

Article

Generative Adversarial Framework with Composite Discriminator for Organization and Process Modelling—Smart City Cases

Nikolay Shilov ^{1,*} , Andrew Ponomarev ¹ , Dmitry Ryumin ¹  and Alexey Karpov ² 

¹ St. Petersburg Federal Research Center of the Russian Academy of Sciences (SPC RAS), 199178 St. Petersburg, Russia; ponomarev@iias.spb.su (A.P.); ryumin.d@iias.spb.su (D.R.)

² Research Center “Strong Artificial Intelligence in Industry”, ITMO University, 191002 St. Petersburg, Russia; karpov@iias.spb.su

* Correspondence: nick@iias.spb.su

Highlights

What are the main findings?

- We successfully apply generative AI techniques (in particular, generative adversarial modeling) to speed up and streamline the development of organizational structures and processes, which is demanded in volatile smart city environments.
- We demonstrate the developed idea of the original composite discriminator, taking advantage of separate evaluation of tacit and explicit domain knowledge on smart city related scenarios, namely logistics system model generation and smart tourist trip booking process model generation
- We show that the developed mechanism of generating and using meaningful augmentations enables learning on limited datasets.

What is the implication of the main finding?

- The developed approach offers an effective solution for the fast generation of organizational structures and process models, accounting for both tacit and explicit requirements and constraints.
- By automating the creation of organization structures and processes, the approach can contribute to the adaptability of smart city structures and processes and their fast adaptation to changing requirements.
- The approach can be applied to other domains and could be a starting point for research efforts in related scientific fields.



Academic Editors: Andrzej Skowron and Pierluigi Siano

Received: 11 January 2025

Revised: 18 February 2025

Accepted: 26 February 2025

Published: 28 February 2025

Citation: Shilov, N.; Ponomarev, A.; Ryumin, D.; Karpov, A. Generative Adversarial Framework with Composite Discriminator for Organization and Process Modelling—Smart City Cases. *Smart Cities* **2025**, *8*, 38. <https://doi.org/10.3390/smartcities8020038>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Abstract: Smart city operation assumes dynamic infrastructure in various aspects. However, organization and process modelling require domain expertise and significant efforts from modelers. As a result, such processes are still not well supported by IT systems and still mostly remain manual tasks. Today, machine learning technologies are capable of performing various tasks including those that have normally been associated with people; for example, tasks that require creativeness and expertise. Generative adversarial networks (GANs) are a good example of this phenomenon. This paper proposes an approach to generating organizational and process models using a GAN. The proposed GAN architecture takes into account both tacit expert knowledge encoded in the training set sample models and the symbolic knowledge (rules and algebraic constraints) that is an essential part of such models. It also pays separate attention to differentiable functional constraints, since learning those just from samples is not efficient. The approach is illustrated via examples of logistic system modelling and smart tourist trip booking process modelling. The developed framework is implemented in a publicly available open-source library that can potentially be used by developers of modelling software.

Keywords: organization modelling; process modelling; smart tourism; machine learning; generative adversarial network; parametrized GAN; parametrized typified graph

1. Introduction

The paradigm of smart cities assumes an increasing level of citizens' comfort, often achieved via the usage of modern information and communication technologies [1,2]. This is done in various aspects of human life: from sustainability and urban planning to mobility and services [3–6]. Solutions supporting smart city operation do not only improve the citizens' experience but also require faster and more efficient decision making in modeling, planning, and management [7,8].

The task of organization and process modelling is complicated due to the variety of interconnections influencing each other, the presence of numerous functional dependencies, and by the presence of the human factor, which causes the emergence of hardly specifiable social connections, which need to be taken into account during modelling [9–11]. It also requires domain expertise and significant efforts from modelers. As a result, such processes are still not well supported by IT systems and still mostly remain manual tasks.

It is not always reasonable or even possible to solve this task by using exclusively analytical approaches due to the high variety of connections. The same applies to the application of exclusively machine learning (ML) models trained using the previous experience of experts due to the low efficiency of identifying functional dependencies from examples. Thus, for their effective modelling, it is necessary to ensure (a) model representation, taking into account the variety of relations and the complexity of their topology, and (b) accounting for factors of different mathematical nature, including the experience of experts as well as strict constraints imposed on various parameters, described in the form of equations or table functions. The aim of this paper is to solve these tasks.

To solve task (a), this paper contributes by considering the modelled systems from the point of view of "System of Systems (SoS) Engineering" as multi-level and requiring configuration both at the level of the entire system and at the level of its components. Moreover, it applies the representation of the modelled system in the form of multilevel graphs with typified nodes and edges, both at the level of the entire system and at the level of the components.

One of the powerful techniques for approximating artifacts using (typically) deep learning is generative adversarial modeling. According to this technique, a pair of models is created, such that one model (the generator) tries to learn patterns in the training set samples and apply those to the input noise and the other one learns to distinguish generated samples from those in the training set.

To solve problem (b), this paper contributes by proposing an approach based on the development of a composite discriminator (additionally taking into account symbolic rules) for training a generative neural network [12–17].

The scientific novelty of the presented approach is threefold. First, we propose to apply generative AI techniques (in particular, generative adversarial modeling), taking into account both tacit and explicit knowledge to speed up and streamline the development of organizational structures and processes, which is demanded in volatile smart city environments. Second, we introduce the idea of the original composite discriminator taking advantage of a separate evaluation of the similarity of the generated sample to the training ones, meeting structural and algebraic constraints and meeting differentiable constraints, which improves the learning speed of the generator. Third, the proposed mechanism

is capable of generating and using meaningful augmentations that enable learning on limited datasets.

The reminder of this paper is structured as follows. Section 2 presents the state of the art in the areas of decision support for modelling and generative adversarial networks. The approach is outlined in Section 3, followed by the presentation of the algorithms used (Section 4), experimentation results (Section 5), and details of the programming library implementation (Section 6). The limitations of the proposed approach are discussed in the Discussion section (Section 7). The main results are summarized in the Conclusions (Section 8).

2. Related Works

2.1. Decision Support in Organization and Process Modelling

Today, decision support in the area of organization and process modelling is in demand, since in addition to the creative component, such process includes tasks related to the analysis of existing solutions for possible reuse, which can be very time-consuming. Existing research aimed at assisting the modeler is at the intersection of recommendation systems (DC) and conceptual design.

There are a limited number of methods and prototypes designed to support the modeler's work. For example, there are approaches focused on the automatic completion of an unfinished model. These approaches were mainly developed in the context of business process modeling and focus on various helper functions, such as facilitating the completion of a partially built model by, for example, presenting relevant fragments of already existing models [18], using pattern-based knowledge for completion [19], or other autofill mechanisms [20] that function by adding the necessary information to implement the model [21] or provide assistance in applying modeling syntax [22].

Another form of recommendation-based support is suggestions for specific modeling actions, also called auto-completion [23]. Work in this area, based on the artificial intelligence paradigm, is currently regarded as very promising [24].

Finally, there are approaches designed to improve the quality of the model [25] or providing information support in a specific subject area in the case of modelling, when certain constraints must be met [26] or when support is required for model changes.

Recently, the introduction of machine learning, for example for parameter determination within a recommendation system, has also been discussed as a future area of research [26].

The idea of using machine learning (which is basically looking for implicit patterns) to support modeling tasks came from using manual or semi-automated template libraries. For example, in [27], the usage of templates is considered an important feature of enterprise modeling tools in today's era of digitalization. Awadid et al. [28] propose a complex system of organizing patterns with an assessment of their consistency, and Khider et al. [29]—a system of recommendations for specific patterns or known patterns. The effectiveness and prospects of using templates in enterprise modeling are also emphasized in [30]. The need to balance between flexibility/variety of patterns and the number of patterns to be analyzed manually is highlighted in [31], which proposes to expand the patterns to the concept of a set ("bag") of fragments (bag of fragments), which are compared with the current model. It can be said that this work serves as a kind of link between the use of templates and machine learning methods. To solve this problem, machine learning methods can be useful, allowing one to both identify patterns by analyzing and summarizing existing big data and to select and offer them to the user depending on the situation (generate suggestions to extend the model or generate an entire model).

All of the above approaches are mainly based on the use of predefined templates, which, on the one hand, makes the modeling process more reliable and predictable but, on the other hand, less flexible and creative. The use of generative design based on machine learning methods is intended to eliminate this limitation.

Today, there already exist machine learning models, including those based on deep neural networks, which are focused on working with network structures and, in particular, with graphs. Jangda et al. [32] propose a method for representing the graph in a form suitable for use in machine learning models and supporting parallelization of the training process for the efficient use of GPUs in order to increase the speed and performance of the training process. Valera et al. [33] use the random forest machine learning paradigm to analyze the main flows in the transport network, representing the latter in the form of a graph. Such models are often successfully used in chemistry today. Chen et al. [34] use the graph to represent chemical compounds to use machine learning techniques to generalize and predict their properties. A similar task has been solved in [35], but using machine learning methods. Nielsen et al. [36] apply machine learning to simulate processes with physical particles.

2.2. Application of Neural Networks to Organization and Process Modelling

One of the typical ways of representing organization or process models is using various network (graph) structures (most modeling techniques are based on using a number of connectivity diagrams—e.g., BPMN and IDEF3). Nodes in these network structures correspond to elements of the model (depending on the type of the model), while connections reflect connections between the elements (dependencies, material and information flows, etc.) There are a number of papers in which Generative Adversarial Networks (GANs) are adapted to work with graph data (and, therefore, can potentially be applied to organization and process models).

