Digital Chemical Engineering 4 (2022) 100038

Contents lists available at ScienceDirect

Digital Chemical Engineering

journal homepage: www.elsevier.com/locate/dche

Original Article

Using artificial intelligence to support the drawing of piping and instrumentation diagrams using DEXPI standard

Jonas Oeing[a,*], Wolfgang Welscher[b], Niclas Krink[b], Lars Jansen[a], Fabian Henke[a], Norbert Kockmann[a]

[a] *Department of Biochemical and Chemical Engineering, Laboratory of Equipment Design, TU Dortmund University, Emil-Figge-Strasse 68, Dortmund 44227, Germany*
[b] *X-Visual Technologies GmbH, James-Franck-Straße 15, Berlin 12489, Germany*

ARTICLE INFO

Keywords:
Artificial intelligence
Process synthesis
P&ID
Process engineering
Detail engineering

ABSTRACT

The design and engineering of piping and instrumentation diagrams (P&ID) is a very time-consuming and labor-intensive process. Although P&IDs show common patterns that could be reused during development, the drawing is usually created manually and built up from scratch for each process. The aim of this paper is to recognize these patterns with the help of artificial intelligence (AI) and to make them available for the development and the drawing process of P&IDs. In order to achieve this, P&ID data is made accessible for AI applications through the DEXPI format, which is a machine-readable, manufacturer-independent exchange standard for P&IDs. It is demonstrated how deep learning models trained with DEXPI P&ID data can support the engineering as well as drawing of P&IDs and therefore decrease labor time and costs. This is achieved by assisted prediction of equipment in P&IDs based on recurrent neural networks as well as consistency checks based on graph neural networks.

List of symbols

$\boldsymbol{b}$	bias
$\boldsymbol{E}$	number of edges
$\boldsymbol{h_u}$	embedding/feature of a node u
$\boldsymbol{l}$	length of the directed walks
$\boldsymbol{k}$	layer of a GNN / iteration step in message passing
$\boldsymbol{m}$	message of an aggregated neighborhood of a node in a graph
$\boldsymbol{N}$	number of nodes
$\boldsymbol{u}$	arbitrary node u of a graph
$\boldsymbol{v}$	arbitrary node in the neighborhood of node u
$\boldsymbol{x}$	input
$\boldsymbol{y}$	output
$\boldsymbol{\alpha}$	weight by which the features of nodes influence each other

Introduction

The need for data-driven modeling and optimization is growing in the process industry due to increasing digitalization of processes and tools. Previous work already shows an acceleration of process development by agent-based environments, which can be understood as the first steps of intelligent process development (Batres et al., 1997). Further work shows the potential for using intelligent process information models by describing chemical plants and processes in terms of a machine-readable format (e.g. colored petri nets), which enables accessibility for deterministic algorithms (Zhao et al., 2005). This paper aims to develop applications that accelerate the development of P&IDs using artificial intelligence (AI). This requires an accelerated development and application of standardized and machine-readable file exchange formats to ensure a sufficiently large and highly available database for the application of AI in P&ID development. In the field of P&IDs, the DEXPI (Data Exchange in Process Industry) standard is becoming increasingly popular., as it enables the uniform description of P&IDs and ensures the vendor-independent exchange of information (Theißen and Wiedau, 2021). At the same time, DEXPI provides the possibility to be used as a platform for digital plant data in process industry (Wiedau et al., 2019), which can significantly reduce the development time of chemical and biotechnological production plants. Additionally, interoperability increases due to the continuous integration of DEXPI into existing engineering software (Fillinger et al., 2017).

DEXPI is a machine-readable P&ID exchange format under development by the DEXPI Initiative. The initiative consists of owner operators, engineering, procurement & construction companies, software vendors and research institutions. The latest data model and the associated DEXPI specification 1.3 (Theißen and Wiedau, 2021) were published in 2021. Within the specification, different international standards for

Abbreviations: AI, artificial intelligence; BRNN, bidirectional recurrent neural network; GNN, graph neural network; GRU, gated recurrent unit; LSTM, long short-term memory; MLP, multilayer perceptron; P&ID, piping and instrumentation diagram; PCE, process control equipment; RNN, recurrent neural network.

* Corresponding author.
E-mail address: jonas.oeing@tu-dortmund.de (J. Oeing).

https://doi.org/10.1016/j.dche.2022.100038
Received 16 March 2022; Received in revised form 30 May 2022; Accepted 12 June 2022

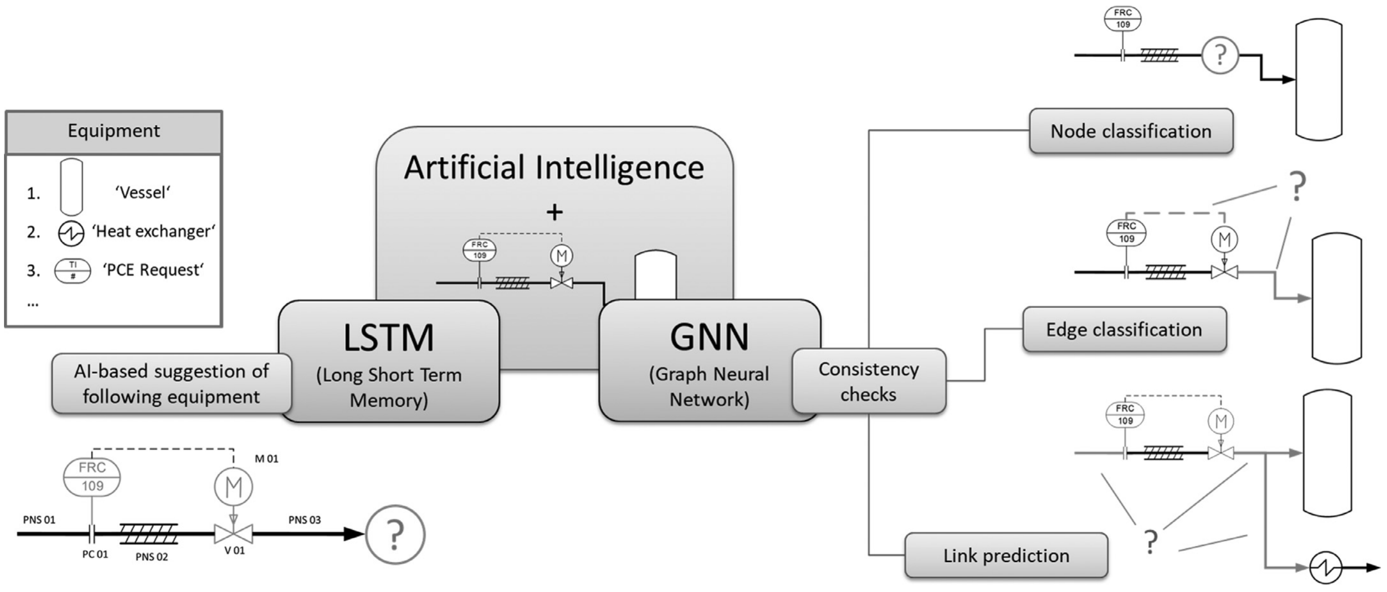


Fig. 1. Use cases of artificial intelligence to accelerate and improve the synthesis of P&IDs.

the description of engineering relevant data for P&IDs are combined (e.g. ISO 15926 (International Organization for Standardization, 2013), ISO 10628 (International Organization for Standardization, 2012a), IEC 62424 (International Electrotechnical Commission, 2016), ISO 10209 (International Organization for Standardization, 2012b). In particular, these include plant breakout structures, instrumentation, properties of equipment and components, and piping topology. The DEXPI information model is already offered by some manufacturers and is exchangeable via a Proteus XML schema (Proteus XML, 2017). At the same time, DEXPI provides the possibility to be used as a platform for digital plant data in process industry (Wiedau et al., 2019), which can significantly reduce the development time of chemical and biotechnological production plants. Additionally, interoperability increases due to the continuous integration of DEXPI into existing engineering software (Fillinger et al., 2017). The uniform and machine-readable format as well as the increasing acceptance of the DEXPI format in the process industry improve the potential for the application in the field of data science and allow the application of artificial intelligence (Wiedau et al., 2021).

