- The embedded system requires access to the processed data and trained machine learning models from the AI processing engine. The interface data types are processed data and trained machine learning models.

Anti-patterns to avoid when implementing this solution include tight coupling, over-engineering, and the God object anti-pattern. These can be avoided by using appropriate architectural patterns and designing components with clear responsibilities and interfaces. The microservices architecture style and the use of messaging components can help reduce coupling, while the use of design patterns such as publish-subscribe and observer can help with component responsibilities and interfaces.

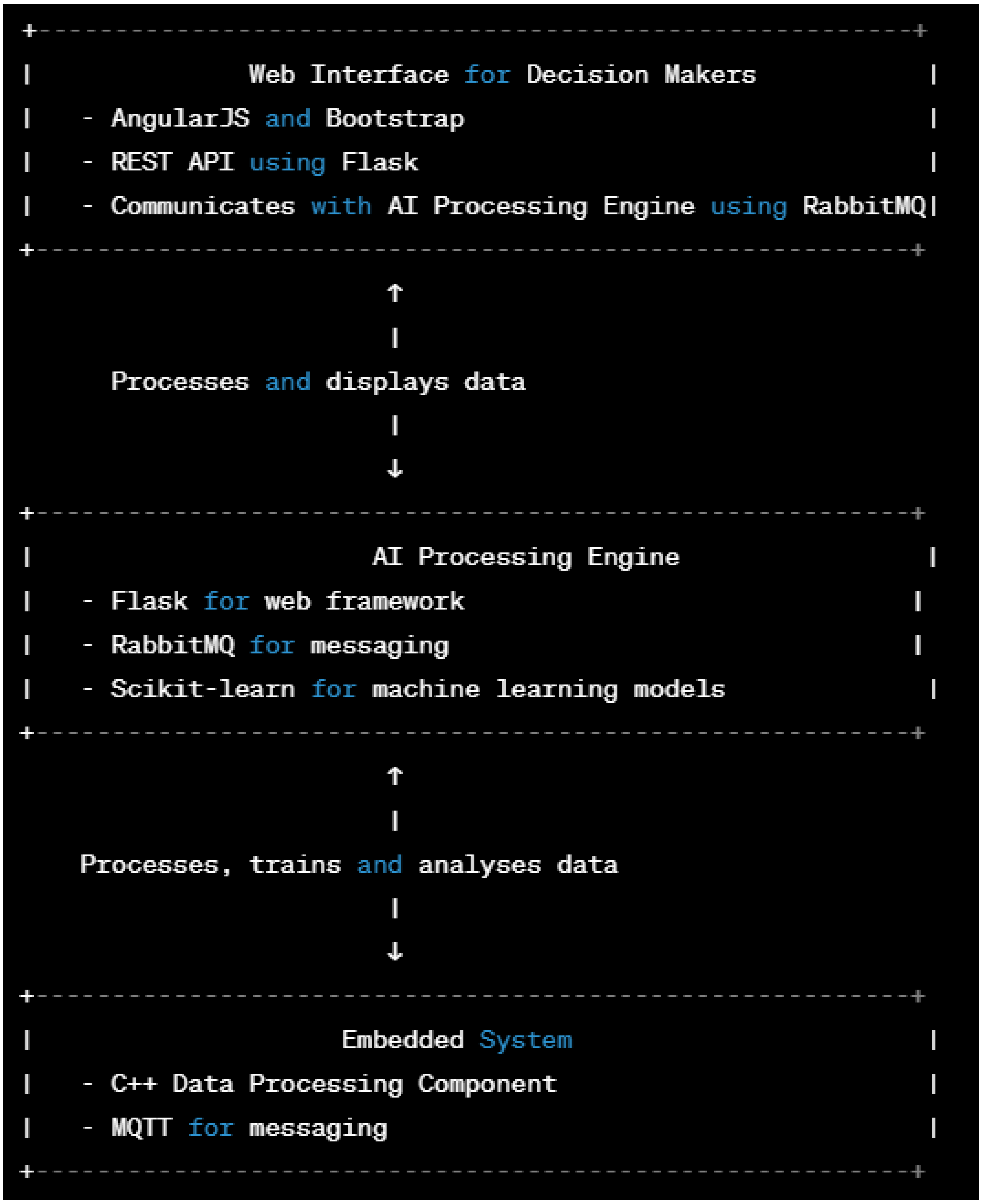
5. Please suggest a toolchain to implement the solution. Consider which tools we can reuse. For the components we need to develop, please suggest a programming language. Please use text (arrows and dots) to visualise the solution. Define data types transferred between components. Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

To implement the proposed solution, we can reuse several existing tools and technologies, such as:

- AngularJS and Bootstrap for the web interface component
- Flask for the web framework of the AI processing engine
- RabbitMQ for messaging between components
- Scikit-learn for the machine learning models in the AI processing engine
- MQTT for messaging in the embedded system

For the components that we need to develop, we can use C++ for the data processing component in the embedded system, and Python for the RabbitMQ, Scikit-learn, and Flask components in the AI processing engine.

The toolchain for the solution can include:



```
+-----------------------------------------------------------+
|              Web Interface for Decision Makers             |
|   - AngularJS and Bootstrap                                |
|   - REST API using Flask                                   |
|   - Communicates with AI Processing Engine using RabbitMQ|
+-----------------------------------------------------------+
                        ↑
                        |
       Processes and displays data
                        |
                        ↓
+-----------------------------------------------------------+
|                      AI Processing Engine                  |
|   - Flask for web framework                               |
|   - RabbitMQ for messaging                                |
|   - Scikit-learn for machine learning models               |
+-----------------------------------------------------------+
                        ↑
                        |
     Processes, trains and analyses data
                        |
                        ↓
+-----------------------------------------------------------+
|                    Embedded System                         |
|   - C++ Data Processing Component                          |
|   - MQTT for messaging                                     |
+-----------------------------------------------------------+
```

The data types transferred between components include raw data, processed data, and trained machine learning models. The connections and interfaces between components include:

- The web interface component sends HTTP requests to the REST API of the AI processing engine, requesting processed data. The interface data type is processed data.
- The RabbitMQ component in the AI processing engine receives raw data from the embedded system, processes it, and sends the processed data and trained machine learning models back to the embedded system. The interface data types are raw data, processed data, and trained machine learning models.
- The MQTT component in the embedded system sends raw data to the RabbitMQ component in the AI processing engine and receives processed data and trained machine learning models from the RabbitMQ component. The interface data types are raw data, processed data, and trained machine learning models.