A large category of network (graph) GANs are designed for processing one (typically large) graph—e.g., a knowledge graph or user–item relations graph. These approaches, also classified as graph representation, learn and model the local structure of the graph and can be used for link prediction [37–39].

Another category of network (graph) GANs are aimed at the generation of whole graphs. These GANs are trained on a set of graphs from a certain domain and learn to generate new graphs from the same domain [40–43]. Inherently, this setting is closer to the problem considered in this paper. Typically, the generator and discriminator of these GANs are based on graph neural networks (GCN [44], GAT [45], etc.). The idea of conditional GANs (generative models that can accept additional condition/context parameters) has also been adapted to graph GANs [41].

There are also several especially relevant papers where the “classical” GAN structure has been augmented by an additional block aimed at accounting for additional domain-specific requirements. In [46], MolGAN has been proposed—a graph GAN for generating small molecules, so that the generated molecules are evaluated according to several properties. This evaluation is done using special software; however, MolGAN contains an additional approximator neural network (along with the generator and the discriminator) which is also trained in order to create a differentiable representation of the evaluation procedure and streamline the training of the generator while accounting for the domain-specific constraints. The problem of molecule structure generation is also addressed in [47].

There are no existing GANs for organization or process models; however, the existing approaches for labeled graph generation and taking into account domain-specific constraints [46] can be adapted to this scenario.

3. Proposed Approach

We propose to consider a modelled process or system as multi-level and requiring configuration both at the level of the entire system and at the level of its components. Thus, within the framework, first the problem of modelling the whole system is solved, followed by modelling its components.

Since modelled systems or process often must meet a few strict requirements (explicit or parametric, that is, formalized and described in the form of equations or table functions), the use of classical generative algorithms is ineffective, since such dependencies are difficult to learn only from examples. The use of evolutionary or swarm algorithms is also ineffective, since the “hit” of values in the required ones is very unlikely.

On the other hand, the use of only methods based, for example, on the technology of constraint programming will also not be effective since they cannot take into account the non-formalizable knowledge of expert modelers.

Thus, within the framework of this concept, the use of a composite discriminator is proposed, the principle of operation of which is presented in Figure 1 on an example of smart logistics system modelling. A composite discriminator combines methods for checking both parametric patterns (analytically) and tacit knowledge of experts using the classic generative–adversarial pair. Below, some examples of these are presented.

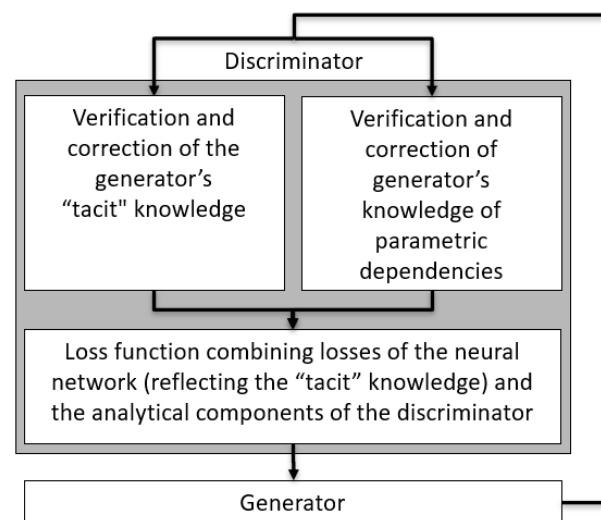


Figure 1. The principle of the composite discriminator operation.

Examples of parametric constraints (that can be fuzzy):

- functional constraint on the number of vehicles for a given volume of deliveries (V): $N_v = f(V)$;
- functional fuzzy constraint on the number of employees for a given supply volume (V): $N_d(\delta) = g(V)$;
- functional fuzzy constraint of the influence of the leader’s leadership qualities (Q_i) on the employee productivity: $N_d(\delta) = g(V, Q_i)$.

Examples of revealed tacit rules:

- if there is a possibility of a stable social connection, it is reasonable to provide additional control;
- the logistics department usually includes roles of the head of logistics department and driver, as well as the resource vehicle;
- products as a rule require assembly.

The main problem in this scheme is the formulation of the error function that combines the estimates of the neural network and analytical components of the discriminator, since it must convey the degree of compliance of the evaluated generated solution with all constraints. Thus, the formulation of an algorithm for creating and training a composite discriminator that combines the properties of neural network and analytical approaches is a novel scientific result.

An illustration of a generated model is shown in Figure 2. Below, the considered initial data for it are presented:

- possible product configurations;
- the volume of product deliveries per month ($V = 500$);
- functional dependencies for calculating the number of vehicles (N_v) and drivers (N_d);
- matrices of candidates' characteristics.

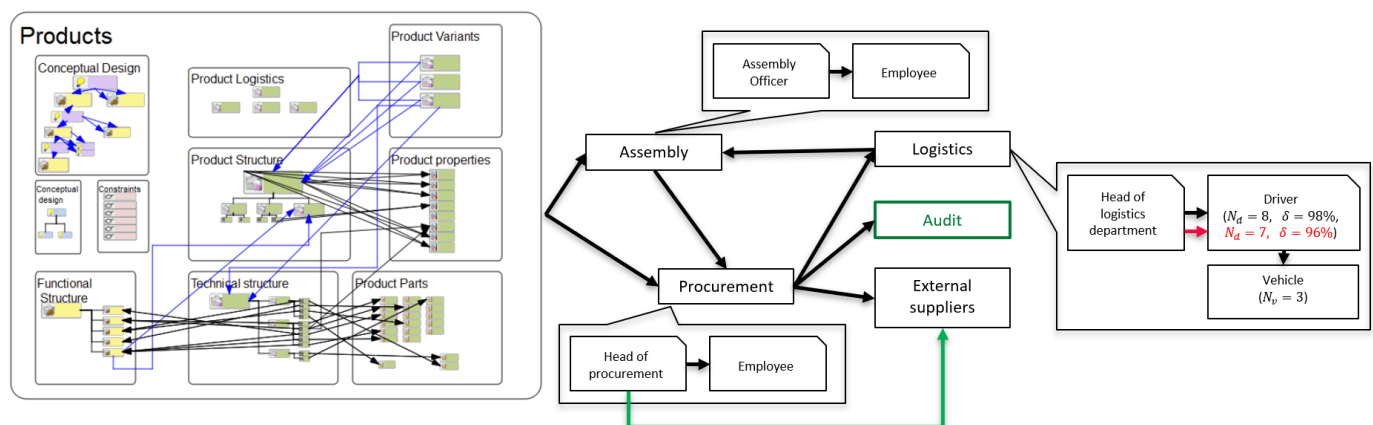


Figure 2. A sample model of smart logistics system modelling.

The black arrows in Figure 2 denote formal relationships (information and material flows, hierarchy, etc.). The red arrow and equation reflect the fact that the presence of high leadership qualities of a candidate for the position of head of the logistics department may reduce the number of required drivers, but at the same time the probability of meeting the initial requirements (volume of deliveries) will slightly decrease. The system can also identify a potential social connection between the head of the procurement department and a potential supplier (the green arrow), and to level it, an additional control element “Audit” (shown in green) is added.

4. Algorithms Used

The developed approach is based on the following three algorithms developed by the authors:

1. An algorithm of training a generative neural network model with elements of autonomous self-learning through the formation of augmented training data that considers both the experience of experts and parametric patterns (ATGM);
2. An algorithm or creation and training a composite discriminator that combines the properties of neural and analytical approaches (ATCD);
3. An algorithm of multi-level model generation with a complex topology of connections at the level of the system as a whole and at the level of its components (AMGS).

4.1. The ATGM Algorithm

The ATGM algorithm is based on the following mathematical relations. The inputs to the algorithm are a set of training examples of models (X^T), a validator of constraints on

the structure and configuration parameters of the models (Φ), and a method for estimating valid input parameters based on known configuration parameters ($\bar{\Phi}$) $X^T = \{x_i^T\}_{i=1}^t$, where t is the size of the training set and any element x_i^T is a tuple (N, C, P, Q) . The elements of the tuple are as follows:

$N \in \{0, 1, \dots, m\}^n$ is the vector of the elements of the model, where each element corresponds to an element type (m is the number of element types) and their number (n is the maximum allowable number of elements);

$C = (C_i)_1^c, \forall i : C_i \in \{0, 1\}^{n \times n}$ are matrices of relationships (c is the number of types of relationships) between the elements of the model;

$P = \mathbb{R}^{n \times p}$ is a matrix of parameter values of the model's elements (p is the maximum number of parameters for one element);

$Q \in \mathbb{R}^q$ is the vector of "input" parameters (q is the number parameters);

X^T is a set of predefined models (the training set), each element x_i^T of which has a label indicating its validity $b_i^T \in B^T$, $b_i^T \in \{0, 1\}$ (0—invalid model and 1—valid model).

