Development of your own chatbot for the automated processing of inquiries from students at a university of applied sciences

Introduction

StudentDesk is a chatbot designed to support students with common university-related tasks. It simplifies access to important information and services, helping students manage their academic activities efficiently.

A chatbot is a program or collection of instructions whose main goal is to mimic human communication. With these chatbots, humans may converse with computers in normal languages like English [1].

Chatbot functionalities:

1. Defining intent via Rule-based approach
2. Defining complexity of request via Natural Language Processing methods (Tokenization, Lemmatization, Part-of-Speech Tagging)
3. Defining entities via Named Entity Recognition model
4. Defining intent via Machine Learning approach
5. Database integration

Chatbot query types:

1. Update personal information (first name, last name, address)
2. Registration for exams
3. Deregistration from exams
4. Query exam status
5. Query exam grades
6. Query student profile

Technologies:

- spaCy – de_core_news_lg – building NER model
- tensorflow – Bidirectional LSTM – building ML model for intent classifier
- sqlite3 – database
- Flask API – API Integration

Data

There are three datasets to train the NER model (tokens.csv) and Intent Classifier model (chatbot_dataset.csv), and to define intent by Rule-based approach (rule_based_intents.json).

tokens.csv is the dataset for the NER model with two columns such as text and entities. There are 1078 rows, and the dataset is shown in Figure 1.

The column text contains a user request to chatbot, and extracted entities with their labels are written in column entities, extracted entities are separated by semicolon.

There are 7 entities such as matriculation_number, course_name, first_name, last_name, address, city, and post_code, and Figure 2 displays the distribution of each entity label.

	text	entities
0	Please change my first name to Carol and my last name to Wilson.	Carol:first_name;Wilson:last_name
1	Change my first name to John and my last name to Smith.	John:first_name;Smith:last_name
2	Set my first name as Emma and my last name as Brown.	Emma:first_name;Brown:last_name
3	Update my first name to Hans and my last name to Müller.	Hans:first_name;Müller:last_name
4	I would like to change my first name to Maria and my last name to Schulz.	Maria:first_name;Schulz:last_name

Figure 1. tokens.csv dataset

	Entity Label	Count
0	matriculation_number	452
1	course_name	452
2	first_name	300
3	last_name	300
4	address	300
5	city	300
6	post_code	300

Figure 2. The Distribution of each entity label

chatbot_dataset.csv is the dataset to train the machine learning model for intent classifier. It has two columns, namely text and intent. There are 2000 rows, and Figure 3 shows the dataset.

	text	intent
0	Please, change my first name to Carol	change_first_name
1	Please, change my first name to Emma	change_first_name
2	I would like my first name to be Liam	change_first_name
3	Change my first name to Mia	change_first_name
4	Can you update my first name to Noah?	change_first_name

Figure 3. chatbot_dataset.csv dataset

The column text stores requests to the chatbot, and the column intent stores the intent of the text. There are eight intent types including change_address, change_first_name, change_last_name, deregister_exam, query_exam_grade, query_exam_status, query_student_profile, and register_exam, and each intent has 250 examples.

	intent	count
0	change_address	250
1	change_first_name	250
2	change_last_name	250
3	deregister_exam	250
4	query_exam_grade	250
5	query_exam_status	250
6	query_student_profile	250
7	register_exam	250

Figure 4. Intents and their counts

rule_based_intents.json is a dataset to implement the Rule-based approach part of the chatbot. This dataset is used to define meta questions or text, for instance, “Hello!”, “Hi!”, “What are you?”, and “What do you do?”. Figure 5 shows intent and their patterns with responses. The count of the patterns and the responses are given in Figure 6.

	Intent	Patterns	Responses
0	greeting	hello, hi, good morning, good afternoon, hey, greetings	Hello! How can I assist you today?, Hi there! What can I do for you?
1	farewell	bye, goodbye, see you, take care, farewell	Goodbye! Have a great day!, See you later! Take care!
2	thanksgiving	thank you, thanks, appreciate it, grateful	You're welcome!, Glad I could help!
3	chatbot_identity	who are you, what are you, tell me about yourself, what is this, who am I talking to, what's your name	I am StudentDesk, your university chatbot. I'm here to assist you with any queries., I'm StudentDesk, your virtual assistant for university-related tasks.
4	chatbot_creation	who created you, who made you, who developed you, who built you, what's your origin	I was created by the university's development team to assist students like you., The university's IT team developed me to help answer your questions and simplify student services.