In this paper, AI algorithms are trained to learn patterns in P&IDs using the DEXPI standard, which can be used as a basis for intelligent and efficient process development. To achieve this, DEXPI P&IDs are converted to graphs, hence are made available for graph theory methods. Based on these graphs two different AI-supported use cases for assisted P&ID synthesis are developed and explained in the following section. The first use case uses sequential processing by recurrent neural networks for the prediction of P&ID equipment. The second use case uses pattern recognition in P&IDs with the aid of graph neural networks (GNN) for consistency checks.

AI-based processing of P&IDs

Fig. 1 shows the two use cases identified in this paper for AI-assisted P&ID synthesis and their respective modeling approaches. In the first use case, a node prediction generates suggestions about subsequent components based on a recurrent neural network (RNN). These suggestions can support the user and decrease the time of the drawing process. The second approach uses graph neural networks (GNN), which are neural networks especially developed for the modeling of graphs. They can learn the topologies of process plants, which are stored in the form of a graph, and enable a consistency check during drawing by comparing the models with drawn P&IDs. The use of a neural network offers the advantage that patterns and rules for generating P&IDs can be learned from existing plant topologies. Therefore, no explicit heuristics need to be stored. Furthermore, the models can be adapted to the user's requirements and preferences by re-training them with data from the user. GNNs can be used to perform various classifications and predictions, which then can be used for consistency checking. The node classification allows for predicting information, such as the equipment classes of individual nodes to check whether components within a P&ID are present at meaningful positions. In contrast, edge classification focuses on the prediction of edge information. In relation to the P&ID, these are for example the connection types (piping or signal lines) or the information, whether a pipeline is insulated or heated. In parallel, a link prediction can be used, which provides information about the probability of a possible link between two components. This enables the validation of connections within a P&ID as well as the suggestion of connections during the drawing process. In the following, the preprocessing for converting the P&IDs into graphs as well as the two modeling concepts is introduced in more detail. The results for a node prediction via RNNs as well as a node classification based on a GNN is presented.

Preprocessing – DEXPI-2-graph

To make the structure of the P&ID available for further processing, the respective DEXPI file of the P&ID is converted into a graph using Python. The plant topology of the P&ID, including parameters relevant for modeling, is stored in a directed graph in the form of a GraphML file (GraphML Project Group, 2017). A directed graph, per definition, consists of a set N of nodes and a set E of edges. The edges are directed, meaning that each edge is defined by ordered pairs of nodes (start and end node) (Turau and Weyer, 2015). The P&ID to be processed is stored in the DEXPI format, which is based on a Proteus XML schema (Proteus XML, 2017) and contains three levels of information relevant for topology extraction. The *Equipment*, which contains a listing of the components present in the P&ID. The *PipingNetworkSystem,* which describes the piping system and the interconnections between the various equipment, and the *PipingComponents,* which contain components embedded in the piping system, like valves. The classes of equipment and components are uniquely defined in DEXPI via the *EquipmentClass* and *ComponentClass*. Furthermore, the *InstrumentationFunction* provides

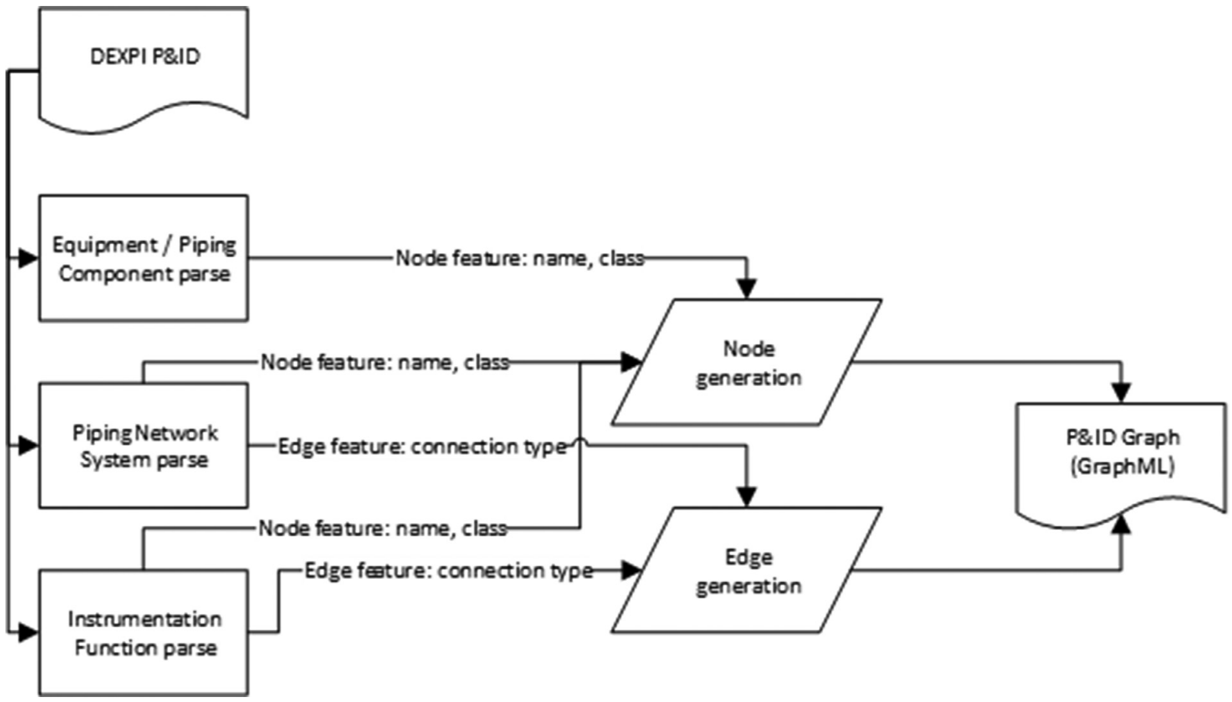

DEXPI P&ID
Equipment / Piping Component parse
Node feature: name, class
Piping Network System parse
Node feature: name, class
Edge feature: connection type
Node feature: name, class
Instrumentation Function parse
Edge feature: connection type
Node generation
Edge generation
P&ID Graph (GraphML)

Fig. 2. Structure of the Python DEXPI-2-graph converter.

information about process control equipment (PCE) and their connections. Fig. 2 shows the procedure used to convert the P&ID via Python.

The basis is an XML-parser (python package: *xml.etree.ElementTree /* version 3.3) to read out all relevant information from the DEXPI-file into Python. In the first step, the parser searches for all *Equipment* and *Piping-Components* and creates a separate node in a graph for each component. At the same time an equipment list is created, which documents the respective *EquipmentClass* and *ComponentClass* of the individual nodes. In a second step, the *PipingNetworkSystem* is searched for connections. These are implemented as directed edges between the already created nodes of the graph. Similarly, the *InstrumentationFunction* of the DEXPI P&ID is scanned and PCE components are added as additional nodes and associated signal lines are added as edges. This allows for considering control loops and signal streams in the graph information model. Subsequently, the graph and the associated equipment list are loaded into a data store and saved for further use. The structure of the P&ID representing graph is shown in Fig. 3.

When parsing the DEXPI files, it becomes apparent that the level of detail in the P&ID description varies greatly depending on the user. Thus, different numbers of attributes of the XML files are filled in. At the same time, the use of the attributes leaves some room for interpretation, such that synonymous information was mapped to different attributes. This requires a certain degree of robustness, which has been considered in the DEXPI-2-graph implementation. Therefore, several attributes (e.g. design temperature, pressure, material, …) are deliberately searched until the desired information for the respective node is found.

For better application, the DEXPI-2-graph converter is equipped with a graphical user interface (GUI), which is shown in Fig. 4. The figure also shows a visualization of an extracted P&ID graph. The path folder containing the DEXPI P&IDs to be converted is selected and the conversion is started via buttons. A console window shows the progress, errors and the generated GraphML files. In addition, a plot window is used to directly check the generated P&ID graphs. The converter including GUI is published as an open-source application and is available on Github as a Python application at https://github.com/TUDoAD/DEXPI2graphML (Oeing, 2022).

