Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Johan Fredrik Alvsaker

R/V Gunnerus Digital Twin Infrastructure

Master's thesis in Marine Technology
Supervisor: Bjørn Egil Asbjørnslett
July 2020

NTNU
Norwegian University of
Science and Technology

Johan Fredrik Alvsaker

R/V Gunnerus Digital Twin Infrastructure

Master's thesis in Marine Technology
Supervisor: Bjørn Egil Asbjørnslett
July 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MSC THESIS DESCRIPTION SHEET

Name of the candidate: Johan Fredrik Alvsaker

Field of study: Marine control engineering

Thesis title (Norwegian): Infrastruktur for en digital tvilling av FF Gunnerus

Thesis title (English): R/V Gunnerus Digital Twin Infrastructure

Background
NTNU's research vessel, R/V Gunnerus, has been in operation since 2006. The vessel serves a wide variety of marine research purposes, covering fields such as biology, technology, geology, archeology, oceanography, and fisheries research. In addition to its academic significance, the research vessel facilitates student activity through a practical, hands-on approach. However, there are added benefits to the research vessel that can be enabled by using generated data from systems and equipment on board. For example, a digital twin represents such an enabler, where a remote representation of the vessel has access to data and can analyze and utilize the data to extract in-operation information or knowledge of other relevant asset conditions. In short, a digital twin is a digital representation of a physical asset, its related processes, systems, and information. Ultimately, a digital twin receiving vessel and signal data can add value to the research vessel, creating a variety of new educational and academic possibilities. Regarding the education of engineering students, the ability to adopt a digital vessel encourages students to relate learning material from different subjects to an existing vessel in operation, providing a practical understanding of theory. The multitude of academic research areas demands a rigid foundation for facilitating different interests.

Through a pre-project carried out in the fall of 2019, an R/V Gunnerus digital twin infrastructure, RVG DTI, was proposed. The infrastructure intends to facilitate the desired functionality of an R/V Gunnerus digital twin. The three main components of the proposed infrastructure were data management, modeling and simulation environments, and software and system realization, and each of the components facilitates different sub-functionalities. Data management revolves around availability, storage, access, and utilization of data. Ultimately, data management should enable learning through data analytics. The modeling and simulation environment provides the necessary elements to represent the real asset in a virtual space. The environment should only be as realistic as necessary for the digital twin functionality. Although the three components are separate, they must be intertwined and work together to represent a complete digital twin foundation. With the concept of these three fundamental building blocks, it is possible to add features for distinct use without compromising the general foundation. As such, the same digital twin infrastructure can be a tool for several different applications in different fields.

Objectives
This report will present use-cases for a digital twin of R/V Gunnerus for educational purposes. More specifically, the use of digital twins in the education of marine engineering students will be in focus. To present use-cases for such a digital twin, it is necessary also to investigate the definition and properties of digital twins and apply these to the specific case of R/V Gunnerus. An objective of the project is to promote further development of an R/V Gunnerus digital twin through theoretical work and developing solutions. In addition, a case study is carried out as an example of how functionality commonly seen in digital twins can be used in engineering education. The case study is related to data management, which is one of the core infrastructure components. In the case study, the goal is to create a framework for real-time anomaly detection on R/V Gunnerus systems through a data-driven approach. The framework will include a modeling environment for developing data-driven models and a web application for implementing, testing, and visualizing the results of the anomaly detection model. The objectives are formalized in a work description.

Work description
1. Perform a background and literature review to provide information and relevant references on:
    - Digital twin definitions and use-cases in a maritime context, focusing on the facilitation of engineering education.
    - Previous initiatives related to R/V Gunnerus as a technological platform and access to vessel data.

NTNU
Norwegian University of Science and Technology
Faculty of Engineering Science and Technology
Department of Marine Technology

- Write a list with abbreviations and definitions of terms and symbols relevant to the literature study and project report.

2. Propose a digital twin infrastructure for R/V Gunnerus, serving as a foundation for continued digital twin development.
3. Present an R/V Gunnerus digital twin as a tool for engineering education, considering pedagogical value, different use-cases, and the digital twin lifecycle.
4. Present a case study where digital twin-related solutions are used to create a tool for engineering education at the Institute of Marine Technology. More specifically, the case study consists of creating a framework for predictive maintenance on R/V Gunnerus which allows implementing and testing data-driven anomaly detection models.
5. Properly document the case study for readability and reusability and make all source code available through GitHub repositories. The web application should be launched into production for demonstration purposes.
6. Propose future work on the topic.
7. Write a report documenting the conducted work, results, and discussion. The report will be in accordance with the specifications below.

Specifications
The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the various steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, problem, design, results, and critical assessments. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 70 A4-pages, 100 B5-pages, from introduction to conclusion, unless otherwise agreed upon. It shall be written in English (preferably US) and contain the elements: Title page, abstract, project specification, list of symbols and acronyms, table of contents, introduction (project motivation, objectives, scope, and delimitations), background/literature review, problem formulation, main parts with design, development, and results, conclusions with recommendations for further work, references, and optional appendices. Figures, tables, and equations shall be numerated. The original contribution of the candidate and material taken from other sources shall be clearly identified. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. *natbib* Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct, which is taken very seriously by the university and cause consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed upon.

The thesis shall be submitted with an electronic copy to the main supervisor and department according to NTNU administrative procedures. The final revised version of this thesis description shall be included after the title page. Computer code, pictures, videos, dataseries, etc., shall be included electronically with the report.

Start date: 15 January, 2020 **Due date:** 1 July, 2020

Supervisor: Bjørn Egil Asbjørnslett
Co-advisor(s): Roger Skjetne

Bjørn Egil Asbjørnslett
Supervisor

2

Preface

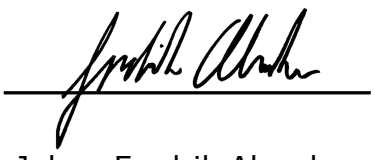
The following report is a master thesis conducted at The Norwegian University of Science and Technology as part of the Department of Marine Technology. The work in this report was carried out during the spring of 2020 as part of the course *TMR4930 – Marine Technology, master thesis* in the field of marine cybernetics. The topic of the master thesis is digital twins, related explicitly to NTNU's research vessel R/V Gunnerus.

As the world has been digitizing at unprecedented speeds, all engineering fields have been intertwined with computer technology to automate, improve, and make industries of all kinds more effective. Digitization has always piqued my interests, and computer science and programming have also become dear to my heart through my studies. I have taken as many courses relevant to computer science as possible, while still maintaining an interest in marine applications and marine cybernetics. As such, it seemed appropriate to fuse my interests in marine technology, digitization, and computer science for my master's thesis. I have also developed an interest in pedagogy and education, and I firmly believe that it is essential for universities to adapt teaching methods to new technology sooner rather than later. Hopefully, the results presented in this report can give coming engineering students a useful introduction to digital topics highly relevant to the maritime industry of the future.

I went into the project with limited programming experience and had high, somewhat unrealistic ambitions. It has been overwhelming at times, but I have come out the other end with more newfound knowledge than I possibly could have imagined. I am grateful for the chance I got to write a thesis about topics that capture my interests. Although the report boils down to discussing how and why code was written, there are thousands of decisions taken and dead ends experienced throughout the project that are hard to get across in a written report.

I would like to thank my supervisor, Professor Bjørn Egil Asbjørnslett, for guiding me throughout my assignment, and for motivating me to focus on the most essential elements and prevent derailing off into new tasks. My co-supervisor, Professor Roger Skjetne, has provided fruitful discussions and valuable insight necessary to get a complete overview of, and real understanding, of related topics. Finally, I would like to thank senior engineer and technical inspector Finn Tore Holmeset at NTNU Ålesund for his openness and hospitality in providing vessel information regarding R/V Gunnerus and access to corresponding data.

Trondheim, July 1, 2020



Johan Fredrik Alvsaker

Abstract

This thesis investigates the potential of digital twins for the education of engineering students through NTNU's research vessel R/V Gunnerus. As digital twins are becoming more and more relevant for increasing the knowledge of assets in operation, it is essential to evaluate the benefits of using digital twins for education purposes as well. This thesis discusses an infrastructure for a digital twin of R/V Gunnerus, how a digital twin could be a useful tool for educating marine engineering students, and exemplifying this through a case study based on signal data from the vessel. The efforts in this thesis are based on a series of previous initiatives related to a digital twin of R/V Gunnerus and is intended to further the work on the topic.

As the defining properties of a digital twin vary based on its intended purpose and area of application, it is necessary to look at the specific case of an R/V Gunnerus digital twin through the overall definition space of digital twins. In an academic setting, the desired functionality of a digital twin varies based on discipline. To be a valuable resource to as many disciplines as possible, the digital twin needs a well-defined foundation. Based on a literary review of digital twin definitions, an R/V Gunnerus digital twin infrastructure (DTI) is proposed as a fundamental building block for a digital twin. The DTI consists of three components, namely data management, modeling and simulation environment, and software and system realization. Each component enables specific functions necessary for a true digital twin, and the facilitation of these functionalities are explored concerning R/V Gunnerus. Next, a digital twin of R/V Gunnerus is considered as a pedagogical tool, and a lifecycle for digital twins is suggested to include students in all life phases of a digital twin.

As an example of how a typical digital twin application can be used in marine engineering education, a case study revolving condition-based maintenance through means of artificial neural networks (ANNs) is conducted. In the case study, a framework for anomaly detection for predictive maintenance is developed, which makes it possible for students to create, implement, and test data-driven algorithms on a selection of R/V Gunnerus systems. The framework is twofold, where the first part consists of a modeling framework made with Python for developing recurrent neural network (RNN) models. The second part consists of creating a web application for uploading and visualizing model predictions and detected anomalies in a real-time environment. The web application is made with a frontend in React through JavaScript, a backend in Flask through Python, and a database through PostgreSQL for storing vessel data. The web application is launched into development through the cloud platform Heroku. Together, the modeling part and web application form an anomaly detection framework for creating, implementing, and testing sequential ANN models, and successively applying the developed models to a practical use-case.

Both the modeling framework and the web application are tested against a simulated error that has been provoked on the exhaust signals on one of the main engines on R/V Gunnerus, where the temperatures rise above the standard operation maxima. Through the modeling framework, a simple model based on a Long Short-Term Memory (LSTM) network – which is a type of RNN – was created as an example to verify functionality. When testing the model through the modeling framework, the prediction model managed to detect 95.8 % of the simulated error interval. When the model was uploaded and tested on the web application, a similar performance was achieved.

Sammendrag

Denne rapporten undersøker potensialet for bruk av digitale tvillinger i utdanningsøyemed for ingeniørstudenter gjennom NTNUs forskningsfartøy FF Gunnerus. Samtidig som digitale tvillinger blir mer og mer relevant for å øke kunnskapen om et fartøy eller system under operasjon, bør man evaluere nytteverdien av digitale tvillinger også for utdanning. I denne oppgaven foreslås en infrastruktur for en digital tvilling av FF Gunnerus, samt hvordan en digital tvilling kan være en nyttig ressurs for utdanningen av mariningeniører. Dette eksemplifiseres i en casestudie basert på signaldata fra FF Gunnerus. Arbeidet i denne rapporten har utgangspunkt i en rekke tidligere initiativer knyttet til en digital tvilling av FF Gunnerus, og formålet er å fremme videre arbeid mot en slik tvilling.

Definisjonen av en digital tvilling varierer med formål og anvendelsesområde. Derfor er det nødvendig å knytte ulike definisjoner opp mot en digital tvilling av FF Gunnerus. Fra et akademisk ståsted er ønsket funksjonalitet avhengig av disiplin. For å være en ressurs for mange ulike disipliner trenger en digital tvilling et robust fundament. Basert på en litteraturstudie rundt definisjoner og bruk av digital tvillinger foreslås en digital tvilligsinfrastruktur som byggestein for en fullverdig digital tvilling. Infrastruktur består av tre deler, nemlig databehandling, modelleringsog simuleringsmiljø, og program- og systemvare. Hver komponent legger til rette ulike funksjoner som gjennomgås i rapporten. Videre diskuteres verdien av en digital tvilling som pedagogisk verktøy, og en livssyklusmodell for digitale tvillinger foreslås for å inkludere studenter i en større del av det teknologiske handlingsrommet knyttet til digitale tvillinger.

Gjennom en casestudie eksemplifiseres et typisk brukseksempel av en digital tvilling til utdanning av mariningeniører. Casestudien tar for seg tilstandsbasert vedlikehold ved hjelp av kunstige nevrale nettverk (ANNs). Gjennom casestudien utvikles et rammeverk for avviksdetektering. Rammeverket kan anvendes som et prediktivt vedlikeholdsverktøy. Gjennom rammeverket har studenter mulighet til å lage, implementere og teste data-drevne algoritmer på utvalgte systemer på FF Gunnerus. Rammeverket er todelt, hvor den ene delen omhandler et modelleringsverktøy utviklet i Python for utvikling av tilbakematede nevrale nettverksmodeller (RNNs). Den andre delen tar for seg utvikling av en webapplikasjon hvor man kan laste opp trente modeller og visualisere predikert data og detekterte avvik i et sanntidsmiljø. Webapplikasjonen er utviklet med en React frontend skrevet i JavaScript, en Flask backend skrevet i Python og et databasesystem for lagring av skipsdata gjennom PostgreSQL. Webapplikasjonen publiseres gjennom skytjenesten Heroku. Både modelleringsverktøyet og webapplikasjonen testes ved å implementere en testmodell gjennom modelleringsverktøyet. Testmodellen er en type RNN kjent som lang kortsiktig hukommelsesnettverk (LSTM). Modellen testes mot en simulert feil på to av eksosutløpene på den ene hovedmotoren. Feilen gjør at eksostemperaturen stiger over forventet maksverdi over et gitt intervall. Gjennom testing klarer modelleringsverktøyet å detektere 95,8 % av de simulerte feilene. Etter å ha lastet opp modellen til webapplikasjon oppnådde sanntidsvisualiseringen liknende prestasjon, som forventet.

Contents

List of Figures

Nomenclature

Abbreviations and Acronyms

ANN	Artificial Neural Network
API	Application Programming Interface
AZ-PM	Azimuthing Permanent Magnet thruster
CBM	Condition-Based Maintenance
CPS	Cyber-Physical System
CSS	Cascading Style Sheets
DP	Dynamic Positioning
DTI	Digital Twin Infrastructure
HTTP	Hyptertext Transfer Protocol
IoT	Internet of Things
KM (CM)	Kongsberg Maritime (Commercial Marine)
LSTM	Long Short-Term Memory
MTTF	Mean Time to Failure
RNN	Recurrent Neural Network
SQL	Structured Query Language
UROP	Undergraduate Research Opportunities Program
V^5	Volume, Velocity, Variety, Veracity, Value
VIS	Vessel Information System
WSGI	Web Server Gateway Interface

1 Introduction

The following report is part of a master's thesis in marine technology at the Norwegian University of Science and Technology (NTNU). More specifically, the report is limited to the field of marine cybernetics. The objective of the thesis is to explore the concept of marine digital twins through NTNU's research vessel, R/V Gunnerus. Digital twins are often regarded in an industrial context, but the topic of marine digital twins is highly relevant for academic purposes as well, and the development of digital twin technologies rely on both academic and industrial efforts.

This report concerns the academic use of marine digital twins, especially related to the education of marine engineering students. Marine digital twins in engineering education is a topic continued from an Undergraduate Research Opportunity Program (UROP) started in 2017, which has had several different activities related to a digital twin of R/V Gunnerus (Asbjørnslett et al., 2019). This thesis represents a continuation of these preceding activities. A pre-project carried out during the fall of 2019 lay the foundation for this thesis, as the pre-project revolved around defining a digital twin infrastructure for R/V Gunnerus. Implied by the name of this report, the concept of a digital twin infrastructure persists in this thesis as well. What a digital twin infrastructure entails, and why it is used in the context of marine digital twins for academic purposes, will be explored in the report.

Figure 1.1: Rendering of R/V Gunnerus and a virtual copy.

1.1 Project Motivation

The motivation from this thesis comes from previous initiatives related to the development of an R/V Gunnerus digital twin. Through several projects, engineering students have been included in digital twin projects alongside professors and industry partners. The projects have shown the value of including students in research related to new technological concepts, which have been beneficial for both students, academia, and industry alike. The students have gained valuable insight into relevant technologies that are not usually lectured in elementary-level marine courses, which can aid the transition into a competitive, technologically driven working environment after finishing the education. Academia can benefit from the perspective of students, both related to developing new ideas, but also related to how digital twins can be used in education, and whether or not the topic of digital twin captures the interest of students or not. For industry, working on company interests alongside resources from the academic field – most notably doctorates and employees – is beneficial as there are no competitive factors between the two parts. Also, collaborating with

academia and students is vital for spreading a company's reputation.

The thesis is intended to motivate similar work for other students. Both when it comes to helping develop resources for digital twins but also learning about new digital technologies through practically using digital twins. From previous initiatives, several use-cases for digital twins have been proposed (Asbjørnslett et al., 2019). The case study in this report has been inspired by these proposals and their intention, which is to use digital twins to aid in the education of engineering students.

1.2 Objectives

The objectives have been outlined in the preliminary thesis description. This report will present use-cases for a digital twin of R/V Gunnerus for educational purposes. More specifically, the use of digital twins in the education of marine engineering students will be in focus. To present use-cases for such a digital twin, it is necessary also to investigate the definition and properties of digital twins and apply these to the specific case of R/V Gunnerus. An objective of the project is to promote further development of an R/V Gunnerus digital twin through theoretical work and developing solutions. In addition, a case study is carried out as an example of how functionality commonly seen in digital twins can be used in engineering education. The case study is related to data management, which is one of the core infrastructure components. In the case study, the goal is to create a framework for real-time anomaly detection on R/V Gunnerus systems through a data-driven approach. The framework will include a modeling environment for developing data-driven models and a web application for implementing, testing, and visualizing the results of the anomaly detection model. The objectives are formalized in the work description below.

1. Perform a background and literature review to provide information and relevant references on:
    - Digital twin definitions and use-cases in a maritime context, focusing on the facilitation of engineering education.
    - Previous initiatives related to R/V Gunnerus as a technological platform and access to vessel data.
    - Write a list with abbreviations and definitions of terms and symbols relevant to the literature study and project report.
2. Propose a digital twin infrastructure for R/V Gunnerus, serving as a foundation for continued digital twin development.
3. Present an R/V Gunnerus digital twin as a tool for engineering education, considering pedagogical value, different use-cases, and the digital twin lifecycle.
4. Present a case study where digital twin-related solutions are used to create a tool for engineering education at the Institute of Marine Technology. More specifically, the case study consists of creating a framework for predictive maintenance on R/V Gunnerus, which allows implementing and testing data-driven anomaly detection models.
5. Properly document the case study for readability and reusability and make all source code available through GitHub repositories. The web application should be launched into production for demonstration purposes.

1.3 Scope and Delimitations

As explored in Section 3, the proposed digital twin infrastructure consists of three main components, namely data management, modeling and simulation environments, and software and system realization. Initially, when the pre-project carried out during the fall of 2019 was concluded, this thesis was intended to further the development of the digital twin infrastructure components. However, as learned through the pre-project, there are several relevant industry-related initiatives currently being developed, which could easily make the works from a single master's thesis futile. Therefore, it was decided to focus on applying potential digital twin functionality to a specific case study instead, in addition to exploring the use-cases for the education of marine engineering students.

A relevant topic that arises when considering machine learning and big data is data quality. Other than having the onboard systems maintained by the vessel vendor, the transmitted data from the vessel utilized for the case study is not verified to any extent. Thus, it is assumed that the data used in the case study is a realistic representation of the vessel's actual, in-operation state. In reality, this may not be the case, but since validating data from the transmitted signals is time-consuming and demands a thorough knowledge of the relevant hardware and software systems, this is disregarded in the project scope.

Further, only data from R/V Gunnerus' main engines will be used in the case study. A simulated error was introduced to the exhaust signals of one of the main engines, causing the temperatures to rise above the standard operation maxima, making the main engines useful for testing purposes. The case study functionality will be tested by creating a sequential model for predicting values on this erroneous data. The intention of the example model is not to perform as best as possible, but to perform sufficiently to detect anomalies successfully. The model should be as simple as possible to demonstrate model capabilities, make the model implementation comprehensible, and save time on tuning the model.

The web application in the case study will be developed without any pre-existing knowledge of web development. This entails that a part of the thesis work revolves around learning the necessary languages and tools to fulfill the intended purposes of the application. Resources that will be used for the case study without pre-existing knowledge include:

- React library (with Redux, component lifecycle, and document object notation),
- Python Flask,
- WebSockets with SocketIO and Flask-SocketIO,
- TensorFlow and Keras,
- Database management with PostgreSQL and Flask SQL Alchemy, and
- launching a web application to Heroku through the gunicorn Web Server Gateway Interface (WSGI), and
- styling through cascading style sheets (CSS).

Familiar resources include:

- Python (and the modules `Pandas`, `Numpy`, and `matplotlib`, which are used throughout the case study),
- vanilla JavaScript,
- basic Hypertext Markup Language (HTML), and
- GitHub.

The resource terms are not defined here, but rather in their respective part of the case study.

1.4 Report Outline

This section is intended to give an overview of the overall structure of the report and its contents, and what the purpose of each section is.

2. **Background**:

   In the background, necessary definitions of digital twins are explored, and a property-driven approach to digital twins is presented. The background is, to a large extent, based on the work carried out in the pre-project during the fall of 2019.

3. **Digital Twin Infrastructure**:

   Here, a foundation for a digital twin of R/V Gunnerus is presented through a proposed digital twin infrastructure. The section documents the pre-project results. These results are used as a guideline throughout the thesis. This section is also based on the work carried out in the pre-project.

4. **Digital twins in marine engineering education**:

   In this section, digital twins are seen from an education perspective, regarding the digital twin as a pedagogical tool, presenting some specific use-cases. A digital twin lifecycle is also suggested to include students in all life phases of a digital twin.

5. **Case study problem formulation**:

   Here, the case study is properly presented. This is done right before the sections related to the actual case study to make the choices taken during the case study more clear.

6. **Anomaly detection for predictive maintenance**

   This section introduces the theoretical background for the case study. The concept of condition-based, predictive maintenance is explored and contextualized through artificial neural networks. The recurrent neural network architecture Long Short-Term Memory (LSTM) is presented in-depth, as this architecture will be used as a sequential model example implemented as a proof of concept for the case study. The section is concluded by presenting the method for detecting anomalies.

7. **Modeling API for anomaly detection**:

This is the first part of the case study, where a modeling framework is developed. The section begins with the concept and methodology used for the modeling API before exploring the actual API functionality and structure. The example Long Short-Term-Memory (LSTM) model is also implemented here. Then, the example model is tested on the simulated error data, and relevant results are presented. The section concludes with potential improvements in the modeling API.

8. **Web application for anomaly detection**:

   This is the second part of the case study. Similarly, this section begins with the concept and methodology used to implement the full-stack web application. Then, the implemented Flask backend and React frontend are discussed separately. For the backend, the Flask application file is discussed. For the React frontend, each developed component and its functions are discussed. Next, the results of uploading and testing the LSTM model on the web application are given, before the section is concluded by discussing potential improvements to the web application.

9. **Discussion**:

   Since the results of the anomaly detection framework are discussed in their respective sections, the discussion revolves around how the case study brings value to the advancement of an R/V Gunnerus digital twin.

10. **Conclusion**:

    Conclude the report content and results.

11. **Further work**:

    Recommend further work on the topics explored in the report.

After showing the references used, some additional material is given in the following appendices:

A **User manuals for anomaly detection framework**:

Necessary user manuals for the anomaly detection framework, including how to

- get access to the network drive containing vessel data,
- install Python, use the `pip` package installer, and use virtual environments,
- how to install the modeling API on a local machine, and
- how to set up a production environment of the web application.

get access to the network drive containing data, using , how to

B **Cloud computing services**:

Extract from the pre-project carried out in the fall of 2019 discussing cloud computing services, which is relevant for the discussion of the proposed digital twin infrastructure.

C **Data ecosystems**:

Extract from the pre-project discussing data ecosystems, which is relevant for the discussion of the proposed digital twin infrastructure.

D **Supplementary code**:

Includes Flask application file and algorithm for removing neighboring outliers in the modeling API.

E **Prediction results for second exhaust temperature signal**:

Provides plots of prediction results for the second exhaust temperature signal not included in the results section of the modeling API in Section 7.3.

F **Additional web application results**:

Provides snapshots of web application functionality not included in the React frontend description in Section 8.3.

2 Background

It is essential to understand the importance of digital twins and why digital twins are likely to become much-used resources for both industry and academia. As the Internet of Things (IoT) has become a household term, and data has become a commodity for a wide range of applications – both industrial and domestic – digital twins fall into the category of data-driven tools. With the increasing availability and resolution of data, both in a historical and real-time context, a digital twin enables a wide variety of use-cases based on analytics and data science. A digital twin could potentially follow an asset throughout its lifecycle, enhancing asset performance, functionality, and risk awareness in all phases of operation.

In addition to having value for industry and academic research within many fields, which is the conventional approach to digital twins, an R/V Gunnerus digital twin can be used for engineering education as well. As a digital tool, an R/V Gunnerus digital twin can help prepare students for the digitalized industry of the future, and provide a more practical approach to theoretical concepts through real-world applications and visualization of systems and operations.

This section seeks to give an overview of digital twin definitions and relate different definitions to the specific case of an R/V Gunnerus digital twin through literary review. As such, it is necessary to briefly present R/V Gunnerus and previous initiatives related to an R/V Gunnerus digital twin. Additionally, the concept of a digital twin infrastructure for R/V Gunnerus is explored. The proposed infrastructure is viewed in light of the digital twin use-cases and subsequent case study following this section.

2.1 R/V Gunnerus

R/V Gunnerus is a multi-purpose research vessel owned and operated by NTNU, contributing to research in the fields of biology, technology, geology, archaeology, oceanography, and fisheries research. The vessel was originally delivered in 2006 with a length overall of 31.25 m and a beam of 9.60 m (NTNU, 2006). The research vessel is seen in Figure 2.1. During the spring of 2019, the midship section was elongated[1]

[1]The elongation was managed by Polarkonsult, also responsible for the complete vessel design.

by 5 m. The propulsion system is diesel-electric with three generators, originally consisting of two fixed pitch, variable speed propellers, and a bow tunnel thruster. The vessel was retrofitted in 2015 when two azimuthing permanent magnet thrusters (AZ-PM) developed by former Rolls-Royce Commercial Marine[2] were installed.

Figure 2.1: R/V Gunnerus at sea. Courtesy of Fredrik Skoglund.

Currently, research is carried out on R/V Gunnerus through expeditions, implying that the vessel's value is extracted through in-operation activities. Since R/V Gunnerus is a physical entity, the number of concurrent tasks is strictly limited – the vessel can only be at one location at a time. By establishing a digital twin of the vessel, the application space increases significantly through data-driven analytics, simulations, and operation insight. A digital twin can be used for monitoring, diagnostics, and prognostics (Alam and El Saddik, 2017; Zhang, 2019), in essence, dealing with situations related to the present, past, and future of the vessel.

2.2 Previous Initiatives

Through initiatives at NTNU in Trondheim and Ålesund, some progress towards establishing an R/V Gunnerus digital twin has already been made. This section is intended to summarize these previous initiatives. Many of the initiatives described here are mentioned in Asbjørnslett et al. (2019).

2.2.1 Ship Technology Platform

Before the vessel was retrofitted in 2015, sea trials were conducted. The purpose of the sea trials was to document speed and maneuverability and generate data for comparing full-scale measurements with measurements from model tests and simulations (Selvik et al., 2015). In 2016, a full-scale test of dynamic positioning (DP) algorithms was carried out on R/V Gunnerus by Skjetne et al. (2017).

The seakeeping data from the sea trials were later used for sea state estimation by both Nielsen et al. (2018) and Brodtkorb et al. (2018). The effect of changing the propulsion system to the AZ-PM thrusters was investigated in the Virtual Prototyping of Maritime Systems and Operations (ViProMa) project by Skjong et al. (2018).

[2]Rolls-Royce Commercial Marine (CM) was acquired by Kongsberg Maritime (KM) in 2019. The integrated service branch is now known as KM CM (Kongsberg Maritime, 2019).

During this project, Coral was developed, which is an academic co-simulation software based on the Functional Mock-up Interface (FMI) standard. Similarly, Hatledal et al. (2018) at NTNU Ålesund developed the software package *FMI4J*, enabling co-simulation based on the FMI standard on a Java Virtual Machine. Many of the systems onboard R/V Gunnerus have been modeled in MATLAB's Simulink, such as the AZ-PM thrusters, hydrodynamic models, and DP algorithms (Zhang, 2019).

2.2.2 Student Activity

An Undergraduate Research Opportunities Program (UROP) was launched at the Department of Marine Technology in the spring of 2018, where master students performed preliminary work towards a digital twin of R/V Gunnerus. A prototype development carried out during the summer of 2018 yielded two important contributions towards an R/V Gunnerus digital twin, namely geometrical 3D models of the ship and structure – created in Siemens NX – and progress towards standardizing vessel signals. The standardization was carried out according to ISO 19848 Annex C through the naming rules of DNV GL–VIS (DNV GL, 2018), where VIS is DNV GL's Vessel Information Structure. The naming rules are compatible with DNV GL's product model (PMod) for describing vessels. The project also utilized some aspects of DNV GL's data platform Veracity, mostly related to storage, hosting, and integration of applications. Sesam Insight[3] was used as a digital twin viewer, integrating the visual models, PMod for R/V Gunnerus, signal data and metadata, and component attributes in Siemens Active Workspace.

(a) Generator set. **(b)** Geometric model. **(c)** Structure model.

Figure 2.2: Visualization of geometric models, taken as screenshots in DNV GL's Sesam Insight.

2.3 Access to Vessel Data

The AZ-PM system has been logging data since installation in 2015 through the KM CM products Ship Intelligence and Equipment Health Monitoring (EHM)[4] (Oksavik, 2019). Some of the services have been less reliable, but most of the data from the thrusters are available through local storage space. During the fall of 2019, a 4G modem was installed on the vessel to transmit data packages from the vessel continuously (Holmeset, 2019). The modem provides a ship-to-cloud connection to an Azure data lake[5] administered by Kongsberg through the Acon Automation

[3] Sesam Insight is a web-based application developed by DNV GL, intended to improve communication and information flow in offshore classification projects (DNV GL, 2019*a*).

[4] Ship Intelligence and EHM was formerly known as Health and Monitoring System (HeMoS), which changed in the mid 2010s (Oksavik, 2019).

[5] A data lake is used to store raw, unstructured data. This differs from databases and data warehouses, which require structured data.

System (Holmeset, 2019; Oksavik, 2019).

Currently, 19 packages related to the AZ-PM logging system are transmitting data. These packages are connected to individual systems or sub-systems, providing information from the thrusters and DP system, as well as some data from weather and electric switchboard elements. For example, the packages provide Seapath data from the thrusters, motion reference units (MRUs), and GPS. Also, one of the packages transmit data from each of the three Scania diesel generators through Scania's *EMS* control system for fuel provision and monitoring (Scania, 2011). Currently, the frequency of each signal is 1 Hz and is divided into comma-separated value (CSV) files with 10-minute intervals. The data is uploaded continuously to the Azure data lake through the modem once per hour. The data is synchronous, where each signal is collected with the same timestamp, removing the need for interpolating values.

During the spring of 2020, a network drive was set up by NTNU Ålesund, which receives data from the Azure data lake (Holmeset, 2019). Thus, the network drive is updated at the same frequency as the Azure data lake. However, the packages are transmitted somewhat at random, which makes it difficult to explore any close to real-time applications considering both transmission latency and irregularity. This network drive is accessible for all employees and students at NTNU. Access to the network drive can be achieved by following the user manual given in Appendix A.1. After connecting to the network drive, the different packages transmitted from the vessel are available as the separate directories shown in Figure 2.3.

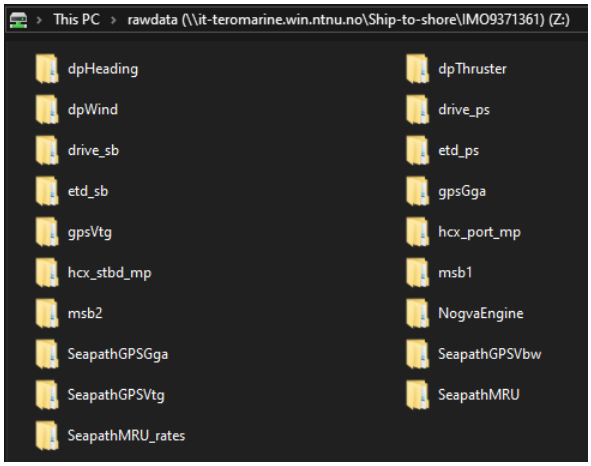


Figure 2.3: Network drive directories representing the different data packages transmitted from R/V Gunnerus.