Figure 5. rule_based_intents.json dataset

	Intent	Pattern Count	Response Count
0	greeting	6	2
1	farewell	5	2
2	thanksgiving	4	2
3	chatbot_identity	6	2
4	chatbot_creation	5	2

Figure 6. Rule-based approach intens with their patterns and responses

Methodology

Rule-based Chatbots

Specifically, rule-based chatbots are designed to respond to pre-established enquiries. Users of this kind of chatbot are limited in their possibilities for input [2]. Chatbots can follow the rule-based approach by relying on pattern recognition rather than necessarily understanding human speech. When chatbots recognise specific words, phrases, or even actions that kick off a series of reactions, this is known as rule-based pattern recognition. Such rules have the advantage of being accurate, enabling developers to add and remove rules to handle novel circumstances and fix bugs with confidence [3]. According to Bahaddad, the rules-based chatbot operates in three stages such as Input Analysis, Response Generation, and Flow Control. In the first stage, the chatbot looks for particular words or phrases that match pre-established criteria in the user's input. Following the identification of a match, the chatbot reacts in accordance with the guidelines, frequently providing a predetermined response or instructing the user to carry out a certain action, such as clicking a button or picking an option. Finally, rule-based chatbots usually adhere to a decision tree. This implies that users have little freedom to stray from the predetermined courses as they are led through an organised series of questions or answers [4].

Machine learning-based Chatbots

Machine Learning (ML) based chatbots are designed to communicate with users in a human-like manner and are able to remember context and vocabulary [2]. Three essential processes are used by ML chatbots to operate, namely, Training Data, Model Learning, and Continuous Improvement. Firstly, large conversational datasets are used to train the ML Chatbot. Next, it learns to find trends and clear up ambiguity to get precise responses. Lastly, as more data is gathered, it improves through retraining [4].

Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on natural language communication between computers and people. It includes the capacity of

computers to meaningfully and practically process, analyse, and comprehend human language, enabling activities like sentiment analysis, translation, and conversational bots [4].
spaCy is an open-source Python library for sophisticated Natural Language Processing (NLP). SpaCy allows to create apps that process and "understand" massive amounts of text and is made especially for production use. It can be used to pre-process text for deep learning or to develop systems for information extraction or natural language understanding [5].

Long Short-Term Memory
Long Short-Term Memory (LSTM) is a popular kind of RNNs, that are made to identify patterns and time-dependencies in sequential data, including audio recordings, texts, and numerical time series [6]. When it comes to language learning, both context-free and context-sensitive, LSTM is a formidable force. For instance, dialogue systems, sometimes referred to as conversational agents, enable spoken interaction between humans and machines [7].

Experiments
Machine Learning Approach in Intent Classifier
Bidirectional LSTM (Long Short-Term Memory) is used to implement the intent classifier model for the chatbot.
The following libraries are used in the implementation:

- tensorflow-keras – building, training, and evaluating the neural network model
- scikit-learn – encoding labels and splitting data into training and testing sets

Before using the text from the dataset, it should be preprocessed. The preprocessing step contains converting all characters to lowercase, removing punctuation and digits using regular expressions, and stripping any leading or trailing whitespace.
A label encoder, LabelEncoder from scikit-learn library, is used to convert text labels (intents) into numeric labels. For instance, intent “change_address” can be 0, and intent “change_first_name” can be 1. It is needed, because the models cannot directly work with categorical data. Consequently, without encoding, models cannot effectively learn or predict categorical outputs like chatbot intents.
The dataset is split into training and testing (80% of the dataset is for training, and remaining, 20%, is for testing).
The next step is Tokenization. Tokenization refers to breaking down a piece of text (chatbot request) into tokens, in this case words. For example, the input text is “register for exam”, and this text is divided into individual words such as “register”, “for”, and “exam”.
In order to handle the words that were not in the training data, but are in the user's query, the special token (<OOV>), out-of-vocabulary words, is used. It is helpful to work with unseen words, otherwise the tokenizer would ignore the unknown objects. For instance, there is the text “register for exam” in the training data, and if the user enters “register for test”, the chatbot does not know how to handle the word “test”. However, this word is mapped to an OOV token, and it preserves the context of the user input.
Following this the vocabulary is built by assigning a unique integer index to each word in the training data, “register” – 1, “for” – 2, “exam” – 3.

