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, and existing architecting approaches have not appropriately dealt with the adequacy of AI for existing software systems and for those to be developed.

Solution: The thesis aims to address these shortcomings by presenting an approach that supports decision-makers in designing an AI-powered system on an architectural level, taking into consideration the most relevant AI-specific properties. The proposed solution should support companies with existing traditional embedded systems to evaluate the adoption of AI into their current portfolio, identify AI-specific architectural properties that influence business, system, and hardware architectures, identify engineering techniques to evaluate the existing system concerning the applicability of AI, and provide a design decision process based on the Solution Adequacy Check or SWOT analysis that maps requirements to potential solutions. It should also enable a discussion about technical and business-relevant concerns, such as added-values, benefits, constraints, and risks of adopting AI.

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.

Please note that as an AI language model, I do not have access to a visual interface to generate diagrams or shapes. However, I can describe the architecture in text form.

System Context Diagram:

- The main system component is an AI-powered software system.
- External systems include data sources, such as databases or IoT devices that provide data for the AI system, and other systems that use the output of the AI system.
- External stakeholders include decision-makers, users of the AI system, and developers who implement the system.
- Data types transmitted between components include data from the data sources, such as sensor data or database records, and output data from the AI system, such as predictions or recommendations.
- The AI system component processes the data from the external data sources and provides output data to other systems or end-users.
- The system also includes components for data processing, storage, and analysis, as well as components for user interfaces and system administration.
- Interfaces between components should be designed to ensure compatibility and security of the transmitted data.

The C4 system context diagram should include boxes representing each component, with arrows indicating the flow of data and dotted lines indicating external connections. Different shapes and colours can be used to differentiate between components and external 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.

Please note that as an AI language model, I do not have access to a visual interface to generate diagrams or shapes. However, I can describe the architecture in text form.

Container Diagram:

- The main stakeholder will use the solution to evaluate the adoption of AI into their existing embedded systems and to design new AI-powered systems. They will interact with the system through a web-based user interface.
- Quality properties that are important include scalability, maintainability, security, and performance.
- The architectural style will be a microservices architecture, with each microservice providing a specific functionality.
- Design patterns that will be used include the repository pattern for data storage and the observer pattern for event-driven communication between microservices.
- Design tactics that will be used include caching to improve performance, load balancing to ensure scalability, and containerization to improve maintainability and deployment.
- Components will include microservices for data processing, machine learning, user interface, and system administration.
- Data types transmitted between components will include data from external data sources, such as sensor data or database records, and output data from the machine learning microservice, such as predictions or recommendations.
- Interfaces between components will be designed using REST APIs, with required and provided interfaces specified for each microservice. Data types will also be specified for each interface.
- Anti-patterns to avoid when implementing this solution include the God Object anti-pattern, where a single component or class becomes too large and complex, and the Monolithic application anti-pattern, where the application is not modularized and is difficult to maintain and scale.

The C4 container diagram should include boxes representing each microservice, with arrows indicating the flow of data and dotted lines indicating external connections. Different shapes and colours can be used to differentiate between microservices and external systems.

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.

Please note that as an AI language model, I do not have access to a visual interface to generate diagrams or shapes. However, I can describe the architecture in text form.

Component Diagram:

- The main stakeholder will use the solution to evaluate the adoption of AI into their existing embedded systems and to design new AI-powered systems. They will interact with the system through a web-based user interface.
- Quality properties that are important include scalability, maintainability, security, and performance.
- The architectural style will be a layered architecture, with each layer responsible for a specific aspect of the system, such as data processing, machine learning, or user interface.
- Design patterns that will be used include the repository pattern for data storage, the observer pattern for event-driven communication between components, and the adapter pattern for integrating with external data sources.
- Design tactics that will be used include caching to improve performance, load balancing to ensure scalability, and containerization to improve maintainability and deployment.
- Components will include modules for data processing, machine learning, user interface, and system administration.
- Data types transmitted between components will include data from external data sources, such as sensor data or database records, and output data from the machine learning module, such as predictions or recommendations.
- Interfaces between components will be designed using REST APIs, with required and provided interfaces specified for each component. Data types will also be specified for each interface.
- Anti-patterns to avoid when implementing this solution include the God Object anti-pattern, where a single component or class becomes too large and complex, and the Monolithic application anti-pattern, where the application is not modularized and is difficult to maintain and scale.

The C4 component diagram should include boxes representing each component, with arrows indicating the flow of data and dotted lines indicating external connections. Different shapes and colours can be used to differentiate between components and external systems. The layers of the architecture can be represented as different coloured boxes, with arrows indicating the flow of data between them.

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.

Toolchain:

- The solution can be implemented using a variety of open source tools and technologies. These include:
    - Docker for containerization and deployment
    - Kubernetes for container orchestration
    - Flask or Django for web-based user interface development
    - Python for machine learning and data processing
    - Apache Kafka for event-driven communication between components
    - PostgreSQL for data storage
    - NGINX for load balancing and reverse proxy
    - Prometheus and Grafana for monitoring and metrics

Programming Language:

- Python can be used as the primary programming language for the data processing, machine learning, and web-based user interface components.

Data Types:

- Data types transmitted between components will include JSON or CSV files for input data and output data, as well as metadata such as timestamps and device IDs.

Interfaces:

- REST APIs will be used to define the interfaces between components, with required and provided interfaces specified for each component. Data types will also be specified for each interface.

Component Communication:

- Apache Kafka can be used to facilitate event-driven communication between components. The data processing and machine learning modules will consume data from Kafka topics and produce output to other Kafka topics. The user interface module can also subscribe to Kafka topics to receive updates and notifications.

The toolchain can be represented using arrows and dots, with the arrows indicating the flow of data between components and the dots indicating the tools and technologies used. The data types transmitted between components can be summarised in the connections and interfaces between components, along with the required and provided interfaces and their associated data types.