The network drive will be the source of data used for the case study in this project. Expressly, the use of data is limited to the Scania diesel generators.

2.4 Digital Twin Definition

Its name can perceive the basic concept of a digital twin; through a virtual realization, a digital twin is intended to replicate and complement a real, physical asset as closely as possible. However, having different types of digital twins with different domains of application has resulted in a surplus of definitions that vary considerably. The following section explores some relevant definitions and their implications, which will serve as the foundation for formulating the R/V Gunnerus digital twin infrastructure

in Section 3.

The term Digital Twin was introduced in 2002 as a "conceptual ideal for [product lifecycle management (PLM)]" by Michael Grieves, and is defined by Grieves and Vickers (2017) as "a set of virtual information constructs that fully describes a potential or actual physical manufactured product ...". The definition provides a general understanding of the digital twin concept, highlighting the connection between physical space and virtual space. The definition of Grieves and Vickers (2017) is motivated by the PLM aspect, suggesting that a digital twin should be connected to the physical system throughout its lifecycle. At all stages[6], the digital twin will mirror a real, physical asset. A general product lifecycle is seen in Figure 2.4, together with a common lifecycle representation for marine vessels. The individual phases of a lifecycle present different needs and challenges, indicating that a complete digital twin must adapt dynamically to the product lifecycle. Due to the fragmentation of the maritime industries, different value chain elements affecting different parts of the lifecycle must also adapt to the introduction of digital twins (Os, 2018). This will require a paradigm shift in the collaboration between different participants, such as the shipyard, owner, and class societies performing approval and inspection. Introducing a digital twin lifecycle of its own can increase the understanding of how a digital twin can support the PLM, and such a lifecycle is proposed in Section 4.1 from an academic perspective.

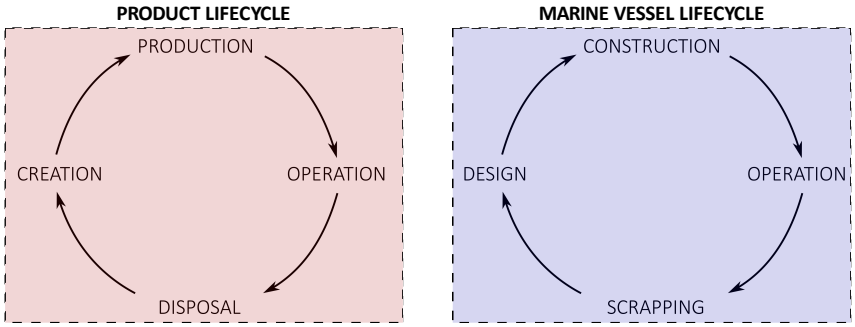


Figure 2.4: Phases of a product lifecycle, according to Grieves and Vickers (2017), and corresponding elements of a marine vessel lifecycle.

Through a simulation-based system engineering approach by NASA in 2010, the first formal definition of a digital twin was to provide "an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin. It is ultra-realistic and may consider one or more important and interdependent vehicle systems" (Shafto et al., 2010). This definition mentions more specific attributes of a digital twin than the definition of Grieves and Vickers (2017), although it is specialized towards aeronautical applications. It is common to use the term digital twin for advanced modeling and simulation of a system, which marks a conceptual misunderstanding of what a digital twin represents – a digital model representation with simulation capabilities is not sufficient to constitute a digital twin (Cabos and Rostock, 2018; Zhang, 2019). To represent a digital twin, the simulations must be

[6]In the creation phase, a digital twin may be a precursor to its real, physical counterpart, providing valuable information for the production phase, when the physical asset is realized, providing the standard for the virtual asset.

able to replicate the exact behavior of the physical asset, in contrast to only using synthetic data.

In different engineering fields, the desired properties and limitations of a concept are related to its intended purpose and functionality. For digital twins, this implies that the properties of, for instance, a digital twin of a research vessel varies substantially from a commercial process plant digital twin. These differences relate to the intended purpose of the digital twin and the corresponding functions needed to fulfill this purpose. As developing digital twins can be both expensive and time-consuming, the digital twin must add value to the physical asset to justify investment costs. Thus, for digital services – such as a digital twin – there should be preexisting, quantifiable utilities and benefits of these services. Such digital services may be embodied by a digital twin, which have inherently different purposes and functions based on the area of application, resources, competencies, and intended degree of realism (Cabos and Rostock, 2018). A way of characterizing purposes and related functions can be achieved by looking at the needs space and solution space of digital service development (Erikstad, 2019), illustrated in Figure 2.5. The asset control levels forming the foundation of the needs space can be categorized as strategic, tactical, and operational (Macchi et al., 2018).

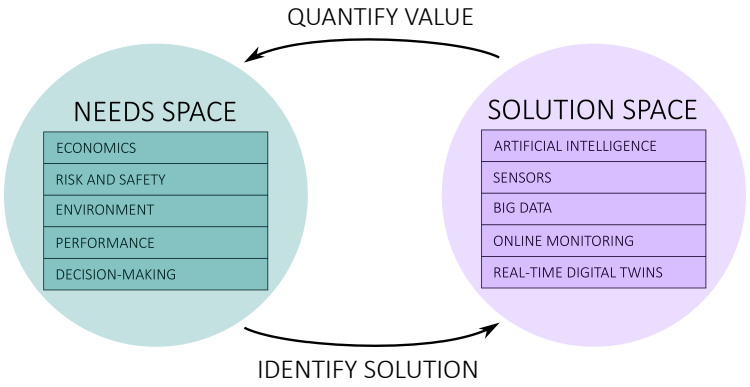


Figure 2.5: Digital service needs space and solution space and their interactions, inspired by Erikstad (2019), where the proposed needs parameters are typical for maritime applications.

As highlighted by Erikstad (2019), large quantities of data from numerous sensors may be collected by vendors without knowing how to utilize the data, essentially making data collection and storage a trivial activity. Therefore, it is necessary to identify the needs space and evaluate these needs against available solutions. By developing a tangible needs space for a known area of application, it is possible to identify potential solutions that can be achieved through digital services. Consecutively, the value of these solutions should be quantified to decide whether to implement them or not. The needs space can consist of unresolved or unconsidered problems that become resolvable through digital services or optimizable problems that can be improved through digital services. The same process can be carried out for digital twins to prevent unnecessary investment, such as creating an ultra-realistic model – following the principles of Shafto et al. (2010) – in a situation where the same results could be accomplished with a less detailed model.

In addition to the introduction of needs space and solution space for digital services,

Erikstad (2019) explores the temporal aspects of service scope and their interaction by emphasizing information in hindsight and foresight. Figure 2.6 illustrates how temporal aspects can be used to gain insight for decision-support. The temporal component is important for distinguishing digital twins from digital models and model simulations. This is emphasized in Erikstad (2018), where a digital twin is defined as "a digital model capable of rendering state and behavior of a unique real asset in (close to) real-time", consisting of the five intrinsic characteristics of identity, representation, state, behavior, and context.

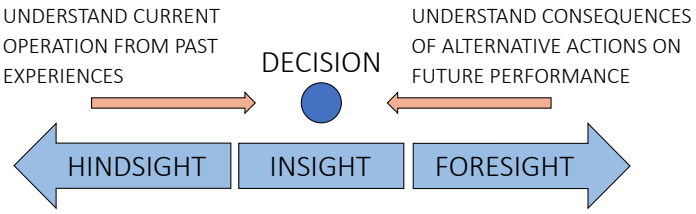


Figure 2.6: Relating hindsight and foresight information to decision-support, courtesy of Erikstad (2019).

The concept displays how decision-making can be affected through insight into past experiences and knowledge of the consequences of different future actions. This principle is especially valuable for in-operation purposes, as it enables adding value to the operational context of an asset through somewhat real-time information feedback and feedforward. Cyber objects analyzing and learning from measured data before feeding it back to real systems are known as Cyber-Physical Systems (CPS) (Alam and El Saddik, 2017). To function properly, the temporal latency for CPS would have to be sufficiently low. Bridging the gap between real-time applications and delay-tolerant services is an important aspect of CPS, attempted to be solved through digital twins (Alam and El Saddik, 2017). The concept of extracting decision-making insight during operation is the topic of the case study, where anomaly detection for predictive maintenance will be implemented in a real-time environment.

2.4.1 A property-driven approach to digital twins

The definitions and considerations discussed in this section contribute to the concept of digital twins to some extent. Since the purpose of a digital twin can be multi-faceted based on the area of application, it is suggested that a digital twin should be defined based on its properties rather than its intended purpose. From the presented material, the following four properties are considered necessary for complete digital twins:

1. Asset representation throughout the lifecycle.

   A real, physical asset exists or will come into existence[7], and has a mirrored digital representation made to a sufficient degree of realism, satisfying the purpose of the digital twin.

2. Simulation environments capable of using synthetic and measured data.

   In addition to simulating with synthetic data, the digital twin must be capable of applying and utilizing data from the vessel in simulations.

[7] When a digital twin is used in the conception phase of a lifecycle, the physical asset may be realized at a later stage.

3. Added value through virtual representation.

   To justify investments, the digital twin must add tangible value to the real asset. The value of the twin should also be evaluated periodically, updating functionality and methodology based on experiences to best reflect the intended purpose of the twin throughout its lifecycle.

4. Insight and decision-support through past knowledge and future predictions.

   Through past and future learning with sufficiently low latency, the established CPS is able to provide feedback based on past experiences or intervene based on event prediction.

The four properties encapsulate the digital twin definitions explored through a property-driven approach. They will be necessary for formalizing the R/V Gunnerus digital twin infrastructure in Section 3.

3 Digital Twin Infrastructure for R/V Gunnerus

Based on the discussion of definitions and a property-driven approach in Section 2.4, it is possible to formalize the concept of a digital twin infrastructure (DTI) for R/V Gunnerus. By definition[8], an infrastructure represents an underlying foundation or basic framework for a system or organization. As such, an R/V Gunnerus DTI should facilitate all of the intended purposes of an R/V Gunnerus digital twin. Since a complete twin will serve a wide variety of purposes based on discipline and academic interests, the underlying infrastructure should represent an enabler for sub-functionality and applications. Initially, a sufficiently functioning subset of functions may be supported with the flexibility to grow at later stages.

By basing a digital twin on an underlying infrastructure, it is possible to add functionality without compromising the integrity of the twin. If changes are made to components of the infrastructure, the changes should be compatible with the structural composition of the infrastructure to maintain functionality. This requires a thorough description of how the digital twin is structured, optimally providing a set of guidelines and rules for unobtrusively modifying the twin. Further, as seen in literature, digital twin definitions are often generalized, making concepts seem abstract and hard to realize. Establishing a DTI is intended to provide a concrete structure and set of requirements. Such a structure is useful for both the initial development and the implementation of new functionality at later stages.

3.1 Digital Twin Properties for R/V Gunnerus

The discussion of digital twin definitions in Section 2.4 was concluded with a property-driven approach to digital twins, presented in Section 2.4.1. The properties indicate that digital twins are individualized based on different purposes and desired functions. Before presenting a DTI for R/V Gunnerus, the intrinsic properties of digital twins are evaluated against the specific case of R/V Gunnerus. Specifically, properties revolving the lifecycle, added value, and data transmission latency.

From a high-level perspective, a digital twin of R/V Gunnerus serves two primary purposes for NTNU, namely contributing to academic research, and providing a learning platform for engineering students. These purposes are contained within the operating phase of the vessel lifecycle provided in Figure 2.4, indicating that the

[8] According to the *Merriam-Webster* dictionary.

R/V Gunnerus digital twin is less dependent on lifecycle adaptations to phases other than the operational phase. However, if the vessel receives substantial modifications affecting operation, a digital twin must subsequently adapt to preserve the true virtual asset representation. The installation of new thrusters in 2015 and the midship elongation in 2019 are examples of such changes. For commercial interests of individual components or systems, such as the AZ-PM thrusters of KM CM, a digital twin will serve different purposes. In a longer perspective, where R/V Gunnerus may potentially be replaced by a new research vessel, the lifecycle aspects of a digital twin become relevant. An approach to the digital twin lifecycle is explored in Section 4.1.

Since R/V Gunnerus is a sophisticated vessel with several sub-systems of interest – such as the DP system, propulsion system, and power management system – there is a different potential for educational benefits in a wide range of engineering disciplines. The vast area of application demands a corresponding digital representation that supports a variety of purposes and functions. The same elements found in the needs space of digital services in Figure 2.5 are relevant for a DTI for R/V Gunnerus. However, the focus is shifted from maximizing commercial profit to improving knowledge and insight through academic research and education.

Reduced data transmission latency can improve the overall performance of a digital twin, and increase the appeal of using the twin. By enabling a close to real-time insight into the vessel's operation, the possibilities of subsequent testing, verification, and validation after various analyses through the connection between the digital twin and real asset. By decreasing latency, it is also possible to use the twin for in-operation purposes through enabling performance evaluation and decision-support. The case study presented in Section 5 attempts to fuse topics which may be valuable from both an educational and commercial operation perspective.

3.2 Infrastructure Proposition

A design structure for digital twins was proposed by Erikstad (2018), which includes structural, creational, insightful, and computational patterns. These patterns are derived from software system implementation patterns presented by Gamma et al. (1995), with changes made to fit the case of digital twins. Furthermore, a set of relevant architecture aspects include runtime environment, structure and content, integration, and API and usage (Malakuti and Grüner, 2018). It is deemed that a DTI should be closely linked to this software and system architecture.

With a software and system architecture, in addition to the implications discussed in Section 2.4 and in this section, a DTI for R/V Gunnerus is proposed with three main components, namely data management, modeling and simulation environment, and software and system realization. A visual representation of these components is given in Figure 3.1. Different functions intended to be facilitated by each of the three main components are also given, providing a complete picture of the DTI functionality and purpose.

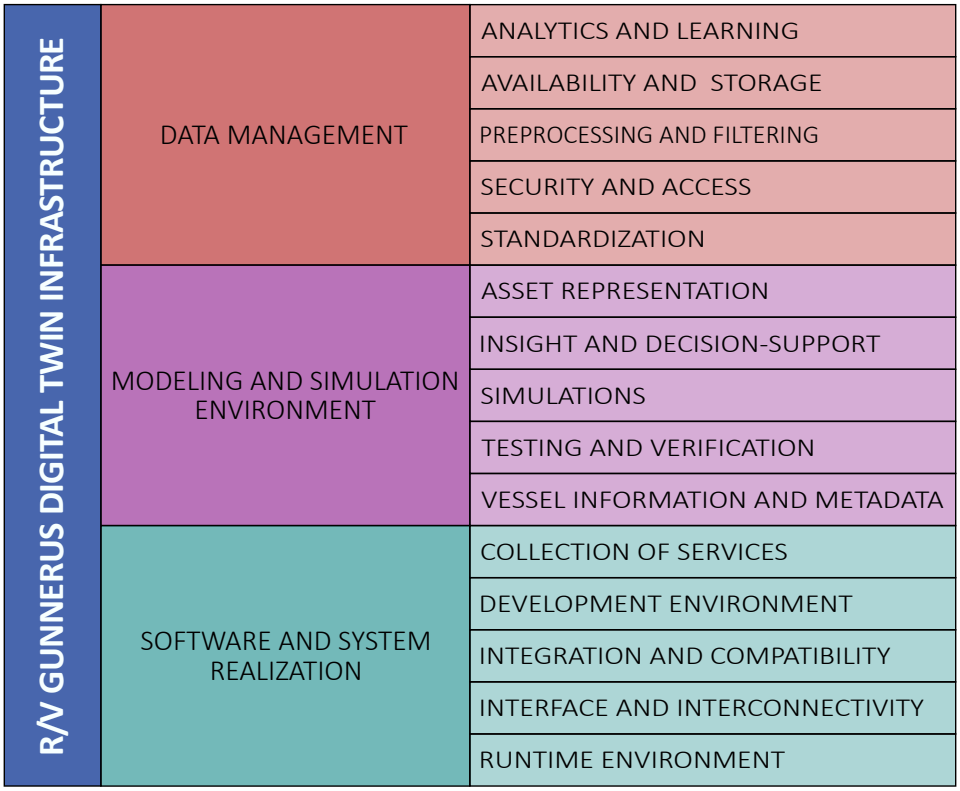


Figure 3.1: Main components of a proposed digital twin infrastructure for R/V Gunnerus. Different functions facilitated by each of the three main components are also given.

The different components are explored in the following sections. Since the case study in this report is most concerned with the data management aspect of the DTI, the data management section will be the most detailed.

3.3 Data Management

In this report, data management for digital twins is related to all aspects of managing data as a resource. This entails retrieving, storing, accessing, transforming, and utilizing data in an efficient, secure, and cost-effective manner. Most of the relevant data will come from sources on the real asset, and the data management system is the mediator between signal data from the real asset and the virtual asset, enabling data utilization for added value. Data quality control is an essential part of data management, and as seen in Figure 3.2, the amount of useful data will decrease as the data management operations progress.

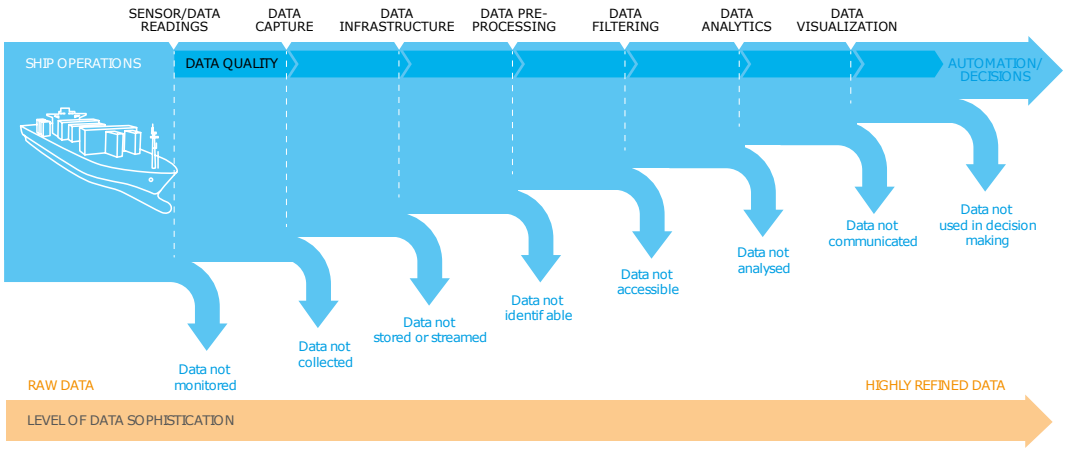


Figure 3.2: Data sophistication progression for ship operations, based on Låg and With (2017).

The facilitated functions proposed in the DTI for R/V Gunnerus in Figure 3.1 highlight essential aspects of a data management system, and will be used to evaluate the data management perspectives for a digital twin of R/V Gunnerus. Based on the facilitated functions for data management, the co-dependent pyramid structure is seen in Figure 3.3 is suggested. The pyramid follows a best-practice hierarchical structure where lower-level functions enable those of higher levels. Similarities can be seen between the structure and the data sophistication progression for ship operations.

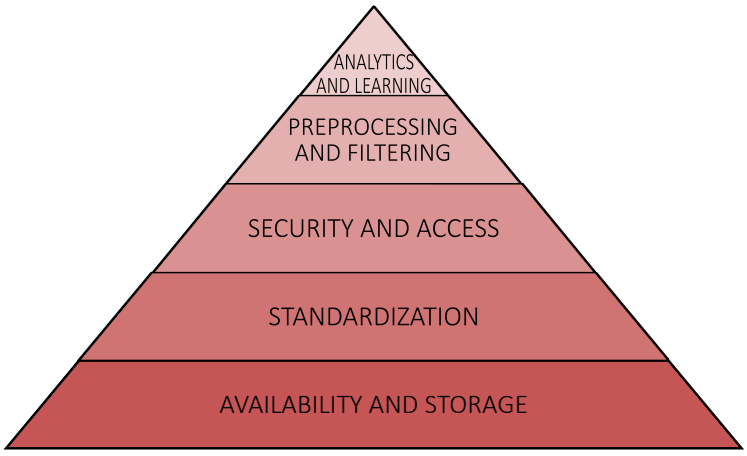


Figure 3.3: Recommended data management functions hierarchy. The hierarchy is based on requirements and complexity of facilitated functions in the data management component of a DTI for R/V Gunnerus.

The following sections will discuss some aspects related to the data management functions for the use-case of an R/V Gunnerus digital twin.

3.3.1 Availability and data storage

The first obstacle related to data management is to gain access to data of interest and store it at an appropriate location. Availability implies connecting information from the physical asset to the virtual asset. In theory, there is infinite information

connected to the operation of a vessel. Thus, there is a need to weigh the expected data value against cost, implementation, and ease of extraction. For existing signals, costs are related to network transmission, storage, and maintenance. This is associated with the discussion of data abundance by Erikstad (2019), as there should be proper reasoning for installing sensors and extracting data from them. For a digital twin of R/V Gunnerus, considerations should be made based on the added educational value, either related to research or education.

Digital twins will typically have different users at different locations, suggesting data can be requested at multiple locations simultaneously. The most prominent solution for data storage is to use cloud computing services. In brief, cloud computing services offer on-demand network access to shared computing resources (Mell and Grance, 2011). The topic of cloud computing services, together with cloud service providers and significant data considerations, was explored in more detail in the pre-project, as seen in Appendix B. With cloud services, it is possible to establish a ship-to-cloud solution, circumventing extra transmission layers by having a direct connection to the ship. With a ship-to-cloud connection, the vessel can transport data in-operation, improving data readiness, and enabling CPS feedback for decision-support. In addition to data transportation costs, a drawback of direct ship-to-cloud transmission is the possible loss of signal, which is undesired if the data is necessary for decision-support. Data transportation can be regarded as the bottleneck of centralized clouds (Shi et al., 2016), which becomes an argument for preventing a ship-to-shore intermediary node. A ship-to-shore node would be the receiver and transmitter of data and information both to and from the vessel and to and from a centralized cloud, which is demanding for data transportation. If a ship-to-cloud connection to a centralized cloud is deemed too slow, it may be advantageous to create a ship-to-edge connection by establishing an edge node. Edge nodes are part of edge computing technology, which is detailed in Appendix B.3. Decision-support is only applicable to established and tested digital twins since the real asset will apply feedback from the CPS into operation. Therefore, the CPS aspect and real-time data requirements will not be of immediate use for the case of R/V Gunnerus, meaning that a centralized ship-to-cloud connection could likely be a sufficient starting point.

If the vessel produces large amounts of data, considerations regarding the challenges of big data may have to be considered. In Appendix B.4, five *Vs* characterizing big data was presented; *Volume, Velocity, Variety, Veracity, and Value*, hereby referred to as V^5. Based on the current state of data packages transmitted from the vessel, as presented in Section 2.3, issues of V^5 are not yet problematic. This can change in the future if more data becomes available for extraction – potentially at higher frequencies – and more historical data is accumulated. In addition to being a best-practice approach, data storage should be optimized to save space and prevent excess data. As an example, unusable data is generated when the vessel is in harbor, but without any storage control, the data will be stored nonetheless. Different data points will also be updated with different frequencies. For instance, data from the DP system and motion reference units (MRUs) must be updated at a significantly higher frequency than the number of available satellites. As highlighted by Tjøswold (2012), a frequency of 1 Hz for the DP system – which is the current data transmission frequency – is too seldom. If all signals are measured at the highest signal frequency to ensure no loss of information, the data points that change rarely will occupy more space than necessary. This could be prevented by having different solutions

for different packages, as well as sub-dividing packages taking update frequency into account. A drawback of having non-uniform logging frequencies is the increased difficulty of combining data for analytic purposes due to size mismatching.

Optimally, storage control measures should be carried out on the vessel before being transmitted to the cloud to limit data velocity and volume challenges encountered in V^5. Instead, if these measures are carried out in the cloud, they would still limit the data volume challenges from V^5. Storage costs could also be reduced by considering different storage readiness tiers, commonly divided into hot, warm, cool, and cold storage (Taylor, 2017). Costs increase based on desired access frequency and storage system performance. On-demand data, such as data used for CPS feedback, needs to be in hot storage, whereas warm storage could suffice for slightly aging, historical data. Cool and cold storage may be a durable way of storing results from archived projects. If it does not affect performance, data should be moved towards colder storage to reduce costs and keep processing power available for more urgent activities.

3.3.2 Standardization

Standardization is important for information transparency and data structuring. For signal data, standardization can be used to create unique identifiers. These identifiers make it easy to exchange and apply information, prevent misunderstandings, and reproduce methods and results. As mentioned in Section 2.2.2, some progress has been made using the DNV GL–VIS naming scheme in Annex C of ISO 19848[9], which takes DNV GL's product model for vessels (PMod) into account. Through the naming scheme, each onboard signal can be given a universal identifier providing relevant information about sensor type, location, configuration and capabilities, quantity, and unit of measure (Låg and With, 2017). Each signal is also given a unique label reference.

3.3.3 Security and access

As mentioned in Section 2.3, Kongsberg has a logging system sustaining data from Scania's EMS control system, AZ-PM system, and DP system. The data from these systems are transmitted through a ship-to-cloud solution. There are other systems of interest, such as the bow tunnel thruster from Brunvoll and the hydraulic Palfinger crane. If vendors grant access, signals from these systems could potentially be included in the logging system through additional packages by re-routing[10] the signal (Holmeset, 2019).

However, gaining access to data from external vendors is not trivial. Although NTNU has ownership of the research vessel itself, it is often the system and component providers who own the data from different onboard systems. Access to component and system data can reveal valuable information in commercial markets, which makes vendors reluctant to releasing sensitive data. This is also relevant when considering the AZ-PM thrusters and DP system. According to Oksavik (2019), it is difficult enough to manage access internally, let alone providing external users access.

Considerations regarding access to data, cybersecurity, and security for vendors, should be documented in a data management plan. Such a plan would promote

[9] ISO 19848 is a sensor naming structure for ship and shipboard machinery and equipment (Låg and With, 2017).

[10] The re-routing would have to be done manually aboard the vessel, and the signals would have to follow a compatible messaging protocol.

transparency for the different parts involved by documenting what data is desired, how it will be used, and by whom, how access will be managed, and restrictions imposed by vendors. A data management plan would also help to share how different users intend to make use of the data, which could prevent overlap in research activities.

3.3.4 Preprocessing and filtering

Applying preprocessing and filtering algorithms is necessary to detect, remove, or adjust faulted and undesired data. For the preprocessing aspect, computer algorithms are used to transform, reduce, and prepare the data. It can also be a tool for verifying data formats and expected values. Examples of common challenges related to data signals can be seen in Figure 3.4. For the filtering aspects, the goal is to extract temporal or spatial data of interest. For instance, data transmitted from the vessel in harbor is most often not of interest and should be filtered accordingly. An intuitive approach would be to divide and label operational data into expeditions, where a single expedition or a set of expeditions can be filtered out of the entire data set. As mentioned in Section 2.3 regarding vessel data access, the data is transmitted through a ship-to-cloud connection directly to a data lake, which in principle contains unstructured data. Some preprocessing measures have already been applied to the data before transmission, such as replacing non-applicable data and interpolating empty rows (Holmeset, 2019). With these inherent preprocessing measures, it becomes easier to handle data for the case study.

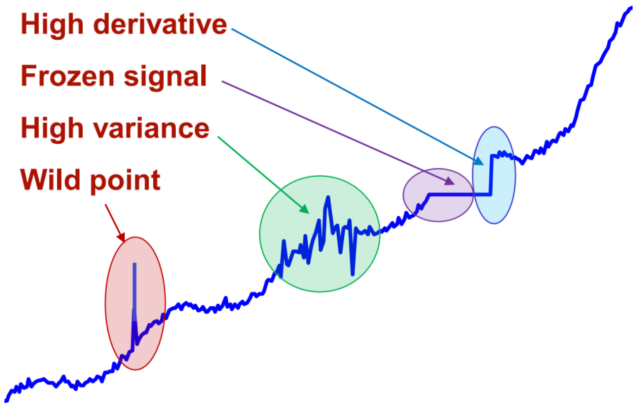


Figure 3.4: Common challenges related to signal data, courtesy of Sørensen (2018).

3.3.5 Analytics and learning

When the preceding data management procedures have been completed, the sophistication is sufficient to utilize the data. This is represented in Figure 3.5, where analytics and learning leads to insight.

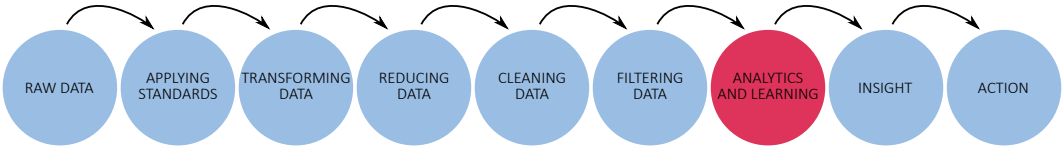


Figure 3.5: Procedural chain from raw data, to analytics and learning, to insight and action.

Data utilization can be related to the hindsight and foresight aspect from Figure 2.6, where insight into the operation of the vessel can be gained through data-driven methods. The insight can be used to improve performance or provide decision-support and is enabled through methods for analyzing and learning from the data. In addition to statistical and empirical ways of analyzing data – for instance, evaluating fuel efficiency during different operational modes – machine learning approaches can be applied to develop data-driven solutions. A data-driven approach to predictive maintenance will be a core topic of the case study.

3.3.6 Modeling and simulation environment

Digital twins are driven by modeling and simulations aspect. The modeling is most often physics-based, where real assets are modeled to a degree of fidelity, either limited by the knowledge of the system or complexity. In the proposed DTI for R/V Gunnerus in Section 3.2, the modeling and simulation environment facilitates

- asset representation,
- insight and decision-support,
- simulations,
- testing and verification, and
- vessel information and metadata.

As a digital twin is the digital representation of a physical asset, the representation should be as close to the real asset as necessary. Thus, the asset representation should encapsulate the physical realization in a virtual format. This would include geometrical models, mathematical models, and physics-based models. In essence, the asset representation includes all of the physical properties necessary to reproduce the real asset. Together with the vessel information and metadata, the asset representation will also be able to describe all parts of the real asset. As mentioned by Erikstad (2017), the models do not necessarily have to be physics-based, as they can also be based on artificial intelligence and machine learning algorithms, promoting data-driven models.

Metadata[11] is data that provides information about other data. As an example, the attributes of vessel components consist of different materials, have different properties, and are instrumented with different signals and standards. By attaching metadata of components to the geometrical representation of the vessel, information can be extracted intuitively and at one distinct location. The geometrical 3D model of R/V Gunnerus from the UROP described in Section 2.2.2 was exported from Siemens NX through the STEP-format[12]. Vessel attributes were stored in Siemens through the PLM Active Workspace, which is disadvantageous when it comes to ease of access and convenience.

Insight and decision-support, which has been explored earlier, should be derived through the virtual model environment. Since the virtual environment is a replication of the real asset, it will – optimally – reproduce the state and behavior correctly. Thus, based on the response from model simulations, the outcome of a particular action can be predicted and applied as feedback or feedforward to the system.

[11] According to the Merriam-Webster dictionary.
[12] STEP: Standard for the Exchange of Product Data

Uncertainties in the modeling become a part of the risk assessment and should be evaluated continuously through increased experience and data material. This highlights the importance of testing and verification to ensure that the system operates correctly and safely.

Simulations intend to demonstrate the underlying principles controlling the behavior of a system. As mentioned in Section 2.4, simulations in digital twins should enable both simulating with synthetic data, but also actual data from the real asset to enable reproduction of the true state and behavior.

3.4 Co-Simulation

The maritime industry is a multi-vendor environment, where different vessel components can be supplied and administered by different parties. As mentioned earlier, many vendors are concerned with sensitive data and do not wish to share their models. In a co-simulation environment, simulation models can be divided into individual modules that can be loosely coupled through black boxes. Thus, the information about the models themselves will be confidential as the different modules only need inputs and outputs to communicate. Both Coral and FMI4J, mentioned in Section 2.2.1, are examples of co-simulation environments. These environments enable co-simulation through the FMI standard, which is a standard for model-exchange and co-simulation (Ludvigsen et al., 2016). The format converts models to C-code, which is likely faster at runtime than the original model language due to being a lower-level programming language. In addition to the model code, an *XML*-file with metadata is provided, as well as all necessary libraries created during the model exchange. Extensible Markup Language (XML) is used to encode data in a format that is both human- and machine-readable. Since the models are converted to C-code, no licenses are needed to run the model in the co-simulation.

