Applied Artificial Intelligence

An International Journal

ISSN: 0883-9514 (Print) 1087-6545 (Online) Journal homepage: www.tandfonline.com/journals/uaai20

Augmentation of Semantic Processes for Deep Learning Applications

Maximilian Hoffmann, Lukas Malburg & Ralph Bergmann

To cite this article: Maximilian Hoffmann, Lukas Malburg & Ralph Bergmann (2025) Augmentation of Semantic Processes for Deep Learning Applications, Applied Artificial Intelligence, 39:1, 2506788, DOI: 10.1080/08839514.2025.2506788

To link to this article: https://doi.org/10.1080/08839514.2025.2506788

APPLIED ARTIFICIAL INTELLIGENCE
2025, VOL. 39, NO. 1, e2506788 (48 pages)
https://doi.org/10.1080/08839514.2025.2506788

Augmentation of Semantic Processes for Deep Learning Applications

Maximilian Hoffmann [a,b], Lukas Malburg [a,b], and Ralph Bergmann [a,b]

[a]Artificial Intelligence and Intelligent Information Systems, University of Trier, Trier, Germany; [b]German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, Trier, Germany

ABSTRACT
The popularity of Deep Learning (DL) methods used in business process management research and practice is constantly increasing. One important factor that hinders the adoption of DL in certain areas is the availability of sufficiently large training datasets, particularly affecting domains where process models are mainly defined manually with a high knowledge-acquisition effort. In this paper, we examine *process model augmentation* in combination with *semi-supervised transfer learning* to enlarge existing datasets and train DL models effectively. The use case of similarity learning between manufacturing process models is discussed. Based on a literature study of existing augmentation techniques, a concept is presented with different categories of augmentation from knowledge-light approaches to knowledge-intensive ones, e. g. based on automated planning. Specifically, the impacts of augmentation approaches on the syntactic and semantic correctness of the augmented process models are considered. The concept also proposes a semi-supervised transfer learning approach to integrate augmented and non-augmented process model datasets in a two-phased training procedure. The experimental evaluation investigates augmented process model datasets regarding their quality for model training in the context of similarity learning between manufacturing process models. The results indicate a large potential with a reduction of the prediction error of up to 53%.

ARTICLE HISTORY
Received 9 September 2024
Revised 13 March 2025
Accepted 19 March 2025

Introduction

Data-driven Artificial Intelligence (AI) methods such as Deep Learning (DL) have recently gained significant importance in practice and research. One of these research areas for the use of DL techniques is BPM (e.g., Di Francescomarino and Ghidini 2022; Nolle et al. 2018; Pfeiffer, Lahann, and Fettke 2021; Rama-Maneiro, Vidal, and Lama 2023). However, to fully utilize the DL techniques and their generalization capabilities, a rich set of training data with meaningful training examples is needed, which leads to considerable

CONTACT Maximilian Hoffmann hoffmannm@uni-trier.de, Artificial Intelligence and Intelligent Information Systems, University of Trier, 54296 Trier, Germany
Supplemental data for this article can be accessed online at https://doi.org/10.1080/08839514.2025.2506788

effort for data acquisition (Hoffmann and Bergmann 2022b; Schuler et al. 2023).

As this effort is not always manageable and sometimes training data is inaccessible, DL methods can be combined with other AI techniques in an approach commonly known as informed machine learning or hybrid AI (von Rueden et al. 2021). There, the disadvantage of not enough or imbalanced data is compensated by symbolic knowledge. A more common technique to mitigate these data issues in practice and research is *data augmentation* (Mumuni and Mumuni 2022; van Dyk, David, and Meng 2001; Zhao et al. 2022; Zhou et al. 2023). Data augmentation aims to increase the amount of available training data to make the DL models trained on it more accurate and robust. In domains such as image processing and Natural Language Processing (NLP) (e. g., Chen, Kornblith et al. 2020), elementary transformational augmentations (e.g., the rotation of an image or the replacement of words with synonyms) or synthesis-based augmentations using generative models have proven effective. These techniques typically exploit the intrinsic structure of their respective data types, ensuring that the augmented examples remain consistent with the original data distribution.

In contrast, in BPM and specifically for process models, the exploration of augmentation techniques is still in its infancy. Previous studies (e. g., de Leoni, van der Aalst, and Dees 2016; Käppel and Jablonski 2023; Käppel, Schönig, and Jablonski 2021; Venkateswaran et al. 2021) have introduced augmentation strategies that are largely tailored to specific domains or are limited to event log data. For instance, while traditional data augmentation methods focus primarily on simple and domain-agnostic transformations, these BPM-specific approaches often rely on ad-hoc modifications that do not generalize well across different types of process models or application contexts. Moreover, many of these studies do not address the challenge of preserving the semantic integrity of the process models during augmentation, a critical aspect to ensure that the augmented data remain useful for training robust DL models.

Our work aims to bridge this gap. The goal of this paper is to discuss approaches of process model augmentation in combination with an effective training method of DL models with augmented and non-augmented process models. The contributions of the paper are the following: 1) we present a literature study on process augmentation techniques that are currently available to be used in DL contexts, 2) we discuss three categories of augmentation from knowledge-light approaches, i. e., deleting and replacing nodes and edges, to knowledge-intensive ones based on automated planning (Marrella 2019; McDermott et al. 1998), 3) we present a training procedure based on semi-supervised transfer learning (Kudenko 2014; Tan et al. 2018; Weiss, Khoshgoftaar, and Wang 2016) to train DL models effectively with augmented and non-augmented data, 4) we conduct an extensive evaluation in which we determine the suitability of different augmentation methods for improving the training procedure of DL models based on our previous work (Hoffmann et al. 2020; Schuler et al. 2023). Throughout the paper, we examine the task of learning similarities between process models from a manufacturing domain (Malburg, Klein, and Bergmann 2020, 2023; Seiger et al. 2022) as the main use case.

The paper is structured as follows: In Section 2, we describe the foundations that consist of the process model representation and the application domain applied in this work, the semantic similarity assessment between process models, and the basics of using graph neural networks for graph embedding. In addition, we present and discuss related augmentation approaches (see Section 3). Based on these foundations and the identified research gaps, a concept is presented to augment process models and use them to train DL models (see Section 4). To assess the utility of the concept for the addressed problems, an experimental evaluation is conducted by using several training data configurations for GNN based on the discussed augmentation techniques (see Section 5). In this context, we use several well-known metrics to evaluate the quality of the trained models for process model similarity learning. Finally, a conclusion is given, and future work is discussed in Section 7.

Foundations

In this section, we introduce the foundations of the semantic process model representation and the manufacturing domain with its underlying domain model that serves as an application scenario (see Section 2.1 and Section 2.2). Since the focus of the work is using DL models for similarity learning, Section 2.3 introduces how similarities are computed between workflows and Section 2.4 follows with GNN that are trained for this task. Finally, the proposed approach is embedded into a broader literature context with a focus on generic graph augmentation methods (see Section 3.1) and more specific approaches from BPM literature (see Section 3.2).

Semantic workflow representation and domain

Process models consist of several components that interact with each other to achieve the common goal of the process. For example, the main components of process models are activities that represent tasks or actions. These activities have relationships with other components such as the participants who perform the activity, the data used during the execution, etc. There are different ways of representing process models, e. g., BPMN, Petri Nets, or BPEL/WS-BPEL (Schultheis et al. 2024). This work uses semantically annotated directed graphs, referred to as *NEST* graphs (Bergmann and Gil 2014). The NEST graph format is a generic graph format that is well-suited for modeling processes, as processes have an inherent graph structure of components and interrelations between them. The strength of NEST graphs is in modeling semantic information with an underlying domain model, and they are well-integrated in our previous work (Malburg, Hoffmann and Bergmann 2023). To ensure compatibility with other more common process model representation formats such as BPMN, we employ a converter from BPMN to NEST and back. The converter is available as part of the ProCAKE framework[1] (Bergmann et al. 2019). All process models used in the experiments are represented in NEST and BPMN, created with this converter. See Section 5 and the statement on data availability at the end of the article for more information.

Each *NEST* graph is a quadruple $G = (N, E, S, T)$, defined by a set of nodes N, a set of edges $E \subseteq N \times N$, a function $T : N \cup E \rightarrow \mathcal{T}$, assigning a type to each node and edge,[2] and a function $S : N \cup E \rightarrow \mathcal{S}$, assigning a semantic description from a *semantic metadata language* (e. g., an ontology) to nodes and edges. While nodes and edges build the structure of each graph, types and semantic descriptions are employed to further capture semantic information. As a result, both nodes and edges always have a type and usually also have a semantic description.

The definition of NEST graphs is rather generic to allow their usage in diverse domains, e.g., to represent cooking recipes (Müller 2018), to solve constraint satisfaction problems (Grumbach and Bergmann 2017), to model scientific workflows (Zeyen, Malburg, and Bergmann 2019), to represent manufacturing processes (Malburg et al. 2020), or to define argumentation graphs (Lenz et al. 2019). As the process models addressed in this work are characterized by certain rules on top of the generic NEST graph definition, we introduce the concept of a *sequential NEST process model*:

> A sequential NEST process model is a NEST graph with a sequential control-flow, a sequential data-flow, and without missing semantic annotations.

By a *sequential control-flow*, we mean that every task node in the process is connected to exactly one preceding task node and one following task node by a control-flow edge. There are two exceptions to this rule, i.e., the start task with only one outgoing control-flow edge and the end task with only one ingoing control-flow edge. The second important extension of NEST graphs toward sequential NEST process models is their *sequential data-flow*. This property is similar to the sequential control-flow but refers to data nodes: Each data node has to be produced by exactly one task and consumed by exactly one task. There is also the exception of two data nodes, i.e., at the beginning of the sequence and at the end, which are consumed or produced only, respectively. A sequential NEST process model with a sequential data-flow prohibits that, for instance, a task consumes or produces multiple data nodes. The third property of sequential NEST process models is about the semantic annotations of nodes and edges that must not be missing. This restriction contributes to the completeness of the process' semantic knowledge and aims to prevent processes with nodes or edges that lack any form of semantic annotation and can only be characterized by their type.

The manufacturing application scenario that we utilize for this work is an example of a domain of sequential NEST process models. The scenario is based on process models that are executed in a smart factory[3] (Malburg, Klein, and Bergmann 2020, 2023; Seiger et al. 2022). These processes are considered as *cyber-physical workflows* (Marrella et al. 2018; Seiger et al. 2019) because they are executed by *actuators* and are driven by *sensors* capturing the production environment. Figure 1 illustrates an exemplary manufacturing process as a sequential *NEST* process model, representing the production steps for producing sheet metals. The illustrated process is a typical example of the processes used in this work. *Task nodes* represent concrete activities during production that are executed by actuators, i. e., machines, in the smart factory by a corresponding service. The state of the product produced is captured by the *data nodes*. Each data node represents the current production state of the sheet metal. The semantic descriptions of task and data nodes are used to describe the properties of these nodes. For example, the semantic description of *Transport from Warehouse to Oven* contains relevant parameters for configuring the machine that executes it, e.g., the start and end position of the transport. Similarly, the semantic description of the first *Sheet Metal* data node is composed of the concrete position of the product, i.e., *ov_1*, and its characteristics, e. g., the *size* and *thickness*.

This example shows that the semantic annotations are a key aspect of the information a process holds. In particular, it shows the tight coupling of the *syntactic information* that is given by the representation as a sequential NEST process model, e. g., node and edge types and graph structure, and the *semantic information* that is expressed by the annotations of nodes and edges, e. g., the position of a workpiece. However, a process such as the one shown in Figure 1 that is consistent with all constraints of sequential NEST

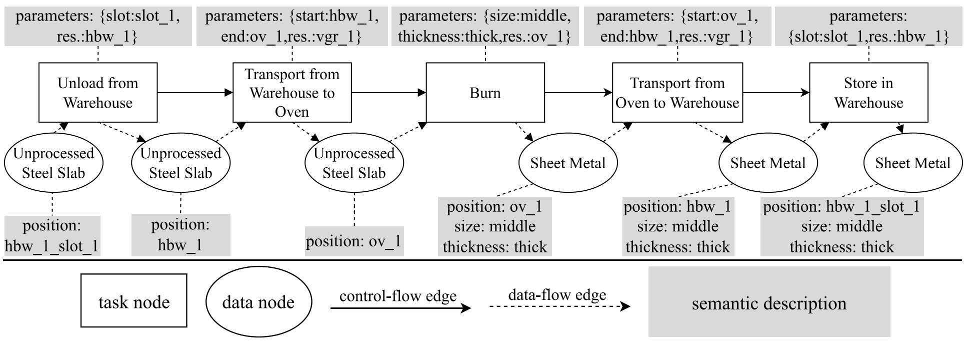
parameters: {slot:slot_1, res.:hbw_1}
parameters: {start:hbw_1, end:ov_1,res.:vgr_1}
parameters: {size:middle, thickness:thick,res.:ov_1}
parameters: {start:ov_1, end:hbw_1,res.:vgr_1}
parameters: {slot:slot_1,res.:hbw_1}
Unload from Warehouse
Transport from Warehouse to Oven
Burn
Transport from Oven to Warehouse
Store in Warehouse
Unprocessed Steel Slab
Unprocessed Steel Slab
Unprocessed Steel Slab
Sheet Metal
Sheet Metal
Sheet Metal
position: hbw_1_slot_1
position: hbw_1
position: ov_1
position: ov_1 size: middle thickness: thick
position: hbw_1 size: middle thickness: thick
position: hbw_1_slot_1 size: middle thickness: thick
task node
data node
control-flow edge
data-flow edge
semantic description

Figure 1. A NEST graph representing a sheet metal manufacturing process (based on: Malburg, Brand, and Bergmann 2023).

process models and has well-formed semantic annotations according to the respective semantic metadata language is not guaranteed to be executable in the smart factory, i. e., the manufacturing environment. The reason for this is the inability of the semantic metadata language to describe the behavior of the process tasks in the environment during execution. For example, the task *Burn* in Figure 1 is parameterized by its semantic annotation to produce a middle-sized thick sheet metal. After being burned, the sheet metal is transported to the first HBW to be stored there. While the semantic annotations of all the tasks involved are within the definitions of the semantic metadata language, there is still room for errors. It might be possible that the first VGR is only capable of moving small-sized workpieces, or that the shop floor layout could prevent the first VGR from reaching the designated storage buckets in the first HBW. Both problems could be solved by using the second VGR that transports the workpiece to the second HBW.

Domain model

An additional knowledge representation, called *domain model*, is utilized to check if a sequential NEST process model is executable in its designated manufacturing environment. The domain model also allows interoperability of a process with other systems of the application context, e. g., ERP systems, MES, or WfMS (Seiger et al. 2022). The complexity and completeness of the information provided by the domain model can vary greatly and depend on the respective domain.

