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: Companies are failing to adopt AI due to a lack of understanding of the technology and its implementation. There is a need for a systematic approach to evaluate the adequacy of AI for existing or new systems in the industrial and embedded domain.

Solution: The thesis proposes an approach to support decision-makers and software system architects in designing AI-powered systems. The approach involves identifying AI-specific architectural properties and engineering techniques to evaluate the existing system concerning the applicability of AI. The design decision process is based on the Solution Adequacy Check or SWOT analysis, which maps requirements to potential solutions. The approach enables a discussion about the benefits, constraints, and risks of adopting AI towards technical and business-relevant concerns. The thesis aims to enhance the adequacy check to address AI-domain-related circumstances, allowing for the evaluation of AI-based architectures. The requirements for the solution include supporting companies with existing traditional embedded systems to evaluate the adoption of AI, getting an overview of the state of the art ML implementation in edge computing, and identifying AI-specific architectural properties that influence business, system, and hardware architectures.

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.

Here is a C4 System Context Diagram for the conceptual solution to the problem of supporting decision-makers and software system architects in designing AI-powered systems:

```
                              +-----------------+
                              | External Systems |
                              +-----------------+
                                       ^
                                       |
                                       |
+---------------+      +-----------------------+
| AI-Powered    |      | Decision-Makers/       |
| System        |      | Software System Architects |
+---------------+      +-----------------------+

  ^                        ^
  |                        |
  |                        |
  +------------------------+
           |
           |
    +-------+-------+
    | Data Transfer  |
    +---------------+
```

In this diagram, we have identified the relevant external stakeholders as decision-makers and software system architects. We have also identified external systems that may be relevant to the AI-powered system, but they are not specified in this diagram.

The AI-Powered System component represents the proposed solution, which will receive data from the decision-makers and software system architects and perform AI-related processing on it. The Data Transfer component represents the communication between the AI-Powered System and the stakeholders.

The data types transmitted between components and stakeholders would include requirements and specifications for the AI-powered system, as well as feedback and analysis of the system's performance. These data types would be transmitted over the Data Transfer component.

Overall, this system context diagram provides a high-level overview of the proposed solution and its relationship to external stakeholders and systems.

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.

Here is a C4 Container Diagram for the proposed solution:

```
  +----------------+
  | User Interface |
  +----------------+
          ^
          |
          |
+--------------------+
| AI-Powered Backend |
+--------------------+
          ^
          |
          |
+-----------------------+
| Data Storage and APIs |
+-----------------------+
```

The main stakeholder, decision-makers, and software system architects will use the User Interface component to interact with the AI-Powered Backend component. The User Interface will allow them to input requirements and specifications for the AI-powered system and view the system's performance metrics and analysis.

The important quality properties for this solution are reliability, scalability, and maintainability. To address these properties, we will use a microservices architecture style for the AI-Powered Backend component, which will consist of multiple independently deployable and scalable services. We will also use containerization and orchestration to improve the reliability and scalability of the system.

The AI-Powered Backend component will consist of several microservices, including a Data Ingestion Service, a Machine Learning Service, and a Decision-Making Service. The Data Storage and APIs component will provide the necessary storage and APIs for the AI-Powered Backend to interact with the data.

The data types transferred between components include requirements and specifications for the AI-powered system, as well as training and testing data for the Machine Learning Service. The Data Storage and APIs component will provide storage for this data and expose APIs for the AI-Powered Backend to access it.

The interfaces between components include the User Interface's required interface with the AI-Powered Backend and the AI-Powered Backend's required interface with the Data Storage and APIs component. The User Interface component will provide input and output data types for the AI-Powered Backend, while the AI-Powered Backend will provide data types for the Data Storage and APIs component.

To avoid anti-patterns, we will follow the best practices of microservices architecture, including using small and focused services, having clear and distinct boundaries between services, and minimizing inter-service communication. We will also ensure proper monitoring and logging to detect and address issues in the system.

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.

Here is a C4 Component Diagram for the proposed solution:

```
+-----------------+   +---------------------+   +-------------------+
| UI Component    |   | AI-Powered Backend  |   | Data Storage      |
|                 |   |                     |   | and APIs          |
|  - Input/Output |   |  - Data Ingestion   |   |  - Input/Output   |
|                 |   |  - Machine Learning |   |                   |
+-----------------+   |  - Decision-Making  |   +-------------------+
                      +---------------------+
```

The main stakeholder, decision-makers, and software system architects will use the UI Component to interact with the AI-Powered Backend. The UI Component will allow them to input requirements and specifications for the AI-powered system and view the system's performance metrics and analysis.

The important quality properties for this solution are reliability, scalability, and maintainability. To address these properties, we will use a microservices architecture style for the AI-Powered Backend component, which will consist of three independent microservices - Data Ingestion, Machine Learning, and Decision-Making. These microservices will communicate with each other using a message queue system, such as Kafka or RabbitMQ. The Data Storage and APIs component will provide the necessary storage and APIs for the AI-Powered Backend to interact with the data.

The data types transferred between components include requirements and specifications for the AI-powered system, as well as training and testing data for the Machine Learning microservice.

The Data Storage and APIs component will provide storage for this data and expose APIs for the AI-Powered Backend to access it.

The interfaces between components include the UI Component's required interface with the AI-Powered Backend, which will provide input and output data types for the microservices, and the AI-Powered Backend's required interface with the Data Storage and APIs component, which will provide input and output data types for the storage and APIs.

To avoid anti-patterns, we will follow the best practices of microservices architecture, including using small and focused microservices, having clear and distinct boundaries between microservices, and minimizing inter-microservice communication. We will also ensure proper monitoring and logging to detect and address issues in the system.

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 use the following toolchain:

- UI Component: React.js for building the user interface.
- AI-Powered Backend: Python for the Machine Learning and Decision-Making microservices, and Java for the Data Ingestion microservice.
- Data Storage and APIs: MongoDB for storing data, and Flask for exposing APIs.

We can reuse the following tools:

- Message Queue System: Apache Kafka for inter-microservice communication.
- Docker and Kubernetes for containerization and deployment.

Here is a diagram visualizing the toolchain and the connections between the components:

```
+-----------------+  +---------------------------------+    +------------------+
| UI Component    |  | AI-Powered Backend                 |    | Data Storage     |
|                 |  |                                 |    | and APIs         |
|  - Input/Output |  |  - Data Ingestion (Java)         |    |  - Input/Output  |
|                 |  |  - Machine Learning (Python)     |    |                  |
+-----------------+  |  - Decision-Making (Python)      |    +------------------+
                      +--------------------+------------+
                                           |   Apache Kafka
                                           |
                                 +-----------------+
                                 | Data Storage    |
                                 |                 |
                                 |     MongoDB     |
                                 +-----------------+
                                 |   Flask APIs    |
                                 +-----------------+
```

The data types transferred between components include JSON-encoded data for requirements and specifications, as well as training and testing data in the form of CSV or JSON files. The Data Storage and APIs component will also provide APIs that return JSON-encoded data.

The interfaces between components include the UI Component's required interface with the AI-Powered Backend, which will provide input and output data types for the microservices, and the AI-Powered Backend's required interface with the Data Storage and APIs component, which will provide input and output data types for the storage and APIs. The Data Ingestion microservice will receive CSV or JSON files as input, and the Machine Learning and Decision-Making microservices will receive JSON-encoded data as input. The microservices will also output JSON-encoded data. The APIs provided by the Data Storage and APIs component will accept and return JSON-encoded data.