3.5 Open Simulation Platform

The Coral software was continued through a joint industry project (JIP) where DNV GL, SINTEF, Kongsberg, and NTNU are founding partners and main developers. The JIP is working on the Open Simulation Platform (OSP), which is an open co-simulation environment intended to facilitate digital twins in the future. OSP consists of the Core Simulation Environment (CSE), which provides the model standard and some reference models. The co-simulation environment can either simulate using subsystems, choosing which modules to include, or execute more comprehensive simulations. The simulation advances at a universal time step, but there can also be individual time steps acting on each module since the mathematical computations are performed within the module. When advancing a global time step, inputs and outputs from all of the modules are exchanged. Co-simulation benefits from the ability of modeling systems as black boxes, in addition to providing specialized solvers, sub-simulator interfaces, loose coupling between subsystems, and distributed simulations (Kyllingstad, 2019). A disadvantage relates to poor performance when simulating tightly coupled systems.

3.6 Software and System Realization

The software and system realization aspect of the DTI for R/V Gunnerus proposed in Section 3.2 facilitates

- collection of services,
- development environment,

- integration and compatibility,
- interface and interconnectivity, and
- runtime environment.

The software and system realization does not add any new insight to the virtual asset. Instead, it is intended to handle all of the enabling factors that are necessary to realize, operate, and sustain the digital twin services. A schematic overview of the software and system realization aspects is seen in Figure 3.6, where a user marks the final destination of the system realization.

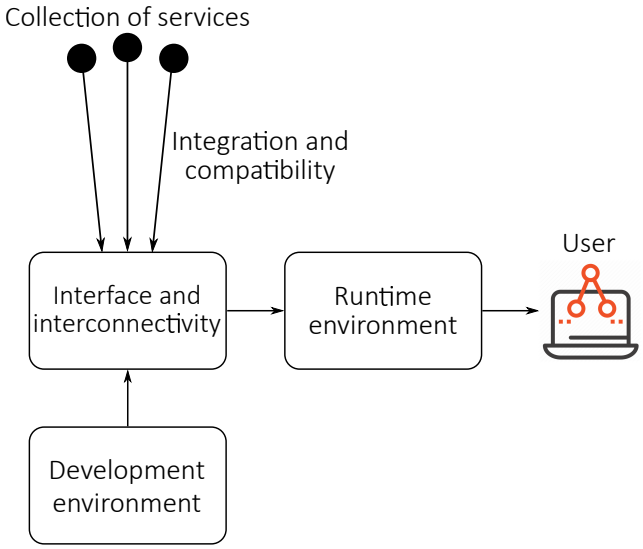


Figure 3.6: Software and system realization.

Collecting services is important for the coherency of a digital twin. If all of the different parts of a digital twin exist but are scattered between several different locations, service hosts, and restricted areas, the usability diminishes. Collecting services will also make the digital twin more appealing, as the virtual asset will be perceived as more authentic to the real asset if it is maintained within a tightly connected virtual space. Contrarily, if it is necessary to navigate between several different services to go from a geometrical component to its attribute metadata, using the digital twin becomes a timely and cumbersome task.

The integration aspect is responsible for the addition and sustainability of new services, whereas compatibility ensures that the services can be integrated into the system. Once integrated and compatible, the interface should provide a seamless transition between different components by enabling the exchange of information. There must be interconnectivity between communicating components. A parallel can be drawn from interconnectivity to the co-simulation aspect explored in Section 3.4, where different modules communicate through inputs and outputs.

A development environment facilitates software realization and is necessary to develop the system as a whole further. Source code, documentation, libraries, and APIs are contained within the development environment, which should only be changed by the system developers. Lastly, the runtime environment is responsible for managing

requests from users, ensuring that the services, system variables, and interconnected functions are executed correctly.

4 Digital Twins in Marine Engineering Education

As mentioned in the project motivation in Section 1.1, this thesis is motivated by the use of digital twins for educational purposes. The industry is incentivized to adapt to new technologies based on economic gains and competitive advantages. This incentive is not as present in the educational system, which can cause the curriculum to fall behind the technological progression. This can create a gap between the students' expectations after graduating and the reality they meet in the industry. In this case, companies would have to train new employees to meet their technological standards, which costs time and money. It is a university's responsibility to prepare the students for the working life, which underlines the importance of continuously updating the curriculum to correspond with the latest technologies.

Some of the engineering educations, especially for master students, have a theoretical approach to most courses. Although it is crucial to understand the underlining theory, the reality is not understood by theory alone. A practical understanding is necessary for the holistic perspective necessary to apply theory to real-world problems. Since solving problems is at the core of engineering, the value of a practical understanding should not be underestimated.

A tool that can help integrate new technologies into engineering education through a practical approach is digital twins. In theory, the potential of digital twins as an educational tool is immense. With the ability to exist as a virtual counterpart to any given asset, all problem definitions applied to an asset can consequently be applied to the virtual twin. However, instead of having to be at the same place as the asset to perform measurements, the digital twin will be available remotely. This opens up the possibilities of using assets that would otherwise be inaccessible. Take, for instance, the case of R/V Gunnerus. If the vessel is in operation, a digital twin will allow others to access the asset as if they were on the vessel. Besides, users from across the globe could connect to the same digital twin, promoting a global collaboration initiative of expensive or rare assets between different institutions. This would be scalable, as long as the digital twin platform can handle the traffic since there is no physical restriction to the number of connected users. Thus, the required interaction with the physical asset becomes less of a priority.

Assuming that a digital twin exists, there are several areas of application relevant for marine engineering education. The applications vary in complexity and can be as relevant for a first-year student as for a graduate student. Thus, a digital twin can follow a student through the entire education, providing new ways of utilizing the twin as the general engineering knowledge of the student increases. For first-year students, there are many new terms and concepts revolving ships and the marine industry in general. As humans are visual creatures, having an interactive digital twin available as a visual tool provides a different level of situational awareness. For second- and third-year students, the structural and hydrodynamic properties of a digital twin could be taken into account, where real asset behavior could be evaluated against simulated behavior, or be used to implement data-driven or model-driven algorithms for health monitoring, decision-support, control systems, and more. For fourth- and fifth-year students, the theoretical principles of the digital twin can

be explored to enhance the twin's properties, capabilities, and performance. New functionality could also be added to the digital twin to promote a student's research interests.

Although there are many possibilities of using a digital twin for marine engineering education – and engineering education in general – creating, managing, and distributing the digital twin is an immediate obstacle with the current technological status of digital twins. Digital twins are not readily available, and if they are available, they usually have some limitations. Currently, the enabling technology for digital twins is still being developed by industry and academic research. Ideally, students should be involved in this process to take part in – and contribute to – the technological advancement of digital twins. Here, the concept of a digital twin lifecycle can be helpful in identifying the different developmental and operational life phases related to a digital twin.

4.1 Digital Twin Lifecycle

When looking at the lifespan of a digital twin of a real asset, it is common to relate the digital twin lifecycle to its product or asset lifecycle counterpart. With this approach, the digital twin follows the same lifecycle phases as the asset does. As shown in Figure 2.4 from the discussion of digital twin definitions, the asset lifecycle is typically divided into four main parts, and, as shown, this asset lifecycle definition could similarly be applied to a marine vessel. However, as digital twins develop and become more prevalent in the design, operation, and decommissioning of an asset, it can be valuable to create a lifecycle model for the digital twin itself. In this section, the different lifecycle phases of a digital twin are explored to propose a complete lifecycle model for digital twins. The concept was developed for a presentation[13] held February 3, 2020, where different professors at the Institute of Marine Technology were invited to discuss the future of an R/V Gunnerus digital twin. The presentation and the following discussions were intended to formulate this master thesis.

The purpose of the life phases of a digital twin presented here is to give a broader perspective of the use-cases of digital twins, and how engineering students could be involved in other aspects of a digital twin that are not purely operational. Through conversations on the topic of digital twin life phases, it was decided that relating the digital twin to the development of humans would provide a comprehensible overview (Skjetne, 2020). The proposed digital twin lifecycle is given in fig. 4.1, along with the defining properties at a given phase.

[13] The presentation is available in PDF-format (Alvsaker, 2020*c*).

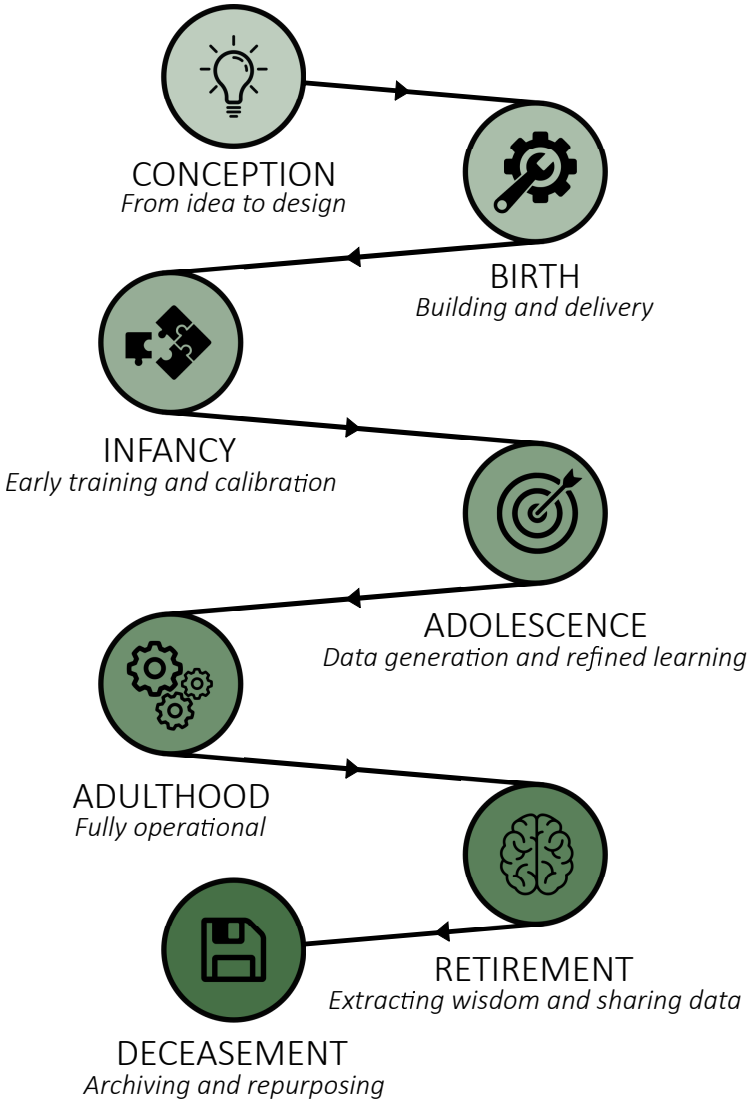


Figure 4.1: Lifecycle of digital twins with seven different phases from conception to deceasement.

The definition of the digital twin lifecycle can affect the use in engineering education as the application space is increased when compared to conventional approaches. Instead of just being concerned with the in-operation aspects of a digital twin, students can be involved in the development, testing, and implementation of a digital twin. Further, a digital twin does not have to be restricted to existing assets. In Figure 4.2, the lifecycle of a real asset is compared to the lifecycle of a digital twin, where the digital twin is conceived at the beginning of the asset's operational phase.

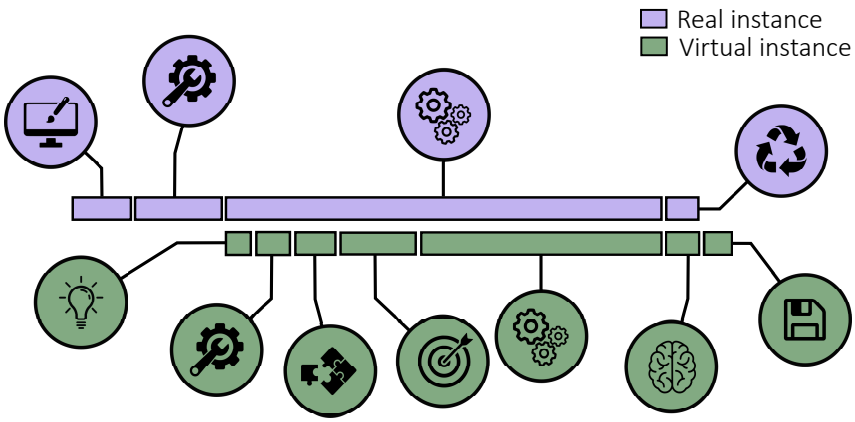


Figure 4.2: Lifecycle of a real asset and its digital twin, where the digital twin is conceived during the operational phase of the asset's lifecycle.

However, the real asset does not have to exist for a digital twin to be made. The asset could be developed in combination with the real asset, where the digital twin foundation is used as a resource for evaluating the asset during conception and construction. Digital twins can also be conceived before the asset and can help in the development and testing of said asset. In some ways, this relates to existing methods for modeling and analyzing assets in the creation phase. The major difference is that the digital twin is simultaneously developed to fulfill its purpose, which requires an unprecedented interaction between the coming asset and the digital twin. A digital twin is also likely to outlive an asset to some extent. Even if the adult phase is finished, it is possible to extract useful information from the retired digital twin for evaluation, which could lead to repurposing or recycling of the twin. An expansion of the digital twin lifecycle in relation to the asset lifecycle is given in Figure 4.3.

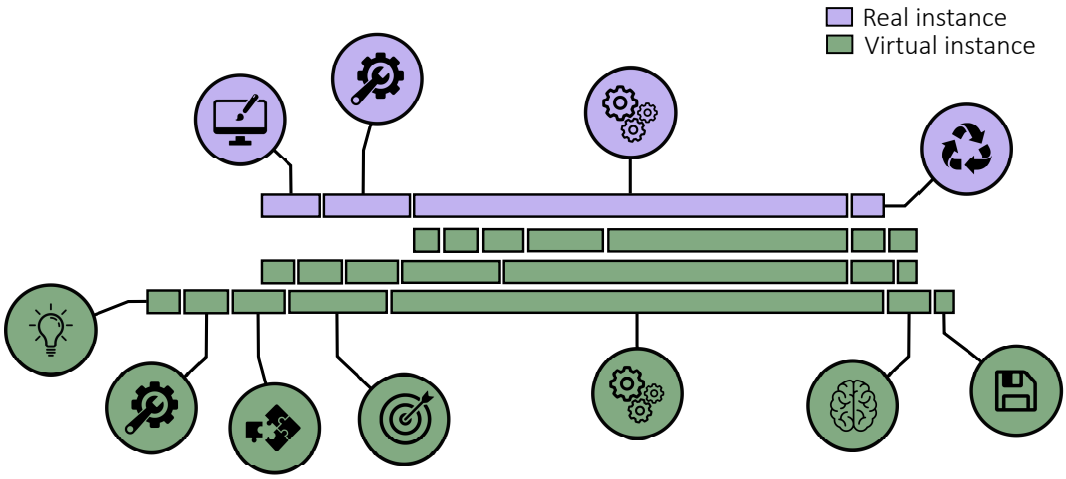


Figure 4.3: Lifecycle of a real asset and its digital twin, where the digital twin is conceived at three different stages of the asset lifecycle; during operation, conception, or predating the asset.

Expanding the lifecycle of the digital twin allows for a broader perspective regarding education. As mentioned in Section 1.1, it is beneficial for students to be part of projects developing new technology. As such, while digital twin technology is still

in its infancy, students should be included in other phases of the digital twin lifecycle than the adult phase. Through course activities, projects, and interest groups, students are encouraged to create algorithms, create simulation models, perform structural analyses, suggest data management plans, and in other ways, become familiarized with the concept and technologies revolving digital twins. This kind of student engagement is exemplified in this thesis, where the following case study is developed as part of a typical digital twin functionality, considering condition-based maintenance through data-driven methods.

5 Case Study Problem Formulation

As specified in the objectives in Section 1.2, the motivation of the case study is to familiarize students with machine learning algorithms through a digital twin approach. By allowing students to create, train, implement, and visualize their own machine learning models, theoretical knowledge can be applied to a practical case that may have actual implications on an operating vessel.

The case study is related to data management, which is one of the core infrastructure components. The purpose of the case study is to create a framework for real-time anomaly detection on systems onboard the vessel through a data-driven approach. Since the framework is made to be available to others, either for using the framework or developing it further, reusability, readability, and documentation are emphasized throughout the case study. The framework will include a modeling environment for developing data-driven models and a web application for implementing, testing, and visualizing the results of the anomaly detection model. As such, the case study is twofold, where one part revolves around a framework for modeling, and the other part revolves around a framework for model evaluation.

The purpose of the modeling framework is to allow engineering students to implement their own algorithms for testing and discussion. The modeling topic of choice, as will be discussed in Section 6.1, is predictive maintenance, which can be approached with data-driven machine learning algorithms. To be easy and quick to learn, a method of creating an application programming interface (API) will be used. An API is a set of reusable functions that can be used without having to learn the complete structure and methodology of an underlying application or operating system. For the case study, the API will consist of functions that can be used to speed up the process of achieving a functioning predictive maintenance model. These functions are related to memory management, file management, modeling, evaluating performance, and visualizing results. Although it is essential to understand the entire procedure from raw data to a predicted outcome, there are parts of the modeling procedure that can be made significantly more effective without losing the understanding of the process. This is primarily related to preprocessing and preparing data used for both training and evaluating models. Since this framework only considers data from R/V Gunnerus, it is possible to perform several generic tasks that make the process of using the framework both simpler to understand and less time-consuming. A simple example will be made to demonstrate functionality.

The purpose of the web application is to connect the modeling API to a realistic scenario where applications of digital twins through predictive maintenance is used. In the web application, users should be able to upload trained models, and test these models against data from the vessel in a *simulated* real-time environment. Since

data is updated irregularly, and since a simulated error performed in 2019 will be used for validation purposes, data will not be updated in real-time. However, the web application will behave as if data is being supplied in real-time, which creates a simulated real-time environment. As data is being visualized and updated frequently, the user will be able to see when anomalies in the data occur. The web application should also be user-friendly and have a visually appealing interface to make the application more appealing. The interaction between the modeling API and web application is demonstrated in Figure 5.1.

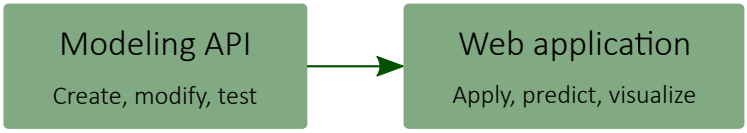


Figure 5.1: Interaction between modeling API and web application in the anomaly detection framework.

To validate the performance of the example model, and represent a method for testing model implementations, a previously simulated error in the exhaust pipes will be used, which was provoked with an intention for research purposes by NTNU Ålesund (Ellefsen, 2020). On November 21, 2019, between 10:50:16 and 10:56:33, the simulated error occurred as the exhaust pipe temperatures rise above the ordinary maximum values for safe operation. The rise in temperatures, together with the simulated error interval, is seen in Figure 5.2. It can be seen that the temperatures also rise quickly when the simulated error commences, which provides another detectable factor for the prediction model.

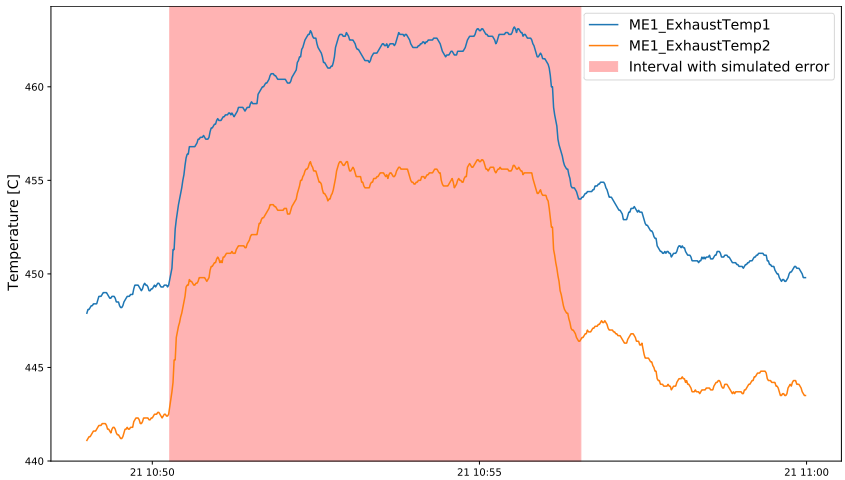


Figure 5.2: Simulated error on exhaust temperatures.

In general, having a good understanding of component behavior before and during a failure is important. However, since the error interval is already given, the process of simulating the error is treated as a black box, and only the result of the simulation error will be considered.

First, anomaly detection with recurrent neural networks (RNN) for predictive maintenance will be presented alongside the specific long short-term memory (LSTM) recurrent architecture used in the modeling example.

6 Anomaly Detection for Predictive Maintenance

The case study explores the topic of predictive maintenance through data-driven methods. This section is intended to give the necessary theoretical background to develop the anomaly detection framework.

6.1 Predictive Maintenance

Industrial maintenance on a system, component, or plant can be divided into preventive and corrective maintenance. As implied by their names, preventive maintenance attempts to interfere before the fault occurs. On the other hand, corrective maintenance is concerned with the repair of a system that has been damaged or degraded to a point where it affects operation. Driving a component to failure can have severe consequences for some systems, as it can cause unnecessary and abrupt strain on components. As the time of failures is not generally known, choosing corrective maintenance can cause unpredictability and sudden downtime. Preventive maintenance is often applied to a system using the mean time to failure ($MTTF$) of a system as a guideline for when to perform maintenance (Utne and Rasmussen, 2017). $MTTF$ is equivalent to the expected lifetime of a component or system, $E(t)$, and is given by

$$MTTF = E(t) = \int_0^\infty R(t)dt = \int_0^\infty t \cdot f(t)dt, \tag{6.1}$$

where $R(t)$ is the reliability function and $f(t)$ is the failure density function. Since $MTTF$ is a statistical variable based on component or system properties, the actual time between performed maintenance, known as the predetermined maintenance interval, will have a safety margin to sufficiently reduce the likelihood of failure occurring before maintenance is performed. This safety margin will inevitably make the predetermined maintenance interval shorter than what is strictly necessary. Excessive maintenance affects both expenses and availability of a component or system, and can also prevent surrounding equipment from being operational when performing maintenance.

Another method of implementing preventive maintenance is through predictive maintenance, which takes the component or system condition into account, also known as condition-based maintenance. A predictive maintenance scheme will predict when failure will occur and suggest when to interfere for maintenance or repair (Trojan and Marçal, 2017).

Predictive maintenance is an important concept for digital twins, as it enables a direct sense of health monitoring by exposing the physical well-being of the vessel through the transmission of data. Feedback from the real asset signals is transmitted to the virtual representation, which is able to extract knowledge and suggested activity based on the asset's behavior and health. Historical data can help prevent failures in normal conditions by evaluating operational patterns. Additionally, predictive maintenance can play a vital role in predicting critical failure due to unforeseen events, which can occur before human operators are able to register and prevent damage from being done.

6.2 Artificial Neural Networks

Artificial intelligence, especially through machine learning, has become an important tool for predicting the behavior of systems transmitting large amounts of data. Artificial neural networks (ANNs), which attempt to replicate the behavior of biological

neurons, are among the most common machine learning algorithms applied to industrial applications (Carvalho et al., 2019). ANNs usually consist of an input layer, a hidden layer, and an output layer, where each neuron in a layer is connected to another layer by weights. A simple neural network (NN) is illustrated in Figure 6.1. Usually, NNs are fully connected, meaning that each neuron in a layer is connected to all of the neurons in adjacent layers. As implied by its name, a weight is used to determine how much *weight* should be added to a connection. In other words, how important the NN should regard the given connection.

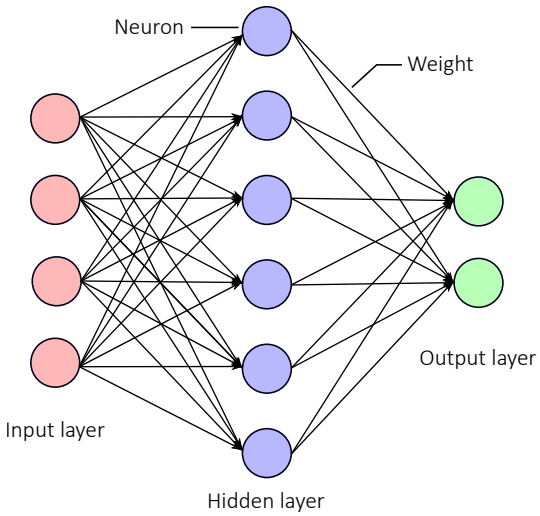


Figure 6.1: A simple ANN configuration, displaying neurons, weights, the input and output layers, and a single hidden layer.

Each node in an NN is concerned with calculating the activation value, which is the value each neuron represents (Nielsen, 2019). The activation value of neuron i at layer n, $a_{i,n}$, is given by

$$a_{i,n} = \varphi(a_{i,n-1}\vec{w}_i + b_i), \tag{6.2}$$

where $a_{i,n-1}$ is the previous activation value at neuron i, and $\vec{w}_i$ is a vector of weights, and and b_i is the neuron bias. As seen in (6.2), a bias is a constant term added to the activation function which affects the activation value independently of preceding layers, whereas the weights represent a connection between the activation neuron and neurons of the preceding layer. Biases can be used to give individual neurons more influence by increasing the bias value, in turn making it easier to *activate* the neuron. $\varphi(\,\cdot\,)$ from (6.2) represents an activation function. Activation functions are non-linear functions that separate NNs from linear regression models, which perform the input transformation which enables the network to learn. The most common activation function type are sigmoid functions, $\sigma(\,\cdot\,)$, which are used to prevent small changes in weights and biases from causing large changes in the activation value (Nielsen, 2019). As such, different sigmoid activation functions have a characteristic S-shape. The principle of sigmoid functions is similar to attenuating feedback in control systems theory to prevent perturbation effects. The most common sigmoid function is the rectified linear unit (ReLU) function, given by

$$\varphi(x) = \max(0, x), \tag{6.3}$$

where x is the output value. As seen, the ReLU function maps an activation value to the positive plane, returning an output value of $x = 0$ if the calculated output value is negative.

The ANN displayed in Figure 6.1 is simple, as it only contains a single hidden layer between the input and output layer. Most often, the NN models used are much more complicated. A particular branch of machine learning is deep learning, which utilizes deep NNs. Their depth characterizes these networks as they have a multi-layered neuron structure with at least two hidden layers. An example of deep NNs is long short-term memory (LSTM) networks, which in turn is a recurrent neural network (RNN). RNNs differ from traditional neural networks in that they can consider time and sequential data. The ANNs discussed in Section 6.2 are examples of feedforward networks, where outputs are calculated without visiting the same neuron twice. In contrast, RNNs are feedback networks, meaning that in addition to utilizing the current input values, the information from the previous input values is also used. The output calculation of a feedback node is illustrated in Figure 6.2, together with the unrolled feedback loop for each timestep. Since information is able to persist from one timestep to another, the looping of input values implies feedback over time.

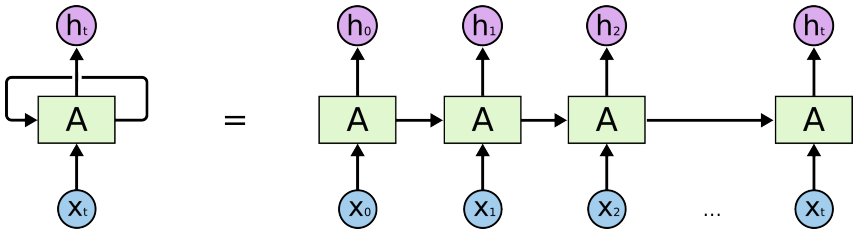


Figure 6.2: A feedback loop for an RNN and the equivalent unrolled RNN. x_i and h_i mark the input values and calculated output values of timestep i, respectively, while A is some NN structure. Courtesy of Olah (2015).

A critical challenge for RNNs is the *vanishing gradient problem*, as highlighted in Jozefowicz et al. (2015). Each neuron in an RNN must maintain a vector of activations for each timestep, which demonstrates that RNNs are deep NNs. Since activation values demand multiplication during the calculation, a problem of repeated multiplication between different timesteps makes small values vanishing and, conversely, high values exploding. The vanishing values turn out to be harder to deal with and makes it difficult for RNNs to learn if the sequence of timesteps becomes too long (Bengio et al., 1994). In other words, as the gradient, which is used to update the RNN weights becomes small, the new weight will not be able to change much, which makes learning time-consuming.

6.3 Long Short-Term Memory Networks

The LSTM network architecture was developed to specifically avoid long-term dependencies, which resolves the vanishing gradient problem inherent in RNNs. As implied by its name, an LSTM network is a type of RNN characterized by its ability to remember previous timesteps in its sequence. Whereas the standard RNN has a single layer, the standard LSTM network has four layers, as illustrated in Figure 6.3.

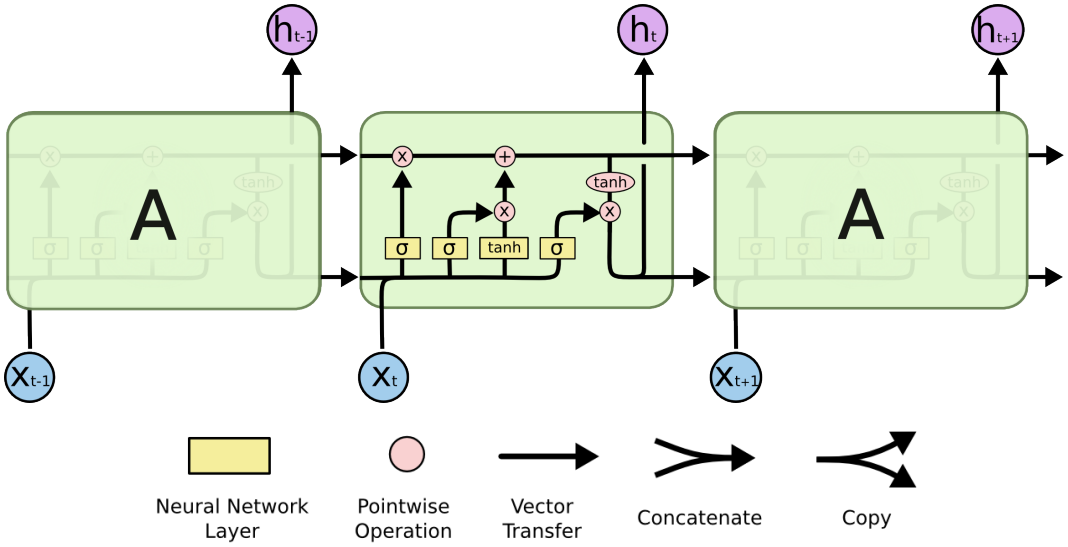


Figure 6.3: The repeating LSTM module with four interacting layers. The notations used are indicated at the bottom of the figure. Courtesy of Olah (2015).

The structure of the LSTM module in Figure 6.3 is described in great detail in Olah (2015). As seen, the four activation functions of the layers, marked by the yellow squares, consist of three sigmoid functions and one hyperbolic tangent function. In addition to the input state, $\vec{x}_t$, and the output state, $\vec{h}_t$, the LSTM has two states that are passed on from one time step to the next, as seen by the horizontal input and output arrows in Figure 6.3. The line passing through on the top can be referred to as the cell state, which receives updates from the neural network as it passes through the cell, as can be observed in the figure. It is this cell state that enables adding and removing sequential information possible. The line passing through on the bottom is simply the output state at time step t, $\vec{h}_t$, which indicates that for the cell in timestep t the output history $\vec{h}_{t-1}$ is used to calculate $\vec{h}_t$.

There are three sigmoid NN layers which are followed by pointwise multiplication operators in the LSTM module. These sigmoid layers control the memory of the LSTM module, and determine what the cell state should forget, add, and modify based on the input values, $\vec{x}_t$, and previous output values, $\vec{h}_{t-1}$.

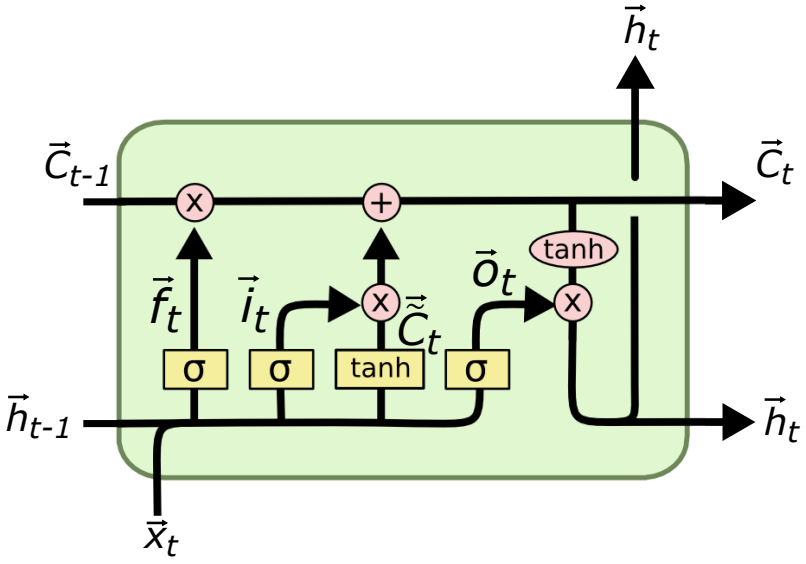


Figure 6.4: LSTM module with labels for input values, output values, and output values from NN layers, inspired by Olah (2015).