In order to learn the data, the text is converted into sequences. The training and testing text data is . Also, all sequences should have the same length, so they are padded, and they are padded at the end of the sequences. In addition, column intent is converted to vectors for multi-class classification.
The model contains Embedding Layer, Bidirectional LSTM Layers, Dense Layer, Dropout Layer, and Output Layer. Word indices are transformed into dense vectors (meaningful representations) in Embedding Layer. Bidirectional LSTM Layers are important to complete understanding of the context. When it goes through the text, it goes both directions (from start to end, and from end to start). Dense Layer is used to map intents to target classes. Dropout Layer prevents overfitting. Finally, the Output Layer predicts the probabilities of each class for the input text.
Early Stopping method is used to prevent overfitting and improve training efficiency by halting training when the model's performance on a validation set stops improving.
The training of the Intent Classifier model can be performed in the train-model.ipynb file, which is in the notebooks directory.

Named Entity Recognition

Overlapping entities

Before using test data to recognize named entities, it is necessary to give them a specific form. For instance, the dataset row contains the following: text – “Please change my first name to Carol and my last name to Wilson.”, intent – “Carol:first_name;Wilson:last_name”, and it is formatted as this – (“Please change my first name to Carol and my last name to Wilson.”, {'entities': [(31, 36, 'first_name'), (57, 63, 'last_name')]}). Because, spaCy’s NER requires the start and end indexes of the entities’ characters to identify. However, there were some problems with overlapping of entities in converting. For example, there is the text – “Please change my address to Dortmunder 12, 44143 Dortmund”, intent – “Dortmunder 12:address;Dortmund:city;44143:post_code”. But, extracting entities from the text has the challenge, as the word “Dortmund” can be found in two places as the address (28, 37) and the city (49, 58). The text of these challenges are extracted from the dataset.

Training NER model

spaCy’s de_core_news_lg (a pre-trained German language model) is used during training NER model. It has the following characteristics:

- language – German
- genre – written text (news, media)
- size – 541 MB

Prior to selecting this model, experiments were conducted with the following models:

- en_core_web_sm
    - language – English
    - genre – written text (blogs, news, comments)
    - size – 12 MB
    - As it is English-language model, it incorrectly recognized German names and addresses
- de_core_news_sm

    - language – German
    - genre – written text (news, media)
    - size – 13 MB
    - this model had inaccuracies in some names and addresses
- de_core_news_md
    - language – German
    - genre – written text (news, media)
    - size – 42 MB
    - this model was better than previous one, but it had inaccuracies in some cases with complex text

For the purpose to train the model to identify entities, custom entity labels ("change_first_name", "change_last_name", etc.) are included into the NER pipeline.
The training model goes to the entire training data 200 iterations, and each time training data is shuffled randomly. It is necessary to compute the loss, and update its weights for improving the model's ability. The loss value of the 1st iteration – 383.025, and the loss value of the last iteration – 1.861 * 10^(-7).
The training of the NER model can be performed in the train-model.ipynb file, which is in the notebooks directory.

Rule-based Approach in Intent Classifier

A rule-based intent classifier was implemented to interact with metatext. It is responsible for greetings, goodbyes, or information about the chatbot itself (meta questions and text). This is implemented as follows. The json file contains templates for user requests and chatbot responses. When a user enters text into a chatbot, it checks the similarity of the specified text with the text in the file, and an answer is randomly selected accordingly.

Chatbot Implementation

Database Operations

There are tables such as students, student_address, courses, exams, student_exams, student_exam_grades. The insertions for these tables can be performed in database.ipynb Jupyter notebook
Separate methods for working with the database are implemented according to the functionality. Each method takes the necessary parameters, checks them against the database, and performs the operation. The check is based on the student's number, by the name of the course, whether the students have a course by a specific name, or whether there was an exam for this course.

User and Confirmation Context

A user and confirmation context was created to implement the chatbot's memory.
User context stores classified intent, detected entities, missed entities and user's previously entered personal information (matriculation number, first name, last name, course name, address, city, postal code).
Confirmation context stores summarization text, classified intent, and detected entities.