Datasets

In the following, several P&IDs in the standardized DEXPI format are used as training data, which were exported using the program *PlantEngineer* from the software vendor X-Visual Technologies GmbH and converted to graphs in GraphML format (GraphML Project Group, 2017) according to chapter 2.1. In total, 35 P&ID graphs from third parties (laboratory and industrial plants) with 1641 nodes and 1410 edges are used. The data set contains 92 different equipment classes (valves, pumps, vessels, instrumentation, etc.) based on the DEXPI specifications (Theißen and Wiedau, 2021) and has three different classes of edges (pipes, signal lines, process connection lines). The ratio of nodes/edges shows that, as expected for P&IDs, these are very linear graphs with rather low connectivity structures. At a closer look there are usually many single nodes along a pipeline (e.g. valves, vessels, pumps, heat exchangers, measuring points, etc.) which results in a kind of dead ends. Additionally, some P&IDs show inconsistencies in their drawn structures, which in some cases lead to isolated nodes or several, smaller graphs. However, these inconsistencies were deliberately included in the data set, as the data is intended to represent the current state of machine-readable P&IDs in the process industry to obtain representative results. The influence of the inconsistencies on the results is examined in more detail in chapter 4.

Sequential node prediction using recurrent neural networks

AI-based suggestions can be used to speed up the process of drawing P&IDs. A sequence of drawn and connected components is used to learn their course with the help of an RNN. Recurrent neural networks are neural networks that can model time series (or in this case linearly structured sequences) in training data based on their structure (Hu and Balasubramaniam, 2008). Based on the learned correlations, a node prediction is carried out, which returns the most probable subsequent P&ID components based on an input sequence. The workflow of the modeling is explained in more detail in the following.

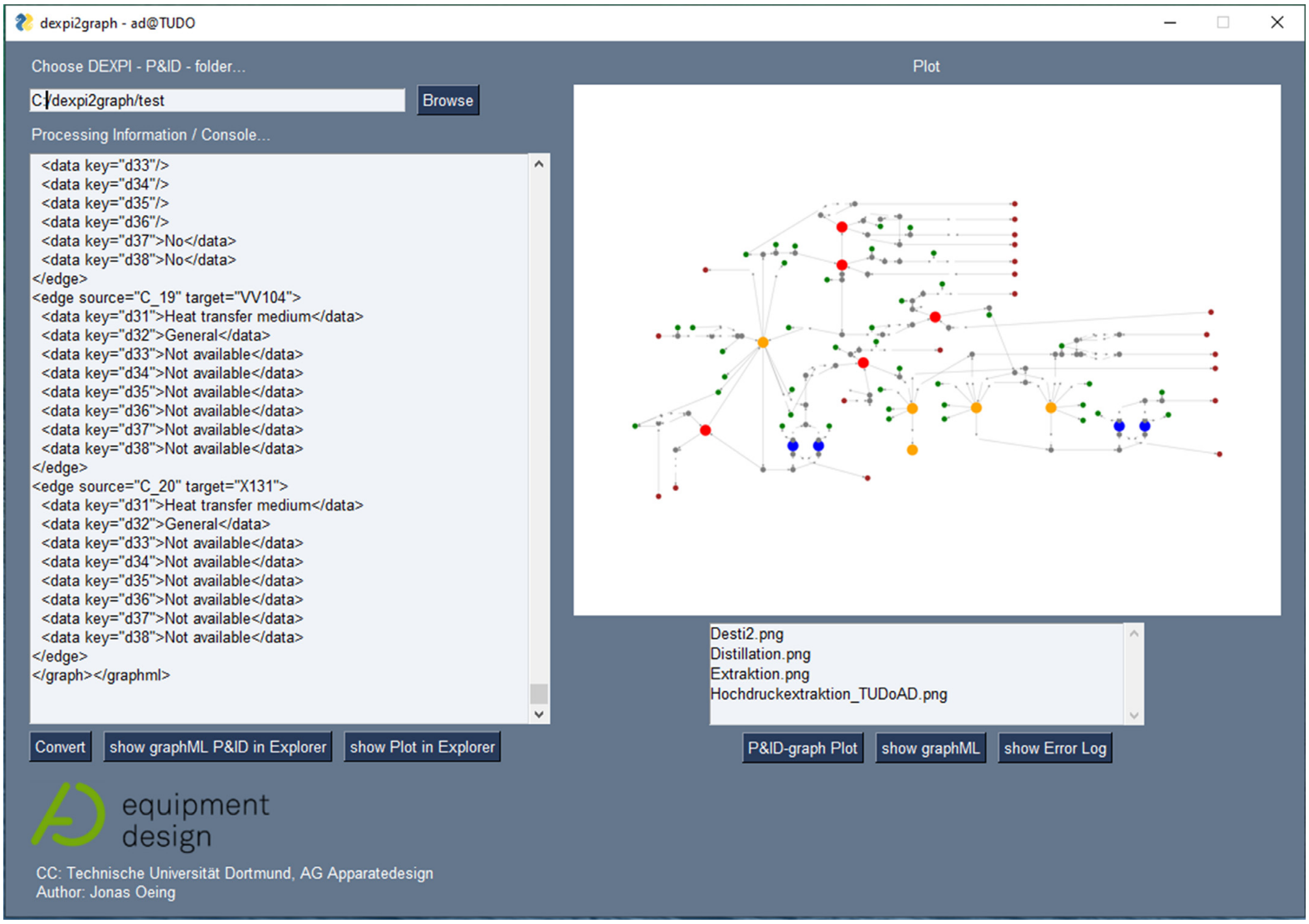
```
<?xml version='1.0' encoding='utf-8'?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

<graph edgedefault="directed">

<node id=PID_label>
  <data key="node_DEXPI_ID">Entry as string</data> #specifies the DEXPI ID
  <data key="node_PID_label">Entry as string</data> #specifies the PID Label
  <data key="node_class">Entry as string</data> #specifies the class
  <data key="node_sub_class">Entry as string</data> #specifies the sub class
  <data key="node_P_max_design">Entry as float</data> #specifies max. design pressure
  <data key="node_P_max_design_unit">Entry as string</data> #specifies the unit of max. design pressure
  <data key="node_agitator">Entry as string (Yes/No)</data> #specifies features regarding agitators
  ...
</node>
...

<edge source=From_PID_label target=To_PID_label>
  <data key="edge_class">Entry as String</data> #specifies the class
  <data key="edge_sub_class">Entry as String</data> #specifies the sub class
  <data key="edge_material">Entry as String</data> #specifies the material
  <data key="edge_insulation">Entry as String (Yes/No)</data> #specifies features regarding insulation
  ...
</edge>
...
</graphml>
```

Fig. 3. P&ID topology representing GraphML structure used for further training.



```
  <data key="d33"/>
  <data key="d34"/>
  <data key="d35"/>
  <data key="d36"/>
  <data key="d37">No</data>
  <data key="d38">No</data>
</edge>
<edge source="C_19" target="VV104">
  <data key="d31">Heat transfer medium</data>
  <data key="d32">General</data>
  <data key="d33">Not available</data>
  <data key="d34">Not available</data>
  <data key="d35">Not available</data>
  <data key="d36">Not available</data>
  <data key="d37">Not available</data>
  <data key="d38">Not available</data>
</edge>
<edge source="C_20" target="X131">
  <data key="d31">Heat transfer medium</data>
  <data key="d32">General</data>
  <data key="d33">Not available</data>
  <data key="d34">Not available</data>
  <data key="d35">Not available</data>
  <data key="d36">Not available</data>
  <data key="d37">Not available</data>
  <data key="d38">Not available</data>
</edge>
</graph></graphml>
```

Desti2.png
Distillation.png
Extraktion.png
Hochdruckextraktion_TUDoAD.png

CC: Technische Universität Dortmund, AG Apparatedesign
Author: Jonas Oeing

Fig. 4. GUI of the DEXPI-2-graph converter.

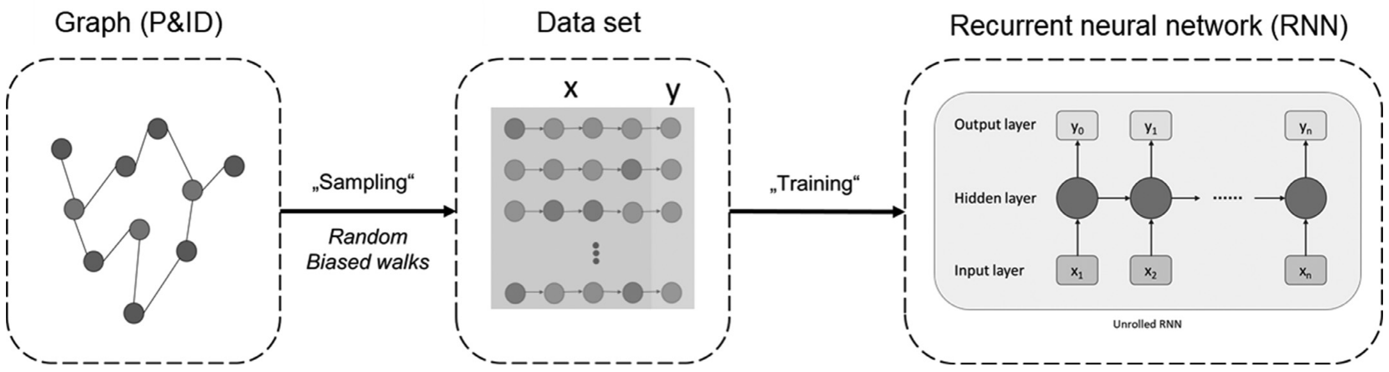


Fig. 5. Workflow of an RNN-based model for predicting subsequent equipment in P&IDs.

