UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Handling anomalies in CAD assemblies: detecting anomalous and suggesting alternative parts

Carola Lenzen, Vinzenz Löffel, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Lenzen, Carola, Vinzenz Löffel, and Wolfgang Reif. 2025. "Handling anomalies in CAD assemblies: detecting anomalous and suggesting alternative parts." In *Computational Science and Computational Intelligence - 11th International Conference, CSCI 2024, Las Vegas, NV, USA, December 11–13, 2024, proceedings, part III*, edited by Hamid R. Arabnia, Leonidas Deligiannidis, Farzan Shenavarmasouleh, Soheyla Amirian, and Farid Ghareh Mohammadi, 257–70. Cham: Springer. https://doi.org/10.1007/978-3-031-94940-1_21.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/



Handling Anomalies in CAD Assemblies: Detecting Anomalous and Suggesting Alternative Parts

Carola Lenzen[1], Vinzenz Löffel[1], and Wolfgang Reif[1]

Institute for Software & Systems Engineering, University of Augsburg, Augsburg, Germany
{lenzen,reif}@isse.de, vinzenz1.loeffel@uni-a.de

Abstract. In the design engineering process of assembly modeling, existing computer-aided design (CAD) models of parts are combined to build new products, termed assemblies. Due to the enormous variety of parts available, designs of inexperienced designers can easily show unusual part combinations stemming from unfamiliarity or a lack of suitable alternatives. This paper addresses the challenge of handling anomalies in CAD assemblies with a twofold goal: first, we aim to identify such anomalies, and second, to provide suggestions for alternative parts to correct these anomalies. We employ a graph-based representation of CAD assemblies and utilize graph neural networks (GNNs) to develop models for the two respective tasks modeled as node classification problems. The models are evaluated both separately and in an end-to-end (E2E) fashion, i.e., considering the task of handling anomalies as a whole. Our experiments demonstrate their effectiveness in improving the quality of CAD assemblies with minimal human intervention.

Keywords: Graph Machine Learning · Assembly Modeling · Anomaly Detection · Recommendation

1 Introduction

Computer-aided design (CAD) systems support design engineers in the creation, modification, analysis and optimization of three-dimensional mechanical designs [10]. Assembly modeling constitutes a core task of CAD, where assemblies are built as a collection of preexisting parts to make up a product. So-called mating conditions may define constraints between pairs of parts, e.g., their relative movement to each other. Consider a metal cabinet composed of five plates, two doors, hinges, and handles as a simple example. Designers are faced with plenty of available parts stemming from numerous part catalogs, which makes the selection of the right parts a cumbersome and time-consuming task – especially for inexperienced designers. If a design engineer has typically designed very robust cabinets with heavy doors and sturdy hinges, they may need to source equivalent parts from entirely different catalogs when designing a lighter-weight version. This scenario often arises in fields like contract manufacturing or special

machine construction, where product designs frequently vary. When unfamiliar with the corresponding light-weight parts, novices might pick an unusual combination, such as pairing light-weight doors with heavy-duty hinges. However, this choice may well be intentional if, for instance, the cabinet is opened frequently, putting considerable stress on the hinges. To better assist CAD designers, our goal is to *identify anomalous parts* within an assembly that are unusual or different from standard, but not necessarily in a negative way. We intend to highlight not only whether the current design deviates, but also *where* and *how*. In this work, we focus on anomalous parts in assemblies occurring due to the lack of knowledge about or the unsuccessful search for a suitable part. In order to support the designer in selecting an actual suitable part, our second goal is to provide *suggestions for alternatives* for detected anomalous parts. Both goals taken together result in the correction of an anomalous assembly. Instead of automatic correction, we strive to offer *user-guided support* that enhances assembly quality. To avoid reducing the limited time of designers [6], we pursue an automated, data-driven approach, minimizing human involvement such as data labeling.

In preliminary work [3], we use graph machine learning to supply designers with recommendations of parts next needed during assembly modeling. Therefore, we represented assemblies as undirected, unweighted graphs $\mathcal{A} = (\mathcal{V}, \mathcal{E})$: The parts within an assembly form the graph nodes $\mathcal{V}$, while edges $\mathcal{E}$ are read off from specified mating conditions or geometric proximity of parts. The assemblies are based on a given set of part types $\mathcal{T} = \{\tau_1, \tau_2, \dots\}$ which serves as a "vocabulary" for the assemblies, with specific parts contained in an assembly being instantiations of these types. Hence, an assembly may contain multiple instances of the same part type. In our considered tasks in this paper, the information about which parts are connected, i.e., the graph structure, is crucial since it is not solely the occurrence of a specific part, but its *use in a certain context*, i.e., its combination with other parts, that makes it anomalous. Thus, we adhere to the graph-based representation of assemblies and employ the real-world assembly datasets of [3] for our experiments. This paper draws on the main results and leverages individual aspects of the approach.