Listing 1 Planning Action for the Burn Service.

```
(:action ov_1_burn
:parameters (?resource - oven ?size - sheet_size ?thickness -
    sheet_thickness ?processID - processID)
:precondition
  (and
  (at ?processID ov_1_pos) (hasPosition ?resource ov_1_pos)
  (not (isInactive ?resource))
  (isReady ?resource)
  (isSteelSlab ?processID) (not (isSheetMetal ?processID)))
:effect
  (and (isSheetMetal ?processID)
  (isSheetMetalWithSize ?size ?processID)
  (isSheetMetalWithThickness ?thickness ?processID)
  (not (isSteelSlab ?processID))
  (increase (total-cost) (Service_OV_Burn_With_Resource_OV_1
      _With_Size_Parameter_With_Thickness _Parameter_Cost
      ?size ?thickness))))
)
```

While rather simple domains might have simpler domain models, more complex domains might also have more complex domain models. The model used in this work is given as a comprehensive planning domain description based on the PDDL (McDermott et al. 1998). The planning domain model consists of the available actions of the smart factory enhanced with semantic information about the preconditions that must be satisfied for execution and the effects that hold after a successful execution. We use a semantic service-based architecture for the smart factory (see Malburg, Klein, and Bergmann 2020; Seiger et al. 2022) and, based on this architecture, a domain expert creates the planning actions contained in the domain description. Listing 1 illustrated a planning action created by the domain expert for the *Burn* service. A planning action in PDDL consists of an action name (Line 1), parameters for configuring the action (Line 2), preconditions that need to be satisfied for execution (Lines 3–8), and effects that represent the state transition after successful execution (Lines 9–14). Each task executed by a service in the smart factory has several parameters to be configured (see Section 2.1). For the illustrated *Burn* task, the executing resource, the size of the burned sheet metal, and the thickness must be specified. In this context, it is important to note that the process ID is a reference to the corresponding process from the smart factory. The preconditions specify the world state in which the action can be executed. For the *Burn* service, the workpiece must be at the location of the oven (Line 5). In addition, the oven must be ready and not inactive (Lines 6–7) as well as the workpiece must be a steel slab, i. e., it must not be burned already (Line 8). After executing the action, the state of the product changes: the workpiece is no steel slab anymore but a sheet metal with a certain size and thickness (Lines 9–13). To capture the costs for executing an action with certain parameters, the total cost function is increased by a predefined value for the service executed (Line 14).

The information in the domain model allows us to check whether a process model can be executed successfully by considering the defined preconditions and effects. It also allows the generation of process models by using *automated planning* (Ghallab, Nau, and Traverso 2016; Haslum et al. 2019), which is applied for process model augmentation in this paper.

Semantic workflow similarity

The combination of syntax and semantics of sequential NEST process models shows the complexity that is involved with all applications working with this data. One task that arises in these applications is the comparison of process models, e. g., for finding a replacement in case the execution of a different process model fails (Malburg, Brand, and Bergmann 2023; Malburg, Hoffmann, and Bergmann 2023). To compare different process models, similarities are a common measurement. We also use similarity computation as a demonstrator in the experiments of the paper, as it is widely used in different fields of AI. One example is CBR: In the problem-solving methodology of CBR, similarities are widely used to determine which process models are alike and can help to solve emerging problems. Thereby, a repository of process models is queried with a different process model to find the best-matching process models of the repository. This is useful since the found process models might be better suited to solve the problem at hand, e. g., a sudden failure of a machine, than the currently executed process model.

There are various methods for calculating similarities between process models. These methods are sometimes consolidated under the term business process matching (Weidlich, Remco, and Jan 2010). Schoknecht et al. (2017) categorize different methods based on seven characteristics, e. g., objective and implementation, and, thus, provide a comprehensive overview of the available literature. We use the semantic similarity measure proposed by Bergmann and Gil (2014) as it is specifically tailored for NEST graphs and focuses on semantic similarity measures on a node-level, which fits our use case.

To compute the similarity between two *NEST* graphs, the similarity measure evaluates the structure of nodes and edges, as well as the semantic descriptions and types of these components. The final similarity at the graph level is determined by the similarities between the nodes and edges of both graphs (Richter 2007). Therefore, a global similarity, denoting the similarity between two graphs, consists of local similarities, i. e., the pairwise similarities of nodes and edges. The similarity between two identical types of nodes is defined as the similarity of the semantic descriptions of these nodes. The similarity value between two edges of the same type includes not only the similarity between the respective semantic descriptions but also the similarity of the connected nodes. The similarities of the semantic descriptions are computed based on the underlying domain model. For instance, in the given exemplary process model illustrated in Figure 1, the similarity between two task nodes would include further similarities between their parameters. This procedure requires the definition of similarity measures for all components of the semantic descriptions as part of the domain's similarity knowledge. These metrics are usually designed with the input of experts in the field and consider the particular data types in use. For example, they could utilize word embeddings (Mikolov et al. 2013) for textual data or the MAE for numerical values (Bergmann 2002).

To determine the global similarity between a pair of graphs from the local similarities of the nodes and edges, an injective partial mapping is computed. The objective of this mapping is to maximize the aggregated local similarities between all mapped pairs of corresponding nodes and edges. Nonetheless, as the number of nodes and edges gets larger, the complexity of finding this mapping increases substantially, given that each node and edge can potentially be mapped to multiple counterparts. To address this complexity, Bergmann and Gil (2014) propose using an A* search algorithm. A* search is faster than exhaustive search, but typically still requires a considerable amount of time and, thus, can result in a slow similarity computation (Hoffmann and Bergmann 2022b; Hoffmann et al. 2020, 2022; Klein, Malburg, and Bergmann 2019; Ontañón 2020; Zeyen and Bergmann 2020).

Graph embedding

Previous work by Hoffmann and Bergmann (2022b) tackles the problem of slow similarity computations by using an automatically learned similarity measure based on a Siamese GNN, denoted as the GEM. The basic idea is to learn to predict the pairwise similarities of workflows with the GEM for faster and sometimes even more accurate similarity computations (Hoffmann and Bergmann 2022b). Therefore, the GEM transforms the graph structure and the semantic annotations and types of all nodes and edges into a whole-graph latent vector representation. These vectors are then combined to calculate a similarity value. An overview of the used GNN architecture is illustrated in Figure 2.

The GEM follows a general architecture composed of four parts: First, the *embedder* starts by transforming the node and edge features into initial embeddings within a vector space. These features encompass semantic annotations and types, and they undergo a specialized encoding and processing procedure tailored for semantic graphs (for more details, see). Next, the *propagation layer* collects information for each node from its local surroundings by transmitting messages, as described by Gilmer et al. (2017). In detail, the vector representation of a node is continuously updated by incorporating the vectors of neighboring nodes linked through incoming edges. This iterative process is repeated multiple times. Following that, the *aggregator* combines the node embeddings at this stage to generate a vector representation of the entire graph. Eventually, the *similarity* between two graphs is calculated by

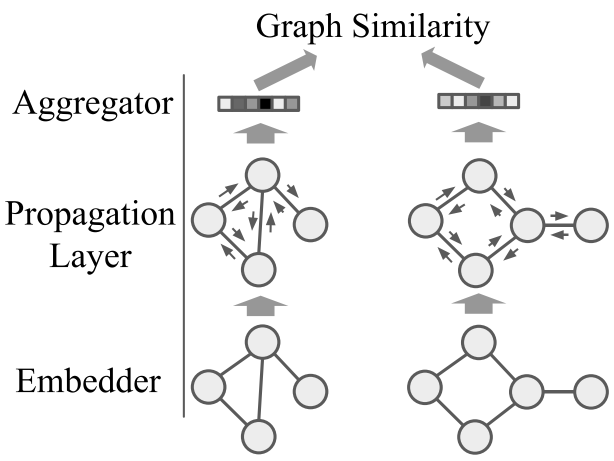

Figure 2. The GEM (Source: Hoffmann et al. 2020).

vector similarity measures, such as cosine similarity, based on the respective graph representations within the vector space.

The trainable parameters of the GEM are modeled as part of the embedder, the propagation layer, and the aggregator. They are trained and adjusted according to a comparison of pairs of predicted similarities and ground-truth similarities by a MSE loss function. After being trained, the neural network can be used as a replacement for the conventional similarity measure based on graph matching (see Section 2.3).

The evaluation of the initial approach (Hoffmann and Bergmann 2022b; Hoffmann et al. 2020) compares the accuracy of the similarity predictions of the embedding-based approach and the graph matching approach, restricting the time budget of the graph matching approach to match that of the embedding-based approach. The results, in general, show more accurate predictions of the embedding-based approach in this scenario. The accuracy changes, however, when increasing the time budget of the graph matching approach, thus, revealing a trade-off between accuracy and computation time. Based on the initial approach, the embedding-based approach has been extended and improved further with, for instance, different components or training scenarios (Hoffmann and Bergmann 2023; Pauli, Hoffmann, and Bergmann 2023; Schuler et al. 2023). The evaluation results of these extensions are transitively applicable to the comparison between embedding-based and graph-matching-based similarity computation in Hoffmann et al. (2020) and Hoffmann and Bergmann (2022b). Overall, the following advantages and disadvantages of both methods are observed:

- The inference runtime of the embedding-based approach is consistent across different runs and only marginally varies based on the size of the input graphs. The runtime of the A*-based graph matching procedure can vary greatly from run to run, due to the integrated search. On the downside, this also means that there are only a few designated mechanisms to configure the runtime budget of the embedding-based approach based on the requirements of different use cases, e. g., general purpose strategies such as integer quantization (Wu et al. 2020). The graph matching approach can be precisely configured by restricting the number of solution candidates to consider during search (for more details, see Bergmann and Gil 2014).
- The embedding-based approach can benefit greatly from modern hardware accelerators for AI workloads (Chen, Xie et al. 2020). The graph matching approach is mainly carried out by the CPU and acceleration on, for instance, GPUs requires custom implementations (e. g., Hoffmann et al. 2022; Zhou and Jianyang 2015).
- It is possible to create and cache embedding vectors with the embedding-based approach and use them in their vector form for similarity

computations. When caching all embedding vectors of a process model repository in an offline phase, time is saved during the online phase of an application, e. g., for process model similarity assessment. Caching at this level is not possible for the graph-matching-based approach.

Related work

Another advantage of the embedding-based approach is a result of its self-learning capabilities, and directly motivates this paper: Larger amounts of training data are expected to improve the accuracy of the predictions. Therefore, this paper investigates data augmentation methods (Mumuni and Mumuni 2022; van Dyk, David, and Meng 2001; Zhao et al. 2022; Zhou et al. 2023) to enlarge the available amount of training data and, ultimately, to increase the robustness and quality of the DL models trained on it.

Mumuni and Mumuni (2022) distinguish between methods based on *transformation*, where an original training example is transformed into an augmented one, and *synthesis*, where new training examples are generated according to the characteristics of the original training set. Both of these categories are covered for the domain of sequential NEST process models in the proposed approach. The remainder of this section discusses specific augmentation methods for graph data (see Section 3.1) and for process data within the research field of BPM (see Section 3.2). While specific augmentation methods for processes are more relevant to this contribution, generic graph augmentation methods are universally applicable and useful as a baseline.

Graph augmentation

There are several recent surveys on generic graph augmentation that summarize and categorize existing methods:

Ding et al. (2022) present different methods for graph augmentation based on a taxonomy of two main use cases: 1) reliable graph learning for enhancing model quality and training data utility, and 2) low-resource graph learning for enlarging the labeled training data. For all augmentation methods, node-level, edge-level, and graph-level tasks are considered. The integration of augmentation in an existing training pipeline is also discussed, with one option being a separate augmentation phase before starting the training, and another option being augmentation as part of model training (with trainable augmenters).

Zhao et al. (2022) structure the presented graph augmentation methods according to three perspectives, i. e., operated data (structure, features, labels), downstream task (node-level, edge-level, graph-level), and whether the augmentations are rule-based or learnable. The methods within these categories are further separated into groups, e. g., methods based on feature removal, addition, and manipulation.

The discussion of augmentation methods by Adjeisah et al. (2023) is less focused on covering as many different methods as possible but rather on experiments with a selection of popular methods. Therefore, the authors examine eight state-of-the-art methods that augment on either the topology-level or the feature-level and do a mathematical analysis. The methods are also applied to three different widely used GNN architectures to test their effectiveness.

Zhou et al. (2023) present four taxonomies of graph augmentation methods with a focus for the graph that is augmented, the addressed target task, whether the augmenter is learnable, and the augmentation mechanism. In addition, suitable evaluation metrics for augmentation methods and possible applications are thoroughly discussed.

Marrium and Mahmood (2022) categorize augmentation methods based on the addressed task, the augmentation technique, and the learning objective. They also provide benchmarks of several methods and a collection of their implementations. This work has a strict focus on the downstream task that augmentation is used in, e. g., node classification, graph classification, and link prediction.

Yu et al. (2022) specifically analyze learnable augmentation methods, framed by the term Graph Augmentation Learning. The strategies are structured as node-, edge-, subgraph-, and graph-level and application scenarios w. r. t. data-specific, model-specific, and hybrid are described. The authors also provide an experimental guideline for choosing a suitable augmentation strategy regarding the application context.

A key takeaway of these surveys is the magnitude of different generic graph augmentation methods available in the literature. Two important factors for distinguishing these methods in our context are the downstream task and the data that is operated on. The downstream task is similarity learning, which is a graph-level task and, thus, makes methods suitable that take a graph as input and a graph as output. The data that is operated on can be any part of the graph, e. g., features and topology, and these aspects should be considered.

Augmentation methods from business process management literature literature

The presented surveys do not discuss specific augmentation methods for processes or semantic graphs. These augmentation methods come from the research field of BPM and are closely related to this work, but only a few approaches can be found. The approaches can, for example, be used for predictive process monitoring with DL methods (for an overview, see Rama-Maneiro, Vidal, and Lama 2023). Although the augmentation methods in this work operate on process models, augmentation of event logs and process instances is also discussed, as literature about augmentation in BPM is scarce.

de Leoni, van der Aalst, and Dees (2016) enrich event logs by adding data to them. Although, their approach is not named augmentation, and it is not set in a deep learning context, the methods could probably be used for this purpose.

Venkateswaran et al. (2021) examine data augmentation methods for facilitating robust deep learning applications in predictive process monitoring. However, concrete augmentation methods are not part of the proposed approach, as simple generic methods are used only in the experiment to test the robustness to changes.

Käppel, Schönig, and Jablonski (2021) present an application of small sample learning (Shu, Xu, and Meng 2018) in BPM applications for dealing with insufficient amounts of event log training data. While this is not directly an augmentation method, the described concept and integration are still very close.

Käppel and Jablonski (2023) present augmentation methods for event logs for the task of predictive process monitoring (Rama-Maneiro, Vidal, and Lama 2023). They present a total of nine augmentation methods, mostly generic methods adapted to event log data. All methods are explained in detail with the intentions behind them. The used augmentation pipeline also discusses the integration of the data augmentation methods into the existing learning procedure.

The presented approaches show that augmentation methods specifically for process models are largely unexplored. The work of Käppel and Jablonski (2023) is the one closest to our approach, but still has a different application context, i. e., process event logs. This makes it challenging to reuse due to the differences between event logs and process models as input data to the augmentation methods. Please note that there is also a branch of work in BPM research that is concerned with the generation of (synthetic) event logs. Grüger et al. (2022) discuss and summarize this branch in more detail and find that most approaches are insufficient to consider semantic data apart from the control-flow.

Synthesis and augmentation of process models

This section discusses three categories of augmentation approaches and their ability to create useful augmented process models. To develop these augmentation approaches, a concrete research methodology has been followed whose individual steps are illustrated in Figure 3.