Workflow – node prediction

The sequential node prediction can be divided into three parts and its workflow is shown in Fig. 5.

The first step is sampling, during which the graph with its networked structure of nodes and edges is sequentially transformed into linear input data. These input data consist of a list of contiguous nodes, which contain the interconnected graph in linear representations. In the sampling process, all possible turns based on the number of output edges are made at branches to obtain a reliable representation of all node interconnections via random walks (Grover and Leskovec, 2016). The sampling is performed with the function *randomBiasedWalk,* which is part of the Python library *StellarGraph (*package: *stellargraph.data.BiasedRandomWalk* / version: v1.0.0rc1*)* (StellarGraph, 2020). The random biased walk requires four input parameters. The *number of walks* defines how many walks are generated from each node in the graph. The *walk length* specifies how many nodes are considered per walk. Important special features of the biased random walk are the return hyperparameter p and the in-out hyperparameter q, which guide the walk. Thus, $1/p$ defines the probability of reversing the sampling direction during the random walk, while $1/q$ describes the probability of discovering new nodes in the graph. In this way, the depth of the search can specifically be controlled (Grover and Leskovec, 2016). Since the generated samples should represent a clean and linear section of the plant topology, the parameters must be chosen in a way that the random walk jumps back as rarely as possible and continuously explores new paths. In this respect, previous investigations have shown that convincing results can be achieved with values of $p = 1000$ and $q = 1$. Smaller values of p, lead to an undesired probability of sampling against the flow direction. The sequential samples represent the actual training data for AI modeling and have a previously defined length l. They are divided in such a way that the first *l-1* entries represent the input sequence x, while the entries at position l are the corresponding output y. The dataset used in this work is composed of a total of 4923 sequences, each consisting of six nodes. For validation, 20 % of the data set are randomly retained as a test set. The remaining 80% are used to train the RNN.

In the second step, the structure of the node sequences is learned with the aid of recurrent neural networks (RNN). For this purpose, four different approaches are used: a simple RNN (Chen, 2021), a Bidirectional RNN (BRNN) (Schuster and Paliwal, 1997), a Gated Recurrent Units (GRU) (Cho et al., 2014) as well as a Long-short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). An RNN, as shown in Fig. 5, has an input layer, a hidden layer, and an output layer. The RNN represents a replication of a neural network with the same dimensions. For each replication, a state is transferred to the next layer. In this way it is possible to learn local relationships in the sequences. Basically, RNNs have a disadvantage that should not be underestimated. The training of weights by more distant information is difficult, since their errors explode or diminish during the backpropagation. (Chen, 2021) To get around this there are further possibilities such as the use of BRNNs. These take into account future information in addition to previous information to increase the accuracy (Schuster and Paliwal, 1997). Another option is the use of GRUs or LSTMs. Both consists of single cells and using cell states and gates to decide which information will be processed and which forgotten. This allows to get the behavior of a short-term memory. GRUs consist of a reset and update gate (Chen, 2021; Cho et al., 2014). A LSTM uses one input, one output, and one forget gate. The gates control the information flow and decide, which information is necessary to make a prediction. A kind of short-term memory is created, which also gives the network its name (Chen, 2021; Hochreiter and Schmidhuber, 1997). Both the GRU and the LSTM are used in state-of-the-art deep learning applications. GRUs have fewer tensor operations, which leads to faster training. For this reason, they are used for modeling in this paper to investigate which of them leads to better results for predicting P&ID equipment based on sequential data.

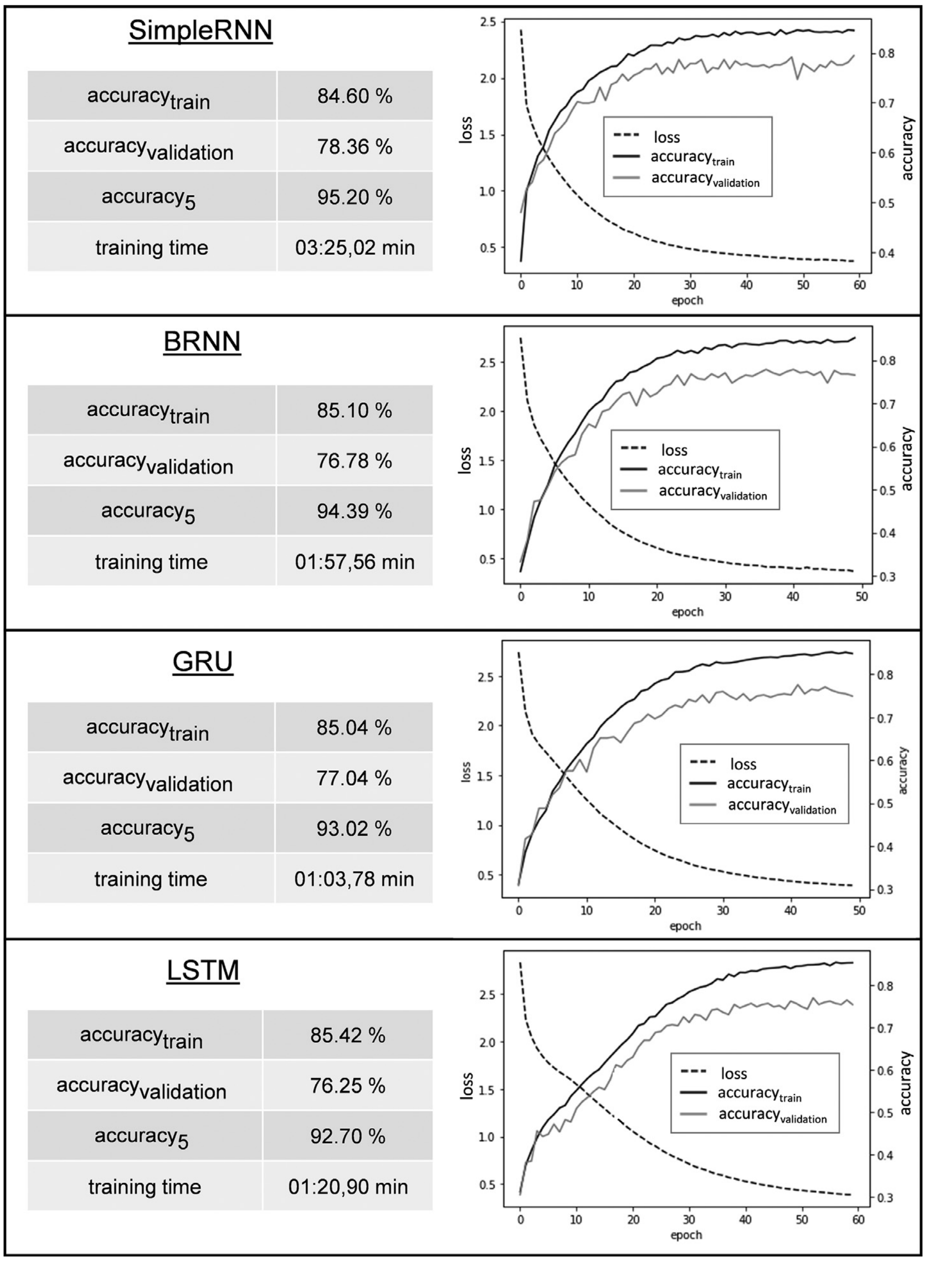
Results – node prediction

In the following, the different RNN models are used and trained with the in chapter 2.2 generated P&ID graphs according to the presented workflow. The implementation is done in Python using the *keras* library (Chollet, 2020). The "Adam" optimizer (Kingma and Ba, 2014) is used for all trainings and the calculation of the loss is performed by the "categorical cross entropy" (Murphy, 2012). The prediction accuracy is used as an evaluation metric and is defined as follows.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

The accuracy is calculated by dividing the sum of true positive (TP) and true negative (TN) by the sum of true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The computations were done on an Intel® Xeon® W-2155 (3.31 GHz) CPU in combination with 128 GB RAM. The results are shown below in Fig. 6. The training accuracy and validation accuracy are shown. In addition, the $accuracy_5$ indicates the correctness, with which the real output of the validation dataset is predicted, when the five most probable outputs are returned. This score is of particular interest to investigate whether the trained models are suitable for a suggestion system that can be used, for example, in a drop-down menu to speed up the drawing process of P&IDs. Furthermore, both the calculated loss and the training and validation accuracy over 60 epochs as well as the training time needed to calculate the 60 epochs are given.