Our Model Pipeline

Aiming to assist a designer during the whole workflow of regularizing anomalous assemblies, an overall task we call *anomaly handling*, we need to tackle two subordinate tasks in our problem domain, thus requiring two distinct, consecutive models: For an input graph fed into our anomaly handling system, a first model is asked to detect anomalies within an assembly, a task we call *anomaly detection* (AD). The resulting output is then assessed by the user who decides which detected anomalies to generate *adaptation suggestions* (AS) for, a task performed by a second model. The suggested alternative part types are meant to correct the anomaly, i.e., replacing the anomalous part with a suggestion should transform the assembly into a regular one. With our anomaly handling system thus consisting of two standalone components, we can either assess each model's performance *separately* or evaluate the overall performance of our two consecutive

models from an *end-to-end* (E2E) perspective. For the separate evaluation of the second model (AS), we use the target values of the previous anomaly detection model (instead of its predictions) to allow for an independent assessment. When evaluating from an E2E perspective, the output of the AD model is directly transformed into inputs to the AS model, replacing the assessment performed by the user by a component we call *AD-AS adapter*. Fig. 1 provides an overview of the overall anomaly handling task as well as its respective subtasks while illustrating the differences between training and evaluation.

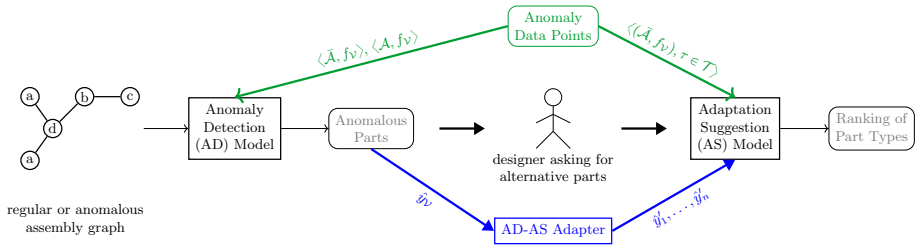


Fig. 1. Overview of our proposed anomaly handling system: approach to detect anomalous parts in CAD assemblies and suggest alternatives for these parts if requested by the designer. The training flow is highlighted in green, while the end-to-end (E2E) evaluation flow is highlighted in blue.

2 Related Work

Anomaly detection in general is a well-known and widely addressed problem, although it has hardly been investigated in the context of CAD assemblies. However, we already elucidated the need to investigate anomalies in CAD assemblies in [6]. There, we present promising use cases within assembly modeling and highlight their suitability for graph machine learning. Anomaly detection problems on graphs are categorized whether the anomaly relates to a graph element (node, edge or subgraph) or the entire graph [8]. In order to support a designer in a targeted manner, we analyze not only whether an assembly is anomalous (i.e., detection on graph-level), but also *where*, by identifying the anomalous node.

Due to the rare occurrence of anomalies in many applications, methods for generating synthetic anomalies have been developed. With respect to the categorization given in [2], our anomaly generation approach falls into the category of modifying real-world graphs to form an anomalous graph.

The field of recommending solutions to detected anomalies has so far been little researched – the same also applies to the combined task which we refer to as anomaly handling. To the best of our knowledge, only Srinivas et al. [12] have tackled this problem in the context of time-series data from large-scale industrial units. Root cause analysis using computational hypergraphs forms

the basis for their recommendations. From a wider perspective, the AS task has to be acknowledged as a recommendation task and is – in its recommendation of part types for CAD assemblies – an extension to the task we researched in [3].

While in that work, we predict the next required part type during assembly design, we now research recommendations of alternative part types for a specific part already existing within an assembly. Moreover, that approach was taken up by Lian et al. [7] which addressed the same questions on the datasets from [3] by developing a neural architecture search framework to automatically customize data-specific GNN models. The wide range of approaches to recommendation systems for CAD tools has been discussed in [3] in detail and most recently by [7]. Our choice of modeling the AS task as node classification is not unusual in the field of GNN-based recommendation [4], albeit link prediction is more common. In combining an anomaly detection task (e.g., identifying fraudsters) with a generic recommendation task (e.g., predicting interesting items for benign users), the GraphRfi study [15] comes close to our overall system, although their recommendation task is not meant to assist at alleviating the anomaly.

3 Creation of Synthetic Anomaly Data

As no real-world CAD anomaly handling dataset is available to us, it was necessary to create such kind of dataset synthetically. It comprises CAD assemblies, accurate descriptions of anomalies possibly contained in such designs and suggestions on how to correct (i.e., *regularize*) the respective design. We based our work on the three assembly datasets A, B and C provided by [3]. They comprise production-ready CAD assemblies originating from customers' orders in different assembly configurators, represented as undirected graphs (cf. Section 1).