When a user enters the text to the chatbot, chatbot classifies the text by intent, detects entities, and defines missed entities. If required entities are missed in the text, the chatbot asks to provide them. When a user enters text, the chatbot checks if any missing entities exist in the user context, and if it exists, it tries to define. If all necessary information is in the text, the chatbot checks if it needs to be confirmed by the user. The functionalities are divided by reading and writing operations. Reading operations include changing personal information, registering for the exam, and de-registering from the exam. Writing operations include querying exam status, exam grades, and student profile. If the user's request needs to be confirmed, the summarization text is generated, and summarization text is stored in the confirmation context. When the user gives the response to the confirmation, the chatbot responds according to the user's text. After that, the chatbot's memory is cleared (the user context, except for the student's personal information, and the confirmation context), and when the user enters a query without specifying entities, the chatbot searches for this entity in the user's context and performs an action.

Reset commands

Special types of requests are defined to clear the memory at the user's request. If the user's query contains certain words to clear the memory, then all stored information in the context of the user and the confirmation will be deleted.

Complexity of the user's text

In order to send input text to the NER model or generate response by Rule-based approach, the complexity of the text is performed. This method uses the de_core_news_lg spaCy model to extract ke linguistic features. It checks the number of verbs, nouns, and content words, and according to their count decides the complexity.

The Block Diagram is shown in Figure 7. It describes how the chatbot works.
StudentDesk Block Diagram link - https://drive.google.com/file/d/1wB_-1D9ba8tljewck8B4VG4f9tdp7LBG/view?usp=sharing

Results

Intent Classifier

The efficiency metrics of the Intent Classifier's model are shown in Table 1. The efficiency of the model is calculated using testing data, 20% of the dataset. An accuracy of 99.5% indicates that the model has classified 398 of the 400 test intents. It can be seen in detail in the Confusion matrix, Figure 8, exactly which intents are incorrectly classified. The classified intents are plotted on the x-axis, and the true values are plotted on the y-axis. One intent text about registration for the exam was incorrectly classified as withdrawal from the exam. In addition, the intent to change the name was incorrectly counted as a request for a student profile. Figure 9 shows the accuracy of the model.