The results show that RNNs are generally able to learn patterns in sequences from P&ID graphs. It is noticeable that the SimpleRNN provides the best results with a validation accuracy of 78.36%. In the case, where the equipment is part of the five most likely predictions, even 95.2% accuracy is achieved. The BRNN reaches an accuracy of 94.39%, while predicting the five most suitable equipment types. The LSTM and GRU



SimpleRNN

$accuracy_{train}$	84.60 %
$accuracy_{validation}$	78.36 %
$accuracy_5$	95.20 %
training time	03:25,02 min

BRNN

$accuracy_{train}$	85.10 %
$accuracy_{validation}$	76.78 %
$accuracy_5$	94.39 %
training time	01:57,56 min

GRU

$accuracy_{train}$	85.04 %
$accuracy_{validation}$	77.04 %
$accuracy_5$	93.02 %
training time	01:03,78 min

LSTM

$accuracy_{train}$	85.42 %
$accuracy_{validation}$	76.25 %
$accuracy_5$	92.70 %
training time	01:20,90 min

Fig. 6. Results of the training of following P&ID equipment with different RNN models.

have slightly lower accuracy, suggesting that the effect of the diminishing gradient for the short sequences involved does not have a significant effect on the training. At the same time, it should be noted that training for the GRU took less than one-third the time of a SimpleRNN model. Given the current small amount of data, this is not a decisive factor with the current setting. However, should the training of the models be done in the future on large data sets or continuously, it is recommended to give more attention to this aspect, as the use of GRUs or LSTMs can save time and resources (Strubell et al., 2019), which should be considered with respect to a sustainable process development.

Node classification using graph neural networks

As mentioned before, the information from the P&ID is interpreted in the form of a graph. This makes it possible to store the relationships between components and the topology in an unambiguous and machine-interpretable way. However, to learn the graph structure as a whole and to solve tasks such as node classification, edge classification or link predictions, machine learning methods of graph analysis are required that can deal with non-Euclidean data structures such as graphs. The modeling of graph structures is particularly interesting in the field of P&ID engineering. By learning connections (e.g. piping, signal lines, …) or components (e.g. valves, equipment, …) based on their neighborhood with the help of AI, it will be possible in the future to perform consistency checks in P&IDs and detect errors in P&IDs. This could reduce the amount of time for drawing P&IDs, which will shorten the time for developing a plants documentation. To achieve this goal, Graph Neural Networks can be used for modeling Graph Neural Networks (GNN) can be used for modelling, which have become increasingly important in recent years (Zhou et al., 2020). A GNN is based on a message passing algorithm that aggregates arbitrary information from the neighborhood of a node, which will convolve the graph (Hamilton, 2020). In general,

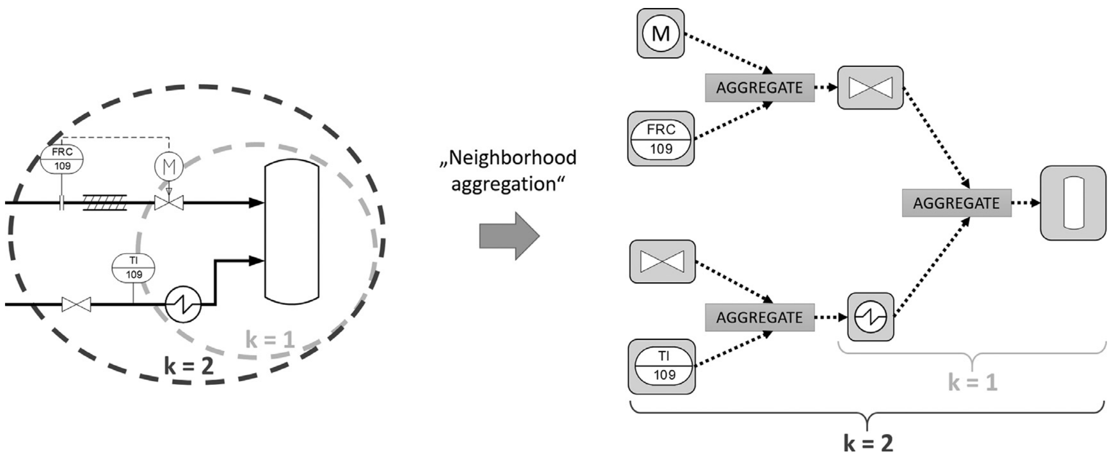


Fig. 7. Example of the neighborhood aggregation of a GNN using a P&ID.

the message passing of a GNN is analogous to the Weisfeiler-Lehman algorithm to test the isomorphism of two graphs (Weisfeiler and Leman, 1968), which was introduced in 1968 and in which information is aggregated from the neighborhood of each node.

Mathematically, the message passing of an GNN (Eq. (2)) can be described as follows (Hamilton, 2020):

$$\begin{aligned} h_u^{(k+1)} &= UPDATE^{(k)}\left(h_u^{(k)},\ AGGREGATE^{(k)}\left(\left\{h_v^{(k)},\ \forall v \in N(u)\right\}\right)\right) \\ &= UPDATE^{(k)}\left(h_u^{(k)},\ m_{N(u)}^{k}\right) \end{aligned} \tag{2}$$

$h_u^{(k)}$ stands for the embedding of a node u at iteration step k. *UPDATE* and *AGGREGATE* are arbitrary, differentiable functions, where the aggregation of the neighborhood *N(u)* of node *u* represents the actual "message" *m*. The parameter *k* defines the number of iterations, at which the message passing proceeds, thus represents the number of hidden layers of the GNN. Since the aggregation of the neighborhood information must be independent of the order, it is important that the *AGGREGATION* is a permutation-invariant function. Based on the embedding for each iteration step *k*, a final embedding for each node u can subsequently be determined using a final layer (Hamilton, 2020).

To better understand the modeling of plant topology by message passing GNNs, an example is given in Fig. 7 that relates the aggregation of neighborhood information to a snippet of a P&ID. The example shows the aggregation by a two-layer neural network. Since the plant topology is to be learned, we focus in the following on the equipment information, such as the classes of each component in the P&ID. Thus, in a first step ($k = 1$), inferences can be made about the vessel based on the information from the valve and the heat exchanger. In a second step ($k = 2$), a valve's and a temperature sensor's information can be aggregated for the embedding of the heat exchanger, while the valves´ embedding is influenced by the connected drive and flow control.

The computational graph shown in Fig. 7 visualizes how it is possible to obtain information of a component based on its neighborhood through the neural networks *AGGREGATE*$_{(k=1)}$ and *AGGREGATE*$_{(k=2)}$. If the neighborhood components of the vessel are known, it is possible to predict the class of the vessel or to analyze in a consistency check how probable it is that a vessel is located at exactly this position in the graph.