Initial data augmentation is performed as follows:

$$C'_{ijk} = \begin{cases} (C_{ijk} + \varepsilon_C) \bmod 2, N_{ij} \neq 0, N_{ik} \neq 0, \\ 0, \text{otherwise} \end{cases}, \text{ where } \varepsilon_C = \text{Bernoulli}(p_i) \quad (1)$$

$$P'_{ij} = P_{ij} + \varepsilon_P, \text{ where } \varepsilon_P = N(0, \sigma_j) \quad (2)$$

To ensure the validity of the augmented models created using the above relations, the feasible input parameters are first selected, $Q'_i = \bar{\Phi}(N, C'_i, P'_i)$, and then the resulting model $b_i = \Phi(x_i)$, $x_i = (N_i, C'_i, P'_i, Q'_i)$ is validated. If $b_i = \text{TRUE}$, the model created through the augmentation is used during the training. For complex tasks, augmentation based on Formulas (1) and (2) can affect only key parameters, which then are used to define the remaining parameters of the model and possible input parameters (as indicated in Figure 3). Definition of key parameters and their dependencies with other parameters are specific for each model.

To ensure that training models received from experts prevail over those created through augmentation, models should be augmented anew at each training epoch.

The training of the generative–adversarial model is carried out in the classical way [48–51], except for the method of evaluating the generated model and training the discriminator, which are performed according to the ATCD algorithm.

Figure 3 shows a block diagram of the ATGM algorithm. It starts by reading the training dataset. Then, a training cycle is performed, the criterion for the end of which is the achievement of the required accuracy by the generator (exit from the cycle according to the condition "Has the required accuracy been achieved?"). One such cycle of learning corresponds to one epoch.

The next nested loop is the generation of augmented data, which ends with the condition "Enough augmented data?". As part of this cycle, the following operations are performed:

A random model is selected from the training dataset and augmented according to the ratios mentioned above: a set is formed. Then, its uniqueness is checked. If the resulting model is unique, the input parameters are selected and then validated. Selection of the input parameters is necessary, since the probability of creating a valid model randomly can be very low. If the resulting model is valid, it is saved.

Once a sufficient number of augmented models have been accumulated, a learning cycle (epoch) is performed using the AOCD algorithm.

The second AOCD algorithm is aimed at creating a mechanism that combines the classical (neural network) component to take into account the tacit knowledge of experts

embedded in training models with the parametric component to take into account the patterns specified analytically.

This algorithm itself does not implement the functions of strong AI, but it is necessary to ensure the operation of the AOGM algorithm, although it can be used outside of it. This need is caused by the fact that structured objects (such as considered organization and process models) must satisfy both the non-formalizable/poorly formalized knowledge of expert designers and a number of strict requirements (explicit or parametric, that is, formalized and described in the form of equations or table functions), for which the use of classical generative algorithms is ineffective. The composite discriminator combines methods for testing both the tacit knowledge of experts using the classical (neural network) component, and parametric patterns (analytically). The proposed formulation of the error function combines the estimates of the neural network and analytical components of the discriminator, conveying the degree of compliance of the evaluated generated solution with all constraints.

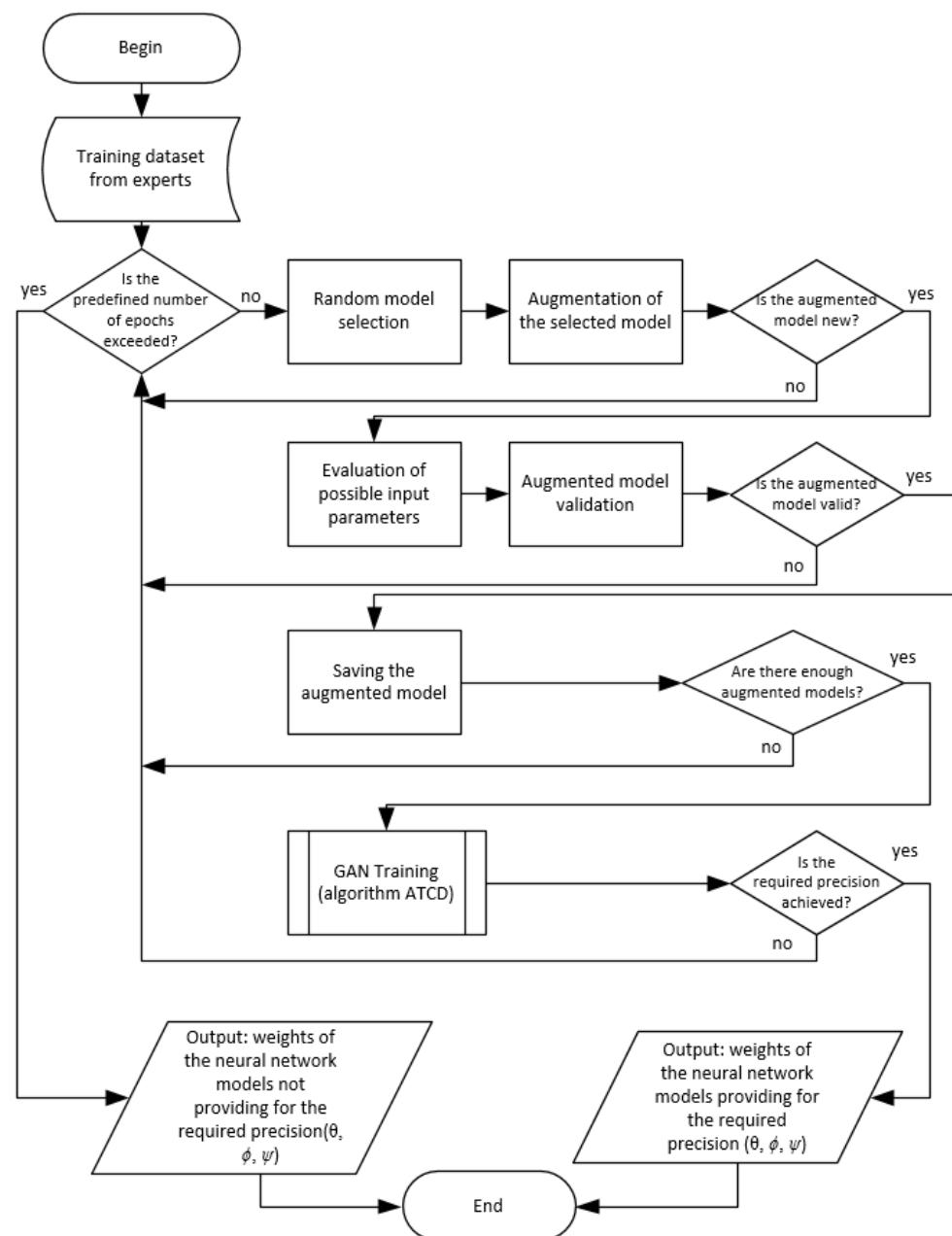


Figure 3. Block diagram of the ATGM algorithm.

4.2. The ATCD Algorithm

The ATCD algorithm is based on the on the approach described in [46], which has been adopted and extended. It uses the following mathematical relations:

The model $x = (N, C, P, Q)$, and the training set X^T have been described above;

G_θ —generator;

D_ϕ —discriminator;

R_ψ —neural network-based approximator of constraints on the model's structure and parameters;

Φ —constraint validator (defined above);

Φ' —validator of differentiable constraints.

The training process is based on the following relations:

$\tilde{x} = G_\theta(z, Q)$ is a probabilistic estimation of the model ($z \in \mathbb{R}^d$ is a vector of random values with a normal distribution from 0 to 1 of the length d , a “noise”).

$x^G = f(\tilde{x}) = (N^G, C^G, P^G, Q^G)$ is the model obtained on the basis of a probabilistic estimation of the model $\tilde{x}$. Differentiable discretization (replacement of probabilistic parameters with binary or integer parameters) is performed in one of three ways (addition of Humbel noise, usage of Gumbel–Softmax, or no discretization usage) described and implemented in [46].

With the help of the validator, each generated model x^G is assigned a label $b^G = \Phi(x^G)$, $b^G \in \{0, 1\}$, $B^G = \{b^G\}$, indicating its validity (0—invalid model and 1—valid model).

To train the neural network-based approximator of constraints on structure and parameters, both the generated models $(\tilde{X}^G, B^G)$, $\tilde{X}^G = \{\tilde{x}^G\}$ and training models (X^T, B^T) are used. The error l^R is calculated using the following formulas: $l^R = \lambda_R^R \cdot l^{RG} + \lambda_R^\Phi \cdot l^{\Phi'G}$ for generated data and $l^R = \lambda_R^R \cdot l^{RT} + \lambda_R^\Phi \cdot l^{\Phi'T}$ for training data, where $l^{RG} = ME(R_\psi(X^G), B^G)$, $l^{RT} = ME(R_\psi(X^T), B^T)$, ME —mean error, $\lambda_R^R + \lambda_R^\Phi = 1$ —trade-off coefficients determining the significance of partial errors within the composite one, and $l^{\Phi'G} = \Phi'(P^G)$ —the error value of the validator of differentiable constraints.

Discriminator training is performed in a classic way. The error l^D is calculated using the following formulas: $l^{DG} = ME(D_\phi(X^G), 0)$, $l^{DT} = ME(D_\phi(X^T), 1)$.

The composite error is used to train the generator $l^G = \lambda_G^R \cdot l^{RG} + \lambda_G^D \cdot l^{DG} + \lambda_G^\Phi \cdot l^{\Phi'G}$, where $\lambda_G^R + \lambda_G^D + \lambda_G^\Phi = 1$ —trade-off coefficients.

Thus, when training a generator, the error l^{RG} provides learning constraints on structure and non-differentiable constraints on parameters, the error $l^{\Phi'G}$ —learning differentiable constraints on parameters, and the error l^{DG} —learning the variety of generated models.