A pointwise multiplication operator succeeds all of the sigmoid NN layers, and the sigmoid layers are used to determine how much of its input is allowed to pass through based on a value between zero and one. Therefore, the output arrow from the three yellow sigmoid layers is a weight between zero and one, which determines how much should be allowed through. Since a multiplicative pointwise operator succeeds these values, everything multiplied with a sigmoid layer output will be filtered by some fraction, which is why these sigmoid layers are known as gates.

The first gate, denoted by $\vec{f}_t$, determines what the cell state should forget from the previous timestep. As seen from Figure 6.4, $\vec{f}_t$ can be derived as

$$\vec{f}_t = \sigma(\mathrm{W}_f \cdot [\vec{h}_{t-1}, \vec{x}_t] + \vec{b}_f), \tag{6.4}$$

with weights and biases similar to those mentioned in Section 6.2. However, since all states are vectors, and assuming the LSTM layer has n neurons, the previous weight vector, $\vec{w} \in \mathbb{R}^n$, now becomes a weight matrix, $\mathrm{W} \in \mathbb{R}^{n \times n}$, and the scalar bias, $b \in \mathbb{R}$ becomes a bias vector, $\vec{b} \in \mathbb{R}^n$. The weight matrix, W_f, contains specific weights for both the previous output values, $\mathrm{W}_{f,h-1}$, and input values, $\mathrm{W}_{f,x}$, but for simplicity, the weights are combined in the expression $\mathrm{W}_f \cdot [\,\cdot\,]$. The forget gate given in (6.3) decides what to keep and what to throw away from the previous cell state based on $\vec{h}_{t-1}$ and $\vec{x}_t$.

After the cell state, $\vec{C}_{t-1}$ has been told what to forget after being multiplied with $\vec{f}_t$, it needs to be told what to add. This process is twofold, where an input gate, denoted by $\vec{i}_t$, is used to decide which values will be updated, and a hyperbolic tangent layer used to calculate a candidate vector, $\tilde{\vec{C}}_t$. As seen in Figure 6.4, the input gate is given by

$$\vec{i}_t = \sigma(\mathrm{W}_i \cdot [\vec{h}_{t-1}, \vec{x}_t] + \vec{b}_i), \tag{6.5}$$

and the candidate vector is given by

$$\tilde{\vec{C}}_t = \tanh\left(\mathrm{W}_C \cdot [\vec{h}_{t-1}, \vec{x}_t] + \vec{b}_C\right), \tag{6.6}$$

where the hyperbolic activation function maps inputs to outputs between unit values as $\tanh(x) = y, y \in [-1, 1]$. With $\vec{f}_t$, $\vec{i}_t$, $\tilde{\vec{C}}_t$, and the previous cell state it is possible to calculate the updated cell state, $\vec{C}_t$, as

$$\vec{C}_t = \vec{f}_t \odot \vec{C}_{t-1} + \vec{i}_t \odot \tilde{\vec{C}}_t, \tag{6.7}$$

where $\odot$ is the pointwise product operator. The final gate layer is the output gate, denoted by $\vec{o}_t$, which decides what the LSTM module with output through $\vec{h}_t$. The calculated cell state, $\vec{C}_t$, is sent through a hyperbolic activation function to map the values between -1 and 1 before it is filtered based on the result of the output gate. The resulting output is given by

$$\vec{h}_t = \vec{o}_t \odot \tanh C_t, \tag{6.8}$$

where

$$\vec{o}_t = \sigma(\mathrm{W}_o \cdot [\vec{h}_{t-1}, \vec{x}_t] + \vec{b}_o), \tag{6.9}$$

The result of the LSTM module presented in this section is a network with the ability to keep information from the first timesteps throughout a sequential process and get rid of information that is deemed invaluable. Since the data from R/V Gunnerus is represented as a sequential time series where previous time variations will help predict future values. There is also an added benefit through the operational profile of a vessel, which may help the model detect recognizable patterns. Due to these benefits, an LSTM network will be implemented as the example model used to verify the modeling API and web application functionality.

It is worth mentioning that there have been developed several alternatives and variations of LSTM with varying results, such as a Gated Recurrent Unit (GRU) (Jozefowicz et al., 2015). The case study will be limited to investigating an LSTM network, but the results from the case study enable testing other sequential RNNs.

6.4 Anomaly Detection

So far, this section has explored concepts of predictive maintenance and creating sequential RNNs with an LSTM architecture. Here, the method for applying RNNs for condition-based maintenance is presented, which is the basis for detecting anomalies during operation.

Considering the states of a subset of signals at timestep t, $\vec{x}_t \in \mathbb{R}^n \subseteq \vec{X}_t \in \mathbb{R}^m$ and $n \leq m$. Here, $\vec{X}_t$ represents all signals related to a system or plant. A subset $\vec{x}_t$ is considered to filter out signals that have the least effect on the output. For example,

a vessel's latitude and longitude may be able to affect the engine speed. However, it is more important for the prediction model to include engine power, for instance. For a prediction model, a set of input signals, such as $\vec{x}_t$, is used to predict a set of output signals, denoted as $\hat{\vec{y}}_t$. $\hat{\vec{y}}_t$ can either be a subset of $\vec{x}_t$ or include other values from $\vec{X}$. With both $\hat{y}_t$ and y_t available, it is possible to calculate the absolute error between the actual value and the predicted value, ε_p, as

$$\varepsilon_p = |y_t - \hat{y}_t|. \tag{6.10}$$

By introducing an anomaly threshold value, denoted by T_a, absolute errors exceeding the given threshold are recognized as anomalies. Denoting anomalies as A which is either 0 (false) or 1 (true), the expression for A is given by

$$A = \begin{cases} 1, & T_a \leq \varepsilon_p \\ 0, & T_a > \varepsilon_p \end{cases}. \tag{6.11}$$

This is the method of condition-based anomaly detection that will be used in the framework developed in the case study. There are other ways of detecting anomalies, but the use of absolute errors makes explicit use of the prediction model, which is a core objective of the case study. In the case study, threshold values will be calculated through tuning. However, it is possible to implement automated methods for calculating threshold values as well. For the case study, these methods will not be explored.

7 Modeling API for Anomaly Detection

The modeling API is the first part of the case study framework and is developed to provide an environment for modeling, training, and testing sequential RNNs. The source code is documented on GitHub (Alvsaker, 2020*a*), and a user manual for the modeling API is given in Appendix A.3 The code itself is heavily documented to provide the necessary information to the user while they are using the API.

The modeling API will be tested against the simulation error presented in Figure 5.2 by implementing a simple LSTM model example. The example model should perform well enough to be able to detect the faulted simulation error while still being as simple as possible. The objective is not to create a model that emphasizes the best possible performance, but rather a model that can exemplify functionality and indicate how the modeling API can be utilized. Therefore, there will be no focus on tuning the example model.

7.1 Concept and Methodology

As mentioned in the problem formulation of Section 5, an API is a set of reusable functions that can be used without having to learn the complete structure and methodology of an underlying application or operating system. Developed models are trained and tested on data from R/V Gunnerus, and the user is able to choose from any available system transmitting data from the vessel. Available data streams are explored in Section 2.3. The user chooses a set of input signals that will be used to make predictions, and can also choose which values will be predicted. The modeling API supports a multivariate input-space and output-space, meaning that models with several inputs and several outputs can be implemented.

Through user settings presented in the execution file of the modeling API, users are able to interact with and choose which systems and data streams are of interest, which interval the model should be trained and tested on, and if the program should use existing files to speed up run time.

The modeling API will be developed using Python. Python is chosen as it is one of the most popular programming languages for beginners as it is open-source, has a human-readable syntax, and has a large developer community for support. Since the modeling API is intended to be used by students with varying knowledge and interests in programming, Python is a favorable choice for the implementation language. Python is also the programming language learned by most first-year engineering students at NTNU.

In addition to the benefits above, *TensorFlow*, which will be the library used for developing models, was originally developed in Python. TensorFlow[14] is an open-source library developed by the *Google Brain* team for numerical computations, and is often used for machine learning and neural networks (Abadi et al., 2015). Since TensorFlow is an end-to-end platform, it emphasizes the ability to export models after training and apply them to production. Since the models from the modeling API will be applied to a web application in a simulated production environment, this is an important functionality. More specifically, the Keras API[15] will be used, which is built on top of TensorFlow as a simple, flexible, and powerful deep learning API (Chollet et al., 2015). Keras is built to be human-readable, which makes it fast to pick up and easier to comprehend and implement than a manual alternative. The aspect of reducing the time spent on tedious tasks is beneficial. However, it is important that the speed of implementation does not compromise the understanding of how the API and ANNs in general work, which is why the framework in this case study relies on complementary teaching material.

7.2 Functionality

The API consists of functions that perform different tasks related to Keras modeling of sequential models. The API itself is divided into four main files that serve different purposes. The structure of the modeling API project is seen in Figure 7.1.

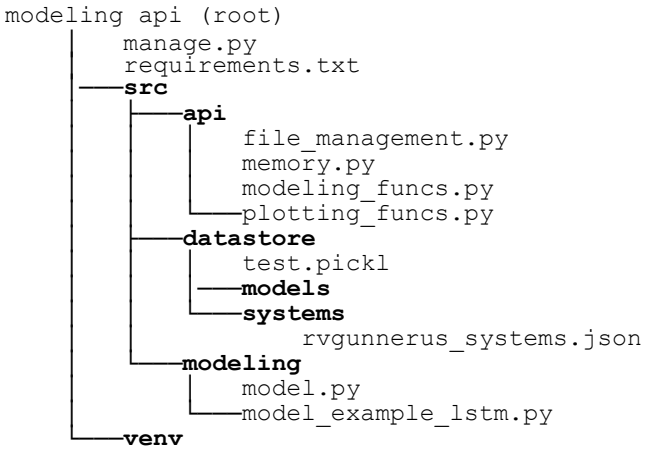


Figure 7.1: Project directory structure for the modeling API.

[14]TensorFlow documentation: https://tensorflow.org/api_docs
[15]Keras API documentation: https://keras.io/api/

At the root directory, two files are located. The `manage.py`-file is used to execute the program and includes all of the user settings related to how the program executes. The execution file is sectioned into different parts, such as file management and modeling operations, which makes it easier to understand what the different settings imply. The `requirements.txt` is used to install dependencies as explained in Appendix A.2.

The remaining Python code-files are found in the `src`-directory, which is subdivided into three folders. `datastore` is the program's native directory for storing files, including developed models and their metadata history files, data files used during runtime, and a systems folder. `rvgunnerus_systems.json` is a JSON[16]-structured file containing the structure for different systems, components, and signals on R/V Gunnerus. Currently, only the three main engines are implemented, but the file can be expanded to include any of the systems transmitting data. It is also possible to define the unit of the signals, which is an option that can be included when generating plots.

The remaining directories are `modeling` and `api`. The `modeling` directory is the main directory the user will use to interact. Here, sequential models and the necessary logic and functions surrounding the models are implemented. The `api` directory contains the four API-files with their respective functionality. In the following sections, the different API-files are introduced.

7.2.1 `file_management.py`

The main purpose of the file management part of the API is to take care of all data-related aspects up until the Keras modeling begins. The situation is analogous to parts of the procedural data chain from Figure 3.5, where `file_management.py` will be used for transformation, reduction, and filtering of data to enable analytics and learning. The analytics and learning part of the data chain, followed by insight and eventually action, is the part of the chain that is made possible through implementing ANN algorithms for condition-based maintenance.

In `manage.py`, the user can specify what data interval the program should use, what system to read from, and which components and signals to extract. If there are known intervals with faulted data, the user is able to add the interval to a blacklist, which will not be read during execution. It is of utmost importance to not include faulted data when training the model, such as the simulated error on the exhaust temperatures in Figure 5.2, which makes the ability to remove faulty intervals essential.

The signal data from the vessel are zipped into files containing 10 minutes of data each, which entails 600 rows per file since the logging frequency is 1 Hz. The file management functions support unzipping and concatenating the user-specified data interval to a single data frame. The file management functions can also filter data based on if the vessel is in operation. This is currently done by removing data where the engine speed is lower than 1770 RPM, which is the start-up speed required for operating the vessel (Ellefsen, 2020). The main file management functionality is summarized in the following list:

[16] JSON is a JavaScript Object Notation (JSON) that represents a lightweight exchange format accessible for humans to read and write and easy for computers to parse and generate (Crockford, 2001).

- check if the user has access to the network drive,
- get a list of existing models which the user can select from for testing,
- filter operation based on a signal and corresponding threshold value,
- navigate the network drive hierarchy, read and unzip files in specified intervals, and filter different systems, components, and signals
- read files from a network drive,
- remove faulty data based on specified intervals, and
- provides progression feedback to the user during file concatenation.

7.2.2 memory.py

The memory file provides functions for storing, loading, and deleting files through Python's native `pickle`-module. The functions are made so that the user can provide the desired file location, file suffix to append to the end of file names to be stored/loaded for recognition, and file prefix at the beginning of file names for type specification. The memory functions come with default values, which means that these input variables do not have to be specified. The memory functionality is summarized in the following list:

- store any Python object as `.pckl`-files,
- load `.pckl`-files,
- store any Keras model file in the designated `.h5`-format,
- load Keras models with the `.h5`-format, and
- delete files.

The memory functions intend to reduce the computation time during execution. Many of the processes related to data management, pre-processing, and Keras model development are time-consuming. Through `memory.py` and the settings in `manage.py`, the user can choose to create new files or use existing files. If, for instance, testing is carried out repeatedly using the same data interval, it is possible to load a pre-existing file rather than create a concatenated file from scratch over and over.

7.2.3 modeling_funcs.py

The modeling functions provide a set of functions useful for creating machine learning models with TensorFlow and Keras. The file includes functions for transforming and reshaping data, as well as evaluating the model after it has been trained.

Keras models require data to be input with specific formats. Using LSTM models as an example, each predicted timestep will look back at a set number of timesteps. Therefore, the shape of the LSTM data passed to the Keras model must be (n_s,t,n_f), where n_s is the number of samples in the data set – which in turn will be modified to correspond with the chosen batch size by Keras – t is the number of timesteps to look back at, and n_f is the number of features (input signals), the model will use for training. The data has then been reshaped from (n_s, n_f) to (n_s, t, n_f). If the model used only requires $t = 1$ the function will return the data frame in the appropriate shape for Keras.

Data transformation functionality is also supported, which maps values to a set interval based on a chosen scaler. This is important for the NN to interpret each signal in the same way without having a pre-existing bias towards certain signals. As an example, a `MinMax` scaler can be used to map values to a domain based on a given minimum and maximum value. If this range in (0,1), all values will be scaled to fit in the range. The scaling process is done per column, which means that the scaling process is completed for each signal independently of the other signals. Scaling is required when the features have different ranges, which is the case for many of the systems at R/V Gunnerus. The supported scalers are currently the `MinMax`-scaler and the `StandardScaler` implemented through `scikit-learn`'s[17] preprocessing scalers, and adding other `scikit-learn`-scalers is supported in the API. The choice of scaler depends on what type of data is used; for instance, the `StandardScaler` is used for data that is known to have a normal distribution.

- reshape training and testing data used to create a Keras model,
- get a user-specified scaler for transforming data,
- add new scalers,
- transform data
- transform data to normalized values based on chosen scaler,
- inverse transform values back to their original magnitude based on the same scaler,
- create a performance evaluation dictionary used to evaluate model performance,
- perform reshaping and transformations needed for testing faulty data,
- calculate threshold value for detecting anomalies,
- identify and label all anomalies exceeding the calculated threshold value,
- remove false anomalies based on an outlier approach explained in Appendix F.2, and
- create a unique string for naming models based on model parameters and time of creation.

As demonstrated, `modeling-funcs.py` provides a toolbox that can be used to speed up essential parts of the modeling process.

7.2.4 `plotting_funcs.py`

Here, basic plotting functionality is included to evaluate the performance of a model. The plotting functions of the API support the following visualizations:

- history plots of training variables, such as mean absolute error (MAE) and root mean square error (RMSE),
- prediction plot showing a time series of actual value and predicted value,
- distribution plots showing histograms and accompanying general kernel density estimates (KDE) for the error values, together with the calculated threshold,

[17] Preprocessing documentation: scikit-learn.org/preprocessing.

- calculated error for each timestep against the calculated threshold value to identify which values are above the threshold, and
- anomaly plot showing time-series data and detected anomalies based on the calculated threshold value.

The plotting functionality also supports adding simple formatting options for the plots. Users are encouraged to make their own plots to evaluate the performance of models, but these plots intend to serve as guidelines and simple model verification.

7.2.5 `model.py` and `model_example_lstm.py`

As mentioned, the `modeling` directory is where users implement their own models. `model.py` consists of a modeling template with empty functions representing the general structure the user should maintain while using the modeling API. The modeling template is given in Listing 7.1. The `model_example_lstm.py` is a helper file showing how an LSTM architecture can be implemented through the API. The LSTM example follows the exact same structure as Listing 7.1. It is recommended to use the LSTM example as a guideline when using the api.

Listing 7.1: Functional template for creating machine learning models.

```
# Standard library:
import os, pickle, sys

# External modules:
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow import keras

# Local API:
from src.api import file_management as filemag
from src.api import memory as mem
from src.api import modeling_funcs as mfnc
from src.api import plotting_funcs as pfnc
#------------------------------------------------------------------------------
def create():
    """Implement your model creation function here."""
    sys.exit('Not implemented.')

    # return model

def train():
    """Implement your model training function here."""
    sys.exit('Not implemented.')

    # return model, history

def test():
    """Implement your model testing function here."""
    sys.exit('Not implemented.')

    # return performance

def visualize():
    """Implement your result visualization function here."""
    sys.exit('Not implemented.')

    # return
```

The actual LSTM model is implemented, as shown in Listing 7.2.

Listing 7.2: Defining the example model through an LSTM architecture.

```
# Model parameters:
UNITS_LSTM = 128
DROPOUT_RATE = 0.2
DENSE_UNITS = 2

# Instantiate a sequential model:
model = keras.Sequential()

# Add LSTM layer:
model.add(layers.LSTM(UNITS_LSTM, input_shape=(input_shape)))

# Add dropout layer to prevent overfitting:
model.add(layers.Dropout(DROPOUT_RATE))

# Add densely-connected neural network layer:
model.add(keras.layers.Dense(DENSE_UNITS))

# Compile model using MAE (mean absolute error), adam optimizer
# (stochastic gradient descent algorithm):
model.compile(loss='mae', optimizer='adam')
```

The example model has a simple structure with one LSTM layer, one dropout layer, and one dense layer. The LSTM layer has the properties of a deep NN, as explored in Section 6.3. The input values to the LSTM layer function is units and input shape. The input shape is the number of timesteps and features as (t, n_f), respectively. This means that the values in each batch will share this shape. The input units, denoted in Listing 7.2, represents the hidden state, or latent dimension. Recalling the discussion of LSTMs and Figure 6.4, the hidden state is $\vec{C}_t$. Thus, `UNITS_LSTM = 64` implies that the cell state vector has 64 elements. Since the shape of the input and output values are decided externally, the LSTM cell state vector indirectly affects the dimensionality of the weights, since multiplying input/output values with the weights have to result in a vector that coincides with the cell state length. The value can, in principle, be *any* value and is chosen through limited testing in this case study. The value affects improves the cell's ability to remember over time, but at the cost of computational power.

The dropout rate is used to drop a random selection of input units during model training by setting their values to zero (Chollet et al., 2015). With a dropout rate of 0.2, 20 % of the input units will be dropped out. A dropout layer is used to prevent overfitting the data where the model performs much better on training data compared to testing data. This is not beneficial for the model as it would perform poorly as soon as it is applied to a real production environment with new data.

The dense layer is a standard dense ANN and is included to add more depth to the model. Setting `DENSE_UNITS = 2` implies that the hidden layer has two neurons.

7.3 Results

The implemented LSTM example is tested on the simulated error values in Figure 5.2. The results shown here are generated by using a selection of the plotting functions in the API. Although the figures will include captions, the title of the plots will not be removed as it is a part of the API. The model was trained for 30 epochs with a batch size of 128, meaning that the training data cycle was repeated 30 times over, and the model completed 128 predictions before it was updated. The number of timesteps in the prediction sequence was set to 30, the number of features in the

input shape was 12, and the model was trained on data from November 2019 to May 2020. The 12 inputs coincide with the 12 available signals on the first main engine, which is where the simulated error in exhaust temperatures occurs. The output signals to predict were accordingly the exhaust temperature on the two exhaust pipes. The anomaly outlier neighborhood described in Appendix F.2 was used with a neighborhood requirement of five consecutive values, indicating that sequences of less than five consecutive anomalies were labeled as outliers and removed from the registered anomalies. The model history for the epochs is given in Figure 7.2, where the mean absolute error (MAE) during training and testing is shown.

Figure 7.2: Model training history over 30 epochs. `loss` is the error during training, while `val_loss` is the validation error during training. Both the training and testing loss values decrease over time and seem to converge.

The loss values start off low. This might be because of the extensive training set, which allowed the value to update many times over before the first epoch was terminated. This was somewhat visually confirmed as the dynamic testing loss value during training started at 1.0, and quickly decreased to around 0.6 during the first few batches. Both the training and validation losses seem to converge, but the validation loss applied to the testing data struggles with fluctuating values. This may be due to overfitting.

Predicted values alongside the actual values for the first exhaust temperature is given in Figure 7.3.

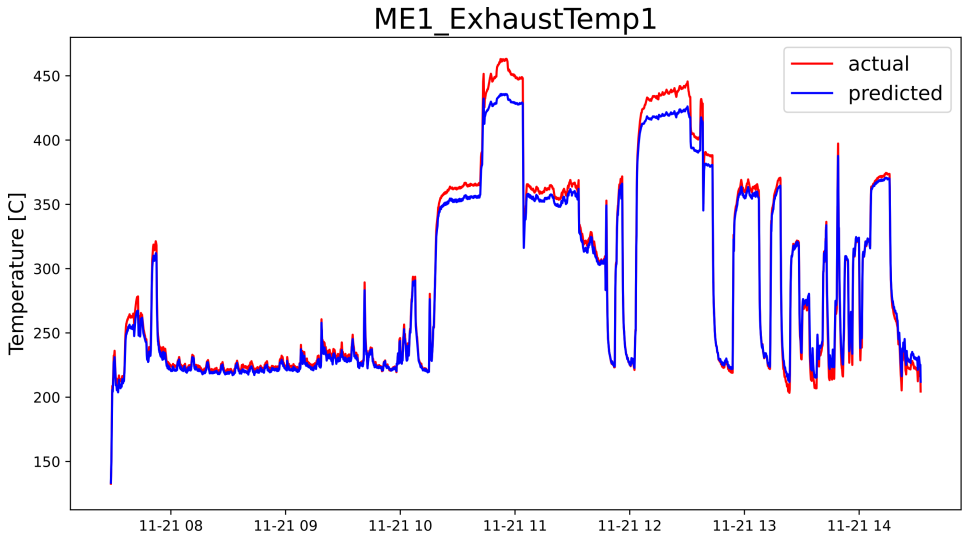


Figure 7.3: Prediction plot for the first exhaust temperature time series.

As can be seen, the model is clearly able to follow the different patterns of the actual values. The model does seem to perform better at some parts of the time series and struggles with a constant offset at other intervals. An enhanced view of the prediction plot is given in Figure 7.4.

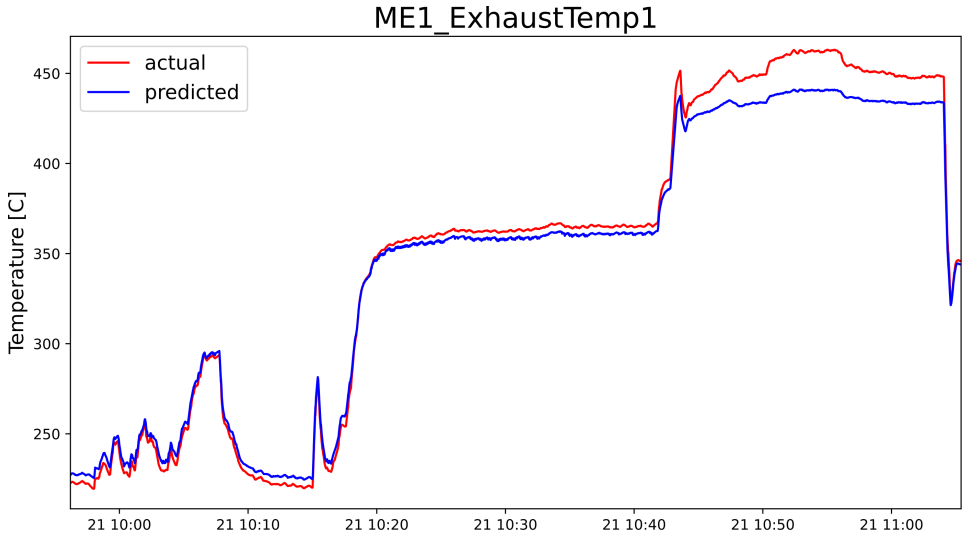


Figure 7.4: Enhanced prediction plot for the first exhaust temperature time series.

Manually changing the bias value to see if the areas of constant offset are affected might be able to change this behavior. A distribution plot of the error values is seen in Figure 7.5 together with a threshold value of $T_a = 7.0$, which was found through tuning.

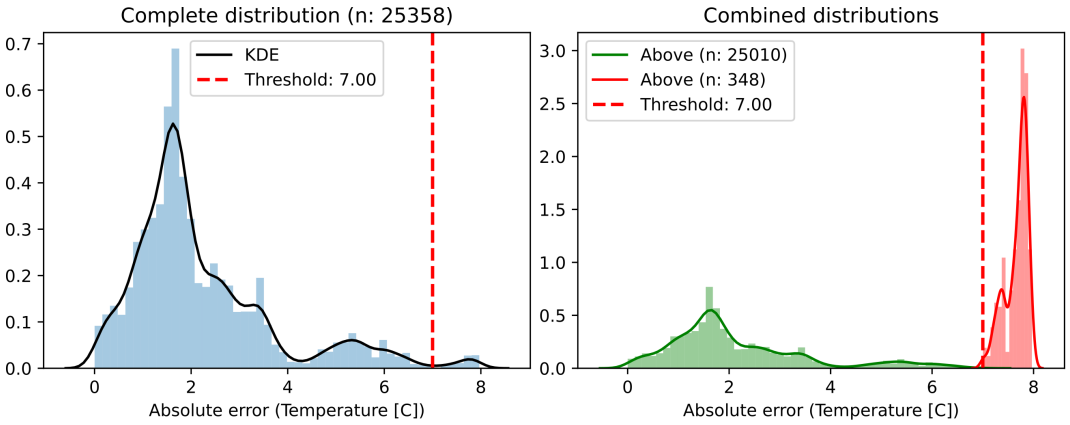


Figure 7.5: Distribution plots for predicted time series error together with a threshold value of $T_a = 7.0$.

With the threshold value found through the help of the distribution plot, the resulting anomalies in Figure 7.6 were identified.

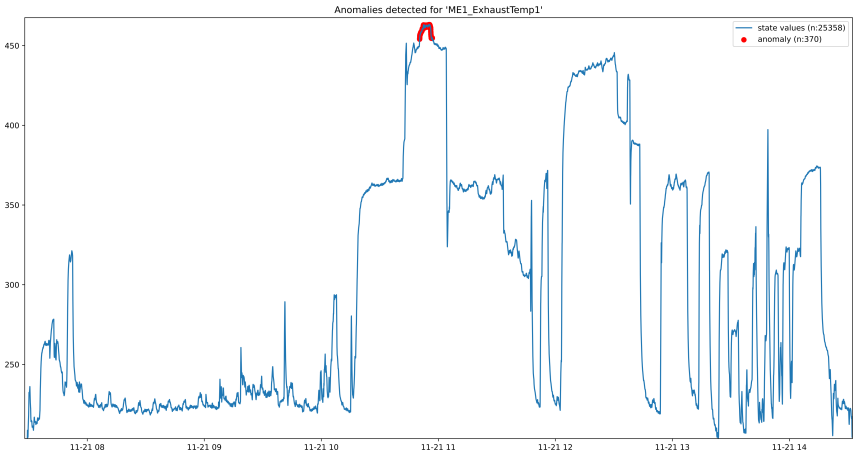


Figure 7.6: Time series plot with detected anomalies highlighted.

Zooming in on the detected anomalies yields the plot in Figure 7.7.

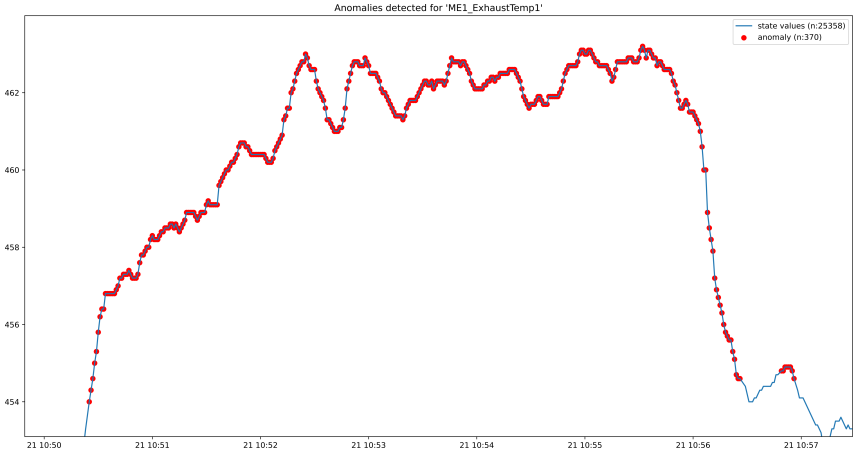


Figure 7.7: Closer view of time series plot with detected anomalies marked.

The first detected anomaly is at 10:50:25, which is 9 seconds after the simulated error has been started. All of the values in the simulated error interval after 10:50:25 up until 10:56:26 are marked as anomalies, while the simulated error actually ends at 10:56:33, missing the last 7 seconds. Thus, the prediction model managed to detect 95.8 % of the simulated error interval. However, as can be seen from Figure 7.7, the prediction model falsely detects a range of anomalies after the simulated error interval ends. This is not optimal behavior, as false positives can lead to unnecessary inspections and increased downtime. However, since the prediction model intends to function as an example, the results provided in this section are deemed sufficient. The model still manages to detect most of the simulated error. Similar results for the second exhaust temperature is given in Appendix D, where a threshold value of $T_a =$ was used, as the performance of the second exhaust temperature had a noticeably larger offset. Again, the effect of adding manual biases may prove valuable and could be a case for further testing.

7.4 Improvements to the API

There are three immediate improvements that will be considered. The first is that the API should have proper API documentation outside of the user manual provided here and the comments in the `manage.py`-file and API itself. This could be located at GitHub or another hosting site, or be created as a PDF. It could also be integrated into the web application created in the second part of the case study.

The second is the use of web applications for implementing the API and executing the project. This could be done with services such as the *Jupyter Notebook*, which is an open-source platform for interactive computing (Pérez, 2014). A benefit of using a web application for implementation is that the requirements for processing power do not depend on each individual computer. In addition, such web applications would allow running parts of the code and storing the results in memory, meaning that code can be executed in blocks, preventing the same problems described in relation to the `memory.py` functionality. The data stored in memory is only temporary, however, and all of the code would have to run upon revisiting the code at a later stage.

The third part is to improve upon the example LSTM model implemented. Since there was little time spent tuning the model, it was not determined how different configurations affect the specific case of the main engines, and there is likely room for substantial improvements in the example model.

8 Web Application for Anomaly Detection

The second part of the case study revolves around creating a web application that can be used to test sequential Keras prediction models made through the modeling API, or independently, in a simulated real-time environment. Since the current data transmission from the vessel is updated at irregular intervals, at most once per hour, it is not ideal for creating a visualization in real-time due to the infrequency of updates. In addition, the web application is meant to be used for anomaly detection, and the only anomalies currently known to have occurred on the vessel are from a historically simulated error from Figure 5.2. Thus, the web application will be made to have a real-time environment revolving around historical data. This means that data will be requested and updated live, but through values that, theoretically speaking, already exist. The web application will, however, be made in such a way that the transition from a simulated real-time environment to an actual real-time environment will be seamless. The web application is tested by uploading an example LSTM model and attempting to detect anomalies on the faulty exhaust temperature data.