3.1 Data Augmentation by Upsampling

As we aim to assist a design engineer during the entire assembly modeling process, we augment each dataset by extracting connected subgraphs from the assembly graphs, which represent all possible intermediate states during the design phase, in a self-supervised manner. This *upsampling* strategy is inspired by [3]: For each complete assembly graph $\mathcal{A}$, we iteratively cut off one leaf node until subgraphs (so-called *partial assemblies*) of a given minimum size are achieved. Unlike [3], the resulting sample set $\mathcal{S}_\mathcal{A}$ from pre-processing an assembly $\mathcal{A}$ consists of each resulting subgraph $\mathcal{A}'$ as well as the original assembly itself, whereby duplicates are eliminated. For an assembly dataset $\mathcal{D}$, all samples in sets $\mathcal{S}_\mathcal{A}$ across all of $\mathcal{D}$'s graphs $\mathcal{A}$ form the *multiset* $\mathcal{S}_\mathcal{D} = \biguplus_{\mathcal{A}\in\mathcal{D}} \mathcal{S}_\mathcal{A}$ of all partial assemblies for dataset $\mathcal{D}$, thus allowing duplicates across multiple sets $\mathcal{S}_\mathcal{A}$.

As is evident in Table 1, this approach generates an adequate number of samples compared to the limited number of assembly graphs in the original datasets. Note that this fact only makes the above approach feasible as the number of samples produced per assembly graph $\mathcal{A}$ is in $O(|\mathcal{V}|!)$, as [3] discussed.

Table 1. Dimensions of the original datasets [3] and after upsampling.

			After Upsampling		
Dataset	# Assemblies	# Part Types	Training	Validation	Test
A	11,826	1,930	74,638	24,151	24,587
B	11,895	3,099	453,477	140,906	145,288
C	11,943	1,924	247,575	85,512	90,060

3.2 Anomaly Generation

As earlier introduced, we focus on anomalies caused by the wrong selection of a part's type relative to its context within the assembly (e.g., attaching a robust hinge to a lightweight door). Understanding our work as a proof of concept in handling anomalous assemblies, we limit ourselves to one anomalous part per design (e.g., a single robust hinge in an otherwise lightweight cabinet). The small average size of the assemblies – especially in datasets A and C – make this choice necessary (cf. [3]). Hence, we make a single node v per assembly graph within our partial assemblies $\mathcal{S}_\mathcal{D}$ anomalous by replacing the part type of v with a type that would be considered anomalous in the context of v. Given that the assemblies in our datasets represent final designs, we assume their part connections to be *regular* and thus comprise a *rule set* $\mathcal{R}_\mathcal{D}$ of regular connections. The rule set is modeled as a dictionary mapping a part type τ to all part types that τ is directly connected to in any assembly within $\mathcal{D}$. Formally: $\tau \mapsto \big\{T(v_2) \mid \{v_1, v_2\} \in \mathcal{E}_\mathcal{D} \wedge T(v_1) = \tau\big\}$, where the type function $T : \mathcal{V} \to \mathcal{T}$ indicates the type of a part and $\mathcal{E}_\mathcal{D}$ unifies all edges across all assembly graphs in $\mathcal{D}$.

Algorithm 1 Convert Assembly $\mathcal{A} = (\mathcal{V}, \mathcal{E})$ to Anomalous Assembly

1: choose arbitrary part $v \in \mathcal{V}$
2: $\mathcal{R}_{N(v)} \leftarrow \bigcup_{n \in N(v)} \mathcal{R}(T(n))$
3: choose arbitrary part type $\tau' \in \mathcal{T} \backslash \mathcal{R}_{N(v)}$
4: $T(v) \leftarrow \tau'$
5: **return** $\mathcal{A}$

To generate the single anomaly within an assembly, we begin by randomly selecting one of its nodes v, and then extracting all part types $\mathcal{R}_{N(v)}$ explicitly allowed by the rule set for at least one of v's direct neighbors $N(v)$. Next, we randomly draw a part type τ' from the subset of $\mathcal{T}$ *not* covered by $\mathcal{R}_{N(v)}$, i.e., the set of part types that do not represent regular connections to any of v's neighbors according to $\mathcal{R}_\mathcal{D}$. Finally, we replace v's original part type with τ'. Algorithm 1 summarizes all steps of anomaly generation.

The described procedure for selecting an anomalous part type may produce quite obvious and unrealistic anomalies by selecting unsuitable types, i.e., a new type τ' that is very different from the original regular type τ, for instance,

replacing a hinge in the cabinet by a drawer slide. Therefore, we additionally limit the possible replacement type τ' (selected in line 3 of Algorithm 1) to stem from the same *part family* [11] as the original type. In part modeling, separate models are not usually created for minor variations of a part; instead, a single model is defined with multiple parametrizations to account for specific details, like different lengths of a screw. These variations within a shared design are collectively referred to as a part family. If there are no remaining part types of the same family, we fall back to employing a potentially obvious anomaly. This was necessary on dataset A for 4.7%, on dataset B for 13.3% and on dataset C for 3.8% of all partial assemblies created by upsampling.