First, key criteria are derived to guide the development process (see Section 4.1). These key criteria, especially the importance of semantic information for the use case of this paper, reveal another aspect to consider for the definition of augmentation methods: the so-called correctness as the second step of the applied research methodology (see Section 4.2). The notion of correctness is based on the observation that arbitrary augmentation of an

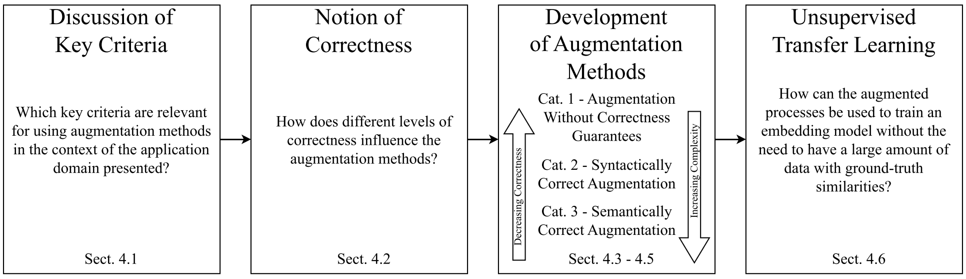
Discussion of Key Criteria
Which key criteria are relevant for using augmentation methods in the context of the application domain presented?
Sect. 4.1
Notion of Correctness
How does different levels of correctness influence the augmentation methods?
Sect. 4.2
Development of Augmentation Methods
Cat. 1 - Augmentation Without Correctness Guarantees
Cat. 2 - Syntactically Correct Augmentation
Cat. 3 - Semantically Correct Augmentation
Decreasing Correctness
Increasing Complexity
Sect. 4.3 - 4.5
Unsupervised Transfer Learning
How can the augmented processes be used to train an embedding model without the need to have a large amount of data with ground-truth similarities?
Sect. 4.6

Figure 3. Overview of applied research methodology.

input process model could result in an output process model that is either syntactically or semantically incorrect, and, thus, violating rules of the process model structure and the underlying domain model, respectively. Given the definition and characteristics of correctness, in the third step of the methodology, three categories of augmentation approaches are developed. These augmentation approaches differ in the level of correctness they can guarantee and are presented in Section 4.3, 4.4, and 4.5. Possible augmentation methods as well as properties of the resulting augmented process models are discussed for each category. As the augmented process models are to be used for training DL models, in the fourth step of the applied methodology, an approach for semi-supervised transfer learning is developed. This approach enables the learning of an embedding model without the need to have a large amount of data with ground-truth similarities. The learning approach is presented in Section 4.6. The idea here is to pair an unsupervised training procedure that uses only augmented data with a supervised training procedure that uses only original, non-augmented data, with no need for potentially costly label acquisition for the augmented data.

Key criteria for the proposed augmentation methods

The discussion of the related work (see Section 3) and the introduction of the process model representation and the domain (see Section 2.1 and 2.2) reveal a discrepancy between existing approaches and the needs of the process models considered in this work. Therefore, several key criteria for the usage of augmentation in the current application context are presented. Please note that these criteria should not be interpreted as requirements but more as guidance for the augmentation of sequential NEST process models.

Focus on semantic information

Semantic annotations are one of the most important parts of sequential NEST process models, which should also be reflected in augmentation methods that operate on the semantic annotations. Thereby, the semantic information is given by the process model, or more precisely its nodes and edges, and the underlying domain model. The semantic annotations can be very complex and heavily customized for the domain, making it difficult for generic augmentation methods to properly handle them. Semantic annotations are also an important part of the similarity computation, i. e., the main learning goal of our use case. Therefore, it should be noted that changes in the semantic annotations by augmentation might have a large impact on the use of the augmented process model as a training example.

Focus on control-flow and data-flow

The process model representation of the manufacturing processes to augment is sequential in its nature and by its definition (see Section 2.1), with the control-flow as one part and the data-flow as the other part (Malburg, Brand, and Bergmann 2023; Malburg, Hoffmann, and Bergmann 2023). This is an important property to consider for augmentation methods, and it is also one property that is not considered by generic graph augmentation methods (see Section 3.1). Although not dealing with process models, Käppel and Jablonski (2023) are also confronted with the sequential nature of process event logs but, generally, only consider control-flow and face an overall lower complexity of the used data with semantics playing a small role. It is important to note that the manufacturing processes in this paper (see Figure 1) are an example of processes that rely mainly on their data-flow. The control-flow is, in these cases, indirectly determined by the data nodes connected via data-flow edges to the corresponding task nodes. There are other types of processes in which the control-flow perspective defined by the task nodes is the main factor, i. e., business processes (Müller 2018).

Small changes may have large effects

Augmentation methods are generally designed to transform some kind of input data into a similar kind of output data, e. g., an image is rotated but still expresses the same content. This property is also desirable for the augmentation methods of the process models used in this work, and the extent of the change should be configurable for the methods. This offers the advantage that the methods can be fine-tuned to each domain or the specific training set, ultimately increasing the applicability.

Notion of correctness

The definitions of the *sequential NEST process model* and the corresponding *domain model* (see Section 2.1) lead to a challenge for augmentation methods to include this syntactical and semantic knowledge into their algorithms. Because the basic assumption of all data augmentation approaches (van Dyk, David, and Meng 2001) is also applicable here: The augmented data should be as close to the original data as possible to ensure a high quality of the augmented data as DL training data. This closeness, however, is hard to measure and strongly depends on the domain. One way could be the computation of the similarity (see Section 2.3) between the original process model and the augmented process model, where a high similarity value is desired. However, the associated computation procedure is infeasible for large-scale augmentation due to the involved computational complexity, and it requires augmented process models that are valid in terms of all constraints by the definition of sequential NEST process models and the corresponding domain model. Especially the fulfillment of the latter requirement can make augmentation methods very complex and vastly increase the knowledge acquisition effort for their implementation. For this reason, we introduce the notion of *correctness*, on the one hand, as an approximation of the quality of an augmented process model and, on the other hand, as a mechanism to allow augmentation methods to steer the trade-off between useful augmentation methods and their complexity and required knowledge. The correctness could also be considered a key criterion in addition to those described in Section 4.1, but it is described separately, as it serves the important purpose of structuring the different augmentation approaches. The correctness of an augmented process model, in the sense that we use it for the remainder of the paper, is further divided into *syntactic correctness* and *semantic correctness*:

> A *syntactically correct process model* is a valid sequential NEST process model (as defined in Section 2.1) and, thus, follows the representation of a NEST graph as well as features a sequential data-flow and control-flow without missing semantic descriptions.

Syntactic correctness addresses the structure of a process model w. r. t. to its nodes and the edges between them. A violation of the syntactic correctness is equal to a violation of the constraints given by the definition of the NEST graph (Bergmann and Gil 2014) and the additional rules of sequential NEST process models. For instance, a simple violation of syntactic correctness would be the removal of a control-flow edge between two tasks, which, in turn, breaks the control-flow within the process model. With the property of being syntactically correct, only the structural conformity of a process model is guaranteed, without any validity checks of the semantic annotations regarding the domain model. Because semantic information is a key aspect of sequential NEST process models in general and manufacturing process models in particular, semantic correctness is defined as follows:

> A *semantically correct process model* is syntactically correct and consistent with the definitions and constraints of the PDDL domain model, thus executable in the manufacturing context. (see Section 2.2)

Semantic correctness completes the aspects of syntactic correctness by incorporating crucial information from the domain model. For example, a manufacturing process model is syntactically but not semantically correct if it is a valid sequential NEST process model, but the workpiece is processed by machines in the wrong order. The underlying factor for determining semantic correctness is the domain model, as it holds the information that describes the proper content and relations of all semantic information of a process model.

These two definitions lead to three levels of correctness that can be determined for an augmented process model: 1) A process model can be neither syntactically nor semantically correct, 2) a process model can be syntactically but not semantically correct, and 3) a process model can be semantically correct, and, thus, the process model is also syntactically correct. The following sections discuss augmentation approaches that are divided into three categories according to the level of correctness that they can guarantee for the resulting augmented process models. Figure 4 depicts these three categories.

We state that augmentation methods that preserve a higher level of correctness of the augmented process models are generally more complex and require more knowledge, i. e., are rather more knowledge-intensive. In turn, augmented process models with a higher level of correctness are typically more representative of their native process domain, and, thus, we assume them to be more useful for learning the properties of these processes. As a more accurate DL model also leads to better results when being applied as a similarity measure, the level of correctness is an important indication of the trade-off between the quality of the augmented process models and the complexity and knowledge requirements of the respective augmentation techniques.

Augmentation without correctness guarantees (Cat. 1)

Augmentation methods from this category are the least complex, but hopefully still suitable for creating a large amount of augmented process

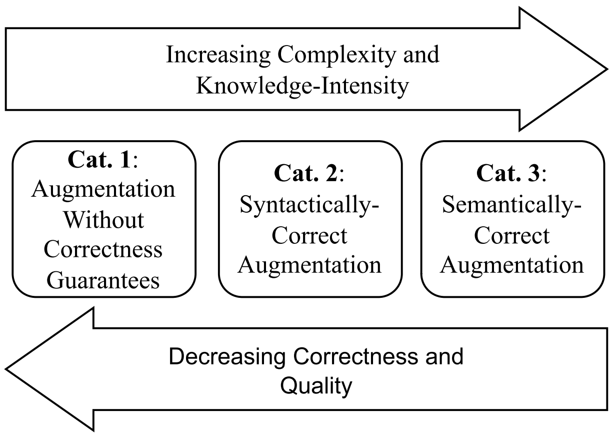
Increasing Complexity and Knowledge-Intensity
Cat. 1: Augmentation Without Correctness Guarantees
Cat. 2: Syntactically-Correct Augmentation
Cat. 3: Semantically-Correct Augmentation
Decreasing Correctness and Quality

Figure 4. Overview of the categories of augmentation methods.

models, representing the basic properties of the process domain. We propose to focus on augmentation based on *deletion* for this category. There are several reasons for this: These methods fit well to the main characteristics of this category, regarding complexity and ease-of-use. Another important factor refers to the knowledge required to implement methods based on deletion. Usually, these methods only require the original process model for the implementation, and no further information, such as data from other process models from a repository, in-depth knowledge about the process model representation or an in-depth understanding of the domain model. In the remainder of the section, we discuss the deletion of nodes, edges, and (parts of) semantic descriptions in more depth and examine the application of these methods for an example.

Randomly deleting nodes from the graph structure representing the process model is a well-known method from generic graph augmentation (see Section 3.1). The implementation is straightforward, but it might result in augmented process models that are not syntactically correct, mainly due to a broken sequential data-flow or control-flow (see Section 4.2). Similar to node deletion, process models can also be augmented by deleting edges. This method is also well-known from existing literature (see Section 3.1) and it is easy to understand and to implement. It is worth discussing nodes and edges separately, as their augmentations also have a different effect on the training procedure of the GEM (see Section 2.4). After the initial embedding of the nodes and edges in the embedder, the node information is propagated along the edge connections. This means that deleting a node removes the information from the propagation procedure and, ultimately, from the graph embedding. Deleting an edge changes the embedding by restricting the flow of information in the propagation layer of the GEM, which can have a significant influence on the resulting embedding.

Apart from the structural components, i. e., nodes and edges, the semantic descriptions are another integral part of the process model representation, which can also be addressed during augmentation. Thereby, the entire semantic description (or parts) of the semantic description of certain nodes or edges can be deleted. The resulting augmented process models are not syntactically correct, due to missing semantic descriptions (see Section 4.2). Although a node or an edge is mainly characterized by its semantic description in a sequential NEST process model, there are differences between deleting only the semantic description or the entire node or edge: On the one hand, the information on the type of node or edge is preserved and still part of the graph embedding procedure, which influences the resulting embedding. On the other hand, there is a node without semantic information, which will lead to a different behavior of the GNN during embedding and propagation. Therefore, it is useful to discuss both aspects separately.

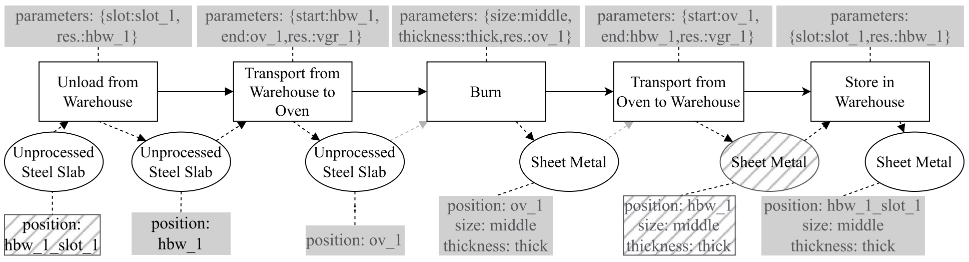
parameters: {slot:slot_1, res.:hbw_1}
parameters: {start:hbw_1, end:ov_1,res.:vgr_1}
parameters: {size:middle, thickness:thick,res.:ov_1}
parameters: {start:ov_1, end:hbw_1,res.:vgr_1}
parameters: {slot:slot_1,res.:hbw_1}
Unload from Warehouse
Transport from Warehouse to Oven
Burn
Transport from Oven to Warehouse
Store in Warehouse
Unprocessed Steel Slab
Unprocessed Steel Slab
Unprocessed Steel Slab
Sheet Metal
Sheet Metal
Sheet Metal
position: hbw_1_slot_1
position: hbw_1
position: ov_1
position: ov_1 size: middle thickness: thick
position: hbw_1 size: middle thickness: thick
position: hbw_1_slot_1 size: middle thickness: thick

Figure 5. Exemplary syntactically incorrect process (based on Figure 1).

An example of the application of three augmentation methods based on deletion is shown in Figure 5. Thereby, the following three augmentations are applied successively to the exemplary process model shown in Figure 1:

(1) Random deletion of one data node
(2) Random deletion of two data-flow edges
(3) Random deletion of the semantic description of one data node

The changes made to the original process model are highlighted with hatched nodes and semantic descriptions, respectively, and edges in bright gray. The augmented process model no longer follows the sequential NEST process model definition (see Section 2.1) due to the broken sequential data-flow in three parts of the process model and a missing semantic description of the first unprocessed steel slab. It is also important to note that the state of the work-piece in the process model is lost due to the deleted data node and the semantic descriptions.

Syntactically correct augmentation (Cat. 2)

For this category of augmentation, we propose to focus on augmentation approaches that involve some kind of *replacement*. The main reason for this is the ability to reuse existing syntactically and semantically correct parts of the original process models to create augmented process models. Thus, it is possible to replace parts of a process model, e. g., a task node, with an equivalent part of the same or a different process model. Syntactic correctness is automatically ensured, since the structure of the augmented process model does not change compared to the respective original. The augmented process model differs from the original process model only on a semantic level. Existing generic graph augmentation methods from the literature can usually only hardly be reused in this regard (see Section 3.1) since they cannot specifically handle process models and, in particular, sequential NEST process models. The remainder of the section discusses replacements of task and data nodes, control-flow and data-flow edges, and (parts of) semantic descriptions in more depth, and examines the application of these methods for an example. We also discuss *swapping* as a special form of replacing, where the part to replace, e. g., a task node, and the replacement come from the same process model.

The control-flow within a process model is given by the control-flow edges between the task nodes in the process model. It determines the order in which tasks are carried out on the data elements, e. g., a metal sheet is drilled. Augmentation can be aimed at the control-flow by replacing task nodes of the original process model with other task nodes, including their semantic description. Therefore, the replaced node receives all ingoing and outgoing edges of the former node after replacement (see tasks *Burn* and *Transport from Warehouse to Oven* in Figure 6). Analogous to the control-flow, the data-flow within a process model, i. e., the data nodes that are consumed or produced along the control-flow, can also be targeted by augmentation similarly. In the manufacturing use case, this manipulates the input goods and the resulting goods of each manufacturing task in the process model. Such an augmentation replaces data nodes along the data-flow with other data nodes and their semantic descriptions, coming from the same or a different process model. As with the replaced task nodes, the connected edges are disconnected before replacing and then connected to the new nodes after the replacement.