Figure 4 shows a block diagram of the ATCD algorithm. The algorithm begins with reading the initial data (the training set X^T), as well as generating a given (determined by the number of training elements within one epoch) number of models by the generator and their subsequent discretization. The resulting models are evaluated by the discriminator and the error l^{DG} is computed (the generated models are considered as fake ones).

At the same time, the validator validates the generated models, which results in the TRUE/FALSE label and the error $l^{\Phi'}$ (the result of the work of the validator of differentiable constraints).

The training dataset is evaluated by the discriminator (the models are considered real) to calculate the error value l^{DT} .

In parallel, both the training and validated generated models are evaluated by the neural network approximator, producing the error values l^{RG} and l^{RT} .

The calculated error values are used to update the weights of the generator (l^{RG} , l^{DG} , $l^{\Phi'G}$), neural network approximator (l^{RG} , l^{RT} , $l^{\Phi'G}$, $l^{\Phi'T}$), and discriminator (l^{DG} , l^{DT}).

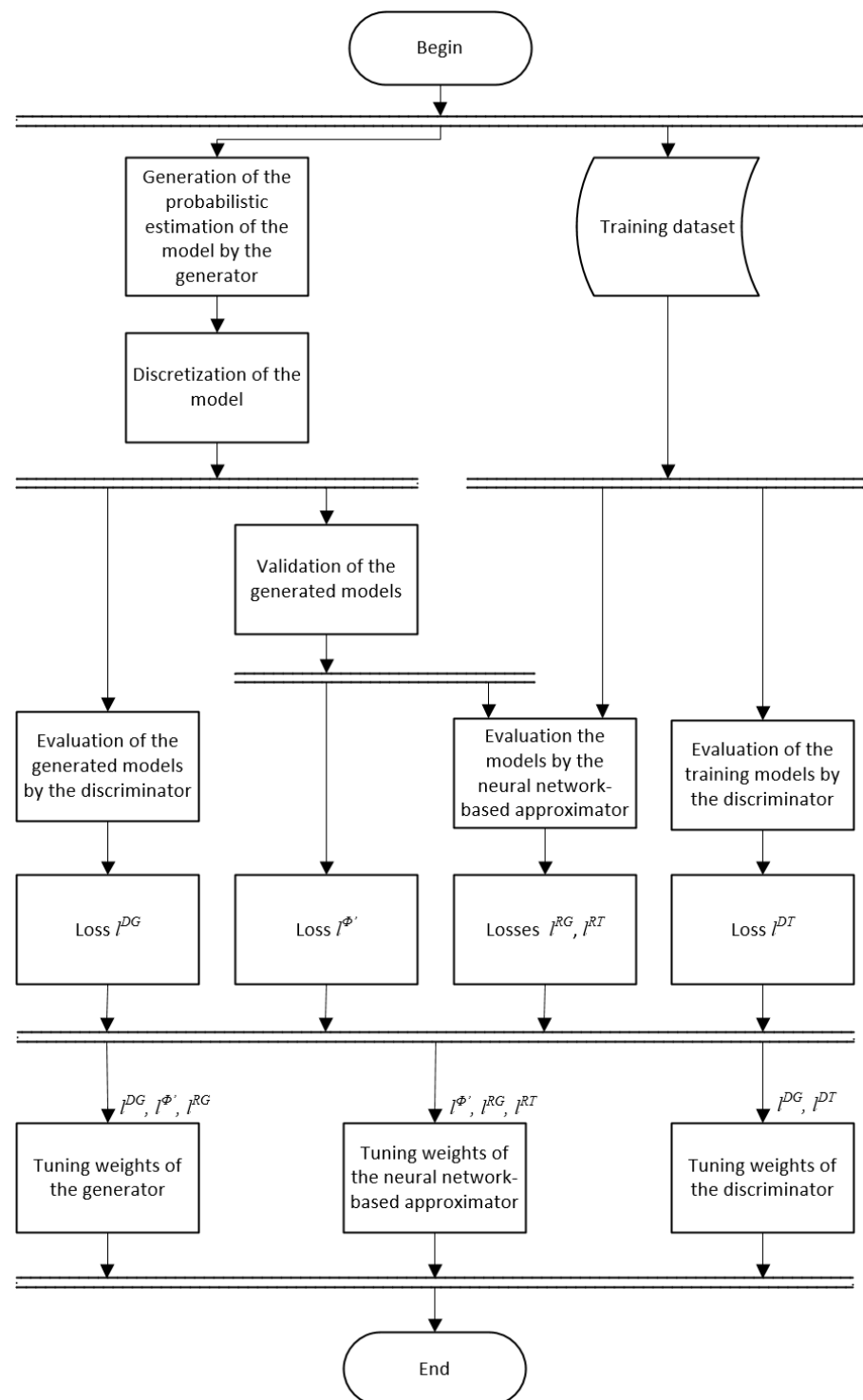


Figure 4. Block diagram of the ATCD algorithm.

4.3. The ATGM Algorithm for Complementing Existing Models

The ATGM algorithm can also be used to complement partially specified models (in addition to generating models from scratch). This functionality is useful if the designer already has a partially defined model (for example, if some key elements have been defined

and the full model needs to be completed, or a model has been defined but some its elements need to be changed to meet the changed requirements).

For this purpose, the vector of input parameters of the elements of the training set $Q \in \mathbb{R}^q x_i^T = (N, C, P, Q) X^T = \{x_i^T\}_{i=1}^t$ is extended to a data structure similar to that describing the models, namely $\tilde{Q} = (N_Q, M_N, C_Q, M_C, P_Q, M_P, Q_Q, M_Q)$, where

$N_Q \in \{0, 1, \dots, m\}^n$ —vector of predefined elements of the model,

$M_N = (m_{Ni})_1^n, \forall i : m_{Ni} \in \{0, 1\}$ —a vector mask that defines the significant values of the vector N_Q (the values of the vector N_Q for which the matrix value is 1 are considered prespecified and the values for which the matrix value is equal 0 are to be ignored);

$C_Q = (C_i)_1^c, \forall i : C_i \in \{0, 1\}^{n \times n}$ —matrices of relationships between the elements of the model;

$M_C = (m_{Ci})_1^c, \forall i : m_{Ci} \in \{0, 1\}^{n \times n}$ —a set of matrices masks that determine the significant values of the matrices C_Q ;

$P_Q = \mathbb{R}^{n \times p}$ —matrix of parameter values of the model elements;

$M_P = \{m_P\}^{n \times p} = \{0, 1\}^{n \times p}$ —a matrix mask that determines the significant values of the matrix P_Q ;

$Q_Q \in \mathbb{R}^q$ —vector of “input” parameters;

$M_Q = \{m_q\} = \{0, 1\}^q$ is a vector mask that defines the significant values of the vector Q_Q .

The augmentation in turn is complemented by a mask generation step, which is then combined with models from the training set to create “incomplete model–complete model” training pairs. Mask elements are generated using the Bernoulli distribution: $m_{Ni} = \text{Bernoulli}(p_N), m_{Ci} = \text{Bernoulli}(p_C), m_P = \text{Bernoulli}(p_P)$, and $m_Q = \text{Bernoulli}(p_Q)$, where the probabilities p_N, p_C, p_P, p_Q are defined empirically.

The block diagram of the ATGM algorithm for model complementing is identical to that in Figure 3; however, after the augmentation generation an additional step of mask generation is introduced.

4.4. The AMGS Algorithm

The AMGS algorithm is aimed for multi-level model generation based on information transfer between levels (AMDS). It is based on the following mathematical relations.

The inputs to the algorithm are a set of generative neural networks $\{G_{i,\theta_i}\}_{i=1}^\Lambda$ trained using the ATCD and ATGM algorithms for the top level model generation and for generation of its components (aspects), a set of input parameter values for each aspect $\{\tilde{Q}_i\}_{i=1}^\Lambda$, a set of relationships between the parameters of aspects $\{\Gamma_{ij}\}_{i,j=1}^{\Lambda,\Lambda} : \tilde{Q}_j' = \Gamma_{ij}(x_i, \tilde{Q}_j)$, and the modelling sequence, which is a sequence of aspect indexes $\{\tau_i\}_{i=1}^\Lambda$. The result of the successful operation of the algorithm is a set of agreed aspect models $\{x_i\}_{i=1}^\Lambda$, where $x_i = G_{i,\theta_i}(\tilde{z}, \tilde{Q}_i')$.

The essence of the algorithm is the sequential generation of models of the system as a whole and its components using trained generative neural networks, with the transfer of parameters (requirements) from more general models to more specific ones. If it is not possible to generate a model that satisfies the input parameters, it is possible to revert to the previous level according to the modelling sequence $\{\tau_i\}$ (“higher-level models”).

Figure 5 shows a block diagram of the AMGS algorithm. It begins with the generation of a batch of upper-level models (τ_1) based on the specified input parameters $\tilde{Q}_{\tau_1}$. Then, the lower level models (according to the sequence $\{\tau_i\}_{i=1}^\Lambda$) are generated using input parameters defined by the set of relationships between the aspect model parameters $\{\Gamma_{ij}\}_{i,j=1}^{\Lambda,\Lambda}$. If at any stage none of the generated models meet the requirements, another model from the previous stage is selected according to the modelling sequence. If there are no such

models left, the process returns to the stage that was even earlier according to the modelling sequence. This process is repeated until all models of the first aspect are exhausted. Thus, the result of the algorithm is either a set of agreed aspect models $\{x_i\}_{i=1}^{\Lambda}$ or the absence of the ability to generate those according to the specified conditions.