As part of the work description in Section 1.2, the web application has been launched into a production environment through a Web Server Gateway Interface (WSGI) for demonstration purposes through the container-based cloud Platform-as-a-Service (PaaS) Heroku (Lindenbaum et al., 2007). The launched website is located at https://rv-gunnerus-anomaly-detection.herokuapp.com/. Note that Heroku is an open-source platform with limited resources for free users. The application will be dormant if not accessed in a while, and since it includes large space-demanding modules, such as TensorFlow, it will take some time to load it the first time. The web application is demanding when it comes to resources since data is being requested, processed, and updated every second. Therefore, the website is prone to be overloaded and run out of memory. If the website is not functioning correctly, or the user is stuck on a loading screen, try refreshing the page. If this does not work, try waiting some time before attempting to access the page again or delete the browser cache and cookies related to the website[18]. The user manual for setting up the development environment is given in Appendix A.4.

8.1 Concept and Methodology

With no pre-existing experience with web development, it seems unnecessary to create a web application to demonstrate the purpose of the modeling API. The choice of developing a web application relates back to the discussion of using digital twins for education. Apart from having pedagogical value, implementing an application tool successfully into an education environment requires the application to be simple, easily accessible, easy to use, portable, and scalable. All of these properties can be embodied in a web application. A web application represents a modern solution to creating applications and can excel in a cloud computing environment, detailed in Appendix B, where performance and speed can be scaled to fit the needs of the application. In brief, the premise of the web application is to create an environment where users can

1. Upload Keras sequential models (or use example files),

[18] If the website still does not work, a set of animations that demonstrate website functionality can be found in the GitHub repository.

2. choose which systems and signals the model is based on,
3. enable and disable signals for visualizing based on these chosen signals,
4. real-time visualization of readings for selected signals with an update frequency of 1 Hz, and
5. visualize predicted values when applicable, as well as to detect anomalies based on a user-specified threshold value.

To achieve this, a full-stack solution will be implemented with a separate client-side frontend and server-side backend. React, which is an open-source JavaScript library for building user interfaces (Facebook, 2013), will be used as frontend. Thus, the frontend is available for the interactions a user has with a web application and its data, as well as creating a visual interface. This includes making requests to the server, in other words interacting with the backend. Flask will be used as the backend, which is a microframework in Python (Ronacher, 2010). Since Flask is a microframework, it functions independently of other tools and libraries, which allows backend flexibility. The standard message communication between a frontend and backend uses the Hypertext Transfer Protocol (HTTP) requests and responses (Fielding et al., 1999). Here, a client sends an HTTP request to the server with information about the request. Based on this information, the backend sends a response back to the client. This type of HTTP messaging is visualized in Figure 8.1.

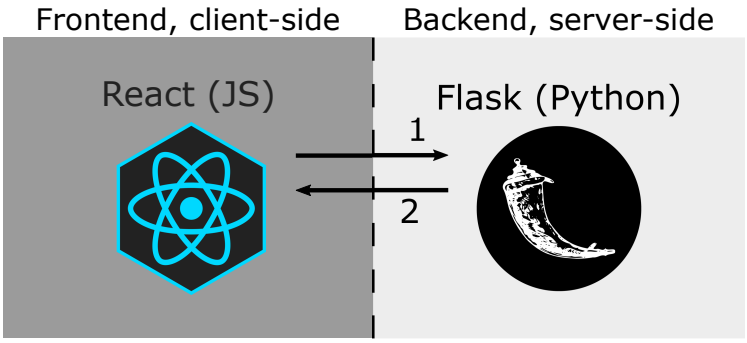


Figure 8.1: Standard HTTP message request$^{(1)}$/response$^{(2)}$ interaction between a React frontend and Flask backend.

A drawback to the standard HTTP messages is that the client has to wait for a request before it can send a response. This is a result of the unidirectional composition of HTTP messages. For the web application in the case study, a key concept is that the backend looks for changes in a database and sends these changes to the frontend. To enable unprovoked message responses to the frontend, WebSocket will be implemented through Socket IO. WebSocket is a bidirectional protocol – whereas HTTP is unidirectional – which allows opening a communication channel between a single connection where both the client and server can send and receive data at will (Fette et al., 2011). The bidirectional WebSocket protocol will be implemented through Socket IO for the React frontend (Rauch, 2014), and Flask-SocketIO for the Flask backend (Grinberg, 2014). Since Socket IO is originally a JavaScript library, Flask-SocketIO is an adapted version of Socket IO for Python.

The bidirectional communication enabled by the Socket IO is seen in Figure 8.2. Here, the database is included on the server-side. The database is able to hold

data in structured tables. For the web application, only the vessel data will be stored in the database, while all other session-dependent data will be kept in the browser memory. There are many different databases, and since the website will be launched to Heroku, a Heroku native database will be utilized, which allows for a cloud-hosted database. The open-source database PostgreSQL will be used through Heroku (PostgreSQL Global, 1996). The backend is able to query data, and upon request, the database sends data to the backend server. An updated communication chart of the client-side, server-side, and database is seen in Figure 8.2.

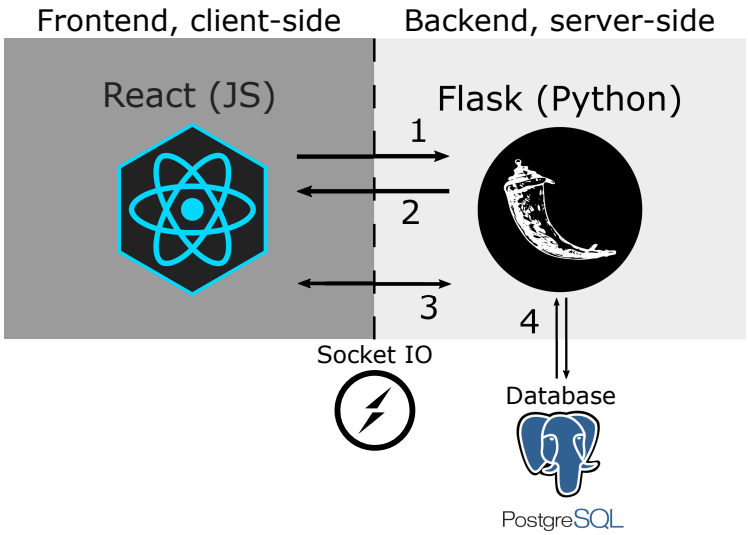


Figure 8.2: Full stack communication channels, with HTTP request(1)/response(2), bidirectional websocket(3), and database querying(4) makes up the communication channels between client, server, and database.

The project file and directory structure is given in Figure 8.3. In this project folder, the backend development is done with Python `api.py` and `models.py` located in `flask-backend`. The frontend development is done with JavaScript in the `src/components` folder of the `react-frontend` directory. Styling of the web page has been done through Sass style sheets with the files located in the `styles` directory (Catlin et al., 2006).

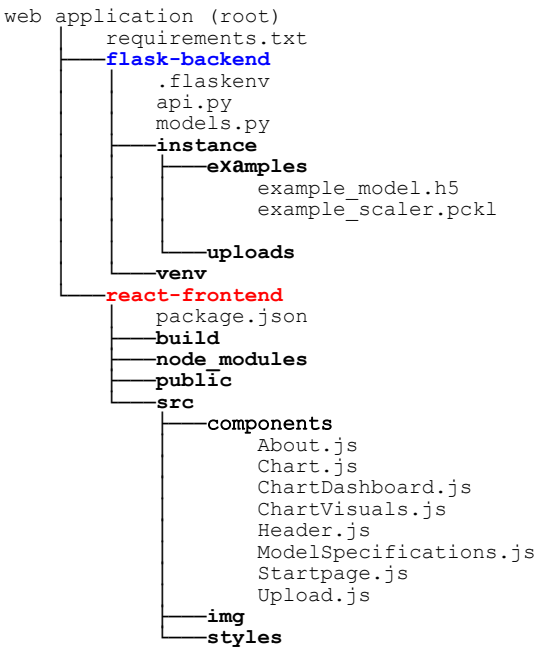


Figure 8.3: Condensed project directory structure for the web application.

8.2 Flask Backend

Flask is used as a backend for several reasons. First, Flask is a Python framework, which coincides with the modeling API. Python is the preferred language when using TensorFlow since TensorFlow was originally developed for Python, and the Flask backend needs to use TensorFlow to make predictions with a Keras model. Further, Flask has its own library supporting the use of SocketIO through Flask-SocketIO, which increases the compatibility between the Flask framework and the React library. Lastly, Flask was chosen over the popular Django framework alternative due to its lightweight implementation, making Flask flexible for smaller applications.

The backend is implemented through the `api.py` and `models.py` files. The `api.py` is the core of the Flask application, including all specifications related to the database, sockets, routing, and computations. The `models.py` file is concerned with the tables in the PostgreSQL database. Interactions with the database are handled through Flask-SQL Alchemy. SQL Alchemy is an object-relational mapper (ORM), implying that the library maps relations in a database to Pythonic objects (Pallets, 2010). A reason for using an ORM is to avoid writing raw SQL code, which interacts with a backend. Nevertheless, it should be noted that it is essential to have an understanding of how the underlying SQL works to utilize the ORM correctly and efficiently. As the documentation specifies, the tables have to be instantiated as classes – even if the tables already exist. The tables are instantiated in the `models.py` file. A snapshot from the Nogva engines table for the main engines defined in the PostgreSQL database is given in Figure 8.4. The snapshot is taken from PostgreSQL's PgAdmin4 software.

```
1  SELECT * FROM Public."Nogva Engines"
```

Data Output | Explain | Messages | Notifications

	time timestamp without time zone	me1_backupbatt double precision	me1_boostpress double precision	me1_enginespeed smallint
1	2019-11-21 10:49:30	26.3	1.4	1783
2	2019-11-21 10:49:31	26.4	1.4	1784
3	2019-11-21 10:49:32	26.3	1.4	1785
4	2019-11-21 10:49:33	26.3	1.4	1784
5	2019-11-21 10:49:34	26.3	1.5	1784
6	2019-11-21 10:49:35	26.4	1.5	1786
7	2019-11-21 10:49:36	26.3	1.5	1783
8	2019-11-21 10:49:37	26.3	1.4	1785

Figure 8.4: Table in PostgreSQL database.

The Flask application embodied in `api.py` is given in its entirety in Appendix F.1. Up until line 66, all of the code is concerned with instantiating the different resources used in the Flask application, in addition to specific global variables.

After the necessary resources have been declared, different Flask routes included in the application are defined. Routes are used as gateways to the frontend application such that a frontend client can make HTTP requests to the backend through the route handlers. The different routes are explained in the application file and include functionality for

- sending static files to the frontend client (necessary for production),
- returning a list of systems based on existing tables in the PostgreSQL database,
- returning a list of signals based on the column entries found for a defined system,
- loading, storing, and reading the properties of Keras models,
- loading, storing, and reading the properties of `scikit-learn` scalers,
- creating, starting, and ending thread activities for multiprocessing, and
- running thread and emitting values at a pre-defined interval (1 Hz) to the frontend client.

In the functionality above, threading is referenced. Threading refers to the `Threading` Python module, which is used in the backend to run multiple processes simultaneously[19]. This is necessary since the backend emits data queried from the database continuously with a given interval, which is 1 Hz. Without threading, the backend server would not be able to handle other requests from the client while emitting values, since the backend is only allowed one process. Threading enabled process parallelism such that values can be emitted to the client while still receiving other

[19] According to the `Threading` module, which is a native module in Python: https://docs.python.org/3/library/threading.html

requests from the client-side. In addition to the routes, two Flask-SocketIO handlers are included to handle WebSocket connection and disconnection.

The last part of the application file is concerned with the actual prediction of values. The route that creates the multiprocessing thread includes the following lines of code:

```
thread = ValueThread(system, input_cols, output_cols, timesteps)
thread.scaler = get_scaler(storage['scaler_path'])
thread.keras_model = get_model(storage['keras_model_path'])
thread.X_pred = thread.get_first_input_values()
```

First, an object variable of the class `ValueThread` is defined as `thread`, with system properties and the timesteps used in the sequential Keras model as inputs. Next, the scaler and Keras model are loaded from a storage cookie, which means that the `thread` object has access to the appropriate scaler and Keras model. Lastly, the thread calls a function of the `ValueThread` class that retrieves the first input values used for prediction. This means that `thread` queries the database for the given number of timesteps, t, in the Keras model, and collects the t first input values. By doing this, the input values are already available as soon as the client asks for predicted values. As an example, if the timesteps used were $t = 30$ and the number of input features was $n_f = 12$, the thread would collect the first 30 timesteps of these 12 features and store them in the session cookie. Now, when a prediction is made, the input values can be used, and the first of the 30 timesteps can be dropped for a new value from the database.

All of the functionality related to prediction of values is defined in the `ValueThread`-class. In addition to the `get_first_input_values()` function mentioned above, the `ValueThread`-class has a `get_data()` function. This is where value prediction and emitting values to the frontend is done. When the values have been queried from the database and predictions have been made through the Keras model, the `get_data()` function calls

```
socketio.emit('values', values)
```

which emits the values to the frontend through a custom socket-emitter called `'values'`. The frontend client listens to messages with this specific emitter-name and catches the values as soon as an incoming message exists.

8.3 React Frontend

React is a component-based library where a web page consists of different, separate components with individual states. Thus, web pages created with React can have states updated independently of other components, which makes the library dynamic. States from a parent component can be passed to child components as properties, and conversely, child components can pass states to parents through methods. This hierarchical approach is efficient for limiting which components have access to the states, making sure that information is only passed on to the components which need them. This helps prevent data leaks and makes the library fast.

In the following sections, the different components developed for the frontend will be explored. Additional snapshots of web application functionality are provided in Appendix E.

8.3.1 Startpage.js

The `Startpage`-component is the first interface the user meets and intends to glue all components together. Based on the user's interaction with the start page, three different display options are controlled by the `Startpage`-component, namely the

- selection page for uploading Keras model and data scaler files, as well as providing specifications related to inputs and outputs used by the Keras model,
- chart dashboard, which is the visualization page for readings, predictions, and anomaly detection based on the provided Keras model and scaler, and
- about page, which gives a summary of the purpose of the web application and links to relevant GitHub repositories.

The `Startpage`-component also has a home button that takes the user back to the default view, which is the selection page without any user selections or uploaded files. Figure 8.5 shows the first view the user sees when navigating to the web page.

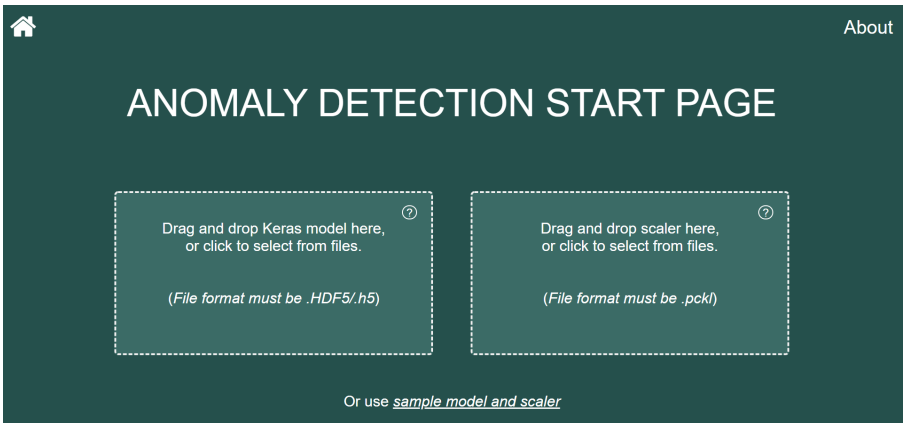


Figure 8.5: Web application start page.

8.3.2 Header.js

The `Header`-component consists of the current dashboard title and the home- and about-button. If the user clicks on the about-page, the Header also displays the GitHub source code repositories of the modeling API and web application.

8.3.3 Upload.js

Here, the user is able to drag and drop or browse files that will be used by the application. The user has to upload two files, a Keras sequential model and a data scaler for transforming and inverse transforming of the data. Scalers were explored in Section 7.2.3. The upload functionality is in part based on the `react-dropzone`[20] component library.

Several mechanisms for preventing erroneous uploads have been implemented, including conflicting file formats, multiple files, files exceeding the maximum file size of 100 MB, and files that can not be identified as either a Keras model or data scaler. If the user's uploads are invalid, the appropriate error is given as feedback. The visual feedback of uploading erroneous files is seen in Figure E.1, and the user

[20] Documentation for `react-dropzone`: https://react-dropzone.js.org/.

can hover over a question mark in the upload box to receive more information about the files to be uploaded, which is seen in Figure E.2.

As seen in the start page in Figure 8.5, there is also an option to use example files. When pressed, the application will load these example files into memory, and the user will be able to proceed.

8.3.4 `ModelSpecifications.js`

When the user has successfully uploaded files, the `ModelSpecifications`-component will allow users to select the system that the model was made with, as seen in Figure E.3. The systems are loaded from the PostgreSQL-database by first sending a request to the backend, which queries the database. Currently, only the Nogva engines table contains data. When a system is selected, the `ModelSpecifications`-component will send a new request to the backend, which queries the database for signals found of the chosen system and sends these signals back to the client. The user must then choose which signals the model uses as input, and consequently, which signals the model have as predicted outputs. This selection process is visualized in Figure 8.6.

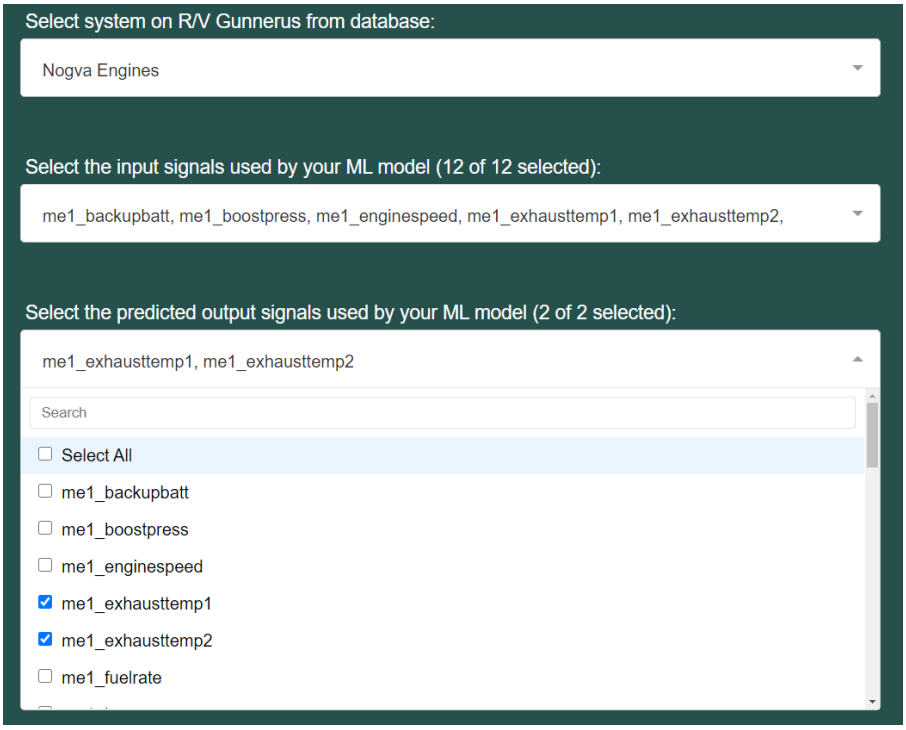


Figure 8.6: Model specifications.

Here, the user is trusted to choose the correct signals, since there is no way of verifying that the chosen inputs and outputs coincide with the uploaded model. As an alternate verification tool, the application reads the number of inputs and outputs from the model file and limits the number of selections the user can make.

When all of the model specifications are chosen, the start page selection phase is finished, and a continue button will become active, as seen in Figure 8.7. If the example files are used, the system, inputs, and outputs are chosen automatically, as seen in Figure E.5.

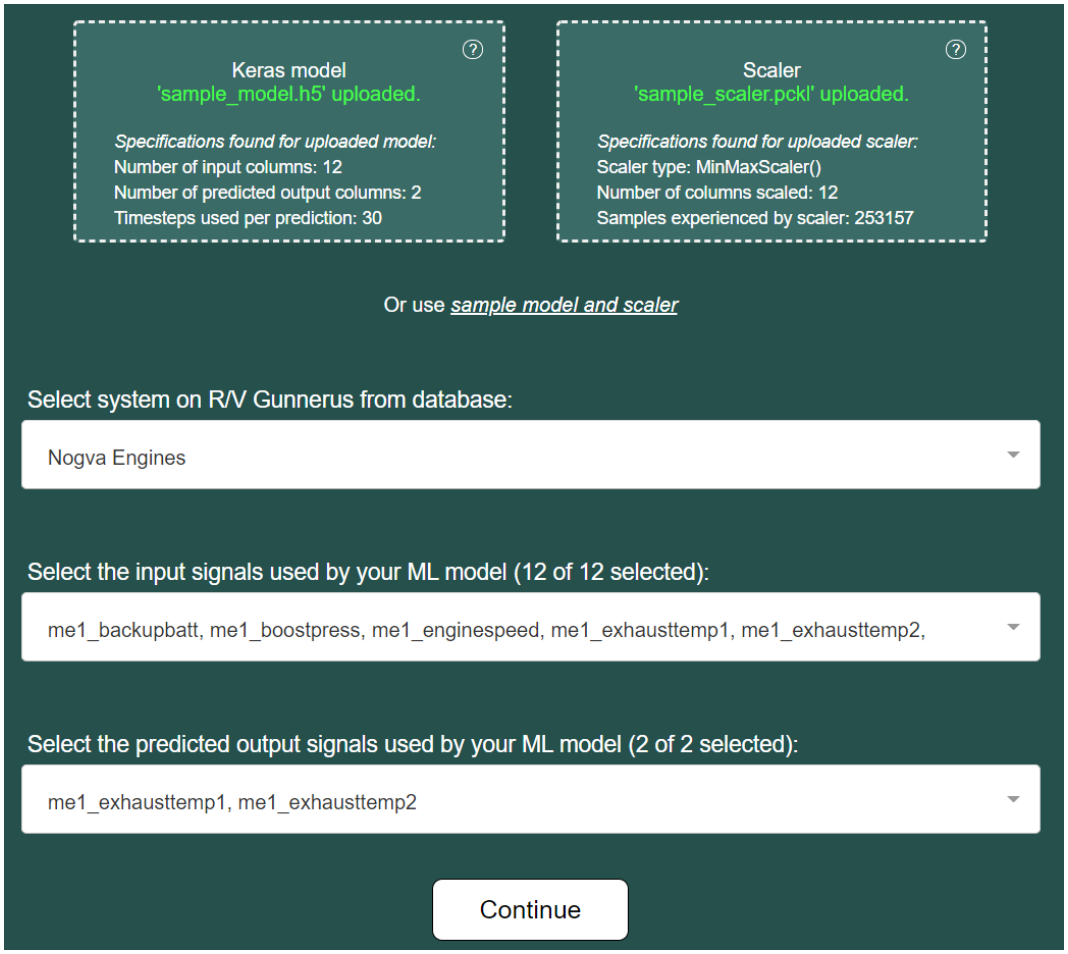


Figure 8.7: Completed model selection process with Keras model and scaler specifications shown, as well as selected system, inputs, and predicted outputs.

8.3.5 `ChartDashboard.js`

When continuing, the `ChartDashboard`-component will become active, which handles all activities related to the visualizing of readings, predictions, and anomalies. The `ChartDashboard` is responsible for

- creating a WebSocket connection with the backend through SocketIO,
- using the SocketIO connection to receive and handle values from the database through the backend,
- giving the user the ability to select which signals to visualize, and
- adding and removing charts based on user selection.

The initial charting dashboard the user sees is given in Figure E.6, where the signal selection menu is open.

8.3.6 `ChartVisuals.js`

Most of the visual representation, and some of the functionality, is taken from a sample IoT-application for peak detection of randomly generated data (Hassan, 2019). There are many open-source visualization modules available online. Therefore, it was deemed better to reuse appropriate resources rather than develop visual function-

ality from scratch. The `ChartVisuals`-component uses the `D3`[21] JavaScript library as a template for the charting visualization. The component has been changed to function with predictions and visualizing anomalies, in addition to some adjusting to the axes.

8.3.7 `Chart.js`

The `Chart`-component bridges the `ChartVisuals`-component and the flow of data through the `ChartDashboard`-component. For each signal the user selects in the chart dashboard, a new instance of a `Chart`-component is instantiated. The `Chart`-component receives data as properties from `ChartDashboard` and transforms the data to its appropriate format for the `ChartVisuals`-component. In addition, the `Chart`-component handles the connection status of the visual component, as well as the threshold selection, which affects the anomaly detection. The `Chart`-component is responsible for

- showing the name of the visualized signal and its last reading,
- showing the connection status of the signal,
- toggling the different series (readings, predictions, and anomalies can be toggled on or off by clicking on their respective labels),
- enabling threshold selection for anomaly detection and displaying the current threshold value,
- calculating the prediction error and determining if an anomaly is active,
- transmitting active anomalies to the `ChartVisuals`-component, which visualizes the anomalies,
- showing the deviation between the reading and predicted value (if applicable), and
- showing a timestamp of the last received reading.

When the user selects signals in the `ChartDashboard`, charts are added as seen in Figure 8.8.

[21] Documentation for `D3`: https://github.com/d3/d3/wiki.

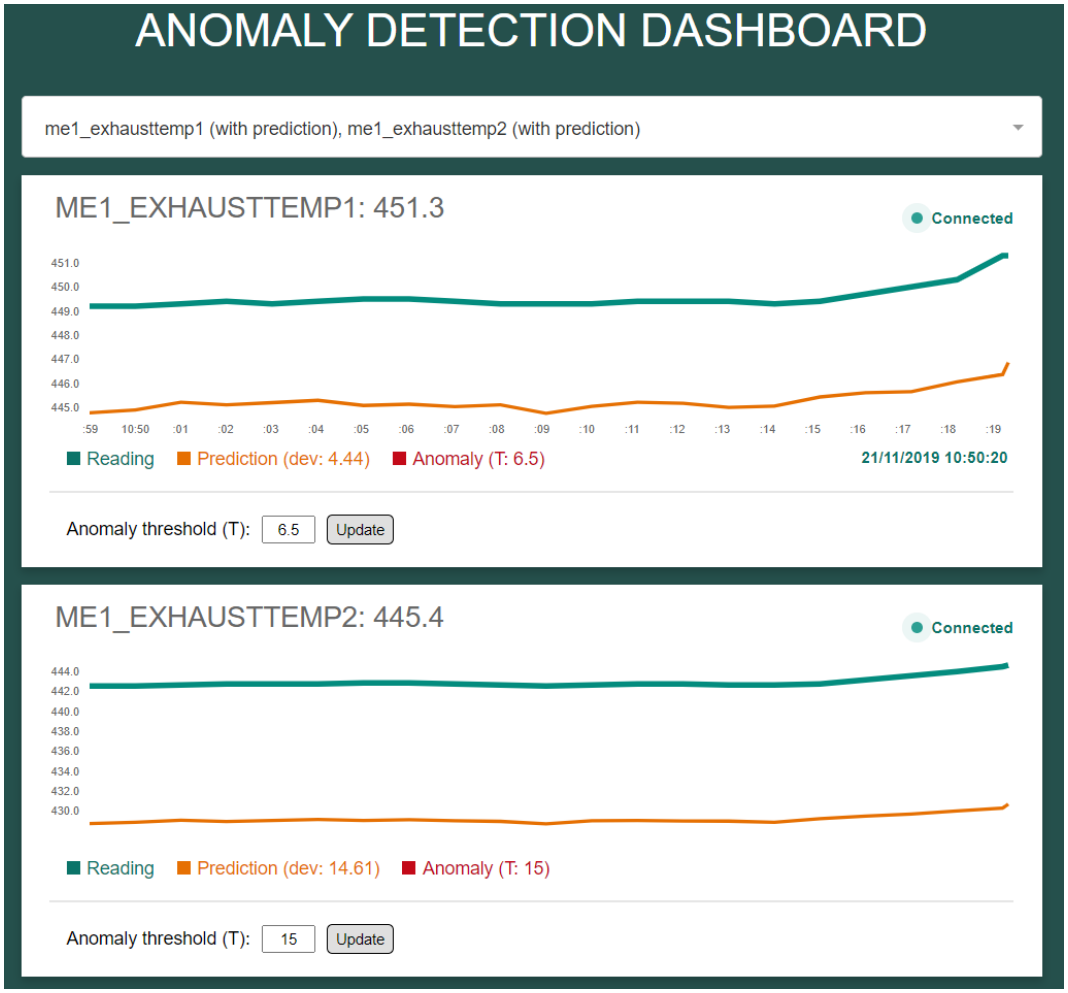


Figure 8.8: Visualization of data from two selected signals.

If the `Chart`-component encountered an error when fetching values, the signal becomes disconnected, as seen in Figure E.8. As mentioned, the user can toggle different series in the plot, which is demonstrated in Figure E.7, where the prediction series is disabled. For signals that are not part of the predicted output signals, meaning they will not be predicted by the model, only the "Reading"-series is available.

8.3.8 `About.js`

The about-page can be seen in Figure E.9.

8.4 Launching the Web Application to Heroku

As mentioned, the web application has been launched to Heroku through a Web Server Gateway Interface (WSGI). More specifically, this was done through *gunicorn*, which is a Python WSGI HTTP server (Chesneau, 2010). When deploying to Heroku, a `Procfile` is necessary for specifying production parameters. The `Procfile` can be seen in the root folder of the web application seen in Figure 8.3, and includes the code seen in Section 8.4.

```
web: gunicorn --preload --no-sendfile --worker-class eventlet -w 1
--chdir flask-backend api:app
```

The `Procfile` includes settings specified by both Heroku, gunicorn, and other parts

of the web application that demands certain behavior. It is especially important to note the restrictions in workers, given by `-worker-class eventlet -w 1`, which sets the maximum number of workers to 1. This is necessary when using Flask-SocketIO together with gunicorn[22].

8.5 Results

Similar to the modeling API, the functionality of the web application is tested against the simulated error on the exhaust temperatures seen in Figure 5.2. Both of the exhaust temperature signals can detect anomalies in the first few seconds of the simulated error. This is seen in Figure 8.9, where the threshold values are based on testing and are similar to the threshold values from Section 7.3.

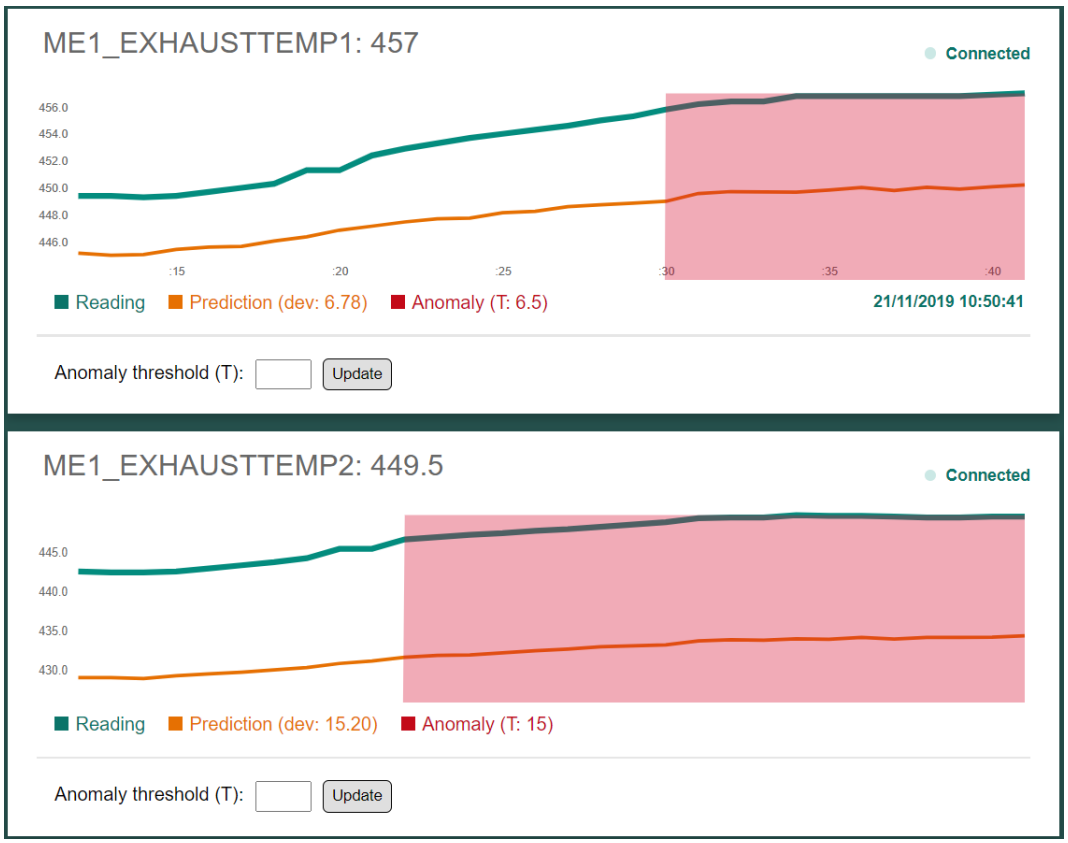


Figure 8.9: Visualization of the first detected anomalies for both exhaust temperatures.