Edges can also be replaced as a form of augmentation. The two most important types of edges to be augmented are control-flow edges and data-flow edges, due to the important NEST process model characteristics of sequential control-flow and data-flow (see Section 2.1).

The semantic descriptions of nodes and edges are also useful for replacements. Replacing the entire semantic description of a node or an edge (given the replacement node or edge has the same type) is equivalent to replacing the node or edge itself, as it is characterized and represented by its semantic description. Parts of semantic descriptions can also be replaced with values from other semantic descriptions of nodes or edges of the same type. Since the replaced parts of the semantic descriptions are from the same or other original process models and, therefore, semantically correct, the resulting process models might not be semantically correct due to the new context that the replaced node is now integrated in.

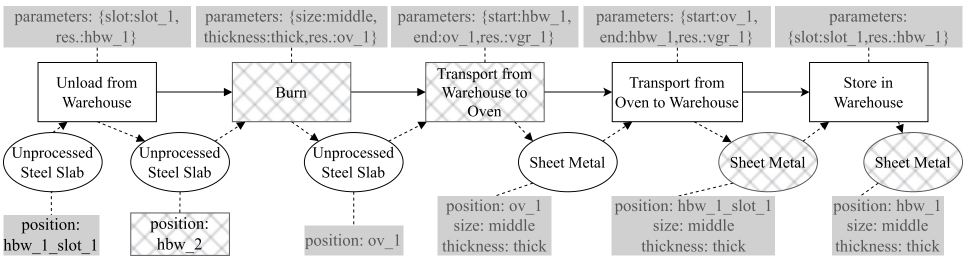
parameters: {slot:slot_1, res.:hbw_1}
parameters: {size:middle, thickness:thick,res.:ov_1}
parameters: {start:hbw_1, end:ov_1,res.:vgr_1}
parameters: {start:ov_1, end:hbw_1,res.:vgr_1}
parameters: {slot:slot_1,res.:hbw_1}
Unload from Warehouse
Burn
Transport from Warehouse to Oven
Transport from Oven to Warehouse
Store in Warehouse
Unprocessed Steel Slab
Unprocessed Steel Slab
Unprocessed Steel Slab
Sheet Metal
Sheet Metal
Sheet Metal
position: hbw_1_slot_1
position: hbw_2
position: ov_1
position: ov_1 size: middle thickness: thick
position: hbw_1_slot_1 size: middle thickness: thick
position: hbw_1 size: middle thickness: thick

Figure 6. Exemplary semantically incorrect process model (based on Figure 1).

The influence of these augmentations on the training of the GEM is different from the deletions previously discussed (see Section 4.3). When comparing the original process model with one augmented by replacing nodes, edges, or parts of semantic descriptions, the structure of the process model remains unchanged, while the embedding procedure nonetheless changes. New information might be part of the process model if data from different process models of the repository is used for the replacement. If data from the same process model is used (i. e., swapping), then the augmented process model is handled differently by the GNN in the propagation and aggregation layers compared to the original one. This consideration between swapping and replacements from different process models might influence the quality of the resulting process models. On the one hand, swapping usually creates augmented process models that are relatively close to the original, since information within the process model is only moved along the control-flow or data-flow. Replacements with information from different process models, on the other hand, might increase the diversity of the training data with new, unseen process models.

An example of the application of three replacement-based augmentation methods is shown in Figure 6. Thereby, the following three augmentations are applied successively to the exemplary process model shown in Figure 1:

(1) One random swap of two neighboring task nodes
(2) One random swap of two neighboring data nodes
(3) One random replacement of the item *position* in the semantic description of one data node

The changes made to the original process model are highlighted with cross-hatched nodes and semantic descriptions. The resulting augmented process model is still a valid sequential NEST process model (see Section 2.1) but it is not semantically correct. For instance, it is not valid to burn the workpiece before it was brought to the oven, as shown by the augmented process model.

Semantically correct augmentation (Cat. 3)

To create semantically correct augmented process models, it is important to do augmentation that fulfills the constraints given by the sequential NEST process model representation (see Section 2.1) and the domain model (see Section 2.2). For example, assume that a manufacturing process has a wrong order of machines or wrong parameter settings for these machines. Training a DL model with these processes could lead to incorrect generalizations and, ultimately, poor model performance. To guarantee semantically correct augmented process models, automated planning techniques in combination with a planning domain description (see Section 2.2) are suitable. Automated planning (Ghallab, Nau, and Traverso 2016; Haslum et al. 2019) is a technique from artificial intelligence and, thus, is often also called AI planning. AI planning can be used as a generative problem solver and is based on the concept of representing a planning problem as an initial state, a goal state that should be reached, and a set of actions that are used for state transitions. The goal of solving the planning problem is to find a sequence of actions so that the goal state is reached based on the facts given in the initial state. In this context, a sequence of actions is rather similar to a sequence of tasks in a process. For this reason, AI planning has also gained considerable interest in the BPM area in recent research works (Malburg, Brand, and Bergmann 2023; Malburg, Hoffmann, and Bergmann 2023; Malburg, Klein, and Bergmann 2023; Marrella, Mecella, and Sardiña 2017, 2018; Masellis et al. 2022; Rodríguez-Moreno et al. 2007). In Marrella (2019), it has been examined in which phases of the BPM lifecycle, AI planning can be utilized. One use case for AI planning in BPM is the generation of process models in the *Design* phase. In this work, we generate new process models and transform existing process models to increase the set of semantically correct process models in the context of data augmentation.

Listing 2 Exemplary Planning Problem for Generating a Manufacturing Workflow.

```
(define (problem factory-problem) (:domain factory)
(:objects workflow_1 - processID)
(:init
    ...

    (isReady vgr_1)    (isReady vgr_2)
    (isReady hbw_1)    (isReady hbw_2)
    (isReady ov_1)     (isReady ov_2)
    (isReady mm_1)     (isReady mm_2)
    (isReady wt_1)     (isReady wt_2)
    (isReady sm_1)     (isReady sm_2)
    (isReady pm_1)     (isReady dm_2)
    (isReady hw_1)

    ; workflow_1 initial state
    (bucketAt hbw_1_slot_1_pos)
    (isSteelSlab workflow_1)
    (at workflow_1 hbw_1_slot_1_pos)

    (=(total-cost) 0)

  )
  (:goal (and (at workflow_1 hbw_1_slot_1_pos)
  (isSheetMetal workflow_1)
  (isSheetMetalWithSize middle workflow_1)
  (isSheetMetalWithThickness thick workflow_1))

  (:metric minimize (total-cost))
)
```

To augment process models, a comprehensive planning domain model consisting of the available planning actions and a planning problem comprised of the initial state and the goal state that should be reached is required. For the manufacturing application scenario used in this work, we use a semantic service-based architecture for the smart factory (Malburg, Klein, and Bergmann 2020; Seiger et al. 2022). Based on this architecture, a domain expert creates a comprehensive planning domain description based on the PDDL (McDermott et al. 1998) for the application scenario, as introduced in Section 2.2. With the planning domain, process models can be either augmented by *synthesis*, i.e., generated from scratch solely based on the planning domain and a randomly generated planning problem, or *transformation*, i.e., based on an original process model see Mumuni and Mumuni (2022).

In addition to the planning domain, a corresponding planning problem is required to generate process models for data augmentation. These planning problems can be created automatically and randomly in the context of the manufacturing application scenario. Given the problems, it is possible to solve them by a corresponding planner. Listing 2 represents a part of the planning problem to generate the workflow illustrated in Figure 1. The random planning problem generator assumes that all machines are available in the smart factory (Lines 6–13) and specifies an object representing the workflow to be planned (Line 2). Subsequently, the initial state of the planning problem is specified by randomly choosing possible initial states (Lines 16–18). Furthermore, the total cost function is initialized (Line 20). To define the goal of the workflow, the goal state of the planning problem needs to be specified. For this purpose, the random planning problem generator determines for each possible property of the workpiece whether it should be included, e. g., drilled or not drilled, and, if so, in which concrete form, e.g., number and size of the holes. In this context, it is important to know that all properties, desired or not, are part of the planning problem. However, please note that Listing 4.5 only contains the desired ones due to space restrictions (Lines 23–26).[4]

Transformational augmentation is possible with AI planning similarly. Instead of directly manipulating the process models (as discussed in Section 4.3 and 4.4), the native planning problems or the configuration of the planner are augmented. In the first case, it is possible to create the initial state and the goal state of the origin process model and then augment the *initial state*, e. g., a change of the position of the workpiece, the *goal state*, e. g., a change in the thickness of the workpiece, or the *final metric*, e. g., a minimization of the number of tasks instead of the total cost. These changes are then processed by the planner and an augmented process model is created. A different way of creating augmented process models with AI planning is an augmentation of the planner configuration.

By changing the heuristic or the configuration of the heuristic that the planner uses to solve the planning problem, it is also possible to create different resulting process models. The augmented process model and the original one are most likely similar for both variants, since they use a similar planning problem and planner configuration. In addition, the proposed approach can also be used for planning parts of process models, e. g., gaps in the control-flow between two activities or alternative routes for certain parts. This offers the advantage that the resulting process models are even more similar as only parts change, and they could be more effective as training examples.

Unsupervised transfer learning with augmented processes

Augmented process models pose a challenge in handling them in the training process of a supervised embedding model such as the GEM (introduced in Section 2.4) due to the required ground-truth similarities. On the one hand, it is unclear if the underlying ground-truth similarity measure can deal with syntactically or semantically incorrect process models. On the other hand, the procedure for computing the ground-truth similarities is usually computationally expensive and, when using measures based on A*, often entirely infeasible for large amounts of data (see Section 2.3 and Schuler et al. (2023). Therefore, we propose to use augmentation in combination with a semi-supervised two-phase training approach, featuring an unsupervised pretraining phase and a supervised graph similarity learning phase (Schuler et al. 2023). This provides the advantage that the augmented process models are used in an unsupervised training procedure without the need for label acquisition, i. e., the computation of ground-truth similarities. The original, non-augmented process models are used in the supervised training procedure, where the model learns to predict the ground-truth similarities (see Section 2.4). Due to this separation, the augmented process models are still used to train the graph embedding model, but label acquisition is greatly reduced, compared to a fully supervised training setup.

The unsupervised and supervised training phases are unified by transfer learning (see Figure 7 for an overview of the components and the main steps). Transfer learning acts as a bridge between the two phases, allowing the knowledge gained during the unsupervised pretraining phase to be transferred to the supervised adaptation phase. This approach leverages the commonalities between the source and target domains, enabling the model to generalize better in tasks where labeled data is scarce. The fundamental concept behind transfer learning is that the knowledge acquired during the pretraining phase from a source domain (see step 2) can be applied in the adaptation phase within a target domain

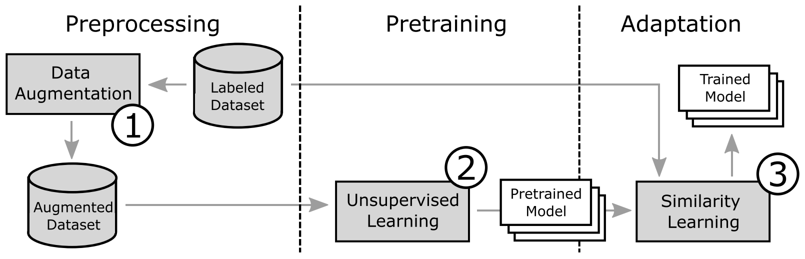

Figure 7. Architecture for semi-supervised transfer learning with its three main steps (Source: Schuler et al. 2023).

(see step 3), as described in Kudenko (2014) and Tan et al. (2018). During the pretraining phase, the model learns a robust representation of the source domain through an unsupervised learning process, where it identifies patterns, structures, and relationships in the input data. In our specific context, a graph embedding model serves as this knowledge repository. The graph embedding captures the structural and contextual features of the data during the unsupervised phase, enabling a seamless transition to the supervised adaptation phase.

The underlying assumption here is that the augmented training data (see step 1) is sufficient for the unsupervised pretraining to learn the characteristics of the process domain and, thus, to diminish the necessity for an extensive amount of labeled data during the subsequent supervised training. By leveraging this assumption, the model can capitalize on the extensive but unlabeled data available in the pretraining phase to build a robust foundational understanding of the domain. This not only accelerates the learning process in the supervised phase, but also enhances performance by minimizing overfitting, which is a common challenge when dealing with limited labeled data. The existing literature (e. g., Weiss, Khoshgoftaar, and Wang 2016) supports this strategy, highlighting its effectiveness in various domains such as natural language processing, computer vision, and graph-based learning tasks.

Unsupervised triplet learning

During the pretraining phase, an SNN with a triplet loss (Hermans, Beyer, and Leibe 2017; Ott et al. 2022; Schroff, Kalenichenko, and Philbin 2015) is used to learn unsupervised graph embeddings (see Figure 8). The specific graph embedding model is the GEM (introduced in Section 2.4). The concept revolves around creating embeddings that are alike for graphs with similarities, and distinct for graphs that differ significantly (in terms of vector space similarity). Given the absence of computed ground-truth similarities for the graphs within the training dataset, the loss function operates on graph triplets comprising an *anchor*, a *negative*,

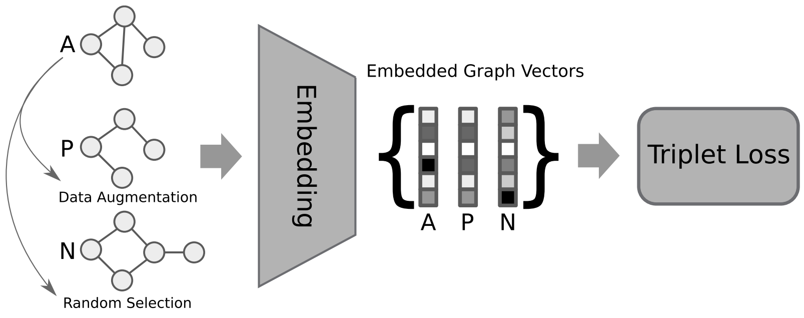

Figure 8. Unsupervised triplet graph embedding using an anchor (A), a positive (P), and a negative (N) (Source: Schuler et al. 2023).

and a *positive*. Thereby, the similarity for the pair of the anchor and the positive is maximized, and the similarity for the pair of the anchor and the negative is minimized.

The formal definition of this triplet loss is given in Equation 1: Let $T = (x^a, x^p, x^n)$ be a triplet consisting of one input vector for anchor x^a, positive x^p, and negative x^n, and let $f(x_i) = m_i \in \mathbb{R}^{\ltimes}$ represent the embedding of an input vector x_i as the n-dimensional embedding m_i. The loss function L is minimized with α as a margin hyperparameter and N as the cardinality of a batch of triplets to train a model Schroff, Kalenichenko, and Philbin (2015).

$$L = \sum_{j}^{N} \max\{0, \| m_j^a - m_j^p \|^2 - \| m_j^a - m_j^n \|^2 + \alpha\} \tag{1}$$

The main challenge for this training method in an unsupervised context is the composition of the triplets, as the aforementioned assumptions regarding the relationships of anchor and positive and anchor and negative should be satisfied. With a given anchor process model, the negative is selected randomly from the training set, which is straightforward. The positive is selected as an augmentation of the anchor process model, since the augmentation methods generally produce augmented process models that are similar to the original ones. Please note the special case when composing triplets of augmented process models generated by AI planning via synthesis (see Section 4.5). Since there is no original process model for each augmented one, it is possible to use the augmented process model as the anchor and the positive.