Note that the extracted rule set is unlikely to be exhaustive as not every connection possible between two part types in the real world can be captured in a limited dataset. We can nevertheless rely on it to select a truly-anomalous part type: without restriction to the same part family, there remain at least 1727, 2087 and 1202 part types for selecting an anomalous type for the three datasets, respectively. It is thus highly likely that τ' does not fit its neighbors – even though τ' could nevertheless be suitable in the context of node v, forming a regular graph not mirrored by our dataset. With a part type τ constituting a specification of the core concept of its part family, we can reasonably assume these specifications to be a necessary detail when employing τ in a specific context and there are no random "hits" when selecting τ' from the same family.

3.3 Anomaly Data Point

After generating synthetic anomalous assemblies, we can consolidate all the information needed for anomaly handling into a so-called *anomaly data point* $\langle \mathcal{A},\ \bar{\mathcal{A}},\ \bar{v} \in \mathcal{V}_{\bar{\mathcal{A}}},\ \tau_0 \in \mathcal{T} \rangle$ consisting of the original regular assembly $\mathcal{A}$, the anomalized assembly $\bar{\mathcal{A}}$, the corresponding anomalous node $\bar{v}$ and its original part type τ_0. By storing the original part type τ_0 corresponding to an anomalized node $\bar{v}$, we preserve the one solution to the AS task accessible to us without expert knowledge (i.e., suggesting how to return to the data point's source graph). As our anomaly generation algorithm only introduces one anomalous node per assembly, we need to be aware of the strong class imbalance between anomalous and regular on node level, especially when evaluating the AD task. Particularly, the percentage of anomalous nodes within all anomaly data points' anomalized graphs is ranging from 13.6% in dataset B to 21.6% in dataset A.

4 Model Architectures

To perform our two separate AD and AS tasks, we define two models which expect task-specific views on our anomaly data points as input and targets (i.e., task-specific samples). Both tasks can be modeled as *node classification*: The AD model is expected to perform *binary classification* on each node, deciding whether its part type is anomalous. Similar to [3]'s approach of recommendations on graph-level, the AS model can be designed as a *multi-class classification*

model assigning each node a ranking over all part types $\mathcal{T}$ (rendering the specific number of recommendations adjustable). By modeling the latter as classification on node-level instead of graph-level, we can take advantage of the fact that each node's predicted ranking is tailored to its positional context. This allows us to "filter" for the specific node we want to generate alternative suggestions for as the model's output. Representing CAD assemblies as undirected graphs, it comes naturally to apply graph neural networks (GNN) – a choice we share with [3]. As a proof of concept, we focused on the two modern GNN flavors *GIN* [14] for its strength in distinguishing between non-isomorphic graphs and *GraphSAGE* [5] for its flexibility in generalizing to unseen instances as base architectures.

4.1 Anomaly Detection Model

Extracting the relevant data from our anomaly data points, we define *AD samples* to consist of an assembly graph $\mathcal{A}$ as input and a binary indicator of anomaly $f_v \in \mathbb{B}$ for each of its nodes $v \in \mathcal{V}$ as target (stacked to a node feature vector $f_\mathcal{V} \in \mathbb{B}^{|\mathcal{V}|}$). To ensure the detector can handle both anomalous and regular graphs, we derive two AD samples from each anomaly data point: one based on its anomalous assembly $\bar{\mathcal{A}}$, the other based on its regular assembly $\mathcal{A}$.

4.2 Adaptation Suggestion Model

The AS model is designed to expect samples containing an anomalous assembly $\bar{\mathcal{A}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ together with its anomalous node $\bar{v}$ as input and $\bar{v}$'s original part type τ_0 as target. As these *AS samples* only include anomalous graphs, but not regular ones, their number is only half the size of the AD samples. In order to achieve a ranking over all part types, it is necessary to define as many output neurons as the number of part types, with each neuron being associated with a specific part type $\tau \in \mathcal{T}$. Then, the output neurons' activations represent an unnormalized probability distribution over all part types, i.e., $\tilde{P}(\mathcal{T} \mid \bar{\mathcal{A}},\ \bar{v} \in \bar{\mathcal{V}})$, indicating how appropriate our model deems every part type for a specific node. By sorting them in descending order, we obtain the ranking we require.