The first chart encounters its first anomaly at 10:50:30 with a threshold value of $T_a = 6.5$, which is 14 seconds after the simulated error begins. The second chart encounters its first anomaly at 10:50:22, which is 8 seconds after the simulated error begins. Both of the signals continue to mark the values as anomalies until the simulated error is close to its end. The last detected anomalies are seen in Figure 8.10.

[22] This is specified by Grinberg (2014) in this discussion thread.

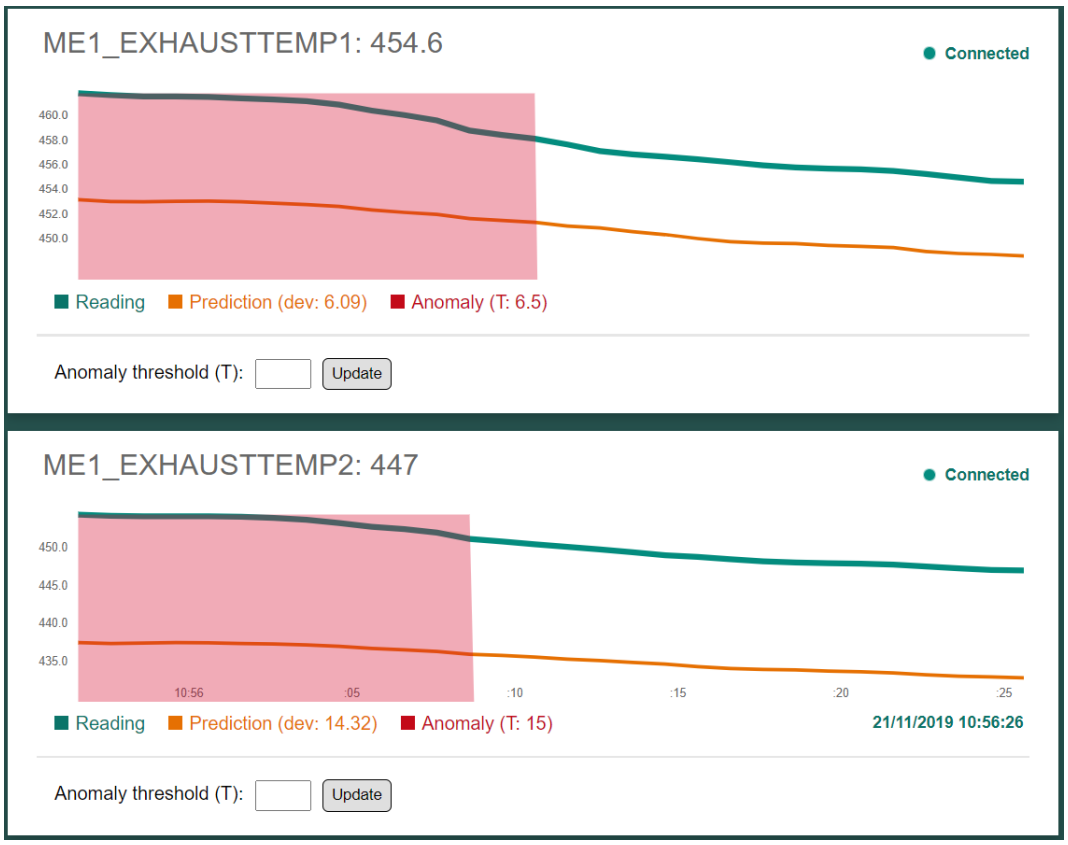


Figure 8.10: Visualization of the last detected anomalies for both exhaust temperatures.

The first chart encounters its last anomaly at 10:56:11, which is 22 seconds before the simulated error ends. The second chart encounters its last anomaly at 10:56:09, which is 24 seconds before the simulated error ends. Although the web application stops detecting anomalies a period before the simulation error ends, the values are seen to normalize in the simulated error, which indicates that the values may be in the operating range. The results are similar to the results from the testing of the modeling API in Section 7.3, which indicates that the web application successfully marks anomalies based on the uploaded Keras model and scaler.

8.6 Improvements to the Web Application

When it comes to web development, there are always ways of incrementally improving an application. Therefore, many small changes would improve the quality of the application for the user and for the maintenance of the server-side backend. The application, as is, lacks several elements of a standard web application, most prominently as authentication and user handling. Authentication was not prioritized during development since this is beside the objectives of the thesis. Also, NTNU has an authentication service that can be integrated into the application. Since user authentication is not included, each user's history is not stored, which means that uploaded files and user specifications are not stored. Adding user areas with file storage would be a significant improvement to the longevity of the application. There also lacks some error handling, which would make it easier for users to know what to do if an error occurs.

Currently, the most significant improvement would be to continue working on the plotting capabilities. The web application only supports a live feeding of data. In contrast, it would be beneficial to add the ability to, for instance, access historical data, save intervals of interests, change the number of data points displayed, and pause the plotting. It would also be possible to introduce an event log at the top of the visualization dashboard, where the user receives a message every time a critical event has occurred. This way, the user does not have to check every signal to know when an anomaly has occurred or when a system has been disconnected. A conceptual idea of such an event logger is given in Figure 8.11. Implementation would not be too demanding since all of the information is available, but due to time constraints, it was not prioritized. At the moment, the database only includes data from the Nogva engines. Adding new data to the database is trivial, and could demonstrate the application's flexibility and open up for testing algorithms on other systems on R/V Gunnerus. Due to database restrictions on Heroku, only the main engines were added for a short time interval. Setting up a database maintained by NTNU would enable scaling the database, which could interact with the streams directly from R/V Gunnerus, giving the application close to real-time functionality.

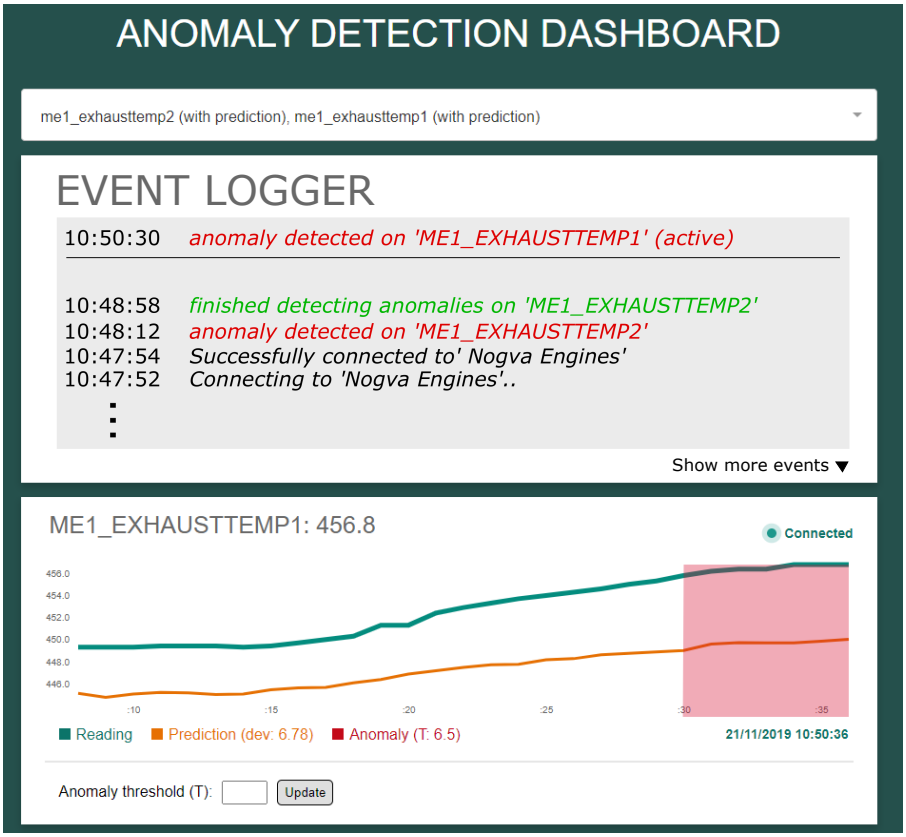


Figure 8.11: Conceptual idea of adding an event logger. Critical, active events are displayed at the top, and historical events are displayed in a list of limited shown entries.

The application is intended to display functionality relevant to digital twins. As such, it could be valuable to implement other elements of a digital twin in the web application, such as implementing different 3D models which the user can interact with

to extract information and data. The models are available, but creating an interaction between model components and a framework such as this anomaly detection application would be too demanding for this project.

9 Discussion

With a functioning framework for anomaly detection on R/V Gunnerus, the case study should be evaluated against its intended purpose. The case study is meant to represent an educational resource for marine engineering students. Although the framework is not a direct derivative of the proposed DTI, it is based on the data management component, where the end-goal is to provide insight into operation based on analytics and learning. What makes the framework a valuable contribution towards a digital twin of R/V Gunnerus is the real-time visualization environment. If it assumed that data is transmitted from the vessel in close to real-time, the web application could be modified to query these values at the same time as they are received by the server. Predicted values are calculated as soon as new values are queried from the database, which means that the web application would be able to detect anomalies and provide feedback close to real-time.

Circling back to what would make a digital twin a valuable resource for education, availability, and ease of access is emphasized. This was the main reason for developing a web application, and the web application was developed with these factors in mind as well. Thus, the application is as simple as possible to use, without any superfluous settings, configurations, or options. These additional functions could perhaps have increased the performance, but if it at the expense of availability or user experience, it would go against its purpose.

Based on the conducted case study, the anomaly detection framework can be used by implementing it into a relevant course in marine technology education at NTNU. This would require a theoretical motivation related to the use of ANNs for condition-based maintenance. After a theoretical introduction, students could be given exercises or a project description where the task is to implement their neural networks for anomaly detection.

10 Conclusion

Through the thesis, several aspects related to a digital twin of R/V Gunnerus have been approached. First, through a property-driven evaluation of digital twin definitions, a digital twin infrastructure was proposed, consisting of three main components. These components were data management, modeling and simulation environment, and software and system realization. These building blocks, and the functions they facilitate, represent a useful approach for creating digital twins for education since there is a vast application space where functionality should be added on top of a strong foundation. Also, the lifecycle of a digital twin suggests that, as digital twins are still in their technological infancy, students should be included in the development of digital twins, and not just the utilization of digital twins.

Through the case study, an anomaly detection framework was developed. The framework consists of a modeling API for creating, modifying, and testing sequential ANN models, and a web application for applying models, predicting values, and visualizing readings, predictions, and detected anomalies in a real-time environment. The modeling API has functionality for managing files, memory, and plotting, in addition to

several functions for accelerating the modeling process. The web application supports uploading a sequential Keras model – or using an example LSTM model – and visualizing data for the chosen system the model is applied to. The web application was developed with a React frontend in JavaScript, and a Flask backend in Python, with a PostgreSQL database. The web application has been launched to production through Heroku and is available at https://rv-gunnerus-anomaly-detection.herokuapp.com/.

To test the functionality of the anomaly detection framework, a simple LSTM sequential model of one of the main engines on R/V Gunnerus was implemented and tested on an interval with a simulated error on the engine's exhaust signals. After using the model to predict the behavior of the exhaust signals, the plotting functions supported by the modeling API were used to verify the results. The LSTM model was able to detect 95.8 % of the values in the erroneous data interval without detecting any false anomalies. The predicted values also followed the measured signal values without any abnormal deviations but struggled with a constant offset at times. This can be a result of overfitting the data. The LSTM model was made as simple as possible to demonstrate how quickly a model could be implemented through the modeling API. When the example model was tested in the web application, similar results were achieved.

The anomaly detection framework illustrates how a typical application of a marine digital twin can be developed to be used in an educational context.

11 Recommendations for Further Work

All of the different topics explored in this thesis have potential for further work. In general, the topics in the thesis have been discussed as means to continuing the development of a digital twin of R/V Gunnerus. The recommendations in this section revolves around specific ways of continuing this development.

The proposed digital twin infrastructure is the first obvious approach. In this report, the infrastructure has been proposed as a foundation for a digital twin of R/V Gunnerus, and if a true digital twin of the research vessel is to be made, the infrastructure is a starting point for achieving the necessary basic functionality. There are different industry projects being developed on the topic of digital twins, and co-operating with industry is mutually beneficial. Creating a closer connection with industry through participating in industry projects, either by contributing to development or through testing functionality, will help promote the use of digital twins in both an academic and educational context. In addition to fusing academic and industry interests, co-operation across institutes could prove beneficial. In experience, different groups working for a large company or university often work on similar topics without being aware of it. As NTNU has a maritime branch in both Trondheim and Ålesund, improving the dialogue between the two institutes would help accelerate the development of an R/V Gunnerus digital twin and prevent duplicate assignments.

When it comes to the case study conducted in this thesis, there are several points of departure for further work. As mentioned in the discussion in Section 9, with a theoretical introduction the results from the case study can be applied directly. Alternatively, it is possible to use the case study as a starting point, either for building upon directly, or by using the case study as a template for creating other, similar resources. Due to the thorough documentation of code for both the modeling API and web application development, building on the developed tools should be

possible if the necessary programming experience is known or taught.

A section for potential improvements were included with the modeling API in Appendix A.3 and with the web application in Section 8. These sections explored some alternatives to continue the development of the anomaly detection framework. For the modeling API, developing a separate documentation for the API would be the immediate action. In addition, exploring cloud platforms for interactive computing to limit the requirements for computing power, making the API more accessible. For the web application, it should be launched to an NTNU server with the appropriate resources available. There are many parts of the web application that needs improvement, as the application was implemented as a minimal bare-bone application with essential functionality such as user authentication and file storage. Implementing other aspects of a digital twin into the web application could also be of interest, but implementing the functionality of the web application into a digital twin application would be a more likely approach.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015), '*TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*', url=https://www.tensorflow.org/. Software available from tensorflow.org.

Alam, K. M. and El Saddik, A. (2017), '*C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems*', *IEEE Access* **Vol. 5**, p. 2050–2062.

Alvsaker, J. F. (2020*a*), '*R/V Gunnerus Anomaly Detection Modeling API*'. GitHub repository, https://github.com/johanfal/rv-gunnerus-anomaly-detection-modeling-api.

Alvsaker, J. F. (2020*b*), '*R/V Gunnerus Anomaly Detection Web Application*'. GitHub repository, https://github.com/johanfal/rv-gunnerus-anomaly-detection-web-application.

Alvsaker, J. F. (2020*c*), '*R/V Gunnerus Digital Twin Infrastructure Presentation*'. Presentation held at the Institute of Marine Tecnology, available at https://drive.google.com/file/d/1CZ19jXreofsXKAhSY3Mdqgx85zQa4-ID/view?usp=sharing.

Asbjørnslett, B. E., Pettersen, S. S., Erikstad, S. O., Rølvåg, T., Alvsaker, J. F., Bjørum, L. O., Borgersen, M., Rølvåg, P. and Stålesen, K. (2019), '*Report on the use of Digital Twins in Engineering Education*'. UROP NTNU, Department of Marine Technology. Available at https://drive.google.com/file/d/1U9MVTphmfkr34dO1_gL8Dwu-GJr65Wuc/view?usp=sharing.

Bengio, Y., Simard, P. and Frasconi, P. (1994), '*Learning long-term dependencies with gradient descent is difficult*', *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* **5**, 157–66.

Berre, A. and Ørnulf Rødset (2018), *From digital twin to maritime data space: Transparent ownership and use of ship information*, *in* 'ISIS – MTE18 Conference', Berlin.

Brodtkorb, A. H., Nielsen, U. D. and Sørensen, A. J. (2018), 'Sea state estimation using vessel response in dynamic positioning', *Applied Ocean Research* **Vol. 70**, p. 76 – 86.

Cabos, C. and Rostock, C. (2018), *Digital Model or Digital Twin?*, *in* 'COMPIT'18 Conference', Pavone.

Cano, J. (2014), '*The V's of Big Data: Velocity, Volume, Value, Variety, and Veracity*', www.xsnet.com/blog/bid/205405/the-v-s-of-big-data-velocity-volume-value-variety-and-veracity.

Carvalho, T. P., Soares, F. A. A. M. N., Vita, R., da P. Francisco, R., Basto, J. P. and Alcalá, S. G. S. (2019), '*A systematic literature review of machine learning methods applied to predictive maintenance*', *Computers & Industrial Engineering* **137**, 106024.

Catlin, H., Weizenbaum, N. and Eppstein, C. (2006), '*Sass: Syntactically Awesome Style Sheets*', https://sass-lang.com/documentation.

Chesneau, B. (2010), '*gunicorn*', https://gunicorn.org/#docs.

Chollet, F. et al. (2015), '*Keras*', https://keras.io.

Crockford, D. (2001), '*JavaScript Object Notation*', https://www.json.org/json-en.html.

de Carvalho, C. A. R. (2019), Personal communication (meeting October 22, 2019). Senior Researcher in Group Technology and Research, DNV GL.

DNV GL (2018), '*DNVGL–VIS Naming rules*', https://data.dnvgl.com/dnvgl-vis/.

DNV GL (2019*a*), '*Collaboration in Offshore Engineering – Sesam Insight*', www.dnvgl.com/services/collaboration-in-offshore-engineering-sesam-insight-115356.

DNV GL (2019*b*), '*Open Simulation Platform: Taking digital twins to the next level*', www.dnvgl.com/feature/open-simulation-platform-osp.html.

Ellefsen, A. L. (2020), Personal communication (e-mails in April and May of 2020). PhD candidate, NTNU Ålesund.

Erikstad, S. O. (2017), *Merging Physics, Big Data Analytics and Simulation for the Next-Generation Digital Twins*, *in* 'HIPER Conference', Zevenwacht.

Erikstad, S. O. (2018), *Design Patterns for Digital Twin Solutions in Marine Systems Design and Operations*, *in* 'COMPIT'18 Conference', Pavone.

Erikstad, S. O. (2019), *Designing Ship Digital Services*, *in* 'COMPIT'19 Conference', Tullamore.

Facebook (2013), '*React.js*', https://reactjs.org/docs/.

Fette, I., Carter, M. and Hickson, I. (2011), '*The WebSocket Protocol*', https://tools.ietf.org/html/rfc6455.

Fielding, R., Irvine, U., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999), '*HTTP Message*', https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html.

Firican, G. (2017), '*The 10 Vs of Big Data*', https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.

FullToTech (2019), '*Top Benefits of using Cloud Computing*', http://fulltotech.info/2019/10/top-benefits-of-using-cloud-computing/.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995), 'Design patterns – elements of reusable object-oriented software code'. Published by Addison-Wesley.

Grieves, M. and Vickers, J. (2017), *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, *in* 'Transdisciplinary Perspectives on Complex Systems', Springer, pp. 85–113.

Grinberg, M. (2014), '*Flask SocketIO*', https://flask-socketio.readthedocs.io/en/latest/.

Gupta, A. and Awasthi, L. K. (2009), *Peer enterprises: A viable alternative to Cloud computing?*, *in* '2009 IEEE International Conference on Internet Multimedia Services Architecture and Applications (IMSAA)', Bangalore, pp. 1–6.

H. Øverby, & J. A. Audestad (2018), *Digital Economics*, Amazon, Great Britain.

Harvey, C. and Patrizio, A. (2019), 'Aws vs. azure vs. google: Cloud comparison', www.datamation.com/cloud-computing/aws-vs-azure-vs-google-cloud-comparison.html.

Hassan, H. (2019), '*D3-TS-Chart*'. GitHub repository, https://github.com/iammowgoud/Peak-Detection-Visualization/tree/master/src/d3-helpers.

Hatledal, L. I., Zhang, H., Styve, A. and Hovland, G. (2018), *FMI4j: A Software Package for Working with Functional Mock-Up Units on the Java Virtual Machine*, *in* 'SIMS'59 Conference', Oslo.

Holmeset, F. T. (2019), Personal communication (audio-meeting October 24, 2019 and May 11, 2020, continuous exchange of e-mails throughout the fall of 2019 and spring of 2020). [1]Head Engineer, NTNU Ålesund, [2]Technical Inspector, R/V Gunnerus.

IntelliPaat (2019), '*AWS vs Azure vs Google – Detailed Cloud Comparison*', https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/.

Jozefowicz, R., Zaremba, W. and Sutskever, I. (2015), *An Empirical Exploration of Recurrent Network Architectures*, *in* '32nd International Conference on Machine Learning', Lille.

Kavis, M. J. (2014), *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*, Wiley, Canada.

Kongsberg Maritime (2019), '*Acquisition of Rolls-Royce Commercial Marine*', www.kongsberg.com/maritime/about-us/who-we-are-kongsberg-maritime/rolls-royce-commercial-marine-information.

Kyllingstad, L. T. (2019), '*Open Simulation Platform – Co-Simulation, Standards, and Software*'. Presentation held at NTNU Trondheim, November 13, 2019.

Laney, D. (2001), *3D Data Management: Controlling Data Volume, Velocity, and Variety*, Technical report, META Group.

Leavitt, N. (2013), '*Hybrid Clouds Move to the Forefront*', *IEEE Computer* **Vol. 46**, p. 15–18.

Lindenbaum, J., Wiggins, A. and Henry, O. (2007), '*Heroku*', https://devcenter.heroku.com/categories/reference.

Ludvigsen, K. B., Jamt, L. K., Husteli, N. and Øyvind Smogeli (2016), *Digital Twins for Design, Testing and Verification Throughout a Vessel's Life Cycle*, *in* 'COMPIT'16 Conference', Lecce.

Låg, S. and With, S. B. (2017), '*Standardisation as an Enabler of Digitalisation in the Maritime Industry*'. Group Technology & Research, DNV GL. Position Paper.

Macchi, M., Roda, I., Negri, E. and Fumagalli, L. (2018), '*Exploring the role of Digital Twin for Asset Lifecycle Management*', *IFAC-PapersOnLine* **Vol. 51**(Num. 11), p. 790–795.

Malakuti, S. and Grüner, S. (2018), *Architectural Aspects of Digital Twins in IIoT Systems*, *in* 'ECSA'18 Conference', Madrid.

Mell, P. and Grance, T. (2011), '*The NIST Definition of Cloud Computing*', https://csrc.nist.gov/publications/detail/sp/800-145/final.

Miller, H. and Veiga, J. (2009), '*Cloud Computing: Will Commodity Services Benefit Users Long Term?*', *IT Professional* **Vol. 11**(Num. 6), p. 57–59.

Mittal, V. (2019), Personal communication (audio-meeting October 2, 2019). Product manager for Data Fabric, Veracity.

Nielsen, M. (2019), '*Neural Networks and Deep Learning*', http://neuralnetworksanddeeplearning.com/.

Nielsen, U. D., Brodtkorb, A. H. and Sørensen, A. J. (2018), 'A brute-force spectral approach for wave estimation using measured vessel motions', *Marine Structures* **Vol. 60**, p. 101 – 121.

NTNU (2006), '*RV GUNNERUS – LNVZ*', https://www.ntnu.edu/oceans/gunnerus/spesifications.

NTNU (2019), '*Research Data*', https://innsida.ntnu.no/researchdata.

Oksavik, T. (2019), Personal communication (mail exchange November 11 to December 5, 2019). Data Scientist, Kongsberg Maritime.

Olah, C. (2015), '*Understanding LSTM Networks*', https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Os, J. V. (2018), *The Digital Twin throughout the Lifecycle*, *in* 'COMPIT'18 Conference', Pavone.

Pallets (2010), '*Flask SQL Alchemy*', https://flask-sqlalchemy.palletsprojects.com/en/2.x/.

Pedersen, T. A. (2019), Personal communication (meeting October 22, 2019). Principal Researcher in Group Technology and Research, DNV GL.

PostgreSQL Global (1996), '*PostgreSQL*', https://www.postgresql.org/docs/.

Pérez, F. (2014), '*Project Jupyter*', https://jupyter.org/documentation.

Qi, Q., Zhao, D., Liao, T. and Tao, F. (2018), *Modeling of Cyber-Physical Systems and Digital Twin Based on Edge Computing, Fog Computing and Cloud Computing Towards Smart Manufacturing*, *in* 'MSEC'13', Texas.

Rauch, G. (2014), '*Socket IO*', https://socket.io/docs/.

Ronacher, A. (2010), '*Flask*', https://flask.palletsprojects.com/en/1.1.x/.

Scania (2011), *Operator's Manual DI16 EMS with S6/PDE Marine engine*.

Selvik, Ø., Berg, T. and Gavrilin, S. (2015), Sea trials for validation of shiphandling simulation models-a case study, *in* 'MTEC'14 Conference', London.

Serrano, N., Gallardo, G. and Hernantes, J. (2015), '*Infrastructure as a Service and Cloud Technologies*', *IEEE Software* **Vol. 32**(Num. 2), p. 30–36.

Shafto, M., Conroy, M., Doyle, R., Glaessgen, E., Kemp, C., LeMoigne, J. and Wang, L. (2010), '*DRAFT Modeling, Simulation, Information Technology & Processing Road Map*'. National Aeronautics and Space Administration.

Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016), '*Edge Computing: Vision and Challenges*', *Internet of Things* **Vol. 3**(Num. 5), p. 637–646.

Skjetne, R. (2020), Personal communication (meetings January 10, 2020, and February 3, 2020). Professor in Marine Control Engineering, NTNU Trondheim.

Skjetne, R., Sørensen, M. E. N., Breivik, M., Værnø, S. A. T., Brodtkorb, A. H., Sørensen, A. J., Kjerstad, Ø. K., Calabrò, V. and Vinje, B. O. (2017), '*AMOS DP Research Cruise 2016: Academic Full-Scale Testing of Experimental Dynamic Positioning Control Algorithms Onboard R/V Gunnerus*'. OMAE 2017, p. 1–10.

Skjong, S., Rindarøy, M., Kyllingstad, L. T., Æsøy, V. and Pedersen, E. (2018), '*Virtual Prototyping of Maritime Systems and Operations: Applications of Distributed Co-simulations*', *Journal of Marine Science and Technology* **Vol. 23**(Num. 4), p. 835–853.

Sverdrup, J. (2017), '*Open Digital Platform Ecosystem – Industry Meets Science: Offshore Wind*'. Presentation held at NTNU Trondheim, November 13, 2019.

Synergy Research Group (2019), 'Strategic market intelligence for emerging it & cloud', https://www.srgresearch.com/.

Sørensen, A. J. (2018), *Marine Cybernetics – Towards Autonomous Marine Operations and Systems*, Department of Marine Technology, NTNU.

Taylor, C. (2017), '*The Cold Cloud: Long-Term Backup Storage in the Public Cloud*', www.enterprisestorageforum.com/storage-services/the-cold-cloud-long-term-backup-storage-in-the-public-cloud-1.html.

TensorFlow (2020), '*Install TensorFlow 2*', https://www.tensorflow.org/install.

Tjøswold, S. (2012), *Verifying and Validation of a Manoeuvring Model for NTNU's Research Vessel R/V Gunnerus*, Master's thesis, Norwegian University of Science and Technology, Department of Marine Technology.

Trojan, F. and Marçal, R. F. M. (2017), '*Proposal of Maintenance-types Classification to Clarify Maintenance Concepts in Production and Operations Management*', *Journal of Business and Economics* **Vol. 8**(Num. 7), p. 560 – 572.

Utne, I. B. and Rasmussen, M. (2017), '*Reliability, Availability, Maintenance and Safety (RAMS) in Design and Operation of Marine Systems*'.

Watts, S. and Raza, M. (2019), '*SaaS vs PaaS vs IaaS: What's The Difference and How To Choose*', www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/.

Zhang, H. (2019), Personal communication (audio-meeting October 10, 2019). Professor on Robotics and Cybernetics, NTNU Ålesund.

A User Manuals for Anomaly Detection Framework

The following user guides will be tailored towards the Windows operating system. However, the commands used for the installation procedure are universally known for all operating systems, and through a search engine, it should not be much different for other operating systems. The section includes user manuals for

- accessing network drives (used to get data in modeling API),
- use of Python, `pip`, and virtual environments,
- modeling API for anomaly detection,
- local web application set up for development (both backend and frontend).

A.1 Access to Network Drive

Data from R/V Gunnerus is transmitted to a server run by Kongsberg, which is the main data provider of the vessel. The data used in the case study is retrieved from a network drive based on some of the data transmitted from the vessel through a 4G transmitter aboard the vessel. The 4G transmitter was set up and is administered by NTNU Ålesund, which provided access to the network drive for this project (Holmeset, 2019). To get access to the network drive, the user must have a valid network connection to NTNU, either through physical presence or through a virtual private network (VPN). NTNU uses Cisco as VPN provider. To connect to a network drive, this NTNU guide for Windows can be used. Similar guides for Mac OS and Linux can be found here and here, respectively. The directory path filled out in step three of the process can be found here. If the network drive was added successfully, it should show up in the file explorer of your operating system. When attempting to access the network drive, the user will be prompted to log in via their NTNU credentials. After adding credentials, the user should have read access to the R/V Gunnerus network drive.

A.2 Python, pip, and Virtual Environments

For both the modeling API and the backend application in Flask, Python, `pip`, and virtual environments will be used to initiate and run the projects. `pip` is a package manager for Python used to install dependencies, and virtual environments are used to maintain project-related packages in a single directory, preventing incompatibility issues regarding package versions in other Python projects. Other package managers can be chosen, but for this project, `pip` will be used.

Python can be downloaded from the official website, python.org/downloads. Be aware that TensorFlow, which is used extensively for both applications, only work with Python releases 3.5-3.8, and it is essential to download a 64-bit version as 32-bit versions are not supported (TensorFlow, 2020). The modeling API and web application is known to work with Python 3.8 and above.

After downloading Python, project dependencies are installed through `pip`, which is integrated into Python for versions 3.4 and higher. However, if `pip` is not installed, there are countless resources online to help installing `pip`. To check if `pip` is already installed, run

```
>pip help
```

in the command line, which will return an error if no valid installation of `pip` is found. Here, the `">"` symbol is used to mark the command line interface symbol,

and is not actually written out. Remember to upgrade `pip` to its latest version. For Windows this is done by the command

```
>python -m pip install --upgrade pip
```

It is recommended to use virtual environments when installing dependencies in Python, as to not interfere with other Python projects on your machine. There are several packages that can be used to handle virtual environments, such as `virtualenv`, which is installed by running

```
>pip install virtualenv
```

Now, a virtual environment can be initiated in the desired directory (which should be the Python project root directory) by running

```
>virtualenv venv
```

in the command line. Here, `venv` is the chosen name of the virtual environment, but this can be changed to an environment name of choice. To activate the virtual environment, run

```
>venv/scripts/activate
```

in the same directory as the location of `venv`. If the activation has succeeded, `(venv)` will be visible at the start of the command line prompt. To deactivate the virtual environment, run

```
>deactivate
```

While the virtual environment is active, the required Python packages for the project can be installed with the `pip` package manager.

A.3 Modeling API for Anomaly Detection

The following section is intended to help setting up the modeling API environment in Python. The application in its entirety is found as a repository on GitHub (Alvsaker, 2020*a*). Setting up the environment is similar to setting up other Python projects.

A.3.1 Installing the project

First of all, the modeling API project must be copied or downloaded to a local machine. This is done by navigating to the GitHub repository and pressing the "clone"-button. The user can eiher choose to clone the repository, which will instantiate a new `git`-project, or download the project as a `.zip`-folder. After downloading the directory, open a command line tool of choice and navigate to the root directory of the downloaded project. The project directory structure for the modeling API was given in Figure 7.1.

As mentioned in Appendix A.2, it is suggested to use a virtual environment for handling necessary modules locally. With a virtual environment active, run

```
>pip install -r requirements.txt
```

in the root project directory to install required modules. If all requirements are installed successfully, and a Python version supporting TensorFlow is installed, the user should be able to run the project through

```
>python manage.py
```

from the command line. Note, however, that it is necessary to alter the user settings in the `manage.py`-file.