To verify the feasibility, a node classification based on a GraphSAGE model is applied in the following. GraphSAGE (Hamilton, 2020; Hamilton et al., 2017; Khosla et al., 2020) is a GNN, which is characterized by a good generalization for unseen structures in graphs (Hamilton et al., 2017), which makes it particularly suitable for the currently small amount of machine-readable P&ID data. A challenging aspect is that GraphSAGE aggregates information of all nodes including the target node. This becomes a problem if the equipment class is part of the aggregated features, since the own equipment class is aggregated. This inevitably leads to an accuracy of 100%, too, since the GNN can read the information in the node. In the case of a consistency check, this does not make sense, since the information is available but must not serve as input for the network, since this is what is to be checked. Hence, it is essential that the features of the target node are not aggregated, but only their neighborhood. The remedy is a recursive neural network, which was modified from the GraphSAGE algorithm. For this purpose, a calculation rule (3) is used, which considers in a case distinction whether information of a node should be included for aggregation. Mathematically the variation can be described as follows, where in contrast to the classical GraphSAGE (Hamilton et al., 2017) it has to be recognized that the model determines the embedding from the starting parameter $h_u^{(0)}$ and *CONCAT* concatenates the individually calculated aggregations of each node.

$$\begin{aligned} h_u^{(k)} &= UPDATE^{(k-1)}\left(W^{(k-1)} \cdot CONCAT \cdot \left(h_u^{(0)},\ AGGREGATE^{(k-1)}\left(\left\{h_v^{(k-1)},\right.\right.\right.\right. \\ &\quad \left.\left.\left.\left.\forall v \in N(u)\right\}\right)\right) + b^{(k-1)}\right) \end{aligned} \tag{3}$$

The model is programmed as a recursive function, since the function *CONCAT* for concatenating the embedding calls itself to compute the message *m* during iteration via all neighboring nodes.

Workflow - GNN node classification

The workflow of the node classification is shown in Fig. 8. First, all nodes of all P&ID graphs in the used dataset are divided into a training dataset (80%) and a test dataset (20%) using a mask. The neural network is then provided with information about the topology of the graph, as well as attributes of the nodes and edges, e.g. equipment class, connection type, etc… From this information, the network generates an embedding for each node and the predicted node class. This is compared with the real node class and the error is reduced via backpropagation. After the training is finished, the trained network can be used for node classification of unseen data (nodes).

Since the structure of the graphs in GNNs is learned by aggregating the neighbors, it is important that the graphs have no missing links. As mentioned before, this is not always the case in the data base from chapter 2.2 due to inconsistencies in real P&ID drawings and their DEXPI exports. For this reason, a new dataset of laboratory plants as well as industrial distillation plants is used. The dataset contains 13 P&ID graphs and have a sufficiently high density of cross-links such that they represent the original P&IDs well. These 13 P&ID graphs with 2020 nodes and 2283 edges are used to train the GNNs in the following. Within the 13 P&IDs, there are a total of 47 different equipment classes. In this context, a feasibility analysis must be performed to investigate the potential of classifying P&ID equipment by GNNs and to identify further challenges.

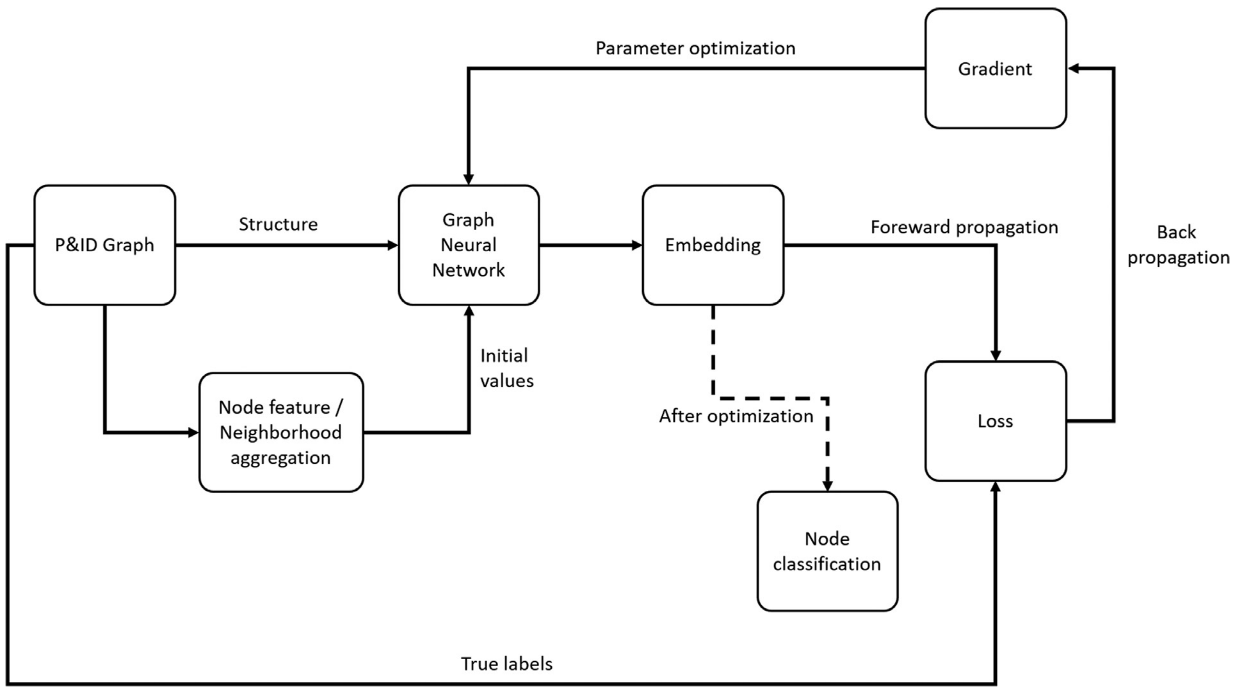


Fig. 8. Workflow of a learning P&ID components using GNNs.

Table 1
Higher level node classes and their quantity in the training dataset.

Node class	Quantity
Valves	1034
Check valves	60
Safety valves	93
Pumps	69
Vessels	75
Heat exchangers	86
Separation units	15
Process control equipment (PCE)	547
Piping equipment	41

Since different components in a P&ID can fulfill the same process engineering function and since the use of meaningful functionalities at the existing positions is to be examined for a consistency check, it is recommended to divide the components within the P&ID into meaningful classes. This has the additional advantage that even single components can be used as training data influencing the classification. Hence, the 47 different components in the training data set are sorted into 9 superior classes, which are shown in Table 1 below. The number of components per class is also shown in relation to the training data.

It appears that the classes have an uneven distribution, which is unavoidable since this data set is a representative cross-section of all components in a process plant. In the context of this work, it is important to investigate to what extent the unequal distribution of training data will affect the results of the classification.

The recursive GNN is used based on the GraphSAGE algorithm presented in the previous chapter 4. The number of layers is $k = 3$. For the activation function, the ReLU function (Manaswi, 2018) and a subsequent normalization are applied. To achieve the most efficient prediction accuracy, different state-of-the-art aggregation functions are used and compared against each other. Basic variants in this respect are the calculation of a sum (4) or an arithmetic mean (5) (Grabisch et al., 2009).

$$m_{N(u)}^{(k)} = \sum\nolimits_{v \in N(u)} h_v^{(k)} \tag{4}$$

$$m_{N(u)}^{(k)} = \frac{1}{|N(u)|} \sum\nolimits_{v \in N(u)} h_v^{(k)} \tag{5}$$

In addition, there is the possibility of adding a multilayer perceptron (MLP) over the states to be aggregated, as shown in Eq. (6) for the sum-MLP (Xu et al., 2019). The idea is that an MLP can transform each state $h_v^{(k)}$ into an one-hot vector, i.e., a vector consisting of zeros except a single entry that is set to one (Gulli and Pal, 2017). The sum of the vectors allows for determining exactly the used one-hot vectors, which were needed for its generation.

$$m_{N(u)}^{(k)} = \sum\nolimits_{v \in N(u)} MLP(h_v^{(k)}) \tag{6}$$

Another option is to form the message via a set pooling approach (7), which is similar to the aggregation of a Graph Isomorphism Network (GIN) (Xu et al., 2019). According to Zaheer et al. (2017), this uses an MLP_θ with trainable parameters θ and applies it to the sum-MLP combination in Eq. (6) (Hamilton, 2020).

$$m_{N(u)}^{(k)} = MPL_\theta \sum\nolimits_{v \in N(u)} MLP_\phi(h_v^{(k)}) \tag{7}$$

As a last investigated variant, the graph attention approach (8) is applied, where additionally for each $v \in N(u)$ a weight factor $\alpha_{u,v}^{(k)}$ is defined (Bahdanau et al., 2014; Hamilton, 2020). In the context of this work, a softmax approach (9) is used to calculate the weight factor α (Bahdanau et al., 2014).