Embedding Exclusion Strategies For predicting a suitable replacement part type for an anomalous node $\bar{v}$, its specific part type in $\bar{\mathcal{A}}$ is irrelevant and could even misguide the model. As the model's ranking is represented by the anomalous node's output neurons, the node itself is however indispensable to the model. To exclude the anomalous node's feature vector from the first round of message passing, we consider two strategies, *default embedding* and *masking*: The common former strategy involves replacing all anomalous nodes' initial embeddings with a shared embedding learned in the first layer before message passing. Masking, a novel method tailored specifically to address the problem, shares its name (as well as the basic idea of excluding nodes from message passing) with a technique proposed by Mishra et al. [9] developed to improve generalization and scalability of GNNs. The key innovation of our strategy lies in excluding (i.e., masking)

the embeddings of those nodes from initial message passing which we want to generate predictions for. Formally, we define our masking strategy as follows:

Definition 1. *Let $\rho^{(k)}$ denote the aggregate function, $\phi^{(k)}$ the combine function of a message-passing-based GNN and $h_v^{(k)}$ the feature vector of node v in the k^{th} layer with $h_v^{(0)}$ giving node v's input features. We define $\overline{\mathcal{M}}_v \subseteq N(v)$ as the subset of v's neighbors that are* not *masked. When applying masking, we alter the update rule for the first layer as follows:*

$$h_v^{(1)} = \begin{cases} \phi'^{(1)}\left(\rho^{(1)}\left(\left\{h_u^{(0)} \mid u \in \overline{\mathcal{M}}_v\right\}\right)\right) & \text{if } v \text{ is masked} \\ \phi^{(1)}\left(h_v^{(0)}, \rho^{(1)}\left(\left\{h_u^{(0)} \mid u \in \overline{\mathcal{M}}_v\right\}\right)\right) & \text{otherwise} \end{cases} \tag{1}$$

with $\phi'^{(1)}(v)$ constituting an adapted version of $\phi^{(1)}(v)$ leaving v's feature vector out of consideration.

5 Experiments

To assess the applicability of our architectural design of the AD and AS models defined in Section 4, we performed separate experiments for each of the three assembly datasets[1] A, B and C introduced in Section 3, covering GIN- and GraphSAGE-based architectures as well as all further architectural choices outlined above. The corresponding number of assemblies as well as distinct part types per dataset is given by Table 1; for detailed statistics on the assembly graph's structure we refer the reader to [3]. We evaluated AD and AS models' performance on their respective tasks separately as well as their working-together on the overall anomaly handling task (compare Section 5.2).

5.1 Experimental Setup

Employing the upsampling method described in Section 3 to each dataset, we achieved a significant amount of partial assemblies (compare Table 1), containing between 3 and 15 nodes. We excluded the few outliers above that upper boundary from sample creation and set the minimum partial assembly size to 3, considering pairs of nodes to be too trivial for our problem. The datasets provided by [3] already contained a reasonable data split of 60:20:20 which we adopted. All GNN-based architectures proposed were implemented using PyTorch leveraging the architecture-specific layers provided by the DGL library [13], using Optuna [1] for hyperparameter optimization. Both task models were trained on batched data using early stopping where the cross-entropy and the top-5 rate was monitored, respectively. Drawing upon our findings in [3] that using pretrained embeddings as node features improves GNN model performance, we used the same 100-dimensional embeddings that proved to perform best for part recommendation in order to represent the input graphs' part types across all tasks in this work.

[1] The assembly datasets based on pseudonymized part types can be accessed via `https://figshare.com/articles/dataset/ECML22_GRAPE_Data/20239767`.

5.2 Evaluation Methodology

Model Framing Breaking new ground with the tasks researched, we strived for contextualizing our GNN models' achieved performance by estimating upper and lower bounds. Suggesting alternative parts compares well to the task studied in [3], therefore we adopt its proposed *upper bound* and simple baseline *Evergreen* (as lower bound). The upper bound should specify the maximum achievable performance for a given number of recommendations k, which may be below 100% due to the multiple adaptation options for the same assembly, especially for small k. The model memorizes a ranking based on the frequency distribution of targets for a partial assembly within the test set (while replacing the anomalous node's part type with a default type not present in $\mathcal{T}$, cf. Section 4.2). In order to assess if the GNNs are capable of processing the graph context, the Evergreen model returns a static ranking over all part types encountered in the training set regardless of the specific partial assembly. Moreover, the strong class imbalance towards regular may cause the AD model to focus on this class. To determine whether it can truly make context-specific decisions regarding anomaly – rather than simply predicting the majority class – we employ a *majority predictor* (MP) that predicts each node to be regular. As there is only a single target for an input in the AD samples, there is no upper limit to its maximum possible performance.

Evaluation Metrics As the AD samples contain both regular and anomalized assemblies, the strong class imbalance is even more pronounced on node-level: among the AD test samples 6.8% (dataset B) to 10.8% (datasets A and C) of nodes are labeled as anomalous. As a consequence, precision and recall constitute the most expressive indicators of an AD model's performance among the canonical binary classification metrics. We selected the best model configuration based on the *F1-score* of the anomaly class as the corresponding precision and recall values best expose the AD models possible deficiencies due to class imbalance. We measure these metrics *on each class separately* and determine aggregates calculating a *macro* average[2]. Having modeled the AS model's output as a ranking, evaluating it using the top-k rate giving the percentage of the target part type appearing in the model's top k predictions is an obvious choice.