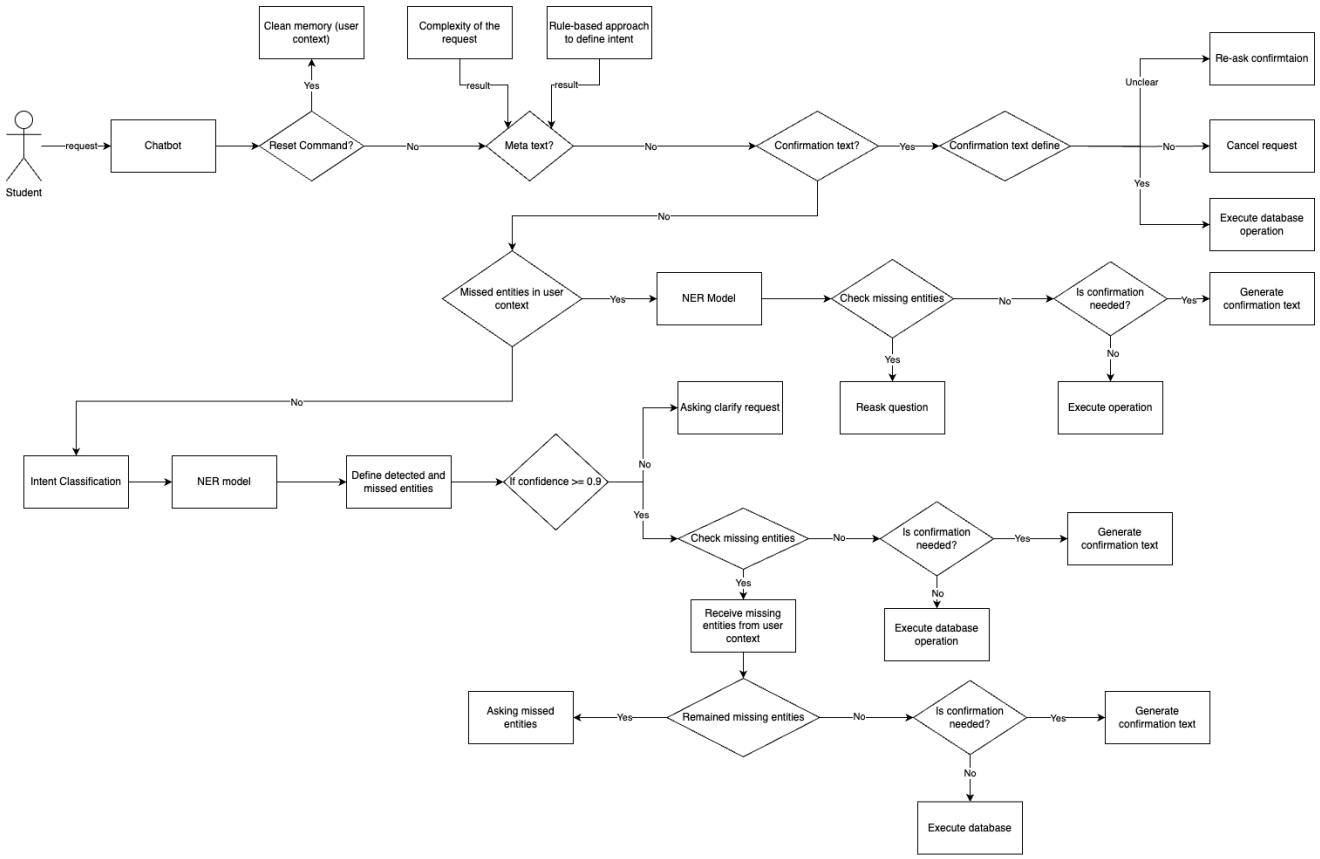

Figure 7. StudentDesk Block Diagram

Table 1. The efficiency metrics of the Intent Classifier's model

	Accuracy	Precision	Recall	F1-score
Intent Classifier Model	99.50%	99.51%	99.50%	99.50%