$$m_{N(u)}^{(k)} = \sum\nolimits_{v \in N(u)} \alpha_{u,v}^{(k)} \cdot h_v^{(k)} \tag{8}$$

$$\alpha_{u,v}^{(k)} = \frac{\exp\left(h_v^T W^{(k)} h_v\right)}{\sum_{v' \in N(u)} \exp\left(h_{v'}^T W^{(k)} h_{v'}\right)} \tag{9}$$

Results - GNN node classification

To quantify how well GNNs are suited to classify individual pieces of equipment based on their location in the P&ID topology, the recursive GNN presented in chapter 3 is trained with the P&ID graphs presented before. For this purpose, the models were implemented and

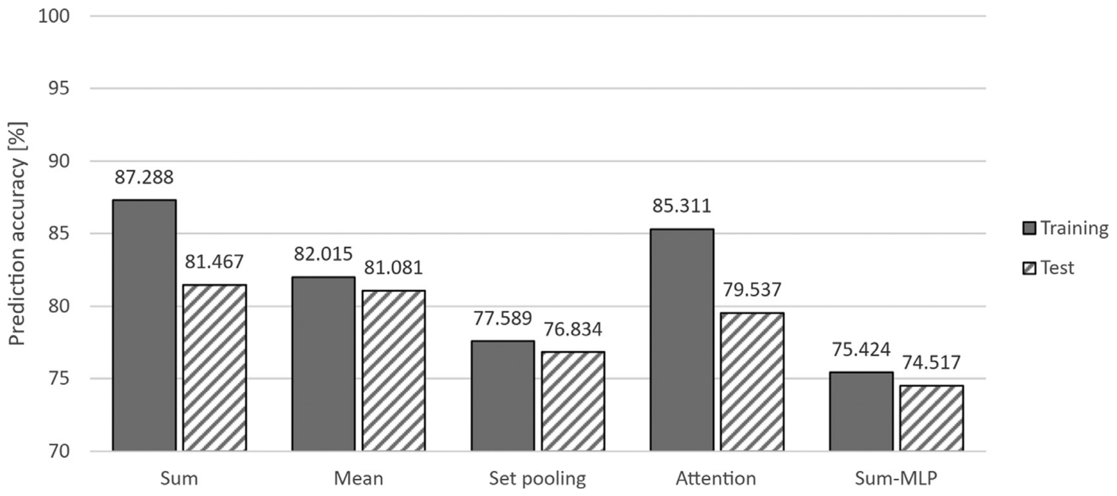


Fig. 9. Results of the node classification in a P&ID graph via recursive GNN grouped by the applied aggregation functions.

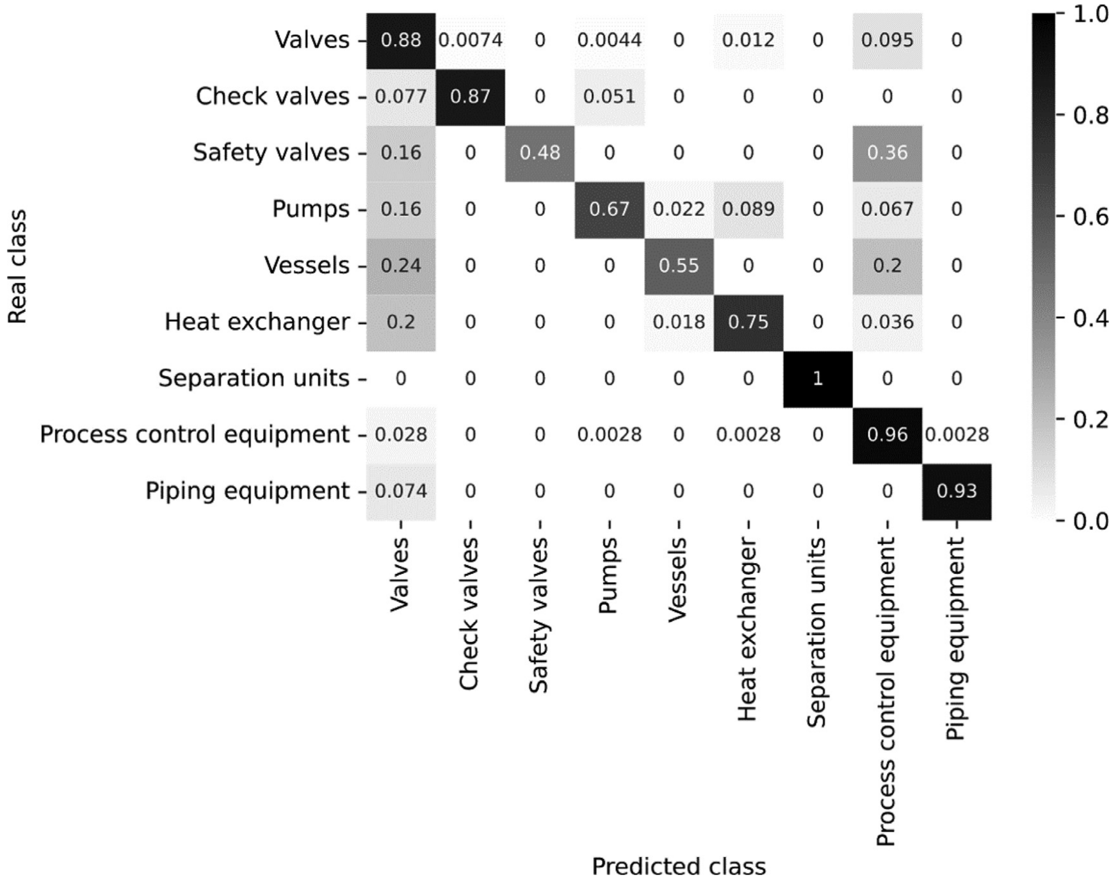


Fig. 10. Normalized confusion matrix of the recursive GNN with sum aggregation for the test data set.

trained using the *Deep Graph Library (version 0.7.2)* with *PyTorch framework* in *Python (version 3.7)*. The calculations were performed on an Intel® Xeon® W-2155 (3.31 GHz) CPU in combination with 128 GB RAM. The aggregation functions listed previously are used in the GNN and compared against each other. The prediction accuracy is used as an evaluation metric, which was defined in chapter 3.2. For the examined models, the accuracy is shown below in Fig. 9.

The results for all models show deviations among each other. The accuracy for the training for all models is between 75.4% (sum-MLP) and 87.3% (sum) while the accuracy for the test data varies between 74.5% (sum-MLP) and 81.5% (sum). It is striking that the gap between the test and training accuracies for the sum aggregation as well as the attention aggregation is larger than for the remaining aggregation functions, at about 6 percentage points. Additionally, the results show that simpler aggregation algorithms such as sum and arithmetic mean achieve higher training accuracies than the more complex aggregations using attention, set pooling or sum-MLP. It is hypothesized that this is due to the fact that all neighborhood information is equally important in predicting the component class. For this reason, learning the individual P&ID components works particularly well when the neighborhood information is aggregated with the same weight, i.e., equally important. This is especially true for the sum or mean.

To check how well the classification can be performed for the different classes, the confusion matrix of the model with sum aggregation using the test data set is also considered, see Fig. 10. The columns in the matrix describe the predicted classes, while the rows represent the real classes. Consequently, the main diagonal displays the number of correctly classified components (TP).

The main diagonal shows that most components of each class are correctly classified. This way, all separation units are correctly classified. Process control equipment (PCE, 96%) and piping components (93%) are also almost completely correctly assigned. With over 87% prediction accuracy, valves and check valves are also classified sufficiently well, although it is noticeable that there is confusion between valves and piping equipment and valves and safety valves. However, with less than 10%, this is still within tolerable limits. The classification of the remaining components is much more difficult for the GNN. Thus, 39% of safety valves are identified as piping equipment and 16% as standard valves. This is not surprising since both classes are usually found in similar positions of a P&ID. Classification of pumps (67%), vessels (55%) and heat exchangers (75%) is only reasonably satisfactory. It is noticeable that all three classes are mainly classified as valves or PCEs. Classes which are particularly strongly represented in the data set according to Table 1. The GNN models should therefore be further optimized in the future. At this point, it is conceivable to integrate the comparatively underrepresented classes more strongly into the training by introducing weighting factors. Furthermore, it would be conceivable to use a larger *k*, which would aggregate more information. However, this results in a larger computational effort.