Graph embedding with the GEM

The GEM, as described in Section 2.4, serves a dual role in both the *pretraining* and *adaptation* phases. In the adaptation phase, its function aligns with its original purpose (see previous work in Hoffmann and Bergmann 2022b), involving the supervised embedding of graph pairs.

Nevertheless, the pretraining phase, which employs the triplet learning method, necessitates specific adjustments to the model. Contrary to its original use with pairs of graphs, the GEM is trained with graph triplets. Each graph in the triplet is embedded using shared parameters. As the GEM supports processing tuples of graphs independent of their number out of the box, this change is more focused on the implementation. These refinements to the GEM enable the successful execution of unsupervised pretraining and the subsequent supervised adaptation of the model within the transfer learning framework.

Experimental evaluation

The experimental evaluation aims to examine the effects of process model augmentation on similarity learning with the GEM (see an overview in Figure 9). Therefore, we compare the GEM trained with different augmented training sets and trained with non-augmented data to a baseline similarity assessment method based on A* search (see Section 2.3). The comparison between the three approaches is made according to several quality metrics, measuring the similarity prediction errors. The augmented training sets are created with different combinations and configurations of the three categories of augmentation, introduced in Section 4.

Hypotheses

We investigate the following hypotheses:

H1 Training the GEM on augmented process models increases the quality compared to the GEM being trained on non-augmented process models.

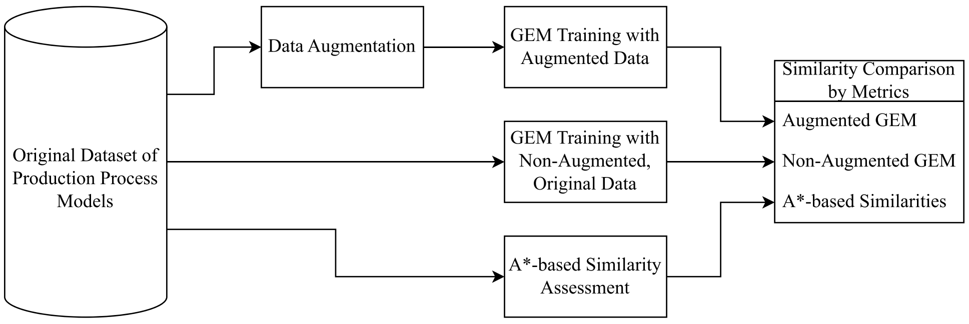
Original Dataset of Production Process Models
Data Augmentation
GEM Training with Augmented Data
GEM Training with Non-Augmented, Original Data
A*-based Similarity Assessment
Similarity Comparison by Metrics
Augmented GEM
Non-Augmented GEM
A*-based Similarities

Figure 9. Overview of the experiments.

H2 Training the GEM on augmented process models of Cat. 2 (see Section 4.4) increases quality compared to the GEM being trained on augmented process models of Cat. 1 (see Section 4.3).

H3 Training the GEM on augmented process models of Cat. 3 (see Section 4.5) increases the quality compared to the GEM being trained on augmented process models of Cat. 1 or 2 (see Section 4.3 and 4.4).

H4 Training the GEM on process models augmented with methods from several categories increases quality compared to the GEM being trained on process models, augmented with the respective methods from the individual categories.

The hypotheses show the expectations of the authors regarding the experimental results. As a baseline, it is expected that using augmented data for training leads to a better model than the original (non-augmented) data. We additionally claim that the level of correctness influences the quality of the model, i. e., higher levels of correctness lead to better models. The final hypothesis addresses the combination of augmentation methods with different levels of correctness that are expected to show an increased quality compared to the respective individual methods.

Datasets

The main dataset (denoted as D_{100}) used throughout the experiments comprises workflows originating from a smart manufacturing IoT environment, representing sample production process models (see Section 2.1 or previous work in Malburg, Hoffmann and Bergmann (2023) for more details). The dataset contains 100 training cases, 20 validation cases, and 20 test cases. To measure the effects of augmentation on differently sized datasets, we also use D_{50} with 50 training, 10 validation, and 10 test cases, D_{200} with 200 training, 30 validation, and 30 test cases, and D_{300} with 300 training, 35 validation, and 35 test cases. For the cases contained in these datasets, $D_{50} \subset D_{100} \subset D_{200} \subset D_{300}$ holds.

Table 1 shows some statistics about the numbers of nodes and edges in these datasets. The foundation for all datasets is the planning approach by synthesis introduced in Section 4.5. It is used to reduce the manual effort of creating a data set of process models. The resulting process models resemble real processes as the underlying planning domain is modeled on the basis of a real production environment. In other scenarios, a manually modeled dataset of process models would probably exist and could be used instead.

Table 1. Statistics on amounts of nodes and edges of the original datasets (see also Table A1).

	All			Nodes task			Data			All			Edges Control-flow			Dataflow		
Domain	Avg	Min.	Max.	Avg	Min.	Max.	Avg	Min.	Max.	Avg	Min.	Max.	Avg	Min.	Max.	Avg	Min.	Max.
50	18.0	39.0	11.6	8.0	17.0	12.2	9.0	18.0	59.3	59.3	40.0	94.0	11.9	7.0	24.0	22.7	16.0	34.0
100	18.0	39.0	11.8	8.0	18.0	12.4	9.0	19.0	58.6	60.6	40.0	94.0	12.4	7.0	24.0	23.0	16.0	36.0
200	15.0	43.0	11.8	7.0	18.0	12.3	7.0	19.0	60.0	60.1	32.0	104.0	12.3	6.0	26.0	22.9	12.0	36.0
300	15.0	43.0	11.8	7.0	18.0	12.4	7.0	19.0	59.9	60.3	32.0	104.0	12.2	6.0	26.0	23.0	12.0	36.0

Table 2. Augmentation strategies in the experiments.

Strategy	Category	Methods	Factor	Sizes (D_{100}, D_{50}, D_{200}, D_{300})
1	1	• Deletion of two control-flow edges • Deletion of one data node • Deletion of one item of the semantic description of two task nodes	4	400, 200, 800, 1200
2	2	• Swapping of two pairs of neighboring data nodes (in terms of data- flow) • Swapping of two pairs of neighboring task nodes (in terms of control-flow) • Swap of one item of the semantic descriptions of three task nodes	4	400, 200, 800, 1200
3.1 & 3.2	3	• Synthesis augmentation of process models via AI planning (3.1) • Transformation augmentation of process models via AI planning (3.2)	4	400, 200, 800, 1200
4	1 and 2	Augmentation according to Strat. 1 and 2	7	700, 350, 1400, 2100
5.1 & 5.2	1, 2, and 3	Augmentation according to Strat. 1, 2, 3.1, and 3.2	10	1000, 500, 2000, 3000

Augmentation strategies

To examine the effects of augmentation on the quality of the GEM, the aforementioned datasets are augmented. The resulting datasets are presented in Table 2.

Each dataset is augmented with different strategies: 1) methods without correctness guarantees (as introduced in Section 4.3), 2) syntactically correct methods (as introduced in Section 4.4), 3) semantically correct methods (as introduced in Section 4.5), 4) the combination of the strategies 1 and 2, and 5) the combination of the Strat. 1–3. Since Strat. 3 and Strat. 5 involve AI planning, we also investigate the differences between synthesis and transformational augmentation. Consequently, Strat. 3.1 and Strat. 5.1 use synthesis while Strat. 3.2 and Strat. 5.2 use transformations. The augmentation methods that are involved in each strategy are applied in parallel, that is, each augmentation method augments the original process model. The result is one original process model that leads to one additional augmented process model. Given $n \in \mathbb{N}$ as the number of original non-augmented process models and $m \in \mathbb{N}$ as the number of augmentation methods, the number of process models after augmenting a dataset with one strategy equals $n + n \cdot m$. Therefore, the original process models as well as one augmented dataset for each augmentation method are part of the final augmented dataset of the strategy, leading to a factor of $m + 1$ for the multiplication of the training data.

We use three deletion-based augmentation methods for Strat. 1, 4, and 5, three augmentation methods based on replacements for Strat. 2, 4, and 5, and two augmentation methods based on AI planning for Strat. 3 and 5. All of these augmentation methods contain random aspects that can be seeded to allow reproducibility. The deletion-based methods work as follows:

Edge deletion

This method randomly deletes edges, including their semantic description, in the original process model. The parameterization possibilities of the method comprise the number of edges to delete and the types of edges to delete. The method is configured to delete two control-flow edges in the experiments.

Node deletion

This method randomly deletes nodes including their semantic description in the original process model. The method can be parameterized by the number of nodes to delete, the types of nodes to delete, and whether the connected edges should also be deleted. The configuration of the method in the experiments deletes one data node without deleting the connected edges.

Deletion within semantic descriptions

Randomly deleting items from semantic descriptions is the core of this augmentation method. Therefore, an attribute of the semantic descriptions of a certain type of node or edge is selected first and then deleted for several nodes. The method can be parameterized by the number of nodes or edges to target and the respective type of node or edge. The method is configured to delete one item in the semantic descriptions of two task nodes in the experiments.

In addition to the methods based on deletion, the experiments also use methods based on replacements. These methods are implemented as follows:

Task order swapping

This method swaps pairs of task nodes that are neighbors according to the control-flow, i. e., task nodes that are directly connected by a control-flow edge. The swapped nodes keep their original semantic descriptions but replace the former node in all edge connections. The method can be configured by the number of swaps to perform, which is two in the experiments.

Data order swapping

Analogous to swapping the task order, this method swaps neighboring data nodes along the data-flow. Two data nodes are neighbors if one of those nodes is consumed and the other one is produced by the same task node. The number of swaps to perform can be set as a parameter, and this value is two in the experiments.

Swapping within semantic descriptions

This augmentation method processes the semantic descriptions of nodes or edges and swaps contained items. Pairs of nodes or edges of the same type are first identified, an item of their semantic descriptions to swap is selected, and then this item is swapped. The method can be parameterized by the number of nodes or edges to target and the respective type of node or edge. The method is configured to swap one item in the semantic descriptions of three task nodes in the experiments.

For the augmentation methods of Strat. 3 and Strat. 5, we distinguish between synthesis (Strat. 3.1 and 5.1) and transformational augmentation (Strat. 3.2 and 5.2):

Synthesis

This method generates new process models by using the PDDL domain model and randomly setting an initial state and a goal state (see Section 4.5 for more information) for the search procedure of a planner. In our experiments, we generate the same number of process models as the augmentation methods of the other strategies to keep the size of the resulting datasets comparable. This means that a dataset augmented with Strat. 3.1 contains three times the number of generated process models compared to the original process models.

Transformation

Transformational augmentation via AI planning (see Section 4.5 for more information) in the experiments is conducted by using different planner heuristics for solving the planning problem of the original process model. Instead of utilizing A* search with the landmark-cut heuristic (*lmcut*) (Helmert and Domshlak 2009) that creates the original process models based on their planning problems, the augmented process models are created with three different variants of a lazy greedy best-first search: The first used heuristic is the context-enhanced additive heuristic (*hcea*) (Helmert and Geffner 2008) and the second heuristic is fast-forward (*hff*) (Hoffmann 2001). Both heuristics are also used in combination as a third variant (*hff-hcea*).

Tables A1 and A2 in the appendix give a statistical comparison of the original datasets and the respective augmented ones. The tables are concerned with the number of nodes and edges of all graphs in the respective datasets. In detail, we present the average, the minimum, and the maximum numbers. While Table A1 shows the absolute values, Table A2 gives a comparison of the average of these numbers w. r. t. the KS statistic. The displayed value is the complement of the statistic, and it measures how well the values of two samples align. Thereby, a value of 1 indicates perfect alignment, meaning that both samples most likely come from the same distribution, and a value of 0 means the worst alignment.

Model training and metrics

The training procedure for each variant of the GEM is two-phased, following the proposed transfer learning approach (see Section 4.6). For each augmented dataset, a pretraining is carried out with the triplet learning approach. The resulting model is then used as the base model for adaptation. Each pretrained model is fine-tuned with the data of the respective base dataset. For example, the dataset D_{100} is augmented with Strat. 1, resulting in the augmented dataset D'. The data of D' is then used for pretraining, and the pretrained model is trained with the data of D_{100} in the adaptation phase. We use identical hyperparameter configurations[5] for all pretraining runs and adaptation runs, respectively, which improves their comparability. These hyperparameters are determined by hyperparameter tuning (Hoffmann and Bergmann 2022a) that is done once for the pretraining of D_{100} augmented with Strat. 1 and the adaptation of the resulting model. All training runs train for a fixed number of 40 epochs, and, similar to early stopping, the best model is selected afterward according to the validation results of each epoch. Each of the adaptation training runs is done three times, and the evaluation results for these three models are averaged to get more consistent results. The GEM and all variants have a computational complexity of $O(n)$ (see Hoffmann and Bergmann (2022b) for more details), with n being the number of graph nodes. Due to the large number of trainable model parameters and training examples, the training times are relatively long, ranging between approx. 3–48 hours. The training times vary depending on the training phase, the specific domain, etc. Using these models during inference enables the similarity assessment for all pairs of cases in the different domains in less than one second.

Different variants of the GEM are compared by using them as a similarity measure for the scenario of a search in a process model repository. Given a single query process model from the test set, similar process models are searched based on the pairwise similarity between the query process model and all process models in the repository. The repository thereby contains all process models from the training set. This setup reflects real-world scenarios where a certain repository exists and new, unknown process models are used as queries. The inspected metrics for the comparison of different variants of the GEM are the MSE, the MAE, and the RMSE. All three metrics measure the error between the predicted similarity and the ground-truth similarity. While the predicted similarity stems from the corresponding GEM variant, we use ground-truth similarities that are computed by an A* graph matching algorithm (see Section 2.3 and Bergmann and Gil (2014)). The MSE computes the error as the average squared error of the difference between the predicted and the ground-truth similarity. The MAE does not use the square of these differences, but rather their absolute value. The RMSE is the square root of the MSE. Lower values of all three metrics are generally better.

Experimental results

The results of the experiments are presented in Table 3. Two tables are provided with the absolute values of the error metrics in Table 3a and the relative differences in Table 3b. The tables cover the computed metrics of the different variants of the GEM for all datasets. These variants include the standard, non-augmented model and the models augmented with different strategies, as introduced in Section 5.2–5.4. The tables also include a variant of the A* graph matching similarity measure (see Section 2.3 and Bergmann and Gil (2014)) that is also used to compute the ground-truth similarities. This variant is included to connect the results of the augmented models to previous work, where embedding-based similarity measures are compared to measures based on A* graph matching (see Section 2.4 and Hoffmann et al. (2020) and Hoffmann and Bergmann (2022b) for more details). This A* measure differs from the one used to compute the ground-truth similarities in the sense that it is restricted in the maximum number of solution candidates to maintain during the search procedure. The value is chosen such that the average computation time is approximately equal to that of the other measures to allow a fair comparison with a similar time budget. The same strategy has also been used in previous work (Hoffmann and Bergmann 2022b; Hoffmann et al. 2020).