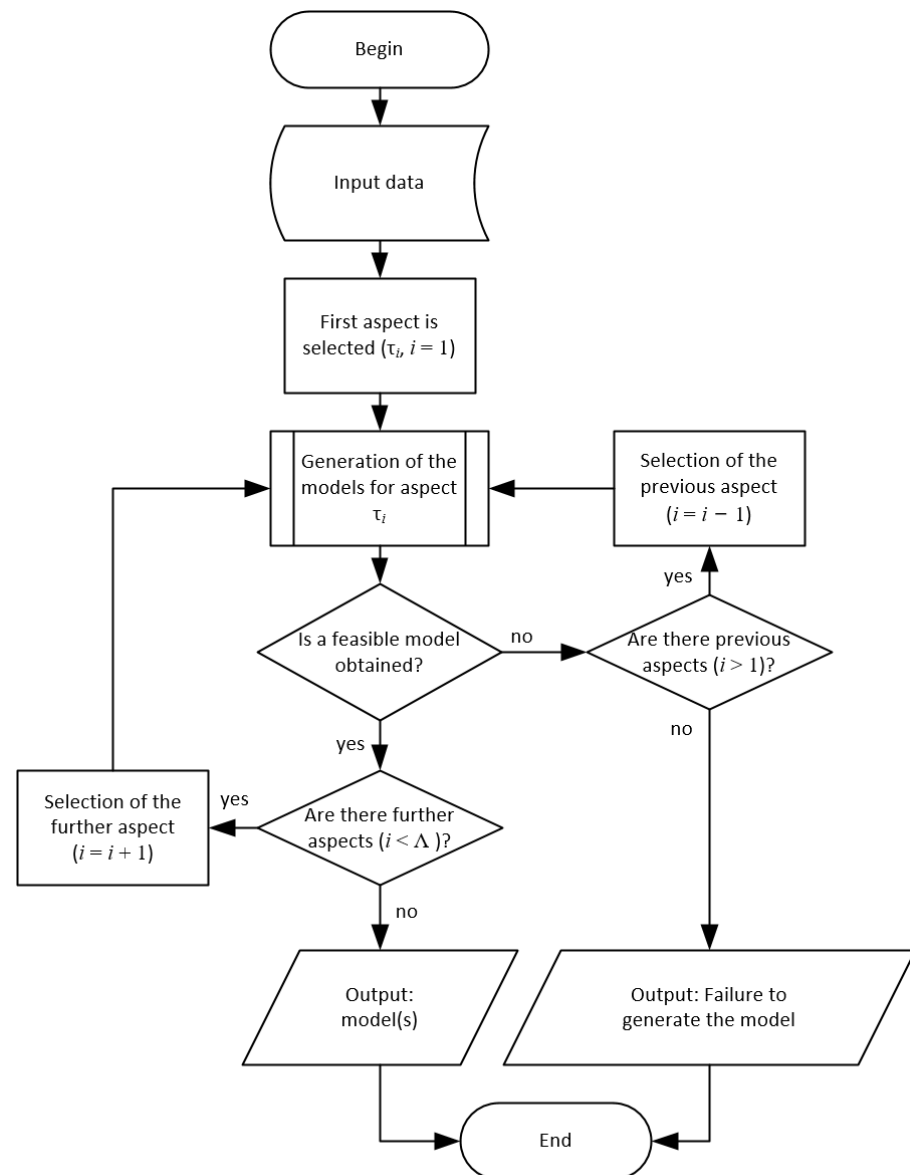


Figure 5. Block diagram of the AMGS algorithm.

5. Experiments

The experimental evaluation of the proposed approach is based on two applications. The first one is from the area of smart logistics and the second one is from the area of smart tourism. The proposed approach is explored in three ways. First, the experiments aim to answer the question of if this approach can actually be used to infuse business constraints (typically, non-differentiable) into a generative model so that the structures generated by a neural network actually conform to these constraints. Second, they aim to analyze the effect of the most important parameters of the approach. Finally, third, they aim to evaluate the usability of the approach.

5.1. Smart Logistics System Modelling Use Case

5.1.1. Smart Logistics System Modelling Dataset

To evaluate the approach, we developed a semi-synthetic dataset of logistics models. The model of a logistic system consists of several possible units, links between them (two types—hierarchical and information flow), and numeric parameters of the units (workload and personnel quantity). In addition to explicit links between units, there are also implicit functional relationships. These relationships are generally “soft” (for example, it is impossible to unambiguously determine the required number of personnel); however, there are constraints that allow assessing the feasibility of a specific set of parameters and the presence/absence of certain units.

The dataset consists of

- 20 logistics system models (note the small number of these models: the size of the datasets models the typical situation when there are only a few samples available for training a generative model);
- a set of business rules and constraints, reflecting the relationships between parameters of units (e.g., possible relationship between a unit’s workload and its personnel).

These two components (samples and rule specification) encode two types of knowledge related to logistics system model—implicit and explicit (symbolic), respectively.

This dataset was used to train a generative–adversarial model for recommending valid logistics system models according to the given criteria for a given set of input parameters (load of certain departments).

5.1.2. Quality Metrics

The main purpose of the generative model is to form valid models (including structural and parametric parts). Validness in this context is defined by the set of problem-specific rules and constraints. Therefore, we measure success of the generative model as the ratio of generated components, conforming the requirements to the total number of generated components: $Acc_R(X) = \frac{|\{x|R(x), x \in X\}|}{|X|}$, where X —is the set of generated models and $R(\cdot)$ is a predicate describing problem-specific constraints. We call this metric constraint satisfaction accuracy, or simply “accuracy” for brevity.

We performed a series of experiments to select optimal training parameter values (see below), then trained 10 generative models with these values. Finally, using each model, we generated 100 structures, yielding a constraint satisfaction accuracy mean of 0.87 and standard deviation of 0.04.

5.1.3. Model Parameters

The convergence rate and accuracy of the resulting model are determined by the complexity of the neural networks (generator, discriminator, and approximator). For the generator, it is the dimensionality of the fully-connected layers, and for the discriminator and the approximator (sharing the same structure) it the dimensionality of the graph convolution layers and fully-connected classification head. Increasing these dimensionalities can result in increased accuracy; however, it also increases the generation time (both iteration time and the number of iterations).

Another parameter influencing the convergence rate (and slightly influencing the accuracy) is batch size. Figure 6 shows learning curves with different batch sizes. It can be seen that with smaller batches (16 samples) the training is slower, but the quality of the generated structures is more stable; with larger batches (500 samples) the generator reaches high accuracy relatively quickly, but the quality is less stable and it frequently drops to near-zero values.

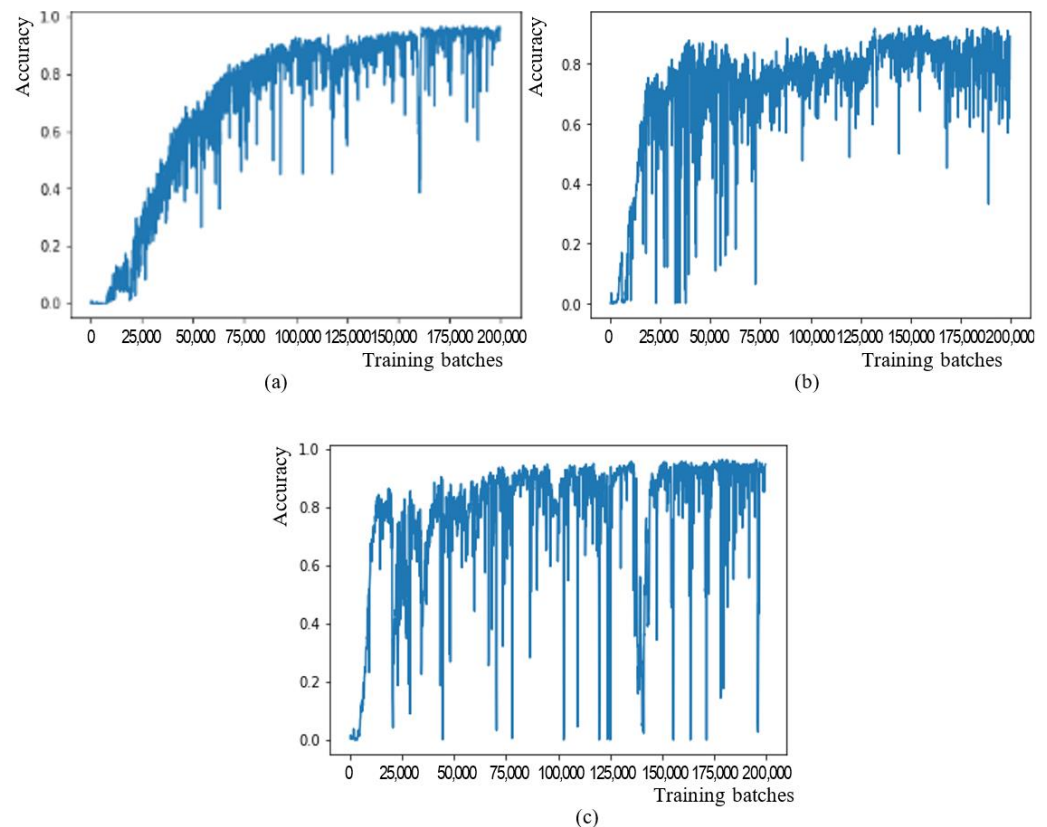


Figure 6. Accuracy for varying batch sizes: (a) 16, (b) 200, and (c) 500.