As $k = 5$ offers a compact window, yet big enough to contain the target in almost all predictions of our upper bound model (cf. Table 3), we based our selection of the best configuration on the top-5 rate. We also paid special attention to the top-10 rate as the first metric considered allowing perfect prediction from a theoretical perspective, agreeing with the assessment by [3] that ten recommendations can still be well integrated into CAD systems. The overall quality of the AS model's ranking across a dataset's test set is reflected by the targets' *mean reciprocal rank* (MRR) indicating its average position within the ranking.

[2] The severe imbalance renders weighted averages almost useless as multiplying class-specific values with their classes' support only exacerbates this.

End-to-End Evaluation Since suggesting alternative parts is the last step in our system, the top-k rate is also a suitable metric for the overall task of anomaly handling, to assess how well AD and AS models interoperate without human intervention in between them. To that extent we define an end-to-end (E2E) evaluation strategy which works by sequentially connecting AD and AS models: The AD-AS adapter (compare Fig. 1) transforms the node-wise classification $\hat{y}_{\mathcal{V}}$ predicted by the AD model into possibly multiple one-hot encodings $\hat{y}'_{\mathcal{V}}$, one for each predicted anomalous node, which are then – together with the assembly – fed into the AS model. From a mathematical perspective, we assess the prediction of a set of rankings covering each node our AD model recognizes as anomalous (which can be empty if the AD model deems the assembly regular). Our E2E evaluation logic is based on the idea that a well performing anomaly handling system should be able to successfully reconstruct the original assembly from its anomalized variant within its top-k recommendations.

For evaluation, we need custom *E2E samples* covering both tasks: As the AS samples $\langle \bar{\mathcal{A}}, f_{\mathcal{V}}, \tau_0 \rangle$ already contain the elements of the AD samples $\langle \bar{\mathcal{A}}, f_{\mathcal{V}} \rangle$ based on anomalized graphs in their input data, we use them as a basis for creating E2E samples to which we only need to append our AD samples based on regular graphs, i.e., $\langle \mathcal{A}, f_{\mathcal{V}} \rangle$, which obviously don't need a target part type τ_0.

We developed two different modes for our E2E evaluation logic, differing in their way of handling *false positives* (i.e., truly regular parts marked as anomalous) by the AD model, we termed *strict* or *result-based*, respectively: For an assembly graph $\mathcal{A}$ constituting the input to our overall anomaly handling system, we employ our AD model to predict a binary anomaly node classification $\hat{y}_{\mathcal{V}}$ as a general first step. If all of the graph's nodes are classified as regular, indicating our system perceives no need for adaptation suggestion, this result can be understood as its output – an empty set from a mathematical perspective – ready for evaluation: In case $\hat{y}_{\mathcal{V}}$ correctly labels $\mathcal{A}$ as all regular, we can consider the not-performed AS prediction as trivially correct and thus count the sample as a hit across all k. In the opposite case or generally if the AD model did not identify the truly anomalous node, there is no way for our system to predict this node's original part type. Hence, we then cannot count any hit regardless of k.

If, after all, the truly anomalous node has correctly been labeled as anomalous, the further procedure depends on the chosen evaluation mode: When evaluating in strict mode, we only continue processing if the truly anomalous node is the *only one* $\hat{y}_{\mathcal{V}}$ asserts to be anomalous. Else – if additional nodes were misidentified as anomalous – we interpret the prediction on the sample as a fail across all ks. If the AD model was perfectly right or when evaluating in result-based mode, we transform its output using the AD-AS adapter as described above and iteratively ask the AS model to predict a replacement part type τ_0 for the respective node $\bar{v}$ of assembly $\mathcal{A}$ represented by the one-hot vector $\hat{y}'_{\mathcal{V}}$.

In strict mode (with only predictions generated for one node), top-k evaluation corresponds to separate AS model evaluation. In result-based mode, we allow our system to recover from a AD-misclassification of a truly regular node by the AS model ranking the part type of that node among its k most confident

Table 2. Anomaly detection (AD) results based on precision and recall, both calculated individually for each class and macro-averaged, with a majority predictor (MP) as baseline. All values are given in percent.

		Regular		Anomalous		Macro	
Dataset	Model	Precision	Recall	Precision	Recall	Precision	Recall
A	**GraphSAGE**	**99.6**	**98.1**	**86.2**	**96.6**	**92.9**	**97.3**
	GIN	99.2	98.1	85.5	93.4	92.4	95.8
	MP	89.1	100.0	0.0	0.0	44.6	50.0
B	**GraphSAGE**	**99.8**	98.9	86.9	**97.9**	93.4	**98.4**
	GIN	99.7	**99.1**	**88.0**	95.9	**93.9**	97.5
	MP	93.3	100.0	0.0	0.0	46.6	50.0
C	**GraphSAGE**	**99.8**	**99.5**	**94.6**	**98.1**	**97.2**	**98.8**
	GIN	**99.8**	99.3	92.3	97.4	96.0	98.4
	MP	92.2	100.0	0.0	0.0	46.1	50.0

suggestions. We thus obtain a hit relative to a specific k in this mode if $\mathcal{A}$'s original graph can be reconstructed by selecting a replacement part type from the respective node's top-k predictions for each node marked as anomalous.