The first observation in the data is that all three metrics behave very similar when comparing the same model, with only small deviations in some cases. Therefore, in the following, we do not describe the data w. r. t. each metric individually but rather as a combination. It is apparent that the best-performing model is never the non-augmented one but rather one of Strat. 1–3.1 for all domains. While these models are generally better than the non-augmented model (indicated by negative percentages in Table 3b) this is not the case for Strat. 3.2–5.2. There, we can observe an increased error for some domains, e. g., Strat. 5 of D_{50}. The metric values for the Strat. 1–3.1 are similar across all domains, and the observed variations do not follow a clear trend between different domains. The data also shows that similar amounts of training data lead to models of similar quality, regardless of the data quality. This becomes apparent, for instance, when comparing the model augmented with Strat. 3.1 of domain D_{50} with the non-augmented model of D_{200}. Both are trained with 200 training process models and, with an MAE of approximately 0.1, the performance is very similar. However, training with more data does not necessarily lead to better results, as shown by the models of Strat. 3.2 and 5 compared to the models of Strat. 1–3.1 of D_{200}. The comparison between synthesis and transformational augmentation with AI planning reveals mixed results. Strat. 3.2 seems to perform worse than Strat. 3.1, on average, and Strat. 5.2 seems to generally outperform Strat. 5.1. It is noticeable that Strat. 3.2 performs very poorly for all domains except D_{50}. The measure based on A* is

Table 3. Evaluation results (bold values highlight the result of the best strategy).

		100			50			200			300		
	Model	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓
(a) Absolute Errors.													
	Standard	0.1772	0.1851	0.0384	0.1603	0.1690	0.0308	0.1087	0.1205	0.0157	0.1234	0.1326	0.0216
	Strat. 1	0.1204	0.1311	0.0187	**0.0746**	**0.0841**	**0.0083**	0.0775	0.0908	0.0101	0.0984	0.1068	0.0139
	Strat. 2	0.1031	0.1142	0.0146	0.0802	0.0904	0.0098	**0.0747**	**0.0894**	**0.0091**	0.1163	0.1229	0.0179
	Strat. 3.1	**0.0911**	**0.0996**	**0.0117**	0.0917	0.1009	0.0125	0.0888	0.1039	0.0117	**0.0972**	**0.1048**	**0.0139**
Augmented	Strat 3.2	0.2754	0.3162	0.1167	0.1253	0.1398	0.0243	0.3195	0.3344	0.1422	0.2439	0.2524	0.1051
	Strat. 4	0.1118	0.1248	0.0173	0.1774	0.1963	0.0450	0.0863	0.1032	0.0130	0.1204	0.1310	0.1310
	Strat 5.1	0.1769	0.1969	0. 0.0438	0.1583	0.1706	0.0339	0.1378	0.1584	0.0281	0.1005	0.1118	0.0153
	Strat. 5.2	0.1566	0.1732	0.0329	0.1363	0.1495	0.0264	0.1096	0.1258	0.0177	0.1729	0.1832	0.0664
	A* measure	0.1656	0.1709	0.0298	0.1764	0.1799	0.0329	0.1816	0.1844	0.0344	0.1560	0.1618	0.0270
(b) Relative Errors.													
	Standard	0.1772	0.1851	0.0384	0.1603	0.1690	0.0308	0.1087	0.1205	0.0157	0.1234	0.1326	0.0216
	Strat. 1	−32.1%	−29.2%	−51.3%	**-53.5%**	**-50.2%**	**-73.0%**	−28.7% -	−24.6%	−35.7% -	−20.3%	−19.5%	−35.5%-
	Strat.2	−41.8%	−38.3%	−62.0%	−50.0%	−46.5%	−68.3%	**-31.3%**	**-25.8%**	**-41.8%**	−5.7%	−7.3%	−17.0%
	Strat. 3.1	**-48.6%**	**-46.2%**	**-69.4%**	−42.8%	−40.3%	−59.4%	−18.3%	−13.7%	−25.3%	**-21.2%**	**-20.9%**	**-35.7%**
Augmented	Strat 3.2	55.4%	70.8%	204.3%	−21.9%	−17.2%	−21.0%	194.0%	177.5%	805.7%	97.7%	90.4%	387.5%
	Strat. 4	−36.9%	−32.6%	−55.0%	10.7%	16.2%	46.4%	−20.6%	−14.4%	−17.2%	−2.4%	−1.2%	−9.6%
	Strat 5.1	−0.2%	6.4%	−55.0%	−1.3%	0.9%	10.2%	26.7%	31.5%	78.8%	−18.6%	−15.7%	−28.9%
	Strat. 5.2	−11.6%	−6.4%	−14.2%	−15.0%	−11.5%	−14.3%	0.8%	4.4%	12.5%	40.1%	38.2%	207.7%
	A* measure	−6.6%	−7.7%	−22.3%	10.0%	6.5%	6.9%	67.1%	53.0%	119.1%	26.5%	22.1%	25.2%

outperformed in almost all comparisons with the embedding-based measures which is in line with the experimental results of previous work (Hoffmann and Bergmann 2022b; Hoffmann et al. 2020). It also appears that the larger domains, i. e., more training data, favor the embedding-based measure more.

When interpreting the results of the embedding-based measures, a great potential of the process augmentation approach is revealed. The augmented datasets lead to a reduction of the prediction error of up to 53% in terms of MAE. This effect weakens with larger datasets, but it is still significant, i. e., a reduction of 21% for the largest domain D_{300}. More original training data is expected to reduce the need for augmented data, which is in line with our results. Furthermore, the results indicate that the augmentation methods of the different categories do not lead to trained models with different levels of performance. The observable differences between the augmented models of the Strat. 1–3 follow no trend across the domains, and are most likely variation. The error reductions appear to be possible with augmented process models of higher and lower quality. Additionally, the results indicate that the highly augmented datasets, i. e., the datasets augmented with Strat. 4 and 5, perform worse than the datasets of Strat. 1–3 on average. This is contrary to the common expectation that more (augmented) training data improves the performance of DL models. And while this expectation holds for the comparison between the differently sized domains, the reason for the adverse effect of Strat. 4 and 5 might be a drift between the original datasets and the highly augmented ones. This means that the augmented data does not adequately represent the original data anymore. Furthermore, the different results of synthesis and transformational augmentation via AI planning might be a result of the different ways the augmented datasets are created. Table A2 indicates that most of the statistical numbers point to a close alignment between the augmented datasets and the respective original ones, except for Strat. 3.2 and 5.2. It appears that the applied method of augmenting process models by using different heuristics to solve the same planning problem leads to large variations in the resulting process models, thus the low values of the KS statistic. In contrast, Strat. 3.1 and 5.1 have much higher KS values that indicate an advantage of synthesis in this scenario.

To accept or reject the hypotheses, the experimental results are assessed statistically and qualitatively. The statistical assessment is conducted by paired, two-sample, one-sided t-tests[6] between the similarity predictions of two individual models to check for unequal mean values. For instance, to get a statistical confidence for Hypothesis 1, the p-value of the t-tests of all pairs of one augmented model and the standard model in all domains (28 in total) are checked to see if the augmented models predict similarities that are different from the standard model. As the t-tests only check for differences of the mean values, the hypotheses are also checked qualitatively for better or worse predictions, given by Table 3. The following conclusions for the hypotheses are drawn based on the statistical and qualitative results: Hypothesis 1 is accepted, due to statistical significance ($p < 0.05$) and since we can see a strong positive influence of the augmented datasets on the model performance compared to the original data. Although some models perform worse than the baseline, a better performing model can still be found in a simple model selection or hyperparameter tuning procedure. Hypotheses 2 and 3 are rejected, as there are no clear trends between models being trained on the augmented data of different categories of augmentation methods. The statistical results for both hypotheses are mostly statistically significant ($p < 0.05$) apart for the comparisons of Strat. 1 and 2 as well as Strat. 2 and 3.1 for domain D_{50}. The t-tests for Hypothesis 4 show statistical significance ($p < 0.05$) except for the comparison between Strat. 5.1 and 2 in domain D_{300}. The hypothesis is still rejected due to the generally worse performance of the Strat. 4 and 5 compared to the Strat. 1–3.

Discussion

One of the key outcomes of the experiments is the observation that DL models trained on augmented process models with a guaranteed correctness do not consistently outperform DL models trained on augmented process models without correctness guarantees. This appears to be contrary to the common expectation that training data with a higher quality leads to trained models of noticeably better quality. We think that this effect cannot be seen here due to the following reason: There are enough augmented process models created such that the individual incorrect parts do not lead to a lowered quality of the trained model, and the important characteristics of the original process models are still represented by the majority of the augmented data. The following example demonstrates this: An augmented process model where a control-flow node is deleted does not allow the trained model to learn the control-flow as it should be. If the other augmented process models of the training dataset are augmented differently, the trained model can still learn the correct characteristics of the control-flow, as the combined characteristics of the training examples represent all the needed information to learn. That is, the (hypothetical) average augmented training example represents the same distribution as the original non-augmented training example. This could be the reason, the different categories of augmentation methods perform similarly. While this is generally positive, there is a risk for small datasets as well as datasets with very distinct individual examples because in these scenarios, each example is more valuable to the training procedure and a significant amount of information might be lost during augmentation by methods without correctness guarantees.

Another noticeable observation refers to the Strat. 4–5 that combine multiple augmentations from the other strategies. The models trained on these augmented process models perform mainly worse than the other augmented models. We interpret the data as an indication of a drift between the non-augmented and augmented datasets due to the highly diverse training sets. It might be the case that too much augmentation with different methods shows the opposite of the effect that was discussed before, ultimately reducing the quality of the trained model. Another factor that might play a role, especially in the training procedure with large amounts of training data, is overfitting. Although the effects are mitigated by early stopping in every training run, there are still hints of it when inspecting the validation loss in comparison to the training loss in the training statistics. The decision to perform hyperparameter optimization only once and reuse the results for all other training runs (see the details in Section 5.4) was made deliberately to keep the experimental results between different models comparable. Nevertheless, there might be room for improvement by performing an individual hyperparameter optimization for each training procedure.

In summary of the experiments, we recommend utilizing augmentation to enlarge the data available for the training procedure of a DL model. The ratio between original and augmented data should be chosen carefully. Larger original training data sets appear to be more robust to augmented data of low quality, but the gain expected by using this augmented data in the training procedure appears to be lower. Analogously, smaller original training data sets might have a higher risk of augmented data of low quality disrupting the training procedure, but the potential gains by training with a larger data set are higher.

Limitations and transferability to other domains

In this section, we discuss the limitations and main assumptions of the proposed approach and, particularly, examine the transferability of the approach to other domains.

Sequential NEST process model representation

One of the main assumptions of the approach is the restriction of the discussed categories of augmentation methods to sequential NEST process models, without any rich control-flow elements such as AND or XOR elements (see Section 4.2). While the NEST graph format itself is no limitation (see Section 2.1), the assumption to have sequential processes might be. This assumption enables this paper to have a proper evaluation of different augmentation methods w. r. t. the correctness of the resulting augmented process models and, thus, the influence of training data with a higher correctness on the quality of the trained embedding models. Transferring the same principles onto other, arbitrary domains seems challenging at first glance, but we still expect it to be possible. For example, it is possible to split the whole process model into several smaller process models, e. g., one subprocess before the control-flow element, one subprocess with the control-flow elements and their branches, and one subprocess with the rest of the process model. The augmentation methods can then recursively be executed on these (sequential) subprocesses to augment them. Thereafter, the processes are put together in a single combined augmented process model.

Selection of augmentation methods and planning domain definition

The presented augmentation categories and the augmentation methods from these categories are selected to fit the assumptions on correctness and the described key criteria. However, we still expect the described augmentation methods to be universally applicable to other domains, especially when correctness is not relevant. The main determining factor to deciding on the concrete methods is most likely the effort that can be put into the implementation of the augmentation and training procedures described in this work. As the evaluation shows no noticeable differences between the augmentation methods of the three categories, even simple augmentations can perform well. But, we are not sure if this also applies to other domains and, therefore, encourage not to disregard methods with correctness guarantees. The use of AI planning in particular can be challenging, even though AI planning is already being used successfully in the BPM area (Malburg, Klein, and Bergmann 2023; Malburg; Hoffmann and Bergmann 2023; Malburg, Brand, and Bergmann 2023; Marrella, Mecella, and Sardiña 2017, 2018; Masellis et al. 2022; Rodríguez-Moreno et al. 2007) and in different lifecycle phases (Marrella 2019). However, for using AI planning, a comprehensive formal planning domain is required, which is difficult to achieve in real-world application scenarios and practice (Malburg, Klein, and Bergmann 2023; Nguyen, Sreedharan, and Kambhampati 2017). To remedy this problem, already available encoded knowledge, e. g., in the form of an ontology, can be reused and converted to a formal planning domain definition (Malburg, Klein, and Bergmann 2023). In addition, there exist approaches (for a comprehensive overview, see Jilani 2020) that try to remedy the efforts for creating a formal planning domain definition. For example, in McCluskey et al. (2009) and in Zhuo, Muñoz-Avila, and Yang (2014) the authors use demonstrations or traces, i. e., in our case available process models or event logs of executed process instances, to automatically derive the planning domain. In addition, special developed user interfaces (e. g., Wickler, Chrpa, and McCluskey 2015) or generative models such as ChatGPT can be used to remedy these efforts (e. g., Guan et al. 2023; Liu et al. 2023; Valmeekam et al. 2023). However, and even if domain experts or already encoded and formal knowledge or demonstrations and traces are available, it means a certain amount of effort that needs to be investigated. If this effort is too high or there is no other formal knowledge available that can be used to create a formal planning domain, augmentation methods of Cat. 1 and Cat. 2 should favorably be used, as they are universally applicable independent of the concrete domain.

Similarity learning use case

The presented approach and its evaluation are built around the use case of similarity predictions between pairs of process models. This use case is chosen, as previous work of the authors (e.g., Hoffmann and Bergmann 2022b) also deals with it, and it is relevant to BPM literature (e.g., Schoknecht et al. 2017). However, the approach is designed to be applicable to other use cases where DL models are utilized to improve BPM applications. That is, the augmentations (see Section 4.3 to 4.5) are not dependent on the use case of similarity prediction but can rather be used to enlarge the training data for arbitrary use cases. Additionally, the transfer learning approach (see Section 4.6) is also rather generic and can be applied to other DL models and training procedures as well. The core feature is the combination of the unsupervised and the supervised training procedures that is not specific to similarity learning. In fact, previous work (Schuler et al. 2023) experiments with a very similar transfer learning setup and shows its suitability across multiple domains. Furthermore, the main contributions might also be interesting to BPM applications that usually deal with process instances or process traces, e.g., predictive business process monitoring (Rama-Maneiro, Vidal, and Lama 2023). The processes used there can be represented in a graph form and also be augmented with the discussed methods, and training can be enhanced with the proposed transfer learning approach.

Conclusion and future work

This paper presents an approach for using augmentation to enhance DL models in the context of similarity learning for semantic processes. The approach is motivated by an analysis of existing literature on process augmentation from BPM research, which shows that the topic of augmentation is largely unexplored. The proposed augmentation methods are organized into three categories that represent their properties regarding complexity and knowledge-intensity and correctness and quality. Additionally, a semi-supervised transfer learning procedure is described to combine training with augmented and non-augmented data. The experimental evaluation compares graph embedding models that are trained on augmented data with models trained on non-augmented data. The results indicate a large potential of the augmented datasets but also strong variations, depending on the specific augmentation methods used and the size of the augmented datasets. As the main takeaway of the experiments, we recommend utilizing augmentation methods to enlarge existing training datasets. It is especially recommended if the size of the original datasets is small, as the potential gains of more data for the training procedures are higher than those with larger original datasets. However, the applicability of the approach is limited due to the assumption of sequential NEST processes, which are not always available. The need for a formal planning domain as well as the focus on similarity learning might also be limitations for certain domains and augmentation methods.