5.2. Smart Tourist Trip Booking Process Use Case

5.2.1. Smart Tourist Trip Booking Process Dataset

Smart tourist trip booking process can involve various operations depending on the client's preferences. To create a training dataset, process models from the SAP-SAM dataset (<https://github.com/signavio/sap-sam>, accessed on 26 February 2025) were utilized. The SAP-SAM dataset (SAP Signavio Academic Models) contains several hundred thousand models in JSON format. Primarily, these are process models in BPMN notation, created over approximately ten years by researchers, educators, and students using a free software platform.

To form the training set, the following operations were performed:

1. An analysis of model types within the dataset was conducted.
2. From the entire dataset, only BPMN models were selected. However, since the type is not defined for most models and models described in BPMN 2.0 notation have different types (e.g., "BPMN 2.0", "Business Process Diagram (BPMN 2.0)"), the presence of the keyword "bpmn2.0#" in the model text was chosen as the selection criterion. This resulted in the selection of 618,861 models.
3. In the final step, models written in English containing the phrase "Travel Booking" were filtered. The resulting set of 29 models was used as the training dataset.

5.2.2. Definition of Explicit Constraints on Process Models

According to the proposed algorithms, explicit (symbolic) constraints were defined to accelerate the process of training the composite discriminator of the generative adversarial model. They include:

1. Each model must have start and end nodes.
2. Any node, except the start node, must have an incoming edge.

3. Any node, except the end node, must have an outgoing edge.
4. The start node must not have incoming edges.
5. The end node must not have outgoing edges.
6. If a “Transaction Start” node is present, the model must include both “Transaction Success” and “Transaction Failure” nodes.
7. If a “Transaction Start” node is absent, the model should not include “Transaction Success” or “Transaction Failure” nodes.

These constraints are processed by a constraint validator, which allows for more precise determination of the gradient during the training of the neural network generator. This enhances the convergence of the training process and, therefore, reduces its duration.

5.2.3. Training Process

The listing of intermediate training results is presented in Listing 1. It includes the following results:

Listing 1. The listing of intermediate training results of the generative adversarial model.

```

The number of model parameters: 57,329
Start training...
Elapsed [0:00:02], Iteration [100/5000], D/loss_real: 0.0198, D/loss_fake: 0.0145,
V/loss: 0.1778, G/loss_fake: 4.2388, G/loss_value: 0.2441, G/loss_compliance: 0.1807,
Accuracy: 1.0000
Elapsed [0:00:05], Iteration [200/5000], D/loss_real: 0.0063, D/loss_fake: 0.0075,
V/loss: 0.0549, G/loss_fake: 5.0640, G/loss_value: 0.1221, G/loss_compliance: 0.2261,
Accuracy: 1.0000
Elapsed [0:00:07], Iteration [300/5000], D/loss_real: 0.0019, D/loss_fake: 0.0033,
V/loss: 0.0211, G/loss_fake: 5.7244, G/loss_value: 0.0946, G/loss_compliance: 0.1965,
Accuracy: 1.0000
Elapsed [0:00:10], Iteration [400/5000], D/loss_real: 0.0007, D/loss_fake: 0.0014,
V/loss: 0.0096, G/loss_fake: 6.6027, G/loss_value: 0.0664, G/loss_compliance: 0.2728,
Accuracy: 1.0000
Elapsed [0:00:12], Iteration [500/5000], D/loss_real: 0.0003, D/loss_fake: 0.0007,
V/loss: 0.7485, G/loss_fake: 7.2087, G/loss_value: 0.0911, G/loss_compliance: 0.2408,
Accuracy: 0.1111
Saved model checkpoints into output/data sap process models...

```

- Iteration: The iteration number and the total number of specified training iterations.
- D/loss_real: The value of the discriminator’s loss function when evaluating “real” samples (samples from the training set).
- D/loss_fake: The value of the discriminator’s loss function when evaluating “fake” samples (samples generated by the generator).
- V/loss: The value of the loss function of the constraint approximator.
- G/loss_fake: The value of the generator’s loss function as determined by the discriminator (indicating how much the generated samples differ from the real ones).
- G/loss_value: The value of the generator’s loss function as determined by the constraint approximator (indicating how well the generated samples conform to the specified constraints).
- G/loss_compliance: The value of the generator’s loss function obtained by comparing how well the generated samples adhere to the specified initial parameters.
- Accuracy: The “accuracy” of the generated samples in terms of structural constraints (the admissibility of combinations of nodes and edges within the model).

From the listing, it can be observed that after just 400 iterations, the training results appear satisfactory—the generator produces process models with permissible structures and relatively low values for the loss functions assessing the conformity of the generated samples to the specified parameters. Subsequently, the accuracy value begins to decline, leading to the decision to terminate the training.

5.2.4. Process Model Generation Procedure

After the generative model is trained, the tourist trip booking process generation procedure consists of the following steps:

1. Specification of input parameters. At this step, the input data vector for the process model generator is formed. For example, one might specify the generation of booking process to include the task “Book flight” and not include the task “Book attraction”.
2. Generation of a set of process models. At this step, the trained generator generates a set of process models according to the input parameters.
3. Evaluation of generated process models. Since neural networks cannot guarantee precise results, after generating a certain number of models, the sampling procedure is applied: only those models that meet the specified parameters and input criteria are selected. If after the sampling procedure the number of generated process models is not sufficient, the generation process (Step 2) may be repeated.
4. Selection of the most appropriate process model. From the obtained set of models, the user selects the most suitable ones. For example, based on the initial condition specified in Step 1, the following model variants were generated (Listings 2–6).

Upon analyzing the generated process models, the user can choose the most appropriate option for the specific task. For instance, in a case where transactions and hotel booking operations are likely needed, the most suitable option would be variant 4, which includes these tasks as illustrated in Figure 7.

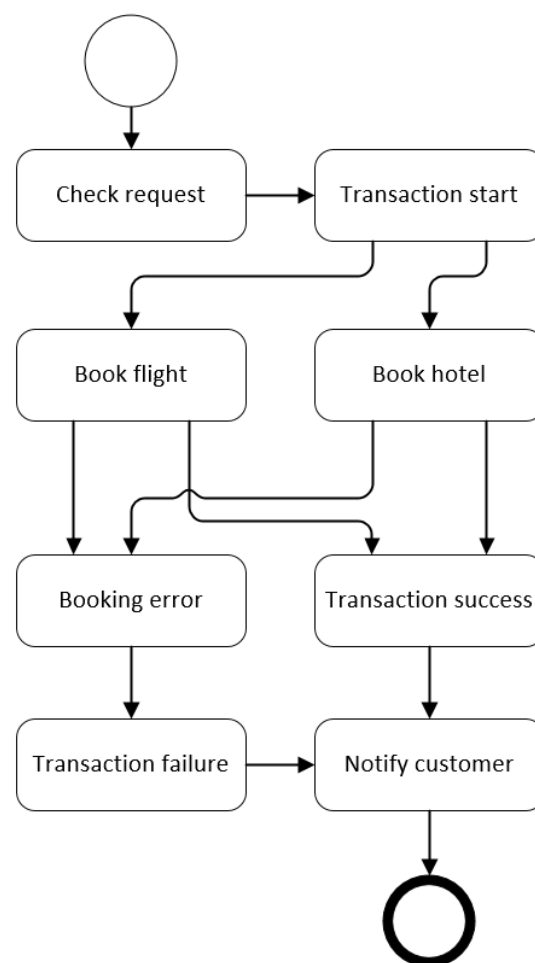


Figure 7. Illustration of the generated process model for smart tourist trip booking (variant 4).

Listing 2. Generated process model for smart tourist trip booking (variant 1).

```

Task 1: Check request
Task 6: Notify customer
Task 7: Success
Task 8: Booking error
Task 9: Book flight
Task 11: Book bus
Connection (1, 9) from Check request to Book flight
Connection (1, 11) from Check request to Book bus
Connection (7, 6) from Success to Notify customer
Connection (8, 6) from Booking error to Notify customer
Connection (9, 7) from Book flight to Success
Connection (9, 8) from Book flight to Booking error
Connection (11, 7) from Book bus to Success
Connection (11, 8) from Book bus to Booking error

```

Listing 3. Generated process model for smart tourist trip booking (variant 2).

```

Task 1: Check request
Task 2: Manual handling
Task 6: Notify customer
Task 8: Booking error
Task 9: Book flight
Task 11: Book bus
Connection (1, 2) from Check request to Manual handling
Connection (1, 9) from Check request to Book flight
Connection (1, 11) from Check request to Book bus
Connection (2, 6) from Manual handling to Notify customer
Connection (8, 6) from Booking error to Notify customer
Connection (9, 8) from Book flight to Booking error
Connection (11, 8) from Book bus to Booking error