A.3.2 Using the modeling API

All of the coding files in the modeling API are heavily commented to ease the use of the modeling tool. It is recommended to use the project files themselves to familiarize with the project. Overall, a user of the API have to interact with at least two of the files given in the modeling API structure in Figure 7.1, namely the `manage.py`-file and `model.py`-file. `manage.py` is used to run the project. In the file, all necessary module imports are included, and the user is able to handle all project settings from the same file. Creating machine learning models is done in the `model.py`-file, which is located in the `modeling`-directory. The `model.py`-file consists of a modeling template with empty functions representing the general structure the user should maintain while using the modeling API. The modeling template is given in Listing 7.1.

As the template shows, developing models in the API is divided into four distinct parts. Suggested return attributes for the different parts are also included. If the user is not sure where to start, it is recommended to use the `model_example_lstm.py`-file as guidance, where a functioning LSTM model structure has been implemented. The example model uses the exact same function structure as the `model.py` template. As described in the `manage.py`-file, it is possible for users to run the program with the example model by changing which model file the `manage.py`-file references. This is easily assessed by inspecting the different module imports in `manage.py` and changing the call from the `model.py` module to the `model_example_lstm.py` module. If `git` is used, it is also possible to change to the `example_model` branch[23] and run `manage.py` from the modeling example branch.

A.4 Web Application for Anomaly Detection

The following section is intended to help setting up the web development environment in Flask and React. The application in its entirety is found as a repository on GitHub (Alvsaker, 2020*b*). The procedure is not different from similar projects, and if some part of the set up procedure fails, there are many resources available online.

First of all, the modeling API project must be copied or downloaded to a local machine. This is done by navigating to the GitHub repository and pressing the "clone"-button. The user can eiher choose to clone the repository, which will instantiate a new `git`-project, or download the project as a `.zip`-folder. After downloading the directory, open a command line tool of choice and navigate to the root directory of the downloaded project. A condensed view of the project directory structure can be seen in Figure 8.3, where the root directory is divided into a `flask-backend` and `react-frontend` directory. When downloading or cloning the project, the `venv` and `node_modules` directories will be missing, as these are package dependencies that have to be installed locally. There are also some additional files in the project, but the structure seen in Figure 8.3 represent the files and directories relevant for development.

A.4.1 Flask backend

All parts of the web application related to the Flask backend is located in the `flask-backend` directory. It is recommended to create a virtual environment in this directory, which can be done as described in Appendix A.2. After creating and activating a virtual environment, the necessary packages can be installed through

```
>pip install -r '../requirements.txt'
```

[23] The branch can be found here.

This will install the packages defined in `requirements.txt`. Since the file containing package requirements is located in the root project directory, the `'../'` notation must be included. If, however, the installation is performed from the root project directory, running

```
>pip install -r requirements.txt
```

is required instead. Since the backend makes use of a PostgreSQL database hosted by Heroku, the access to the database must be added to the backend application file. In production, the access is handled automatically by Heroku's configuration variables, and the secret key and database URL are set in the Flask application file through

```
app.secret_key = os.environ.get('SECRET')
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ['DATABASE_URL']
```

In development, however, these configuration variables are added manually for this project. Changing the configuration variables is done by navigating to the Flask application file, `api.py`, in the `flask-backend` directory, and changing the secret key and database URL configurations found in the application as

```
# Add Heroku configuration variables here:
app.secret_key = "ADD_SECRET_KEY_HERE"
app.config['SQLALCHEMY_DATABASE_URI'] = "ADD_DATABASE_URL_HERE"
```

Hard-coding the configuration variables like above is not optimal, as the configuration variables change routinely. Therefore, for development over time, it is suggested to create a separate Heroku application where the owner can manage the configuration variables in a more flexible way. It is also possible to share access to the Heroku application through e-mail address, which would allow access to individual configuration variables. As a quick fix, the configuration variables can be changed to the values given here.

If the packages are installed successfully and the configuration variables have been changed, the Flask backend can be started by running

```
>flask run
```

in the command line from the `flask-backend` directory. By default, the Flask app will run on port 5000.

A.4.2 React frontend

For the React frontend, `Node.js` will be used for handling all client-side server activities, and must be installed to receive necessary command line functionality for JavaScript. `Node.js` can be downloaded for the appropriate operating system from the official website, nodejs.org/download. To check if `Node.js` has been installed successfully, run

```
>node -v
```

in a terminal, which should provide the current Node version. It is possible to use Node's native package manager, `npm`, but for better performance, `yarn` is a better option. `yarn` can be installed through `npm` with

```
>npm install yarn
```

After installing `yarn` (or deciding to stick with `npm`) navigate to the `react-frontend` directory in the root project directory. To install the necessary Node modules, run

```
>yarn install
```

This command will search the `package.json` file in the `react-frontend` directory, install the modules found, and add these to a new directory `node_modules`. If all modules have been succesfully installed, a server for the React application can be created by running

```
>yarn start
```

in the command line. By default, the React application will run on port 3000. If both the Flask and React applications have been initiated successfully, the web application should be up and running on `localhost:3000/`, and the start page seen in Figure A.1 should be visible.

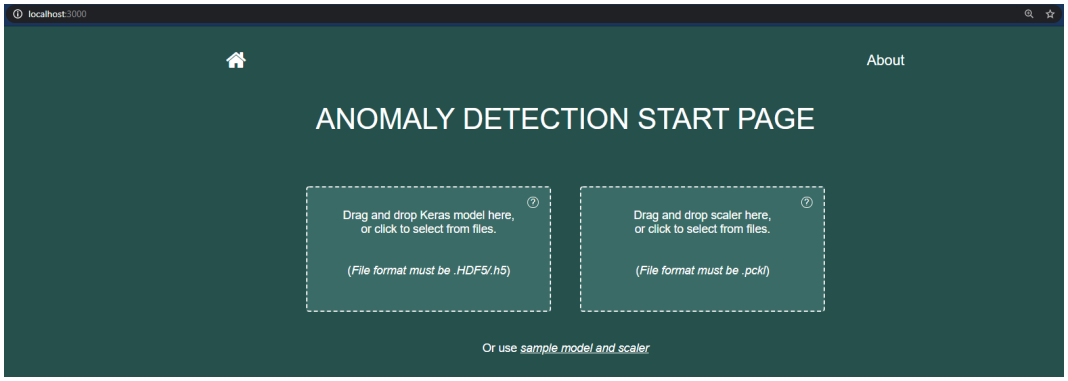


Figure A.1: The start page should be visible from `localhost:3000/` if the React application were launched successfully.

B Cloud Computing Services

Disclaimer: the following section was part of the pre-project delivered in the fall of 2019, and the contents have not been altered since then.

Cloud services refer to any remote service made available via the Internet through a cloud computing service provider. Similarly, cloud computing describes on-demand network access to shared computing resources (Mell and Grance, 2011), which facilitates the existence of all cloud services. As opposed to on-premise services, cloud services are hosted on the centralized servers of the service provider. Although there exist viable alternatives – for instance, peer enterprises with reduced environmental impact and energy usage (Gupta and Awasthi, 2009) – cloud computing is becoming the de facto mode of computing for industry applications. Figure B.1 display the types of clouds and overarching services of cloud computing. Private clouds are often used by solitary businesses through private networks for improved security and ability to perform big data analytics. Hybrid clouds can be used to prevent cloud-bursting through sudden activity spikes, where private clouds are combined with public clouds that manage activities that exceed expected behavior (Leavitt, 2013).

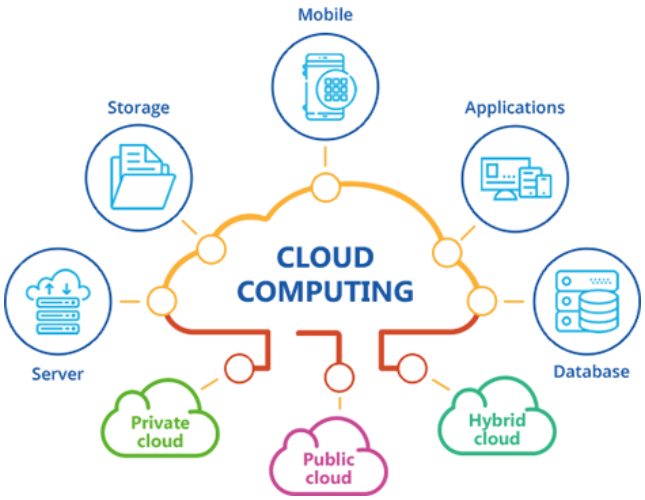


Figure B.1: An overview of the types of clouds that comprise cloud computing, as well as their overarching services. Courtesy of FullToTech (2019).

By outsourcing services covered by cloud computing, clients receive a reliable solution from established companies who possess bandwidth capabilities and web security systems that are often unattainable for small companies and single-purpose applications. As cloud services have dynamic scalability, service providers can easily adjust to a client's demand. Cloud services are an important enabler for digital twins, providing a foundation for aspects related to data management, namely storage, access, sharing, computing, and analysis.

B.1 Cloud Computing Models

Different components within cloud computing are known as *Anything-as-a-Service* (XaaS). XaaS can be sub-divided into three cloud service models for *Software-*, *Platform-*, and *Infrastructure-as-a-Service* (Kavis, 2014), or SaaS, PaaS, and IaaS, respectively. The different models relate to the number of services outsourced to an external provider.

From a client perspective, the different models give access to:

- SaaS, on-demand software applications administered and ran at a remote location;
- PaaS, an environment for developing, running, and managing application;
- IaaS, computing resources and high-level APIs in a virtual environment.

The virtual machine is a common denominator for each cloud service model, providing processing power, memory, and permanent storage. The differences between the cloud service models – in addition to a division of responsibilities between client and vendor – is given in Figure B.2, according to Watts and Raza (2019). In addition to the XaaS models, an on-premise scenario is included where there is no involvement from a cloud service provider.

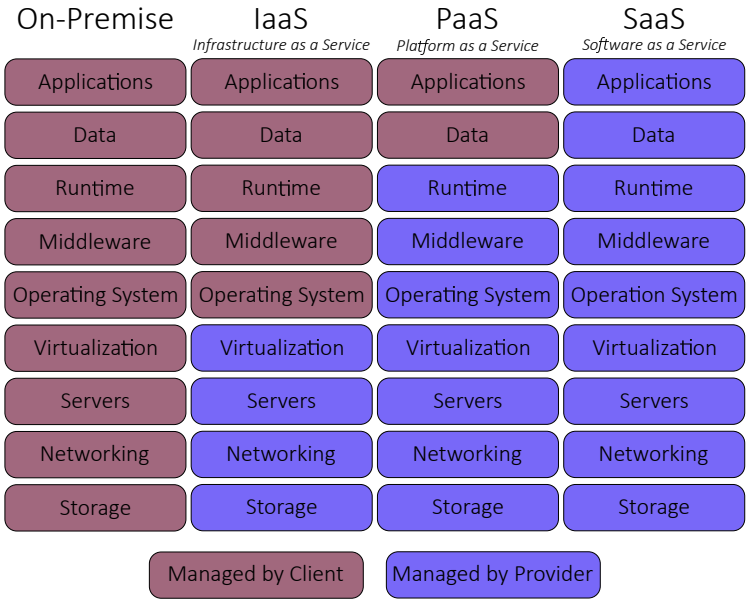


Figure B.2: Categories of XaaS, displaying the pyramid structure from an on-premise situation managed by the client, to a SaaS situation managed by the provider.

For XaaS, clients purchase services from major cloud service providers to fit their business needs. This is beneficial for clients, who do not need to develop their own solutions. Due to its dynamic scalability and service model hierarchy, XaaS is highly adaptable to client demands.

B.2 Comparison of Cloud Service Providers

The requirements of established infrastructure and capacity result in a cloud computing market dominated by large organizations (Gupta and Awasthi, 2009). This resembles an oligopoly, although some companies have pursued smaller, more niche segments. A market overview of different cloud computing providers and corresponding service segmentation has been made by Synergy Research Group[24]. The market overview shows that for the cloud service models of interest – SaaS, PaaS, and IaaS, as well as private cloud hosting – American organizations Amazon, Microsoft, and

[24]Synergy Research Group provides periodic reports of market development in cloud-related markets (Synergy Research Group, 2019)

Google are at the forefront of the competition. Table B.1 shows some of the main providers and their corresponding competitive segment(s). Other relevant vendors include IBM, Oracle, Salesforce, and Rackspace. The Chinese vendors Alibaba and Tencent have considerable market shares, however, they are mostly targeting domestic markets.

Table B.1: Overview of the most prominent cloud computing providers, together with their respective competitive market segment(s) (Serrano et al., 2015; Synergy Research Group, 2019).

Cloud Service Provider	**Competitive segment(s)**
Alibaba Cloud	IaaS & PaaS, hosted private clouds
Amazon Web Services	IaaS & PaaS, public clouds
Google Cloud Platform	IaaS & PaaS, hosted private clouds, Enterprise SaaS
IBM Cloud	Hosted private clouds
Microsoft Azure	IaaS & PaaS, hosted private clouds, Enterprise SaaS
Oracle Cloud	Enterprise SaaS, IaaS & PaaS, hosted private clouds
Rackspace Cloud	Hosted private cloud services
Salesforce	Enterprise SaaS
Tencent Cloud	IaaS & PaaS

Each of the main providers have benefits and drawbacks to their solutions which must be weighed against individual organizations' needs for cloud computing. Table B.2 gives an overview of PaaS and IaaS services[25]. As the first company to offer cloud computing services in 2006 (Miller and Veiga, 2009), Amazon Web Services (AWS) is the most dominant, but Azure and Google Cloud Platform (GCP) have had larger market growths in recent years.

Table B.2: Overview of PaaS, IaaS, and virtual private cloud (VPC) for Amazon, Microsoft, and Google. Each company has several SaaS, such as Amazon Connect, Microsoft Office 365, and Google G Suite.

Service	**Amazon Web Services**	**Microsoft Azure**	**Google Cloud Platform**
PaaS	Elastic Beanstalk	Azure Cloud Services	App Engine (GAE)
IaaS	Elastic Compute Cloud (EC2)	Azure Virtual Machines	Compute Engine (GCE)
VPC	Amazon VPC	Virtual Network (VNet)	Google VPC

AWS is mostly concerned with public clouds, which is not ideal for digital twins. GCP and Azure target many of the same segments and have some overlapping services. GCP is not as established but benefit from Google's experience and corresponding services within big data, analytics, and ML (Harvey and Patrizio, 2019). Azure is well-established and favored by businesses that already use Microsoft services, as Azure emphasizes seamless compatibility and integration. The large and mostly pre-existing user-base, a variety of applications and services, and private cloud hosting make Azure suitable for businesses.

[25] A more complete overview of services for the main providers is given in IntelliPaat (2019).

Currently, the cloud computing services provided by Amazon, Microsoft, and Google are different, but they may become commoditized with time (Miller and Veiga, 2009), rendering the services interchangeable. A fungible development is common for digital services (H. Øverby, & J. A. Audestad, 2018), exemplified by the file-hosting SaaS solutions OneDrive and Dropbox, which are near indistinguishable from a customer's point of view.

B.3 Edge Computing

For CPSs, data should be transmitted with sufficiently low latency to fulfill their intended purposes. If computations are carried out in centralized cloud servers, there may be significant delay. For IoT devices and CPSs, new data may be produced in large volumes at great velocities. If this data is supposed to be used for feedback in decision-support, centralized cloud servers may become problematic, since data must be queried, uploaded, processed, downloaded, and applied to make appropriate decisions. It is possible to reduce latency by introducing edge clouds as a supplement to centralized clouds, illustrated in Figure B.3. Such edge servers represent a decentralization of services and will serve as intermediary nodes moving computation power and data storage closer to where traffic originates.

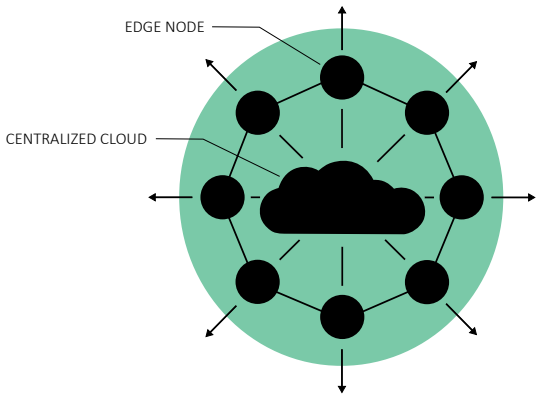


Figure B.3: Centralized cloud server connected to multiple edge nodes, which serve as close-proximity cloud servers for connected users, reducing data transmission latency for IoT devices and CPSs.

The processing speed of centralized clouds are superior to edge clouds, but the transmission speed from, for instance, IoT devices and CPSs becomes a bottleneck due to limitations in speed of data transportation (Shi et al., 2016). Since both data producers and data consumers are included at the edge node, unit-level CPSs and digital twins will benefit from edge computing (Qi et al., 2018). The edge nodes can connect to multiple devices simultaneously, as well as other nodes.

B.4 Big Data and Storage Considerations

When transmitting data from a multitude of signals at high frequency, the amount of data will ramp up quickly. Laney (2001) highlighted the need to control three of the main characteristics of big data, namely *Volume*, *Velocity*, and *Variety*. Later, several additional characterizations have been considered, most notably *Veracity* and *Value* (Cano, 2014; Firican, 2017), which will be denoted V^5.

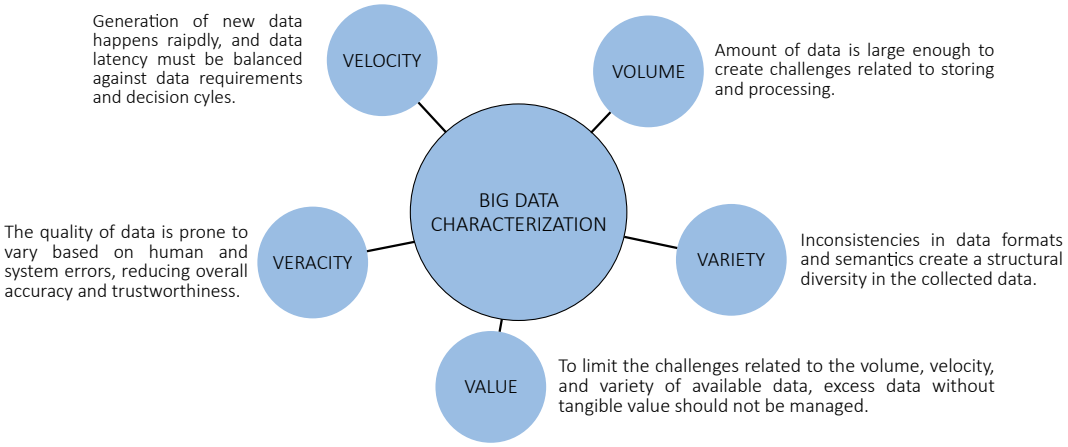


Figure B.4: V^5 of big data characterization.

These characterizations, which present ever-increasing challenges as IoT devices and CPSs become even more prominent, represent a prevalent challenge for digital twins. Thus, considerations related to V^5 are important when establishing data management systems, especially related to the realization of CPSs through digital twins.

C Data Ecosystems

Disclaimer: the following section was part of the pre-project delivered in the fall of 2019, and the contents have not been altered since then.

Since there are many different aspects of ship operation, it has become a trend for the maritime industry service providers to offer a wide variety of data services to their customers, typically managing activities and providing operation insight through data-driven services. These services are cloud-based solutions based on the presented cloud computing service models from Appendix B.1. Some companies have also created data platforms, intending to offer a complete ecosystem for all aspects of operation, such as secure cloud storage, applications for data insight, and analytics and learning. The software and system realization part of an R/V Gunnerus DTI can, to some extent, be achieved by using an appropriate data ecosystem.

The cloud computing service providers – consisting of large technology companies such as Amazon, Microsoft, and Google – provide the foundation for smaller companies that operate in a wide variety of industries. These companies do not necessarily have the capacity or competence to develop self-sustained software infrastructures or platforms, making existing cloud computing services desired. The value chain from the cloud computing service to a data platform user with a service mediator in-between is illustrated in Figure C.1.

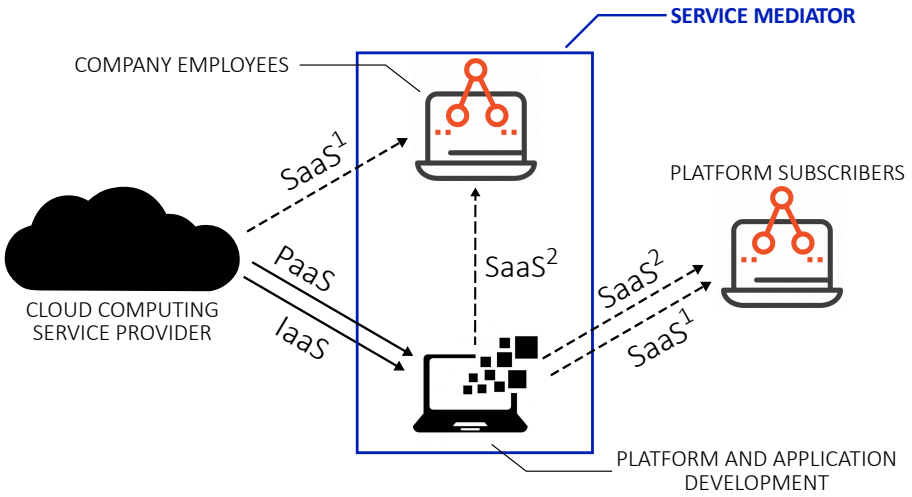


Figure C.1: Company mediator for distributing SaaS through a data platform. The dotted lines indicate that the receiving end chooses what to receive based on a pool of on-demand services. $SaaS^1$ includes applications developed by the cloud computing service provider, hosted on their platform, whereas $SaaS^2$ are developed by the service mediator, hosted on the service mediator's platform.

In this scenario, the service mediator receives all three cloud computing service models, IaaS, PaaS, and SaaS, from a provider. The IaaS is the foundation for the data ecosystem developed by the service mediator, which enables creating and maintaining applications based on the service mediator's desired operating system, middleware, and runtime environment, as evident from Figure B.2. The data ecosystem can then distribute on-demand SaaS to company employees internally, and to customers externally. In addition, pre-existing SaaS developed by the cloud computing service

provider is available to the company employees and platform subscribers if desired. If a service mediator utilize both IaaS and PaaS from a cloud computing service provider, a hybrid data ecosystem is achieved, in contrast to an ecosystem where only IaaS is utilized.

Typical mediators will be industry service providers, such as through Kongsberg's *Kognifai*, DNV GL's *Veracity*, Maersk's *TradeLens*, Siemens' *Xcelerator*, and *Cognite*. In this section, the data ecosystems provided by Kongsberg and DNV GL will be considered. Kongsberg is responsible for a substantial part of the instrumentation onboard R/V Gunnerus, including the DP system and AZ-PM thruster system. As Kongsberg is responsible for much of the data generated onboard, there exists an inherent compatibility with their data platform, Kognifai. DNV GL has been involved with preceding R/V Gunnerus digital twin initiatives, and is a leading partner in the development of the Open Simulation Platform (OSP), which, to some extent, will be customized to fit DNV GL's data platform Veracity (Pedersen and de Carvahlo, 2019).

C.1 Comparing Kognifai and Veracity

In many ways, Kognifai and Veracity are similar. The data ecosystems were both launched in the first quarter of 2017, and are built on top of Microsoft Azure. The Azure cloud computing service is responsible for the ecosystems' infrastructure through its IaaS, Azure Virtual Machines. Customers can choose desired SaaS through a marketplace, which exists both in Kognifai and Veracity. In addition to providing on-demand applications, SaaS, the data ecosystems facilitate secure data management, sharing, and storage. Both platforms are open, which means that external users are encouraged to develop own applications. External applications will have to be developed based on the PaaS of the data ecosystem and must be implemented and launched following the ecosystems' guidelines. The services of the ecosystems are available for both employees internally and customers externally, and access is administered by the PaaS. A comparison of available functions for Kognifai and Veracity is seen in Table C.1 (Mittal, 2019; Sverdrup, 2017).

Table C.1: Comparison of Kognifai and Veracity as data ecosystems.

	kognifai	VERACITY
Built on top of Azure	✓	✓
Based on Azure Virtual Machines (IaaS)	✓	✓
Has developed in-house PaaS	✓	✓
Has marketplace for SaaS	✓	✓
Data management	✓	✓
Secure storage and sharing	✓	✓
Third-party solutions	✓	✓
Centralized cloud solutions	✓	✓
Edge-solutions	✓	✓
Supports data analytics, ML, and AI	✓	✓
Open ecosystem	✓	✓
Can be integrated with hardware	✓	✗
Will have OSP compatibility	✗	✓
Focus on digital assurance and class solutions	✗	✓
Focus on supporting commercial products	✓	✗

As illustrated, many of the functions are supported by both ecosystems. The dis-

similarities are based on specific applications in each respective marketplace – which will not be elaborated in this report – as well as the overall vision of each ecosystem. Whereas DNV GL is a class society, Kongsberg is a commercial company focusing on selling advanced technological equipment to a range of industries. Thus, it is reasonable that Kognifai is customized to enhance their products. Comparatively, Veracity is intended to promote digital innovation and collaboration for the entire maritime industry, in addition to own interests of digital assurance and class solutions. Since many of the systems onboard R/V Gunnerus are supplied by Kongsberg, there is a clear benefit of the ability to integrate hardware into the overall ecosystem. On the other hand, the Core Simulation Environment (CSE), model catalogue, and all platform mechanisms related to OSP will be hosted on Veracity (DNV GL, 2019*b*).

DNV GL has previously been involved in initiatives related to developing an R/V Gunnerus digital twin, and has been positive to continuing the collaborative efforts (Mittal, 2019). Kongsberg has vested interests in R/V Gunnerus due to the installed systems onboard, especially related to the AZ-PM thrusters, which represents new technology that is under development and testing. Kongsberg have several employees working with the vessel, its instrumentation, and data communication. Unlike DNV GL, Kongsberg is a pure commercial company. Still, they have often been involved in academic projects with the intent of furthering technology and research, which is similar to DNV GL.

C.2 Alternative Approaches

In addition to the presented ecosystems, a Maritime Data Space (MDS) is being developed by SINTEF as a data exchange and sharing platform, which takes data governance and shared database problems into consideration (Berre and Ørnulf Rødset, 2018). NTNU has also started a research data initiative, intended to provide a storage, archive, and data management plan for employees and students (NTNU, 2019). The initiative is, however, deemed too underdeveloped at the moment.

C.3 Discussing the use of Data Ecosystems

There are advantages and disadvantages to using external cloud computing services and data ecosystems. The most appealing advantage – which is also the underlying reason cloud computing services exist – is that an established IaaS shifts the focus from software architecture and development, to platform and service development that directly contribute to the digital twin functionality. The biggest drawback is the loss of control, flexibility and predictability. Systems may change without notice, causing loss of functionality or the need to make big changes to maintain integrity. Data ecosystems, such as Kognifai and Veracity, offer SaaS which can provide immediate functionality to the vessel. Nevertheless, these ecosystems are driven by commercial companies, and the academic purpose of an R/V Gunnerus digital twin is not necessarily facilitated by available applications. Since the explored data ecosystems are open, it is possible to develop and launch applications independently of the ecosystem SaaS. Thus, since it greatly reduces the threshold for establishing a digital twin foundation, it is advised to utilize data ecosystems, at least in the developing phases.

D Prediction Results for Second Exhaust Temperature Signal

Here, the same plots as presented in the result section of Section 7 is shown for the second exhaust temperature, which resulted in an anomaly threshold value of $T_a = 14.0$. The results are similar to the ones for the first exhaust temperature, but the predictions perform worse.

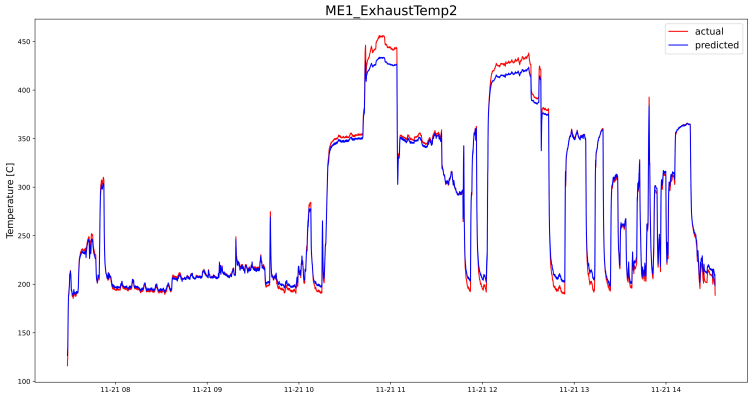


Figure D.1: Prediction plot for the first exhaust temperature time series.

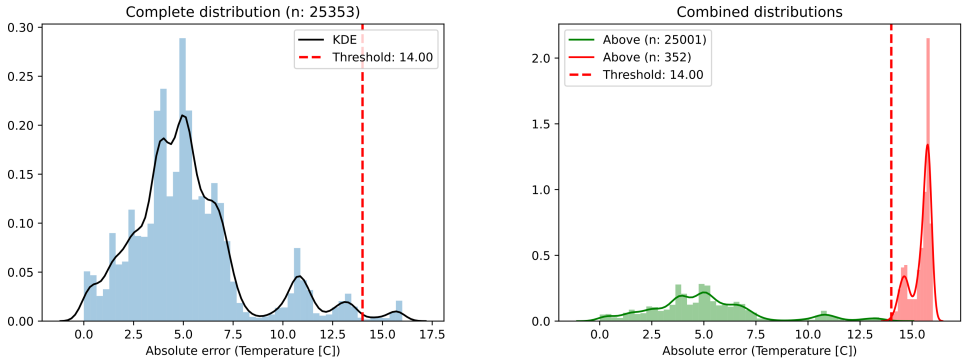


Figure D.2: Distribution plots for predicted time series error together with a threshold value of $T_a = 7.0$.

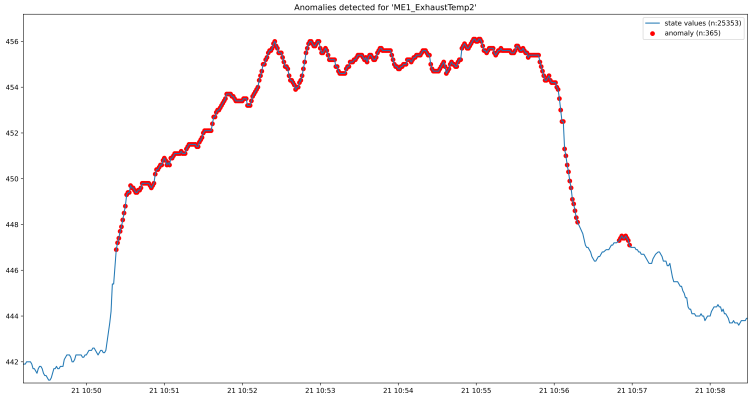


Figure D.3: Time series plot with detected anomalies marked.

E Additional Web Application Results

Additional features of the web application are documented here through snapshots. The website is available at https://rv-gunnerus-anomaly-detection.com/ and the code is documented in a GitHub repository (Alvsaker, 2020*b*).

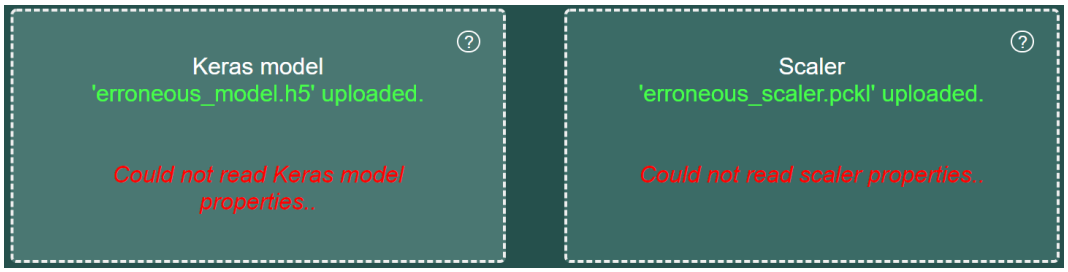


Figure E.1: Example of erroneous file uploads.