In future work, we aim to extend and refine the proposed approach. Due to our focus on the manufacturing domain in this work, it is not apparent how the approach performs for process models of other domains (see Section 6). In this regard, the applicability of the approach in the domain of event logs should also be examined. It could complement other literature, such as Käppel and Jablonski (2023) or Grüger et al. (2022). In this context, we also want to investigate how LLM can be combined and used with other AI methods such as automated planning (e. g., Guan et al. 2023; Liu et al. 2023; Valmeekam et al. 2023) to support process augmentation further. Furthermore, the usage of other GNN than the GEM (Hoffmann and Bergmann 2023; Hoffmann et al. 2020), is part of future work. This provides insights on the effects of the proposed augmentation methods for other DL models. In addition, these approaches can be used in combination with process model adaptation approaches (Müller 2018; Zeyen, Malburg, and Bergmann 2019; Malburg, Hoffmann and Bergmann 2023; Malburg, Brand, and Bergmann 2023), as self-learning process adaptation requires large amounts of data. Finally, it is planned to apply the proposed approach in a greater comparison with related work on process matching, e. g., other process similarity measures (Assy, van Dongen, and van der Aalst 2018; Dijkman et al. 2011; Jabeen, Leopold, and Reijers 2017; Yan, Remco, and Paul 2010).

Notes

1. Available at https://gitlab.rlp.net/procake/procake-framework.
2. The defined node types comprise task, data, control-flow, and workflow nodes. The defined edge types comprise control-flow, data-flow, part-of, and constraint edges.
3. More information on the used smart factory can be found at: https://iot.uni-trier.de.
4. All PDDL files created for the paper are publicly available. See the statement on data availability at the end of the article for more information.
5. A list of all hyperparameters is available as part of the supplementary data. See the reference in the data availability statement at the end of the paper.
6. The detailed results of the statistical tests are part of the supplementary data, as referenced in the data availability statement at the end of the paper.

Acknowledgements

The publication was funded/supported by the Open Access Fund of Universität Trier and by the German Research Foundation (DFG).

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Maximilian Hoffmann http://orcid.org/0000-0002-7932-5822
Lukas Malburg http://orcid.org/0000-0002-6866-0799
Ralph Bergmann http://orcid.org/0000-0002-5515-7158

Data availability statement

The source code, the datasets, the trained models, and other supplementary resources are available online at https://gitlab.rlp.net/iot-lab-uni-trier/aai-journal-2024. The data was last accessed on August 8th, 2024.

References

Adjeisah, M., X. Zhu, H. Xu, and T. A. Ayall. 2023. To- wards data augmentation in graph neural network: An overview and evaluation. *Computer Science Review* 47:100527. doi: 10.1016/j.cosrev.2022.100527.

Assy, N., B. F. van Dongen, and W. M. P. van der Aalst. 2018. Similarity resonance for improving process model matching accuracy. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ed. H. M. Haddad, 86–93. New York, NY: ACM Conferences.

Bergmann, R., ed. 2002. *Experience management: Foundations, development method- ology, and internet-based applications*, vol. 2432. Berlin Heidelberg: LNCS. Springer.

Bergmann, R., and Y. Gil. 2014. Similarity assessment and efficient retrieval of semantic workflows. *Information Systems* 40:115–27. issn: 03064379. doi: 10.1016/j.is.2012.07.005.

Bergmann, R., G. Lisa, M. Lukas, and Z. Christian. 2019. Pro- CAKE: A process-oriented case-based reasoning framework. In *Workshops Proceedings for the Twenty-seventh International Conference on Case-Based Rea- soning co-located with the Twenty-seventh International Conference on Case-Based Reasoning (ICCBR 2019), September 8-12, 2019*, ed. S. Kapetanakis and H. Borck, vol. 2567, 156–61. Otzenhausen, Germany: CEUR Workshop Proceedings. CEUR-WS.org.

Chen, T., S. Kornblith, M. Norouzi, and G. E. Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proc. of the 37th ICML, virtual event*, ed. H. Daumé and A. Singh, vol. 119, 1597–607. PMLR.

Chen, Y., Y. Xie, L. Song, F. Chen, and T. Tang. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6 (3):264–74. doi: 10.1016/j.eng.2020.01.007.

de Leoni, M., W. M. P. van der Aalst, and M. Dees. 2016. A generalprocess mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems* 56:235–57. doi: 10.1016/j.is.2015.07.003.

Di Francescomarino, C., and C. Ghidini. 2022. Predictive process monitoring. In *Process mining handbook*, ed. W. van der Aalst and J. Carmona, vol. 448, 320–46. Cham: Springer. Lecture Notes in Business Information Processing. isbn: 978-3-031-08847-6.

Dijkman, R., M. Dumas, B. van Dongen, R. Käärik, and J. Mendling. 2011. Similarity of business process models: Metrics and evaluation. *Information Systems* 36 (2):498–516. doi: 10.1016/j.is.2010.09.006.

Ding, K., Z. Xu, H. Tong, and H. Liu. 2022. Data augmentation for deep graph learning. *ACM SIGKDD Explorations Newsletter* 24 (2):61–77. issn:1931-0145. doi: 10.1145/3575637.3575646.

Ghallab, M., D. Nau, and P. Traverso. 2016. *Automated planning and acting*. Cambridge: Cambridge University Press.

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proc. Of the 34th ICML, Australia*, ed. M. D. Precup and Y. W. Teh, vol. 70, 1263–72. PMLR.

Grüger, J., T. Geyer, D. Jilg, and R. Bergmann. 2022. SAMPLE: A semantic approach for multi-perspective event log generation. In *ICPM 2022 Workshops, 2022*, ed. M. Montali, A. Senderovich, and M. Weidlich, vol. 468, 328–40. Cham: LNBIP. Springer.

Grumbach, L., and R. Bergmann. 2017. Using constraint satisfaction problem solving to enable workflow flexibility by deviation (Best technical paper). In *Artificial intelligence XXXIV*, ed. M. A. Bramer and M. Petridis, vol. 10630, 3–17. Cham: Springer. LNAI.

Guan, L., K. Valmeekam, S. Sreedharan, and S. Kambhampati. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *CoRR* Abs/2305.14909.

Haslum, P., N. Lipovetzky, D. Magazzeni, and C. Muise. 2019. An introduction to the planning domain definition language. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*, ed. N. Lipovetzky, D. Magazzeni, C. Muise. San Rafael, CA: Morgan & Claypool.

Helmert, M., and C. Domshlak. 2009. Landmarks, critical paths and Abstractions: What's the difference anyway? In *19th ICAPS*, ed. A. Gerevini, A. Howe, A. Cesta, and I. Refanidis, 162–69. Washington, DC: AAAI.

Helmert, M., and H. Geffner. 2008. Unifying the causal graph and additive heuristics. In *18th ICAPS*, ed. J. Rintanen, B. Nebel, J. C. Beck, E. A. Hansen, 140–47. Washington, DC: AAAI.

Hermans, A., L. Beyer, and B. Leibe. 2017. In defense of the triplet loss for person Re-identification. *CoRR* abs/1703.07737. arXiv: 1703.07737.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI Magazine* 22 (Septem– ber):57–62.

Hoffmann, M., and R. Bergmann. 2022a. Improving automated hyper-parameter optimization with case-based reasoning. In *Case-based reasoning research and development*, ed. M. T. Keane and N. Wiratunga, vol. 13405, 273–88. Cham: Springer. LNCS.

Hoffmann, M., and R. Bergmann. 2022b. Using graph embedding techniques in process-oriented case-based reasoning. *Algorithms* 15 (2):27. doi: 10.3390/a15020027.

Hoffmann, M., and R. Bergmann. 2023. Ranking-based case retrieval with graph neural networks in process- oriented case-based reasoning. *The International FLAIRS Conference Proceedings* 36. doi: 10.32473/flairs.36.133039.

Hoffmann, M., L. Malburg, N. Bach, and R. Bergmann. 2022. GPU- Based graph matching for accelerating similarity assessment in process-oriented case-based reasoning. In *30th ICCBR*, ed. M. T. Keane and N. Wiratunga, vol. 13405, 240–55. Cham: LNCS. Springer.

Hoffmann, M., L. Malburg, P. Klein, and R. Bergmann. 2020. Using siamese graph neural networks for similarity-based retrieval in process- oriented case-based reasoning. In *28th ICCBR 2020*, ed. I. Watson and R. O. Weber, 229–44. Spain: Springer.

Jabeen, F., H. Leopold, and H. A. Reijers. 2017. How to make process model matching work better? An analysis of current similarity measures. In *Business information systems*, ed. W. Abramowicz, vol. 288, 181–93. Cham, Switzerland: Springer. Lecture Notes in Business Information Processing.

Jilani, R. 2020. Automated domain model learning tools for planning. In Chap. 2 in *knowledge engineering tools and techniques for AI planning*, ed. M. Vallati and D. E. Kitchin, 21–46. Cham: Springer.

Käppel, M., and S. Jablonski. 2023. Model-agnostic event log augmentation for predictive process monitoring. In *Advanced information systems engineering*, ed. M. Indulska, I. Reinhartz-Berger, C. Cetina, and O. Pastor, vol. 13901, 381–97. Cham: LNCS. Springer.

Käppel, M., S. Schönig, and S. Jablonski. 2021. Leveraging small sample learning for business process management. *Information and Software Technology* 132:106472. 10.1016/j.infsof.2020.106472.

Klein, P., L. Malburg, and R. Bergmann. 2019. Learning workflow em- beddings to improve the performance of similarity-based retrieval for process-oriented case-based reasoning. In *Case-based reasoning research and develop- ment*, ed. K. Bach and C. Marling, vol. 11680, 188–203. Cham: Springer. LNCS.

Kudenko, D. 2014. Special issue on transfer learning. *KI - Künstliche Intelligenz* 28 (1):5–6. doi: 10.1007/s13218-013-0289-5.

Lenz, M., S. Ollinger, P. Sahitaj, and R. Bergmann. 2019. Semantic textual similarity measures for case-based retrieval of argument graphs. In *27th ICCBR 2019*, ed. K. Bach and C. Marling, 219–34. Germany: Springer.

Liu, B., Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. 2023. LLM+P: Empowering large language models with optimal planning proficiency. *CoRR* abs/2304.11477.

Malburg, L., F. Brand, and R. Bergmann. 2023. Adaptive management of cyber-physical workflows by means of case-based reasoning and automated planning. In *26th EDOC workshops*, ed. T. Prince Sales, H. A. Proper, G. Guizzardi, M. Montali, F. M. Maggi, and C. M. Fonseca, vol. 466, 79–95. Cham: LNBIP. Springer.

Malburg, L., M. Hoffmann, and R. Bergmann. 2023. Applying MAPE-K control loops for adaptive workflow management in smart factories. *Journal of Intelligent Information Systems* 61 (1):83–111. doi: 10.1007/s10844-022-00766-w.

Malburg, L., P. Klein, and R. Bergmann. 2020. Semantic web services for AI-Research with physical factory simulation models in industry 4.0. In *1st IN4PL*, ed. H. Panetto, K. Madani, and A. V. Smirnov, 32–43. Setúbal: ScitePress.

Malburg, L., P. Klein, and R. Bergmann. 2023. Converting semantic web services into formal planning domain descrip- tions to enable manufacturing process planning and scheduling in industry 4.0. *Engineering Applications of Artificial Intelligence* 126:106727. doi: 10.1016/j.engappai.2023.106727.

Malburg, L., R. Seiger, R. Bergmann, and B. Weber. 2020. Using physical factory simulation models for business process management research. In *BPM workshops*, ed. A. Del Río Ortega, H. Leopold, and F. M. Santoro, vol. 397, 95–107. Cham: LNBIP. Springer.

Marrella, A. 2019. Automated planning for business process management. *Journal on Data Semantics* 8 (2):79–98. issn: 1861-2040. doi: 10.1007/s13740-018-0096-0.

Marrella, A., M. Mecella, and S. Sardiña. 2017. Intelligent process adaptation in the SmartPM system. *ACM Transactions on Intelligent Systems and Technology* 8 (2):1–43. doi: 10.1145/2948071.

Marrella, A., M. Mecella, S. Sardiña, and C. Linares López. 2018. Supporting adaptiveness of cyber-physical processes through action-based formalisms. *AI Communications* 31 (1):47–74. doi: 10.3233/AIC-170748.

Marrium, M., and A. Mahmood. 2022. Data augmentation for graph data: Recent advancements." *CoRR* abs/2208.11973. arXiv: 2208.11973.

Masellis, R. D., C. Di Francescomarino, C. Ghidini, and S. Tessaris. 2022. Solving reachability problems on data-aware workflows. *Expert Systems with Applications* 189:116059. doi: 10.1016/j.eswa.2021.116059.

McCluskey, T. L., S. Cresswell, N. E. Richardson, and M. M. West. 2009. Automated acquisition of action knowledge. In *1st ICAART*, ed. J. Filipe, and A. L. N. Fred, and B. Sharp, 93–100. Setúbal: INSTICC Press.

McDermott, D., M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. *PDDL - The planning domain definition language. Technical report CVC TR-98-003/DCS TR-1165.*

Mikolov, T., K. Chen, G. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. In *1st ICLR, Scottsdale, Workshop Track Proceedings*, ed. Y. Bengio and Y. LeCun. Ithaca, NY: arXiv.

Müller, G. 2018. *Workflow modeling assistance by case-based reasoning*. Wies- baden: Springer. isbn: 978-3-658-23558-1.

Mumuni, A., and F. Mumuni. 2022. Data augmentation: A comprehensive survey of modern approaches. *Array* 16:100258. doi: 10.1016/j.array.2022.100258.

Nguyen, T. A., S. Sreedharan, and S. Kambhampati. 2017. Robust planning with incomplete domain models. *Artificial Intelligence* 245:134–61. doi: 10.1016/j.artint.2016.12.003.

Nolle, T., S. Luettgen, A. Seeliger, and M. Mühlhäuser. 2018. Analyzing business process anomalies using autoencoders. *Machine Learning* 107 (11):1875–93. issn: 0885-6125. doi: 10.1007/s10994-018-5702-8.

Ontañón, S. 2020. An overview of distance and similarity functions for structured data. *Artificial Intelligence Review* 53 (7):5309–51. issn: 0269-2821. doi: 10.1007/s10462-020-09821-w.

Ott, F., D. Rügamer, L. Heublein, B. Bischl, and C. Mutschler. 2022. Cross-modal common representation learning with triplet loss functions." *CoRR* abs/2202.07901. arXiv: 2202.07901.

Pauli, J., M. Hoffmann, and R. Bergmann. 2023. Similarity-based retrieval in process-oriented case-based reasoning using graph neural networks and transfer learning. *The International FLAIRS Conference Proceedings* 36. doi: 10.32473/flairs.36.133040.

Pfeiffer, P., J. Lahann, and P. Fettke. 2021. Multivariate business process representation learning utilizing gramian angular fields and convolutional neural networks. In *Business process management*, ed. A. Polyvyanyy, M. T. Wynn, A. Van Looy, and M. Reichert, vol. 12875, 327–44. Cham: LNCS. Springer.

Rama-Maneiro, E., J. C. Vidal, and M. Lama. 2023. Deep learning for predictive business process monitoring: Review and benchmark. *IEEE Transactions on Services Computing* 16 (1):739–56.