5.3 Experimental Results

Anomaly Detection In our experiments, the GNN-based AD models both clearly outperform the baseline model. While the majority predictor inherently fails to identify any anomalous nodes – indicated by precision and recall of 0.0% – the GNN-based models single out anomalous nodes with a high recall while achieving reasonable precision over all datasets, see Table 2. The higher recall compared to precision for anomalous nodes in both GNN flavors shows the practicality of our model, ensuring most anomalies are detected. The wide margin of this difference (at least for datasets A and B) is mirrored in the macro average. Performance on the regular class also proved superior with the GNN models missing only a minimal share of regular nodes (which the baseline model identified perfectly at the expense of regular precision and overall performance on anomalous nodes) while raising precision on the nodes identified as regular by ten percentage points to almost perfect 99% across all experiments.

The superior performance of both AD model architectures on dataset C can not be traced back to the fallback rate during anomaly generation (4.8% dataset A, 12.1% dataset B, 4.1% dataset C on their test sets) but most plausibly to the interplay between their respective number of distinct part types and the number of partial assemblies created by upsampling for each dataset (compare Table 1). With nearly identical fallback rates and numbers of part types, dataset C contains more than three times as many samples than dataset A. The fact that dataset B contains almost double the amount of samples than dataset C cannot compensate for the third more part types C encompasses. GraphSAGE-

based architectures prove superior on all metrics for datasets A and C, as well as in the critical metric of recall for the anomalous class on dataset B.

Adaptation Suggestion In our experiments, all variants of our GNN-based AS models fulfilled their task with success, standing out from Evergreen models by a wide margin while – from five suggestions onwards – nearing the theoretical upper bound by less than 20%, with the best variants providing a useful preselection of five part types to the user for 84.8% to 87.9% of anomalous graphs evaluated (compare Table 3). Across all datasets, the GraphSAGE-based models proved to perform better (regardless of the specific embedding exclusion strategy) with their respective best variants differing between 1.9% and 3.0% for A, between 0.9% and 1.1% for B and between 0.4% and 0.6% for C.

For datasets A and C, we can identify a performance advantage of using a default embedding compared to masking during the first round of message passing for both GNN flavors. This corresponds well to the observation that the performance boost achieved by the default embedding strategy for this model variant gradually declines with increasing k. For dataset B, the GraphSAGE model employing masking turned out to be the best performing one across all configurations, however with an only slight lead at the beginning and ending in a draw for the significant number of recommendations $k \in \{3, 5, 10\}$. This deviation could possibly result from the size and the high ambiguity of this dataset (with an average assembly consisting of three times as many nodes and the dataset containing over half more part types than the others; compare Tables 1 and [3]). It is remarkable that the best model for dataset C initially performs relatively poor (with its top-1 rate 30% below the upper bound), then relatively abruptly achieves extraordinary results for $k \in \{5, 10, 15\}$ indicating that this model tends to assign the target part a relatively low confidence still within a larger prediction window noticeable in the targets' mean reciprocal rank (MRR). The best model for dataset B shows the opposite development: starting with comparably high rates for $k = 1$ and $k = 3$ yet resulting in the lowest performances for $k > 5$ across all models which does not surprise in the light of the dataset's complexity.

E2E Evaluation We applied E2E evaluation of the overall anomaly handling task to the AD and AS models proven to perform best on each dataset (marked as bold in Tables 2 and 3), achieving top-5 rates of 74.4%, 76.5% and 78.3%, as well as top-10 rates of 79.9%, 81.3% and 83.7% in result-based mode on the three datasets, respectively. These results suggest that our system in general meets its purpose of successfully assisting engineers during the whole workflow of regularizing assemblies, indicating that in three quarters of cases designers can achieve a regular assembly from the assembly they hand over to the system – which can be regular or not – by only following our system's recommendations.

Assuming that novice engineers are likely more comfortable selecting from provided alternative suggestions rather than determining a part's anomalous nature without further context, this result provides a lower bound estimate achiev-

Table 3. Adaptation suggestion (AS) results based on the top-k rate for $k = 1, \ldots, 15$ recommendations and mean reciprocal rank (MRR). The top-k rate is given in percent.