Conclusion & outlook

Graph-based P&ID formats are a promising way to improve machine readability of important process information. The standardized DEXPI format, which is defined and will be improved continuously by the DEXPI Initiative, is able to store the topology of process plants as well as all apparatus specifications in a structured way and make them available in a machine-interpretable manner. This standardization enables the retrieval of information and a simple application of AI models. Thus, in addition to the possible accessibility of P&ID data, two approaches have been presented in this work, which make it possible to recognize patterns in P&IDs and use AI to help support and increase efficiency in P&ID synthesis. First, P&ID graphs are decomposed into sequences, and subsequent components can be predicted based on the drawn partial structure by RNNs. In this regard, high accuracies were achieved and, in particular the use of AI-based suggestion algorithms was quantified. On the other hand, the use of GNNs allows for learning correlations in plant topologies. In the present case, existing components were classified into P&IDs for a consistency check. In this case, too, good results were achieved in a first feasibility analysis with a recursive GNN and the data basis is expandable for the application of AI. However, further fields of application for GNNs are possible. For example, the prediction of connections is possible, e.g. pipelines, signal lines, as well as the detection of subgraph structures which could help to automatically detect functional equipment assemblies (as a part of the engineering of modular plants).

For a wide acceptance and usage of AI in process engineering, it is necessary that digital, machine-readable formats are further implemented and are integral part of software packages. The analysis of the current, still very small data base shows that P&IDs are often drawn in such a way that they are available as a pdf document or printout, while important information such as interconnections (e.g. piping, signal lines) are often missing in the digital representation due to incorrect drawn P&IDs by the user. If there is a continuous increase in machine-readable P&IDs in the future, the approaches demonstrated in this paper show the potential to accelerate the engineering process and save labor time and costs, when integrated into P&ID software. Primarily, it is recommended to suggest components during P&ID synthesis and to identify and reduce possible errors through validation. In addition, further AI solutions can be developed that recognize functional groups in P&IDs and consider the implementation of these modules when developing processes.

Further opportunities are safety assessment and HAZOP studies, where machine-readable HAZOP scenarios could be mapped to graph-based plant topologies using search algorithms. Furthermore, in future a possible detection of subgraphs in P&IDs i.e. functional equipment assemblies could facilitate the mapping between PFD simulations based on unit operations and P&IDs. This could provide the foundation for automated generation of PFD simulations from a DEXPI plant topology.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research work was supported within the KEEN project (grant number: 01MK20014S) and has been funded by the Federal Ministry of Economic Affairs and Climate Action (BMWK).

References

Bahdanau, D., Cho, K., Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate. In: Proceedings of the ICLR, p. 2015.
Batres, R., Lu, M.L., Naka, Y., 1997. An agent-based environment for operational design. Comput. Chem. Eng. 21, S71–S76. doi:10.1016/S0098-1354(97)87481-9.
Chen, L., 2021. Deep Learning and Practice with MindSpore, Cognitive Intelligence and Robotics. Springer, Singapore, Singapore doi:10.1007/978-981-16-2233-5.
Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y., 2014. On the properties of neural machine translation: encoder-decoder approaches. arXiv Prepr. arXiv1409.1259.
Chollet, F., 2020. Keras API - documentation vers. 2.4.0 [WWW Document]. URL https://keras.io (accessed 2.20.22).
Fillinger, S., Bonart, H., Welscher, W., Esche, E., Repke, J.-U., 2017. Improving interoperability of engineering tools - data exchange in plant design. Chem. Ing. Tech. 89, 1454–1463. doi:10.1002/cite.201700032.
Grabisch, M., Marichal, J.-L., Mesiar, R., Pap, E., 2009. Aggregation Functions (Encyclopedia of Mathematics and its Applications). Cambridge University Press, Cambridge doi:10.1017/CBO9781139644150.
GraphML Project Group, 2017. GraphML specification [WWW Document]. URL http://graphml.graphdrawing.org/specification/dtd.html (accessed 2.10.22).
Grover, A., Leskovec, J., 2016. node2vec. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, pp. 855–864. doi:10.1145/2939672.2939754.
Gulli, A., Pal, S., 2017. Deep Learning with Keras: Implementing Deep Learning Models and Neural Networks With the Power of Python. Packt Publishing, Birmingham.
Hamilton, W.L., 2020. Graph representation learning. Synth. Lect. Artif. Intell. Mach. Learn. 14, 1–159. doi:10.2200/S01045ED1V01Y202009AIM046.
Hamilton, W.L., Ying, R., Leskovec, J., 2017. Inductive representation learning on large graphs. In: Proceedings of the NIPS, p. 17.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735.
Hu, X., Balasubramaniam, P., 2008. Recurrent Neural Networks. InTech doi:10.5772/68.
International Electrotechnical Commission, 2016. IEC 62424, Representation of process control engineering – requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools. International Electrotechnical Commission, Geneva.
International Organization for Standardization, 2013. ISO 15926-2 - industrial automation systems and integration – integration of life-cycle data for process plants including oil and gas production facilities – part 2: data model. Beuth Verlag, Geneva.
International Organization for Standardization, 2012a. ISO 10628-2 - Diagrams for the Chemical and Petrochemical Industry – Part 2: Graphical Symbols. International Organization for Standardization.
International Organization for Standardization, 2012b. ISO 10209, technical product documentation – vocabulary – terms relating to technical drawings, product definition and related documentation. Geneva.
Khosla, M., Setty, V., Anand, A., 2020. A comparative study for unsupervised network representation learning. IEEE Trans. Knowl. Data Eng. 33, 1807–1818. doi:10.1109/TKDE.2019.2951398.
Kingma, D.P., Ba, J., 2014. Adam: a method for stochastic optimization. In: Proceedings of the ICLR, p. 2015 ArXiv ID 1412.6980.
Manaswi, N.K., 2018. Deep Learning with Applications Using Python. Apress, Berkeley doi:10.1007/978-1-4842-3516-4.
Murphy, K.P., 2012. Machine Learning : A Probabilistic Perspective. MIT Press, Cambridge.

Oeing, J., 2022. DEXPI2graph converter application [WWW Document]. URL https://github.com/TUDoAD/DEXPI2graphML (accessed 5.18.22).

Proteus XML, 2017. Proteus schema for P&ID exchange [WWW Document]. URL https://github.com/ProteusXML/proteusxml (accessed 5.18.22).

Schuster, M., Paliwal, K.K., 1997. Bidirectional recurrent neural networks. IEEE Trans. Signal Process. 45, 2673–2681. doi:10.1109/78.650093.

StellarGraph, 2020. StellarGraph machine learning library - documentation [WWW Document]. URL https://stellargraph.readthedocs.io/en/stable/README.html (accessed 3.1.22).

Strubell, E., Ganesh, A., McCallum, A., 2019. Energy and policy considerations for deep learning in NLP. In: Proceedings of the ACL, p. 19.

Theißen, M., Wiedau, M., 2021. DEXPI - P&ID Specification [WWW Document]. Version 1.3. URL https://dexpi.org/specifications/ (accessed 3.1.22).

Turau, V., Weyer, C., 2015. Algorithmische Graphentheorie, De Gruyter Studium. DE GRUYTER, Berlin doi:10.1515/9783110417326.

Weisfeiler, B., Leman, A., 1968. The reduction of a graph to canonical form and the algebra which appears therein. NTI Ser. 2, 12–16.

Wiedau, M., Tolksdorf, G., Oeing, J., Kockmann, N., 2021. Towards a systematic data harmonization to enable AI application in the process industry. Chem. Ing. Tech. 93, 2105–2115. doi:10.1002/cite.202100203.

Wiedau, M., von Wedel, L., Temmen, H., Welke, R., Papakonstantinou, N., 2019. ENPRO data integration: extending DEXPI towards the asset lifecycle. Chem. Ing. Tech. 91, 240–255. doi:10.1002/cite.201800112.

Xu, K., Hu, W., Leskovec, J., Jegelka, S., 2019. How powerful are graph neural networks? In: Proceedings of the ICLR, p. 2019.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., Smola, A., 2017. Deep sets. In: Proceedings of the NIPS, p. 2017.

Zhao, C., Bhushan, M., Venkatasubramanian, V., 2005. PHASuite: an automated HAZOP Analysis tool for chemical processes. Process. Saf. Environ. Prot. 83, 509–532. doi:10.1205/psep.04055.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M., 2020. Graph neural networks: a review of methods and applications. AI Open 1, 57–81. doi:10.1016/j.aiopen.2021.01.001.