Richter, M. M. 2007. Foundations of similarity and utility. In *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, May 7–9, 2007*, ed. D. Wilson and G. Sutcliffe, 30–37. Key West, FL, USA: AAAI Press.

Rodríguez-Moreno, M. D., D. Borrajo, A. Cesta, and A. Oddi. 2007. Integrating planning and scheduling in workflow domains. *Expert Systems with Applications* 33 (2):389–406. doi: 10.1016/j.eswa.2006.05.027.

Schoknecht, A., T. Thaler, P. Fettke, A. Oberweis, and R. Laue. 2017. Similarity of business process models—A state-of-the-art analysis. *ACM Computing Surveys* 50 (4). issn: 0360-0300. 1–33. doi: 10.1145/3092694.

Schroff, F., D. Kalenichenko, and J. Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, 815–23. USA: IEEE.

Schuler, N., M. Hoffmann, H.-P. Beise, and R. Bergmann. 2023. Semi-supervised similarity learning in process-oriented case-based reasoning. In *Artificial intelligence XL - 43rd SGAI conference 2023*, ed. M. Bramer and F. T. Stahl, vol. 14381, 159–173. Cham: LNCS. Springer.

Schultheis, A., D. Jilg, L. Malburg, S. Bergweiler, and R. Bergmann. 2024. Towards flexible control of production processes: A require- ments analysis for adaptive workflow management and evaluation of suitable process modeling languages. *Processes* 12 (12):2714. doi: 10.3390/pr12122714.

Seiger, R., S. Huber, P. Heisig, and U. Aßmann. 2019. Toward a framework for self-adaptive workflows in cyber-physical systems. *Software & Systems Modeling* 18 (2):1117–34. doi: 10.1007/s10270-017-0639-0.

Seiger, R., L. Malburg, B. Weber, and R. Bergmann. 2022. Integrating process management and event processing in smart factories: A systems architecture and use cases. *Journal of Manufacturing Systems* 63:575–92. doi: 10.1016/j.jmsy.2022.05.012.

Shu, J., Z. Xu, and D. Meng. 2018. Small sample learning in big data era. https://arxiv.org/abs/1808.04572.

Tan, C., F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. 2018. A survey on deep transfer learning. In *27th international ICANN*, ed. V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, vol. 11141, 270–79. Greece: LNCS. Springer.

Valmeekam, K., S. Sreedharan, M. Marquez, A. Olmo, and S. Barao Kambhampati. 2023. On the planning abilities of large language models (A critical investigation with a proposed benchmark). *CoRR* Abs/2302.06706.

van Dyk, A. D., and X.-L. Meng. 2001. The art of data augmentation. *Journal of Computational and Graphical Statistics* 10 (1):1–50.

Venkateswaran, P., V. Muthusamy, V. Isahagian, and N. Venkata- Subramanian. 2021. Robust and generalizable predictive models for business processes. In *Business process management*, ed. A. Polyvyanyy, M. T. Wynn, A. Van Looy, and M. Reichert, vol. 12875, 105–22. Cham: LNCS. Springer. Isbn: 978-3-030-85468-3.

von Rueden, L., S. Mayer, K. Beckh, B. Georgiev, S. Gies- Selbach, R. Heese, B. Kirsch, M. Walczak, J. Pfrommer, and Pick. 2021. Informed machine learning - a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering* 1:1–1. doi: 10.1109/TKDE.2021.3079836.

Weidlich, M., D. Remco, and M. Jan. 2010. The ICoP framework: Identification of correspondences between process models. In *Advanced informa- tion systems engineering*, ed. B. Pernici, 483–98. Berlin, Heidelberg: Springer Berlin Heidelberg. isbn: 978-3-642-13094-6.

Weiss, K., T. M. Khoshgoftaar, and D. Wang. 2016. A survey of transfer learning. *Journal of Big Data* 3 (1). doi: 10.1186/s40537-016-0043-6.

Wickler, G., L. Chrpa, and T. L. McCluskey. 2015. Ontological sup- port for modelling planning knowledge. In *Knowledge discovery, knowledge engineering and knowledge management*, ed. L. Ana, N. Fred , J. L. G. Dietz, D. Aveiro, K. Liu, and J. Filipe, vol. 553, 293–312. Cham: CCIS. Springer.

Wu, H., P. Judd, X. Zhang, M. Isaev, and P. Micikevicius. 2020. Integer quantization for deep learning inference: Principles and empirical evalu- ation." *CoRR* abs/2004.09602. arXiv: 2004.09602.

Yan, Z., D. Remco, and G. Paul. 2010. Fast business process similarity search with feature-based similarity estimation. In *On the move to meaningful internet systems*, ed. R. Meersman, T. Dillon, and P. Herrero, vol. 6426, 60–77. Berlin: Springer. Lecture Notes in Computer Science.

Yu, S., H. Huafei, N. D. Minh, and X. Feng. 2022. Graph augmentation learning. In *Companion Proceedings of the Web Conference 2022*, ed. F. Laforest, 1063–72. ACM Digital Library. New York, NY, United States: Association for Computing Machinery. isbn: 9781450391306.

Zeyen, C., and R. Bergmann. 2020. A*-based similarity assessment of semantic graphs. In *28th ICCBR*, ed. I. Watson and R. O. Weber, vol. 12311, 17–32. Cham: LNCS. Springer. Isbn: 978-3-030-58341-5.

Zeyen, C., L. Malburg, and R. Bergmann. 2019. Adaptation of scientific workflows by means of process-oriented case-based reasoning. In *27th ICCBR*, ed. K. Bach and C. Marling, vol. 11680, 388–403. Cham: LNCS. Springer.

Zhao, T., G. Liu, S. Günnemann, and M. Jiang. 2022. Graph data augmentation for graph machine learning: A survey." *CoRR* abs/2202.08871. arXiv: 2202.08871.

Zhou, J., C. Xie, Z. Wen, X. Zhao, and Q. Xuan. 2023. Data augmentation on graphs: A technical survey." *CoRR* abs/2212.09970. arXiv: 2212.09970.

Zhou, Y., and Z. Jianyang. 2015. Massively parallel A* search on a GPU. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015*, ed. B. Bonet and S. Koenig, 1248–55. Austin, TX, USA: AAAI Press.

Zhuo, H. H., H. Muñoz-Avila, and Q. Yang. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artificial Intelligence* 212:134–57. doi: 10.1016/j.artint.2014.04.003.

Appendix A Statistics

Table A1. Statistics on amounts of nodes and edges of the datasets in the experiments.

		All			Nodes Task			Data			All			Edges control-flow			Dataflow		
Domain	Dataset	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
	Original	25.7	18.0	39.0	11.6	8.0	17.0	12.2	9.0	18.0	59.3	40.0	94.0	11.9	7.0	24.0	22.7	16.0	34.0
	Strat. 1	25.4	17.0	39.0	11.6	8.0	17.0	12.0	8.0	18.0	58.1	37.0	94.0	11.4	5.0	24.0	22.2	14.0	34.0
	Strat. 2	25.7	18.0	39.0	11.6	8.0	17.0	12.2	9.0	18.0	59.3	40.0	94.0	11.9	7.0	24.0	22.7	16.0	34.0
50	**Strat. 3.1**	25.6	12.0	39.0	11.7	5.0	18.0	12.3	6.0	18.0	58.9	25.0	94.0	11.6	4.0	24.0	22.7	10.0	34.0
	Strat. 3.2	30.0	18.0	58.0	14.3	8.0	27.0	14.1	9.0	24.0	69.5	40.0	140.0	14.2	7.0	35.0	26.3	16.0	48.0
	Strat. 4	25.5	17.0	39.0	11.6	8.0	17.0	12.1	8.0	18.0	58.6	37.0	94.0	11.6	5.0	24.0	22.4	14.0	34.0
	Strat. 5.1	25.5	12.0	39.0	11.7	5.0	18.0	12.1	6.0	18.0	58.6	25.0	94.0	11.6	4.0	24.0	22.5	10.0	34.0
	Strat. 5.2	27.3	17.0	58.0	12.7	8.0	27.0	12.8	8.0	24.0	62.9	37.0	140.0	12.6	5.0	35.0	24.0	14.0	48.0
	Original	26.2	18.0	39.0	11.8	8.0	18.0	12.4	9.0	19.0	60.6	40.0	94.0	12.4	7.0	24.0	23.0	16.0	36.0
	Strat. 1	26.0	17.0	39.0	11.8	8.0	18.0	12.1	8.0	19.0	59.4	37.0	94.0	11.9	5.0	24.0	22.6	14.0	36.0
	Strat. 2	26.2	18.0	39.0	11.8	8.0	18.0	12.4	9.0	19.0	60.6	40.0	94.0	12.4	7.0	24.0	23.0	16.0	36.0
100	**Strat. 3.1**	26.2	12.0	39.0	11.9	5.0	18.0	12.4	6.0	19.0	60.4	25.0	94.0	12.1	4.0	24.0	23.1	10.0	36.0
	Strat. 3.2	31.0	18.0	60.0	14.7	8.0	27.0	14.5	9.0	26.0	72.1	40.0	146.0	14.9	7.0	35.0	27.2	16.0	52.0
	Strat. 4	26.1	17.0	39.0	11.8	8.0	18.0	12.2	8.0	19.0	59.9	37.0	94.0	12.1	5.0	24.0	22.8	14.0	36.0
	Strat. 5.1	26.1	12.0	39.0	11.9	5.0	18.0	12.3	6.0	19.0	60.0	25.0	94.0	12.1	4.0	24.0	22.9	10.0	36.0
	Strat. 5.2	28.0	17.0	60.0	13.0	8.0	27.0	13.1	8.0	26.0	64.7	37.0	146.0	13.2	5.0	35.0	24.5	14.0	52.0
	Original	26.0	15.0	43.0	11.8	7.0	18.0	12.3	7.0	19.0	60.1	32.0	104.0	12.3	6.0	26.0	22.9	12.0	36.0
	Strat. 1	25.8	14.0	43.0	11.8	7.0	18.0	12.0	6.0	19.0	58.9	29.0	104.0	11.8	4.0	26.0	22.4	10.0	36.0
	Strat. 2	26.0	15.0	43.0	11.8	7.0	18.0	12.3	7.0	19.0	60.1	32.0	104.0	12.3	6.0	26.0	22.9	12.0	36.0
200	**Strat. 3.1**	26.3	12.0	43.0	12.0	5.0	18.0	12.5	6.0	19.0	60.8	25.0	104.0	12.2	4.0	26.0	23.3	10.0	36.0
	Strat. 3.2	30.8	15.0	67.0	14.6	7.0	31.0	14.4	7.0	30.0	71.6	32.0	163.0	14.8	6.0	39.0	27.1	12.0	58.0
	Strat. 4	25.9	14.0	43.0	11.8	7.0	18.0	12.1	6.0	19.0	59.5	29.0	104.0	12.0	4.0	26.0	22.6	10.0	36.0
	Strat. 5.1	26.0	12.0	43.0	11.9	5.0	18.0	12.3	6.0	19.0	59.9	25.0	104.0	12.0	4.0	26.0	22.9	10.0	36.0
	Strat. 5.2	27.8	14.0	67.0	12.9	7.0	31.0	13.0	6.0	30.0	64.2	29.0	163.0	13.1	4.0	39.0	24.4	10.0	58.0
	Original	26.1	15.0	43.0	11.8	7.0	18.0	12.4	7.0	19.0	60.3	32.0	104.0	12.2	6.0	26.0	23.0	12.0	36.0
	Strat. 1	25.8	14.0	43.0	11.8	7.0	18.0	12.1	6.0	19.0	59.1	29.0	104.0	11.7	4.0	26.0	22.6	10.0	36.0
	Strat. 2	26.1	15.0	43.0	11.8	7.0	18.0	12.4	7.0	19.0	60.3	32.0	104.0	12.2	6.0	26.0	23.0	12.0	36.0
300	**Strat. 3.1**	26.3	12.0	43.0	12.0	5.0	18.0	12.5	6.0	19.0	60.9	25.0	104.0	12.2	4.0	26.0	23.3	10.0	36.0
	Strat. 3.2	31.2	15.0	67.0	14.9	7.0	34.0	14.6	7.0	32.0	72.7	32.0	163.0	15.0	6.0	39.0	27.5	12.0	62.0
	Strat. 4	25.9	14.0	43.0	11.8	7.0	18.0	12.2	6.0	19.0	59.6	29.0	104.0	11.9	4.0	26.0	22.8	10.0	36.0
	Strat. 5.1	26.1	12.0	43.0	11.9	5.0	18.0	12.3	6.0	19.0	60.0	25.0	104.0	12.0	4.0	26.0	22.9	10.0	36.0
	Strat. 5.2	28.0	14.0	67.0	13.1	7.0	34.0	13.2	6.0	32.0	64.7	29.0	163.0	13.1	4.0	39.0	24.6	10.0	62.0

Table A2. Kolmogorov-smirnov-statistics of the average number of nodes and edges of the datasets in the experiments.

	Attribute	Compared strategy						
Domain		1	2	3.1	3.2	4	5.1	5.2
	No. Nodes (avg.)	0.970	1.000	0.940	0.725	0.983	0.976	0.896
	No. Task Nodes (avg.)	1.000	1.000	0.940	0.000	1.000	0.976	0.008
50	**No. Data Nodes (avg.)**	0.945	1.000	0.955	0.020	0.969	0.970	0.020
	No. Edges (avg.)	0.920	1.000	0.940	0.125	0.954	0.966	0.050
	No. Control-Flow Edges (avg.)	0.925	1.000	0.940	0.140	0.957	0.968	0.140
	No. Dataflow Edges (avg.)	0.950	1.000	0.955	0.585	0.971	0.978	0.698
	No. Nodes (avg.)	0.970	1.000	0.943	0.745	0.983	0.968	0.899
	No. Task Nodes (avg.)	1.000	1.000	0.943	0.010	1.000	0.977	0.017
	No. Data Nodes (avg.)	0.943	1.000	0.958	0.020	0.967	0.970	0.020
	No. Edges (avg.)	0.935	1.000	0.940	0.153	0.963	0.952	0.061
	No. Control-Flow Edges (avg.)	0.920	1.000	0.943	0.170	0.954	0.966	0.177
100	**No. Dataflow Edges (avg.)**	0.955	1.000	0.958	0.565	0.974	0.972	0.703
	No. Nodes (avg.)	0.978	1.000	0.936	0.738	0.987	0.979	0.895
	No. Task Nodes (avg.)	1.000	1.000	0.936	0.014	1.000	0.975	0.023
	No. Data Nodes (avg.)	0.946	1.000	0.941	0.019	0.969	0.983	0.024
	No. Edges (avg.)	0.938	1.000	0.936	0.144	0.964	0.970	0.064
200	**No. Control-Flow Edges (avg.)**	0.923	1.000	0.936	0.169	0.956	0.977	0.174
	No. Dataflow Edges (avg.)	0.958	1.000	0.944	0.556	0.976	0.978	0.706
	No. Nodes (avg.)	0.975	1.000	0.941	0.715	0.986	0.982	0.887
	No. Task Nodes (avg.)	1.000	1.000	0.941	0.018	1.000	0.976	0.027
	No. Data Nodes (avg.)	0.950	1.000	0.948	0.025	0.971	0.982	0.028
	No. Edges (avg.)	0.933	1.000	0.941	0.163	0.961	0.979	0.071
300	**No. Control-Flow Edges (avg.)**	0.923	1.000	0.941	0.152	0.956	0.977	0.161
	No. Dataflow Edges (avg.)	0.958	1.000	0.950	0.550	0.976	0.977	0.697