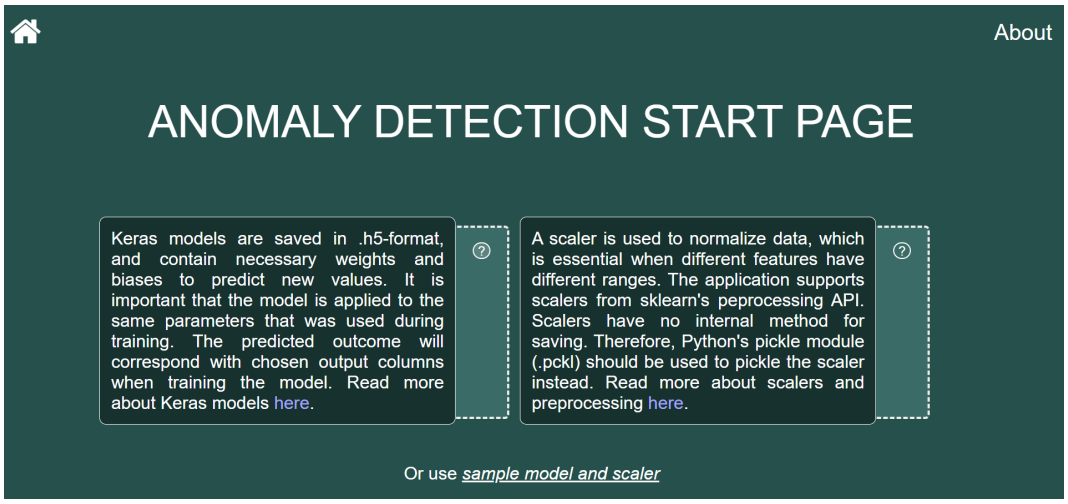


Figure E.2: Additional information about file uploads on hover.

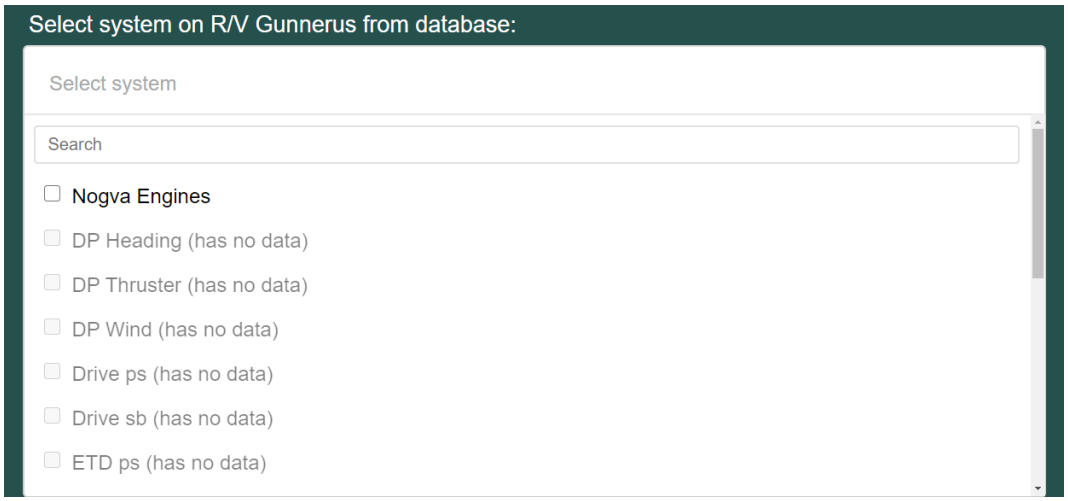


Figure E.3: Selecting system on R/V Gunnerus.

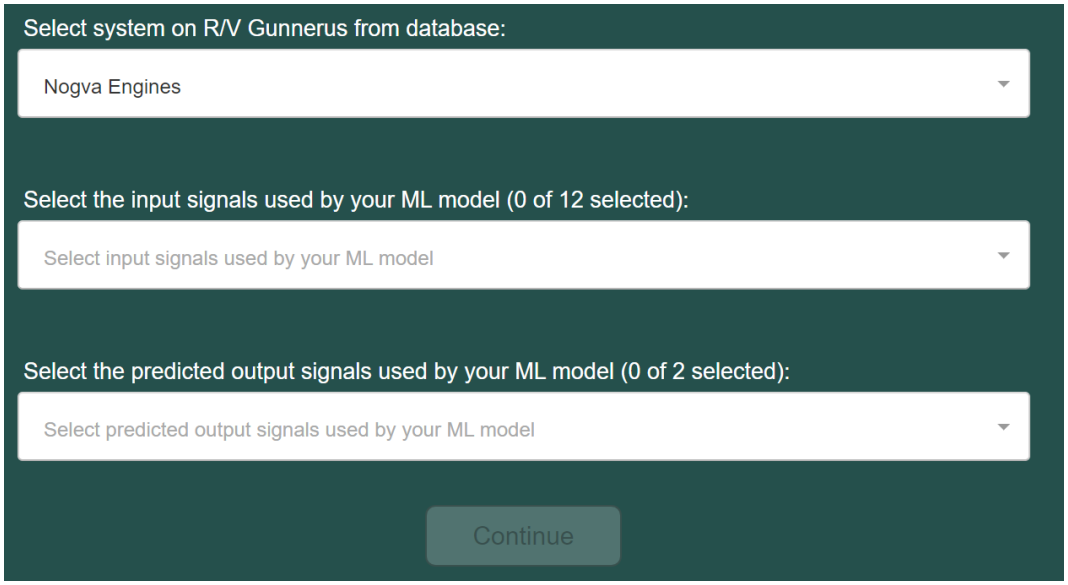


Figure E.4: Selecting input and predicted output signals.

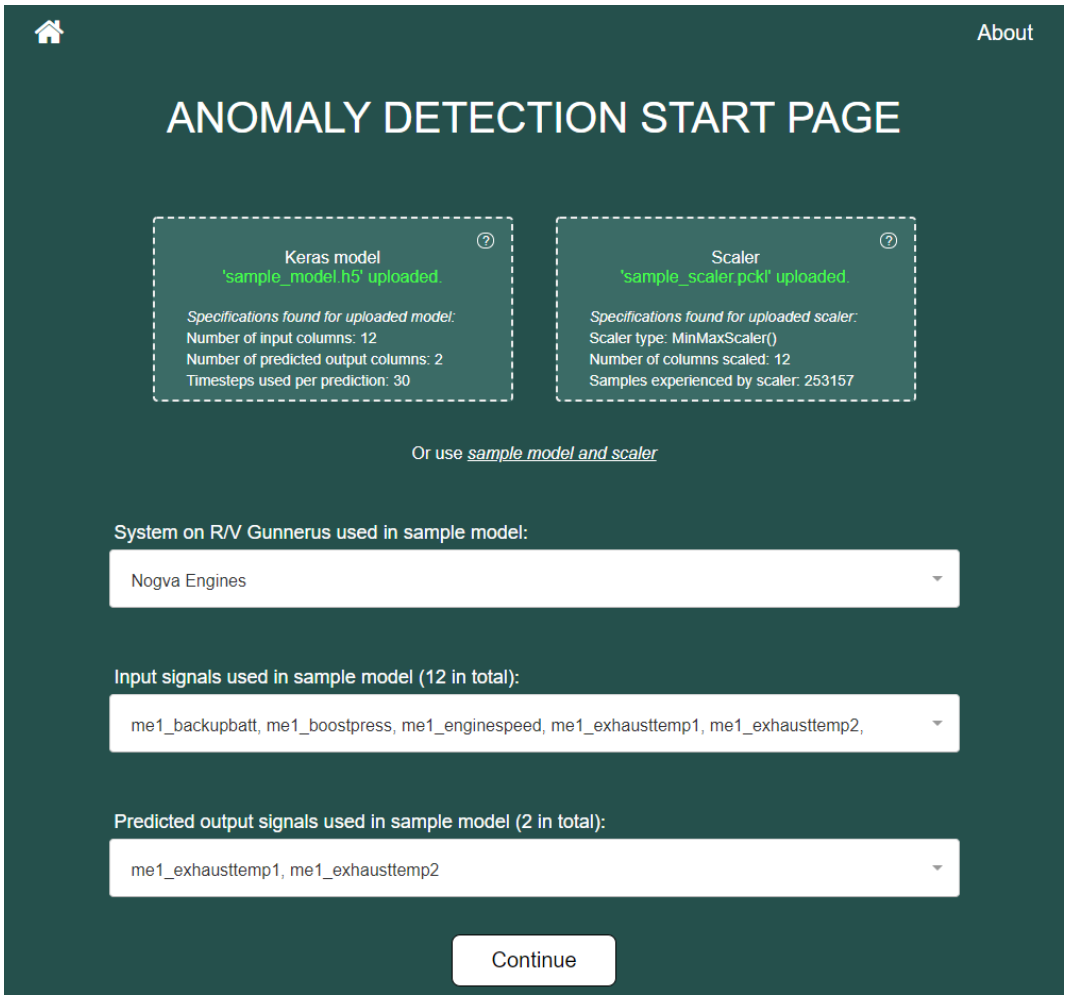


Figure E.5: Automatic configuration when using example files.

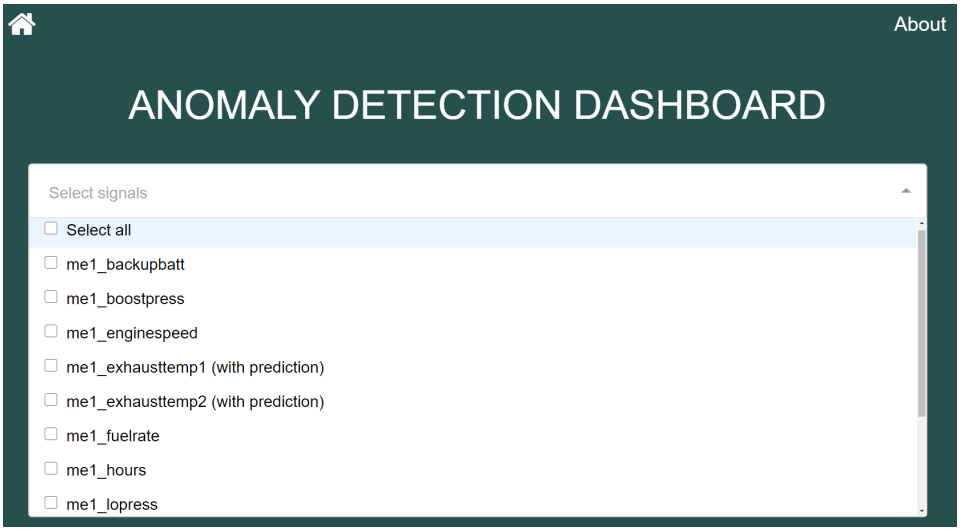


Figure E.6: Signal selection for charting.

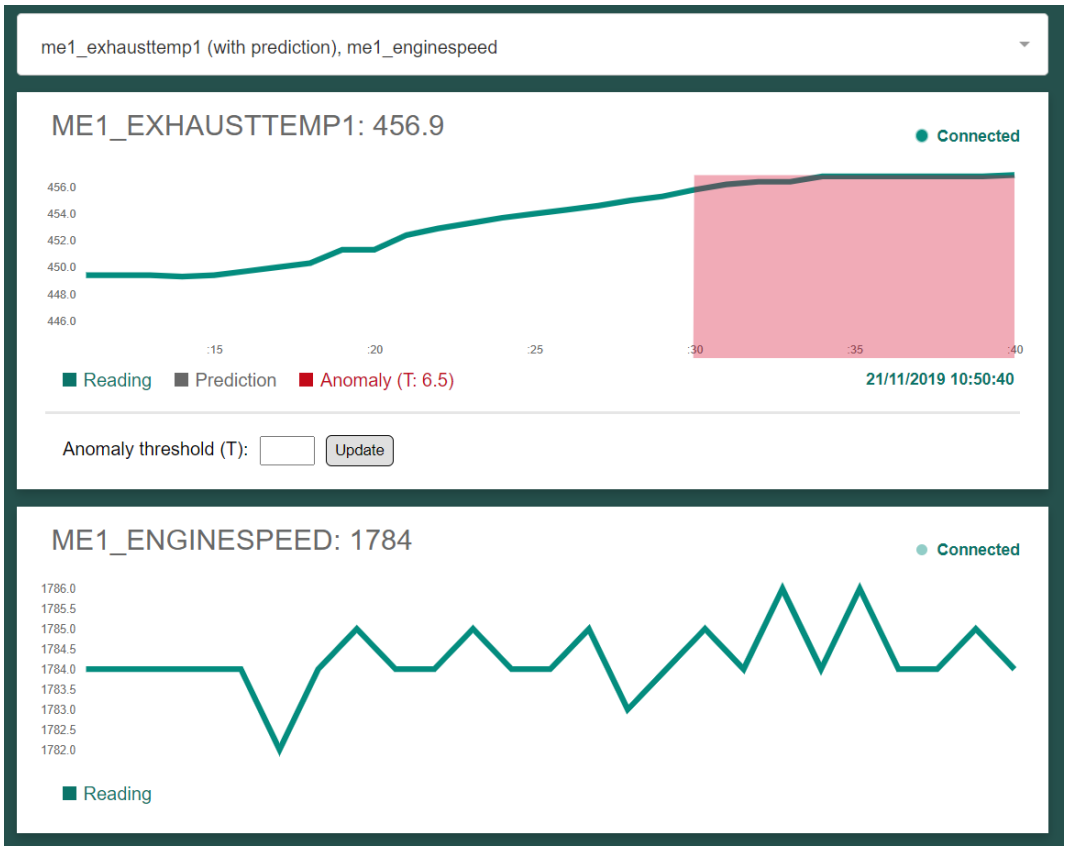


Figure E.7: Disabled prediction series, and showing a chart for a signal that is not part of the predicted output signals.

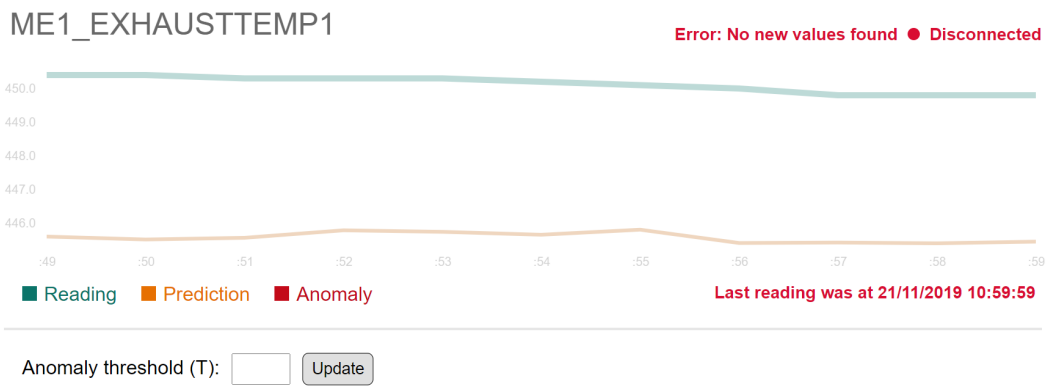


Figure E.8: Error display when no new values are found. The last valid reading is also shown.

Figure E.9: About page.

F Supplementary Code

F.1 Flask Application File

Here, the Flask application file `api.py` is documented. The file is also found in the GitHub repository for the web application (Alvsaker, 2020*b*).

Listing F.1: Main application file for running Flask backend.

```
from werkzeug.utils import secure_filename
from models import *  # all table classes and function get_table_classes()
from tensorflow.keras.models import load_model
import os
import pickle
import time
from threading import Event, Thread

import eventlet
import numpy as np
from flask import Flask, redirect, render_template, request, url_for
from flask import session as storage
from flask_socketio import SocketIO, emit, send
from sqlalchemy import create_engine, inspect
from sqlalchemy.orm import load_only, session

# Instantiate Flask application
app = Flask(
    __name__,
    static_folder='../react-frontend/build',
    static_url_path='/')

eventlet.monkey_patch()  # ensure appropriate threading behavior

in_production = os.environ.get('IN_PRODUCTION', None)
if in_production:
    # Add secret key
    app.secret_key = os.environ.get('SECRET')
    # Configure PostgreSQL Heroku database with the database URL
    app.config['SQLALCHEMY_DATABASE_URI'] = os.environ['DATABASE_URL']
else:
    # Add Heroku configuration variables here:
    app.secret_key = "ADD VALID SECRET KEY HERE"
    app.config['SQLALCHEMY_DATABASE_URI'] = "ADD DATABASE URL HERE"

# Directories for uploaded files and sample files:
UPLOADS_DIR = os.path.join(app.instance_path, 'uploads')
SAMPLES_DIR = os.path.join(app.instance_path, 'samples')
os.makedirs(UPLOADS_DIR, exist_ok=True)

# Prevent unnecessary console warning:
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Initiate database engine:
engine = create_engine(
    app.config['SQLALCHEMY_DATABASE_URI'],
    pool_size=40,
    max_overflow=80)

# Initialize PostgreSQL Heroku database with SQL Alchemy:
db = SQLAlchemy(app)
db.init_app(app)

# Initialize Flask-SocketIO:
socketio = SocketIO(app)
socketio.init_app(app, cors_allowed_origins='*')

INTERVAL = 1  # fetch interval from database in seconds (data frequency)

# Instantiating variables for global scope:
thread = Thread()  # define thread object
```

```
62  thread_stop_event = Event()  # define threading-event (used for termination)
63
64
65  @app.route('/')
66  def index():
67      return app.send_static_file('index.html')
68
69
70  @app.route('/systems', methods=['GET', 'POST'])
71  def get_systems():
72      """Return a list of systems based on the entry tables in the PostgreSQL
73      database, which are instantiated through SQL Alchemy in 'models.py'."""
74      # Dict of all model classes from models.py with table names as key:
75      table_classes = get_table_classes()  # from models.py
76      systems = {}
77      tables = engine.table_names()
78      for table in tables:
79          if not table_classes[table].query.first():
80              systems[table] = False  # if result is None (empty table)
81          else:
82              systems[table] = True  # if result is not None (non-empty table)
83      # return boolean dictionary with tables as keys
84      return {'systems': systems}
85
86
87  @app.route('/signals/<system_table>', methods=['GET', 'POST'])
88  def get_signals(system_table):
89      """Return a list of signals based on the entry table in the PostgreSQL
90      database, which is instantiated through SQL Alchemy in 'models.py'. The
91      function takes a string 'system_table' as input variable, which is the
92      name of the selected system from the database."""
93      # Properties of current system table, used to retrieve table columns:
94      table_props = inspect(engine).get_columns(system_table)
95      signals = []  # placeholder for signals
96
97      # Name of each column of the selected system:
98      for col in table_props:
99          signals.append(col['name'])
100     return {'signals': signals}
101
102
103 @app.route('/start_thread')
104 def start_thread():
105     """Start thread for processing parallelism."""
106     if not thread.is_alive():
107         thread.start()
108     return {'thread_alive': True}
109
110
111 @app.route('/reload', methods=['GET'])
112 def stop_thread():
113     """If the client window is reloaded and a thread is active in the
114     background, this function discontinues the thread, meaning that upon page
115     reload, a new thread will be initiated with new properties."""
116     global engine, thread, thread_stop_event
117     if thread.is_alive():
118         thread_stop_event = Event()  # define stop event
119         thread_stop_event.set()  # set stop event
120         del thread  # delete thread to prevent double-initiation
121         thread = Thread()  # create new thread object
122         print(f'Upon page reload, the thread has been discontinued.')
123         return {'thread_stopped': True}
124     engine.dispose()
125     return {'thread_stopped': False}
126
127
128 @app.route('/keras_model/<use_sample>', methods=['GET', 'POST'])
129 def save_uploaded_model(use_sample):
130     """Receives uploaded Keras model file from frontend client. If use_sample
131     is true, the sample model, uploaded locally, is used instead."""
```

```
132     # Delete model path if already defined in flask storage session:
133     storage.pop('keras_model_path', None)
134     if use_sample == 'true':  # load sample model
135         storage['keras_model_path'] = os.path.join(
136             SAMPLES_DIR, 'sample_model.h5')
137         keras_model = load_model(storage['keras_model_path'])
138     else:  # load uploaded model based on filename request from client
139         file = request.files['file']  # request from client
140         storage['keras_model_path'] = os.path.join(
141             UPLOADS_DIR, secure_filename(
142                 file.filename))
143         # save to UPLOADS_DIR with provided filename
144         file.save(storage['keras_model_path'])
145         print(f"succesfully saved model to '{storage['keras_model_path']}'")
146         try:
147             # Attempt to load Keras model with native Keras function:
148             keras_model = load_model(storage['keras_model_path'])
149         except BaseException:
150             # Something went wrong during Keras model loading:
151             storage['model_properties'] = False
152     try:
153         # Attempt to set properties through native Keras attributes:
154         storage['model_properties'] = {
155             'inp': keras_model.input_shape[2],
156             'out': keras_model.output_shape[1],
157             'timesteps': keras_model.input_shape[1]
158         }
159     except BaseException:
160         # Something went wrong when reading model properties:
161         storage['model_properties'] = False
162     return {'fileprops': storage['model_properties']}
163
164
165 @app.route('/scaler/<use_sample>', methods=['GET', 'POST'])
166 def save_uploaded_scaler(use_sample):
167     """Receives uploaded sklearn scaler file from frontend client. If
168     use_sample is true, the sample scaler, uploaded locally, is used
169     instead."""
170     # Delete scaler path if already defined in flask storage session:
171     storage.pop('scaler_path', None)
172     if use_sample == 'true':  # load sample scaler
173         storage['scaler_path'] = os.path.join(
174             SAMPLES_DIR, secure_filename('sample_scaler.pckl'))
175         with open(storage['scaler_path'], 'rb') as f:
176             scaler = pickle.load(f)[0]  # retrieve scaler from pickle object
177             # Set scaler properties:
178             scaler_properties = {
179                 'type': str(scaler),
180                 'features': scaler.n_features_in_,
181                 'samples': scaler.n_samples_seen_
182             }
183     else:  # load uploaded scaler based on filename requested from client
184         file = request.files['file']  # request from client
185         storage['scaler_path'] = os.path.join(
186             UPLOADS_DIR, secure_filename(file.filename))
187         # Save to UPLOADS_DIR with provided filename
188         file.save(storage['scaler_path'])
189         print(f"succesfully saved scaler to '{storage['scaler_path']}'")
190         try:
191             with open(storage['scaler_path'], 'rb') as f:
192                 # Attempt to load scaler with native pickle function:
193                 scaler = pickle.load(f)
194             # Scaler part of modeling API exported file, containing
195             # [scaler, df_train, df_test]:
196             try:
197                 scaler = scaler[0]
198             except BaseException:
199                 pass  # scaler uploaded independently of modeling API:
200             # Attempt to set properties through native Sklearn attributes:
201             scaler_properties = {
```

```
202                 'type': str(scaler),
203                 'features': scaler.n_features_in_,
204                 'samples': scaler.n_samples_seen_
205             }
206         except BaseException:
207             # Something went wrong when loading scaler or reading properties:
208             scaler_properties = False
209     return {'fileprops': scaler_properties}
210
211
212 @app.route('/create_thread/<system>/<input_cols>/<output_cols>',
213            methods=['GET', 'POST'])
214 def initiate_thread(system, input_cols, output_cols):
215     global thread, thread_stop_event
216     # convert strings from client to lists with comma-delimiter:
217     input_cols = input_cols.split(',')
218     output_cols = output_cols.split(',')
219     timesteps = storage['model_properties']['timesteps']
220     if not thread.is_alive():
221         print(f'Creating thread object..')
222         thread = ValueThread(system, input_cols, output_cols, timesteps)
223         thread.scaler = get_scaler(storage['scaler_path'])
224         thread.keras_model = get_model(storage['keras_model_path'])
225         thread.X_pred = thread.get_first_input_values()
226     else:
227         print(f'Attempting to connect while thread is active')
228     return {'thread_created': True}
229
230
231 @socketio.on('connect')
232 def on_connect():
233     """SocketIO connect event."""
234     global thread, thread_stop_event
235     sio_id = request.sid
236     thread.sio_id = sio_id  # socket IO identification
237     thread_stop_event.clear()
238     print(
239         f"New client '{request.args.get('system')}' connected with "
240         f"connection id: {sio_id}"
241     )
242
243
244 @socketio.on('disconnect')
245 def disconnect():
246     """SocketIO disconnect event."""
247     global engine
248     sys_id = request.args.get('system')
249     engine.dispose()
250     print('Engine disposed after disconnecting')
251     print(f"Client '{sys_id}' has been disconnected.")
252
253
254 class ValueThread(Thread):
255     """Handles signal thread, which allows the dynamic relationship between
256     the database and client, allowing for a realtime transmission of data
257     through websockets."""
258
259     def __init__(self, system, input_cols, output_cols, timesteps):
260         """Instantiate class object."""
261         self.system = system
262         self.input_cols = input_cols  # inputs used for prediction in model
263         # Get table model for current system (used for querying database):
264         table_classes = get_table_classes()
265         self.sys_table = table_classes[system]
266         # Variable for columns that will be queried from database:
267         self.fetch_columns = ['time']
268         self.fetch_columns.extend(self.input_cols)
269         self.output_cols = output_cols  # predicted output columns of model
270         self.output_indices = self.get_output_indices()
271         self.delay = INTERVAL  # frequency of updates
```

```
272         # model timesteps used
273         self.timesteps = timesteps
274         # Set initial index (database follows a not-null approach, meaning the
275         # first row index is 1):
276         self.index = self.timesteps  # initial index
277         # Get the input values for the first timestep-values:
278         super(ValueThread, self).__init__()
279 
280     def get_first_input_values(self):
281         """First input values corresponding with the Keras model timesteps.
282         If the model uses n timesteps, the function fetches the first n values
283         of the requested system from the database, and returns a scaled numpy
284         array-vector of size n, which is used for predicting new values. The
285         input_columns variables coincide with the columns used to create the
286         model, and are the same as the chosen inputs during model and scaler
287         uploading."""
288         timesteps = self.timesteps
289         input_cols = self.input_cols
290         system = self.system
291         with app.app_context():  # enable database access through SQLAlchemy
292             X_pred = []  # Placeholder for each timestep of signal values
293             for i in range(1, timesteps):
294                 ordered_values = []
295                 values = db.session.query(self.sys_table).options(
296                     load_only(*input_cols)).get(i).get_dict()
297                 for col in input_cols:
298                     ordered_values.append(values[col])
299                 # append current timestep values
300                 X_pred.append(ordered_values)
301         # Return transformed values with shape (timesteps, features):
302         db.session.close()
303         return self.scaler.transform(X_pred)
304 
305     def get_output_indices(self):
306         """Returns a list with the position of each output column in the
307         ordered input list. During reverse transform of data (from normalized
308         to original magnitude of values) the order of the signals matters.
309         Since the original data is fetched, and can be transmitted to the
310         client before normalization, only the predicted values must be reverse
311         transformed before being sent to the client."""
312         indices = []
313         for pred_sig in self.output_cols:
314             indices.append(self.input_cols.index(pred_sig))
315         return indices
316 
317     def get_data(self):
318         """Continuously emit information about the current database index to
319         the client, which fetches data based on the index."""
320         # Create placeholder list for predicted values, which must be filled
321         # with all input columns to reverse transform properly:
322         pred_vals_list = [[None] * len(self.input_cols)]
323         while not thread_stop_event.is_set():
324             with app.app_context():
325                 start_time = time.time()
326 
327                 # Query values as dictionary containing input_cols and time:
328                 try:
329                     values = db.session.query(self.sys_table).options(
330                         load_only(*self.fetch_columns)).get(
331                         self.index).get_dict()
332                     values['time'] = str(values['time'])
333                     ordered_values = []
334                     # Order values correctly according to model, and exclude
335                     # time:
336                     for col in self.input_cols:
337                         ordered_values.append(values[col])
338                     # Scale new values:
339                     scaled_values = self.scaler.transform([ordered_values])
340                     # Add new values to the end of X_pred:
341                     self.X_pred = np.append(
```

```
342                         self.X_pred, scaled_values, axis=0)
343
344                     # Predict values at the next timestep (the input must be a
345                     # numpy array with shape (1,timesteps, features)):
346                     pred_values = self.keras_model.predict(
347                         np.array([self.X_pred]))
348                     pred_value_counter = 0
349                     # Add values to placeholder list for predictions:
350                     for index in self.output_indices:
351                         pred_vals_list[0][index] = pred_values[0][
352                             pred_value_counter
353                         ]
354                         pred_value_counter += 1
355                     # Inverse transform predicted values:
356                     pred_vals_list = self.scaler.inverse_transform(
357                         pred_vals_list)
358                     # Add predicted values from placeholder list to values
359                     # dict:
360                     pred_value_counter = 0
361                     for pred_col in self.output_cols:
362                         pred_key = f'{pred_col}_pred'
363                         values[pred_key] = pred_vals_list[0][
364                             self.output_indices[pred_value_counter]
365                         ]
366                         pred_value_counter += 1
367
368                     socketio.emit('values', values)
369                     self.X_pred = self.X_pred[1:]
370                     # Time used for prediction, manipulations, and emission:
371                     calculation_time = time.time() - start_time
372                     if calculation_time > 1:
373                         sleep_time = 0
374                     else:
375                         sleep_time = self.delay - calculation_time
376                     time.sleep(sleep_time)
377                     self.index += 1
378
379                 except BaseException:
380                     thread_stop_event.set()
381         db.session.close()
382         try:
383             thread_stop_event.set()
384         except BaseException:
385             pass
386         socketio.emit('values', False)
387         engine.dispose()
388         print('Engine disposed after threading')
389
390     def run(self):
391         self.get_data()
392
393
394 def get_scaler(path):
395     """Loads the pickle-file containing data scaler specified by the
396     'scaler_path'."""
397     with open(path, 'rb') as f:
398         scaler = pickle.load(f)
399     try:
400         return scaler[0]
401     except BaseException:
402         return scaler
403
404
405 def get_model(path):
406     """Loads the Keras model-file specified by the 'keras_model_path'."""
407     return load_model(path)
408
409
410 if __name__ == '__main__':
411     app.run(host='0.0.0.0', debug=False, port=os.environ.get('PORT', 80))
```

F.2 Removing False Anomaly Outliers

At times, unwanted anomalies are detected by the ML model. Such anomalies can represent outliers or rapid changes in state values upon engine start-up. To prevent unwanted anomalies, which cause the threshold value necessary to successfully detect true anomalies to ramp up, an approach of evaluating the neighboring values surrounding a detected anomaly was implemented in the modeling API. In this approach, a neighborhood, $\mathcal{N}$, of size $|\mathcal{N}| = n$, is used to determine the validity of a detected anomaly. In short, the algorithm presented below checks for a given number of consecutive anomalies. If the required number of consecutive anomalies is not met, the anomaly will be deemed an outlier and is removed from the list of anomalies.

Listing F.2: Functions used to implement outlier algorithm for false anomalies.

```
1
2  def get_anomaly_range(df_loss: pd.DataFrame, threshold: float, size: int = 0
3                        ) -> pd.DataFrame:
4      """Calls the neighborhood checking function with a boolean dataframe where
5      loss values above the input threshold is True."""
6
7      df_bool = df_loss > threshold
8      return _check_neighboring_bools(df_bool, size=size).values
9
10
11 def _check_neighboring_bools(df_bool: pd.DataFrame, size: int = 0
12                               ) -> pd.DataFrame:
13     """Checks if the neighborhood surrounding a boolean True value is also
14     True. If not, the boolean value is changed to False. The function helps
15     remove false outliers which will otherwise trigger unnwanted anomalies.
16     The input variable 'size' defines the number of elements to include
17     in the neighborhood of each timestep. If size=2n, timestep t will
18     result in a neighborhood containing values t-n, t-n+1,...,t,...,t+n-1,t+n,
19     thus yielding a neighborhood of 2n+1 elements (including t). If desired
20     neighborhood size is odd with neighborhood=2n+1, the function will
21     include an extra comparison value at a later timestep, resulting in the
22     values t-n,...,t,...t+n+1."""
23
24     df_nh = _get_neighborhood(df_bool, size)
25     df_center_anoms = df_nh.all(axis='columns')
26     df_center_anoms_nh = _get_neighborhood(df_center_anoms, size)
27
28     return df_center_anoms_nh.any(axis='columns')
29
30
31 def _get_neighborhood(df_bool: pd.DataFrame,
32                       neighborhood: int = 0) -> pd.DataFrame:
33     df_neighborhood = pd.DataFrame(df_bool.values)
34     """Gets the neighborhood on each side of a selection column through the
35     dataframe shift-function. If the 'neighborhood' input value is odd, an
36     extra value is added below the selected column, skewing the symmetry by
37     one entry."""
38     for i in range(1, int(neighborhood / 2) + 1):
39         df_neighborhood[f'-{i}'] = df_bool.shift(-i, fill_value=False).values
40         df_neighborhood[f'+{i}'] = df_bool.shift(i, fill_value=False).values
41     if neighborhood % 2:
42         j = int((neighborhood + 1) / 2)
43         df_neighborhood[f'+{j}'] = df_bool.shift(j, fill_value=False).values
44     return df_neighborhood
```

Johan Fredrik Alvsaker

R/V Gunnerus Digital Twin Infrastructure

NTNU
Norwegian University of
Science and Technology