Dataset	Model	Strategy	1	3	5	10	15	MRR
A	Upper Bound		91.4	98.8	99.6	100	100	0.951
	GraphSAGE	**Default Rep.**	**64.6**	**83.1**	**87.8**	**92.5**	**94.3**	**0.750**
		Masking	64.1	82.6	87.2	92.0	93.8	0.744
	GIN	Default Rep.	62.7	80.1	85.4	90.2	92.2	0.728
		Masking	60.5	79.5	85.2	90.7	92.7	0.714
	Evergreen		3.2	7.2	9.2	13.2	16.8	0.063
B	Upper Bound		92.2	98.5	99.3	99.8	99.9	0.954
	GraphSAGE	Default Rep.	60.6	**79.8**	**84.8**	**89.7**	**91.7**	0.714
		Masking	**61.1**	**79.8**	**84.8**	**89.7**	**91.7**	**0.718**
	GIN	Default Rep.	60.2	78.8	83.8	88.6	90.8	0.708
		Masking	57.3	76.1	81.8	87.9	90.5	0.683
	Evergreen		4.2	11.0	18.0	30.1	37.0	0.111
C	Upper Bound		84.3	97.8	99.2	99.8	99.9	0.910
	GraphSAGE	**Default Rep.**	**53.8**	**79.8**	**87.9**	**93.8**	**95.5**	**0.683**
		Masking	53.1	79.3	87.7	93.7	**95.5**	0.677
	GIN	Default Rep.	53.2	79.4	87.5	93.3	95.0	0.677
		Masking	53.0	78.9	86.8	93.1	95.0	0.674
	Evergreen		5.8	15.8	21.0	27.9	34.2	0.129

able with minimal cognitive burden. As the result-based metrics perform only marginally better, there remains much potential for optimization in future work.

6 Conclusion and Outlook

In this work, we present a framework for detecting anomalous parts in CAD assemblies (AD) and suggesting alternatives (AS) to support the assembly design process based on a company's common design practices implicitly comprised in their past designs. Our experiments show that our two proposed separate GNN-based AD and AS models not only significantly outperform non-ML models in their respective tasks, but also work well together for the anomaly handling task. Our adaptation suggestion model appears especially promising with its wide range of use cases going beyond handling anomalies: It is capable of suggesting suitable part types at every position within an assembly – either for offering alternatives to an existing part's type or for recommending part types for newly added parts to the assembly. This constitutes an extension to the recommendation systems presented in [3] and its customization proposed in [7].

In light of the success of our proof of concept, a next logical step will be addressing multiple and more complex anomalies within an assembly, such as replacing multiple connected anomalous parts by a single regularizing part.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A Next-generation Hyperparameter Optimization Framework (2019)
2. Chandola, V., Banerjee, A., Kumar, V.: Anomaly Detection: A Survey. ACM computing surveys (CSUR) **41**(3), 1–58 (2009)
3. Gajek, C., Schiendorfer, A., Reif, W.: A Recommendation System for CAD Assembly Modeling based on Graph Neural Networks. In: ECML PKDD. pp. 1 – 17 (2022)
4. Gao, C., Zheng, Y., Li, N., Li, Y., Qin, Y., Piao, J., Quan, Y., Chang, J., Jin, D., He, X., Li, Y.: A Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions. ACM Trans. Recomm. Syst. **1**(1) (2023). https://doi.org/10.1145/3568022, `https://doi.org/10.1145/3568022`
5. Hamilton, W., Ying, Z., Leskovec, J.: Inductive Representation Learning on Large Graphs. NeurIPS **30** (2017)
6. Lenzen, C., Schiendorfer, A., Reif, W.: Graph Machine Learning for Assembly Modeling. In: LoG (2022)
7. Liang, F., Zhao, H., Quan, Y., Fang, W., Shi, C.: Customizing Graph Neural Network for CAD Assembly Recommendation. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. p. 1746–1757. KDD '24 (2024)
8. Ma, X., Wu, J., Xue, S., Yang, J., Zhou, C., Sheng, Q.Z., Xiong, H., Akoglu, L.: A Comprehensive Survey on Graph Anomaly Detection With Deep Learning. IEEE Transactions on Knowledge and Data Engineering **35**(12), 12012–12038 (2023)
9. Mishra, P., Piktus, A., Goossen, G., Silvestri, F.: Node Masking: Making Graph Neural Networks Generalize and Scale Better **abs/2001.07524** (2020)
10. Sarcar, M., Rao, K.M., Narayan, K.L.: Computer Aided Design and Manufacturing. PHI Learning Pvt. Ltd. (2008)
11. Schoonmaker, S.J.: The CAD Guidebook: A Basic Manual for Understanding and Improving Computer-Aided Design. CRC Press (2002)
12. Srinivas, S.S., Kumar Sarkar, R., Runkana, V.: Hypergraph Learning based Recommender System for Anomaly Detection, Control and Optimization. In: IEEE International Conference on Big Data. IEEE (2022)
13. Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., Huang, Z., Guo, Q., Zhang, H., Lin, H., Zhao, J., Li, J., Smola, A.J., Zhang, Z.: Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
14. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How Powerful are Graph Neural Networks? (2019)
15. Zhang, S., Yin, H., Chen, T., Hung, Q.V.N., Huang, Z., Cui, L.: GCN-Based User Representation Learning for Unifying Robust Recommendation and Fraudster Detection (2020), `https://arxiv.org/abs/2005.10150`