```

Listing 4. Generated process model for smart tourist trip booking (variant 3).

```

Task 1: Check request
Task 3: Transaction start
Task 4: Transaction success
Task 5: Transaction failure
Task 6: Notify customer
Task 7: Success
Task 8: Booking error
Task 9: Book flight
Task 11: Book bus
Connection (1, 3) from Check request to Transaction start
Connection (1, 9) from Check request to Book flight
Connection (1, 11) from Check request to Book bus
Connection (3, 9) from Transaction start to Book flight
Connection (3, 11) from Transaction start to Book bus
Connection (4, 6) from Transaction success to Notify customer
Connection (5, 6) from Transaction failure to Notify customer
Connection (7, 4) from Success to Transaction success
Connection (7, 6) from Success to Notify customer
Connection (8, 5) from Booking error to Transaction failure
Connection (8, 6) from Booking error to Notify customer
Connection (9, 7) from Book flight to Success
Connection (9, 8) from Book flight to Booking error
Connection (11, 7) from Book bus to Success
Connection (11, 8) from Book bus to Booking error

```

Listing 5. Generated process model for smart tourist trip booking (variant 4).

```

Task 1: Check request
Task 3: Transaction start
Task 4: Transaction success
Task 5: Transaction failure
Task 6: Notify customer
Task 8: Booking error
Task 9: Book flight
Task 10: Book hotel
Connection (1, 3) from Check request to Transaction start
Connection (3, 9) from Transaction start to Book flight
Connection (3, 10) from Transaction start to Book hotel
Connection (4, 6) from Transaction success to Notify customer
Connection (5, 6) from Transaction failure to Notify customer
Connection (8, 5) from Booking error to Transaction failure
Connection (9, 6) from Book flight to Transaction success
Connection (10, 6) from Book flight to Transaction success
Connection (9, 8) from Book flight to Booking error
Connection (10, 8) from Book hotel to Booking error

```

Listing 6. Generated process model for smart tourist trip booking (variant 5).

```

Task 1: Check request
Task 2: Manual handling
Task 3: Transaction start
Task 4: Transaction success
Task 5: Transaction failure
Task 6: Notify customer
Task 8: Booking error
Task 9: Book flight
Task 10: Book hotel
Connection (1, 2) from Check request to Manual handling
Connection (1, 3) from Check request to Transaction start
Connection (2, 6) from Manual handling to Notify customer
Connection (3, 9) from Transaction start to Book flight
Connection (3, 10) from Transaction start to Book hotel
Connection (4, 6) from Transaction success to Notify customer
Connection (5, 6) from Transaction failure to Notify customer
Connection (8, 5) from Booking error to Transaction failure
Connection (8, 6) from Booking error to Notify customer
Connection (9, 8) from Book flight to Booking error
Connection (10, 8) from Book hotel to Booking error

```

6. Programming Library

We have developed and published an open-source programming library, OrGAN (<https://github.com/cais-lab/organ> accessed on 6 December 2024), implementing the proposed techniques. To train a generative–adversarial neural network model, the OrGAN library provides two interfaces: a command line interface and an application programming interface. The command line interface may be useful to a user who is satisfied with the basic architectures of neural networks (generator and discriminator) and the application programming interface provides richer options for configuring a generative–adversarial model. The library is based on the PyTorch framework (v. 1.8.1) and allows users to define their own architectures for the generator, discriminator, and validator, as well as implement specific non-differentiable verification logic for the considered class of organizational structures.

7. Discussion

The paper proposes a GAN training approach based on the idea of the original composite discriminator, taking advantage of separate evaluation of tacit and explicit domain knowledge on smart city-related scenarios, namely logistics system model generation and smart tourist trip booking process model generation. It provides a higher automation level of the modelers' work in the context of organization and process modelling.

The experiments prove the validity of this approach as an effective solution for the fast generation of organizational structures and process models, accounting for both tacit and explicit requirements and constraints. They demonstrate the approach on two scenarios related to smart cities; however, it can be potentially extended to other domains and serve as a starting point for research efforts in related scientific fields.

Unfortunately, there are no direct competitive models to compare the developed approach with. One of the closest models is MolGAN [46], but it does not address the generation of numerical parameters of graph nodes or the partial definition of generated graphs. A comparison of the GAN training process using the developed composite discriminator with a conventional GAN is presented in Figure 8. It is based on the training of a simplified Smart Logistics System Modelling Use Case with the application of only structural constraints. One can see that the application of the composite discriminator enabled a reduction of the incorrect generated models of 36.4%: the achieved accuracies after 22,000 batches for the composite discriminator and the conventional discriminator are 88.3% and 81.6%, respectively.

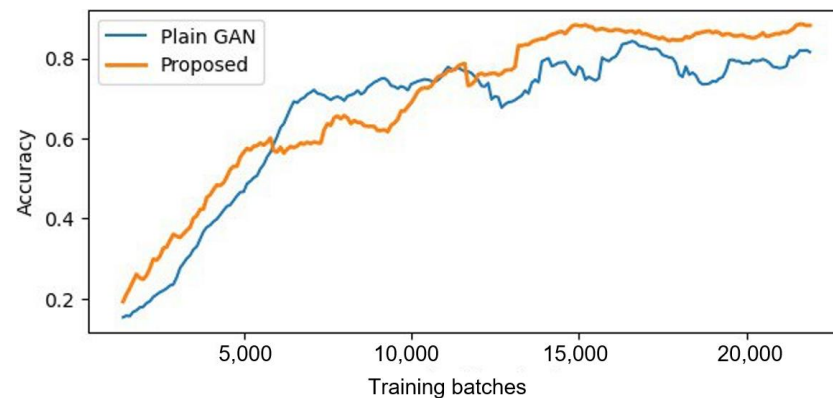


Figure 8. Comparison of the GAN training process using the developed composite discriminator with a conventional GAN.

The approach is still subject to several limitations discussed below.

1. Generating large-scale complex organizational systems can be a challenging task for GANs. The proposed approach mitigates this limitation in the following ways. First, the usage of the composite discriminator enables reducing the complexity of GAN training due to separate handling of explicit constraints and, especially, differentiable constraints, which makes it possible to simplify and speed up the training of GANs for such models. Second, the introduced the AMGS algorithm for the step-by-step detailing of the generated model makes it possible to generate several simpler models instead of one complex model. These means do not remove the limitation completely, but in many cases they do.
2. The definition of constraints for a particular model, as well as the augmentation generation procedure, is a complex and often iterative process. It might require numerous tests to achieve convergence of the GAN training. Further, depending on the model complexity, the structures and sizes of the neural networks used can require adjustment. We have not studied the dependency of the sizes of the GAN models on the complexity of the dataset models. In fact, the selection of the GAN models' sizes can be challenging due to the necessity to find the balance between the modelling graph complexity and the training set available. This issue is currently considered as one of the topics of future research.

3. One more limitation is related to the absence of a standard for describing datasets in the developed library. However, the library is still under development and since it is open-source we invite other developers to contribute.

Normally, an important metric for GANs is the diversity of the generated models. However, in this study we do not present such an analysis, since in the presented scenarios this is not the case. Good specification of the enterprise or process model being designed does not allow for a wide range of diverse models. In fact, what is demanded from the GAN is several similar (but different in few aspects) models that meet the given requirements, and the modeler can select one as the basis and introduce needed adjustments. The diversity can also be affected by the small training dataset (Section 5.2.1); however, as was mentioned in the paper, today there are no high-quality datasets for the considered domain (enterprise and process modelling), which was the main reason for developing the augmentation procedure presented in this paper. Nevertheless, one can observe the diversity of generated models in the process model scenario: for this reason, we presented five various generated models (Section 5.2.4, Listings 2–6).

Another important point in enterprise and process modelling is the adaptability of the models to changes in dynamic environments. The suggested framework does not support adjustment of the models; instead, the modeler can modify the input parameters for the model to generate a new one that would meet the changed requirements. The input parameters in this case can be not only numeric ones, but also a partially defined model—so the modeler can “freeze” the fragments of the model that have to be preserved, update the input parameters, and use the normal generation process to create new models.

8. Conclusions

This paper proposes an approach to generating and complementing graph-based organization and process models using a GAN with examples related to the smart city area, namely for decision support in logistics system and smart tourist trip booking process modelling. The developed model considers a composite discriminator consisting of three components: a “classical” discriminator for learning tacit patterns in the training dataset, an analytical constraint validator for learning differentiable constraints on parameters, and a constraint approximator for learning constraints on structure and non-differentiable constraints on parameters. A procedure for generating augmentations is also considered that makes it possible to extend the training set with meaningful models satisfying required constraints. The multi-aspect modelling algorithm enables the multi-level generation of complex models. The carried out experiments prove the feasibility of the proposed approach. A publicly-available open-source library, OrGAN, has been developed which may be useful for both developers and researchers.

Future work is considered in the following directions. First, as was already mentioned, we are going to study the dependencies between GAN neural network sizes, modelling graph complexity, and the training set size. This issue also covers the problem of the scalability of this approach. The other direction aims at developing a universal framework for the definition of the problem domain constraints to simplify the symbolic knowledge representation. We also plan to extend the approach to other domain and cross-domain applications.

We believe that the proposed approach can be useful for developing GAN models, not only for generating models in the identified domains but also for other ones, and can be a starting point for research efforts in related scientific fields.

Author Contributions: Conceptualization, N.S.; methodology, N.S. and A.P.; software, N.S. and A.P.; validation, A.P.; writing—original draft preparation, N.S. and A.P.; writing—review and editing, A.P., A.K. and D.R.; visualization: N.S., A.P., A.K. and D.R.; project administration, A.K.; funding acquisition, A.K. and D.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Analytical Center for the Government of the Russian Federation (IGK 000000D730324P540002), agreement No. 70-2021-00141.