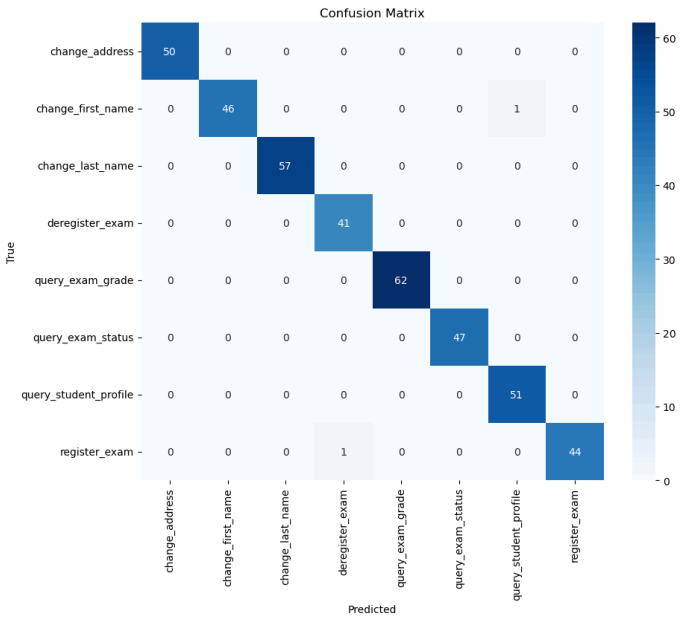

Figure 8. Intent Classifier Model Confusion Matrix

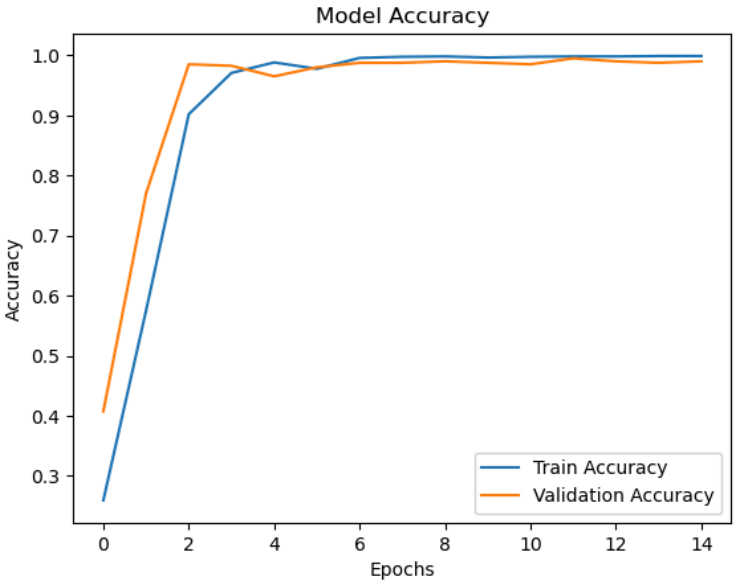

Figure 9. Intent Classifier Model Accuracy

NER

The performance indicators of the NER model are shown in Table 2. The precision metric is the result of how the model correctly recognized entities in the text, while a few non-existent entities were recognized incorrectly, and the recall is the result of how the model in most cases identified the required entities and sometimes missed entities that should have been identified. The F1-score is the average harmonic value of precision and recall.

Table 2. The efficiency metrics of the NER model

	Precision	Recall	F1-score
NER Model	98.96%	98.45%	98.70%

Figure 10 displays the user interface of the chatbot, how it starts a conversation and how it processes the user's text to display the exam results. The user entered the matriculation number and the course name, and the chatbot immediately provided the exam score.

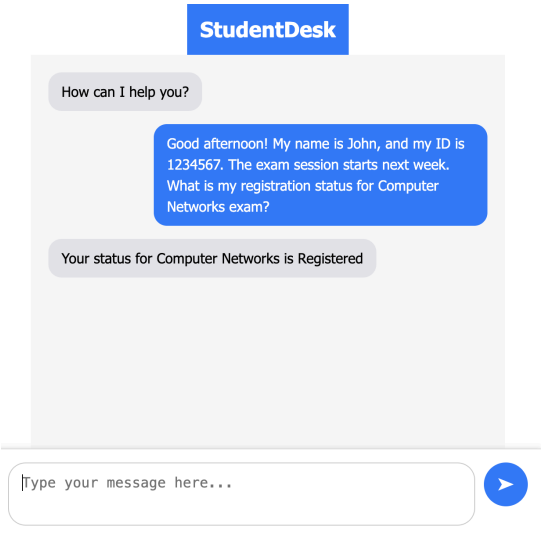

Figure 10. StudentDesk User Interface

Conclusion

A chatbot called StudentDesk was created to assist students with typical university-related duties. It makes key information and services easier to access, assisting students in effectively managing their academic workload. It contains both approaches to chatbots (rule-based and ML-based), and a trained NER model. The efficiency metrics of Intent Classifier model and NER model, accuracy, precision, recall, and f1-score, shows high results. The accuracy of the Intent Classifier model is 99.5%, which indicates that the model correctly classifies intents in most cases, and 98.70% of the F1-score for the NER model shows that the required intents are recognized correctly in most cases, however, there are several cases when the model incorrectly identifies intents or omits the correct intents. Moreover, there are the implementation of the rule-based approach for interaction with meta text, and database integration to read or write students' data.

References

1. Choudhary, P., & Chauhan, S. (2023). An intelligent chatbot design and implementation model using long short-term memory with recurrent neural networks and attention mechanism. Decision Analytics Journal, 9, 100359. https://doi.org/10.1016/j.dajour.2023.100359
2. Ramakrishna, K. & Maha, M. (2020). A Review on Chatbot Design and Implementation Techniques. International Research Journal of Engineering Science Technology and Innovation. 07. 2791.
3. Singh, J., Joesph, M. H., & Jabbar, K. B. (2019). Rule-based Chabot for student enquiries. Journal of Physics: Conference Series, 1228(1), 012060. https://doi.org/10.1088/1742-6596/1228/1/012060
4. Bahaddad, A. A. (2025). Increasing the degree of acceptability for Chatbots in technical support systems for educational sector by using ML and semantic web. Alexandria Engineering Journal, 116, 296–305. https://doi.org/10.1016/j.aej.2024.12.028
5. Spacy 101: Everything you need to know · spacy usage documentation. spaCy 101: Everything you need to know. (n.d.). https://spacy.io/usage/spacy-101
6. Lindemann, B., Müller, T., Vietz, H., Jazdi, N., & Weyrich, M. (2021). A survey on long short-term memory networks for time series prediction. Procedia CIRP, 99, 650–655. https://doi.org/10.1016/j.procir.2021.03.088
7. Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. Artificial Intelligence Review, 53(8), 5929–5955. https://doi.org/10.1007/s10462-020-09838-1