Data Availability Statement: The developed open source programming library OrGAN is openly available at <https://github.com/cais-lab/organ>, accessed on 6 December 2024.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Yaqoob, I.; Salah, K.; Jayaraman, R.; Omar, M. Metaverse Applications in Smart Cities: Enabling Technologies, Opportunities, Challenges, and Future Directions. *Internet Things* **2023**, *23*, 100884. [\[CrossRef\]](#)
2. Pandya, S.; Srivastava, G.; Jhaveri, R.; Babu, M.R.; Bhattacharya, S.; Maddikunta, P.K.R.; Mastorakis, S.; Piran, M.J.; Gadekallu, T.R. Federated Learning for Smart Cities: A Comprehensive Survey. *Sustain. Energy Technol. Assess.* **2023**, *55*, 102987. [\[CrossRef\]](#)
3. Huda, N.U.; Ahmed, I.; Adnan, M.; Ali, M.; Naeem, F. Experts and Intelligent Systems for Smart Homes' Transformation to Sustainable Smart Cities: A Comprehensive Review. *Expert Syst. Appl.* **2024**, *238*, 122380. [\[CrossRef\]](#)
4. Tupayachi, J.; Xu, H.; Omitaomu, O.A.; Camur, M.C.; Sharmin, A.; Li, X. Towards Next-Generation Urban Decision Support Systems through AI-Powered Construction of Scientific Ontology Using Large Language Models—A Case in Optimizing Intermodal Freight Transportation. *Smart Cities* **2024**, *7*, 2392–2421. [\[CrossRef\]](#)
5. Kolhe, R.V.; William, P.; Yawalkar, P.M.; Paithankar, D.N.; Pabale, A.R. Smart City Implementation Based on Internet of Things Integrated with Optimization Technology. *Meas. Sens.* **2023**, *27*, 100789. [\[CrossRef\]](#)
6. Müller-Eie, D.; Kosmidis, I. Sustainable Mobility in Smart Cities: A Document Study of Mobility Initiatives of Mid-Sized Nordic Smart Cities. *Eur. Transp. Res. Rev.* **2023**, *15*, 36. [\[CrossRef\]](#)
7. Kutty, A.A.; Kucukvar, M.; Onat, N.C.; Ayvaz, B.; Abdella, G.M. Measuring Sustainability, Resilience and Livability Performance of European Smart Cities: A Novel Fuzzy Expert-Based Multi-Criteria Decision Support Model. *Cities* **2023**, *137*, 104293. [\[CrossRef\]](#)
8. Olaniyi, O.O.; Okunleye, O.J.; Olabanji, S.O. Advancing Data-Driven Decision-Making in Smart Cities through Big Data Analytics: A Comprehensive Review of Existing Literature. *Curr. J. Appl. Sci. Technol.* **2023**, *42*, 10–18. [\[CrossRef\]](#)
9. Doctorarastoo, M.; Flanigan, K.; Bergés, M.; McComb, C. Exploring the Potentials and Challenges of Cyber-Physical-Social Infrastructure Systems for Achieving Human-Centered Objectives. In Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, Istanbul, Turkey, 15–16 November 2023; ACM: New York, NY, USA, 2023; pp. 385–389.
10. Zang, T.; Wang, S.; Wang, Z.; Li, C.; Liu, Y.; Xiao, Y.; Zhou, B. Integrated Planning and Operation Dispatching of Source–Grid–Load–Storage in a New Power System: A Coupled Socio–Cyber–Physical Perspective. *Energies* **2024**, *17*, 3013. [\[CrossRef\]](#)
11. Metta, M.; Dessein, J.; Brunori, G. Between On-Site and the Clouds: Socio-Cyber-Physical Assemblages in on-Farm Diversification. *J. Rural Stud.* **2024**, *105*, 103193. [\[CrossRef\]](#)
12. Kusiak, A. Generative Artificial Intelligence in Smart Manufacturing. *J. Intell. Manuf.* **2024**, *36*, 1–3. [\[CrossRef\]](#)
13. Krichen, M. Generative Adversarial Networks. In Proceedings of the 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 6–8 July 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–7.
14. Chakraborty, T.; Reddy, K.S.U.; Naik, S.M.; Panja, M.; Manvitha, B. Ten Years of Generative Adversarial Nets (GANs): A Survey of the State-of-the-Art. *Mach. Learn. Sci. Technol.* **2024**, *5*, 011001. [\[CrossRef\]](#)
15. Lin, H.; Liu, Y.; Li, S.; Qu, X. How Generative Adversarial Networks Promote the Development of Intelligent Transportation Systems: A Survey. *IEEE/CAA J. Autom. Sin.* **2023**, *10*, 1781–1796. [\[CrossRef\]](#)
16. Liu, M.; Wei, Y.; Wu, X.; Zuo, W.; Zhang, L. Survey on Leveraging Pre-Trained Generative Adversarial Networks for Image Editing and Restoration. *Sci. China Inf. Sci.* **2023**, *66*, 151101. [\[CrossRef\]](#)
17. Kao, P.-Y.; Yang, Y.-C.; Chiang, W.-Y.; Hsiao, J.-Y.; Cao, Y.; Aliper, A.; Ren, F.; Aspuru-Guzik, A.; Zhavoronkov, A.; Hsieh, M.-H.; et al. Exploring the Advantages of Quantum Generative Adversarial Networks in Generative Chemistry. *J. Chem. Inf. Model.* **2023**, *63*, 3307–3318. [\[CrossRef\]](#)
18. Koschmider, A.; Hornung, T.; Oberweis, A. Recommendation-Based Editor for Business Process Modeling. *Data Knowl. Eng.* **2011**, *70*, 483–503. [\[CrossRef\]](#)

19. Kuschke, T.; Mäder, P. Pattern-Based Auto-Completion of UML Modeling Activities. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, Vasteras, Sweden, 15–19 September 2014; ACM: New York, NY, USA, 2014; pp. 551–556.
20. Wieloch, K.; Filipowska, A.; Kaczmarek, M. Autocompletion for Business Process Modelling. In *Business Information Systems Workshops, Proceedings of the BIS 2011 International Workshops and BPSC International Conference, Poznań, Poland, 15–17 June 2011*; Lecture Notes in Business Information Processing; Springer: Berlin/Heidelberg, Germany, 2011; Volume 97, pp. 30–40. [\[CrossRef\]](#)
21. Born, M.; Brelage, C.; Markovic, I.; Pfeiffer, D.; Weber, I. Auto-Completion for Executable Business Process Models. In *Business Process Management Workshops, Proceedings of the BPM 2008 International Workshops, Milano, Italy, 1–4 September 2008*; Lecture Notes in Business Information Processing; Springer: Berlin/Heidelberg, Germany, 2009; Volume 17, pp. 510–515. [\[CrossRef\]](#)
22. Mazanek, S.; Minas, M. Business Process Models as a Showcase for Syntax-Based Assistance in Diagram Editors. In *Model Driven Engineering Languages and Systems, Proceedings of the 12th International Conference, MODELS 2009, Denver, CO, USA, 4–9 October 2009*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5795, pp. 322–336. [\[CrossRef\]](#)
23. Clever, N.; Holler, J.; Shitkova, M.; Becker, J. Towards Auto-Suggested Process Modeling—Prototypical Development of an Auto-Suggest Component for Process Modeling Tools. In Proceedings of the Enterprise Modelling and Information Systems Architectures (EMISA 2013), St. Gallen, Switzerland, 5–6 September 2013; Gesellschaft für Informatik e.V.: Bonn, Germany, 2013; pp. 133–145.
24. Fellmann, M.; Zarvić, N.; Thomas, O. Business Processes Modelling Assistance by Recommender Functionalities: A First Evaluation from Potential Users. In *Perspectives in Business Informatics Research, Proceedings of the 16th International Conference, BIR 2017, Copenhagen, Denmark, 28–30 August 2017*; Lecture Notes in Business Information Processing; Springer: Cham, Switzerland, 2017; Volume 295, pp. 79–92. [\[CrossRef\]](#)
25. Li, Y.; Cao, B.; Xu, L.; Yin, J.; Deng, S.; Yin, Y.; Wu, Z. An Efficient Recommendation Method for Improving Business Process Modeling. *IEEE Trans. Ind. Inform.* **2014**, *10*, 502–513. [\[CrossRef\]](#)
26. Nair, A.; Ning, X.; Hill, J.H. Using Recommender Systems to Improve Proactive Modeling. *Softw. Syst. Model.* **2021**. [\[CrossRef\]](#)
27. van Gils, B.; Proper, H.A. Enterprise Modelling in the Age of Digital Transformation. In *The Practice of Enterprise Modeling, Proceedings of the 11th IFIP WG 8.1. Working Conference, PoEM 2018, Vienna, Austria, 31 October–2 November 2018*; Lecture Notes in Business Information Processing; Springer: Cham, Switzerland, 2018; Volume 335, pp. 257–273. [\[CrossRef\]](#)
28. Awadid, A.; Bork, D.; Karagiannis, D.; Nurcan, S. Toward Generic Consistency Patterns in Multi-View Enterprise Modelling. In Proceedings of the ECIS 2018 Proceedings; Association for Information Systems (AIS), Portsmouth, UK, 23–28 June 2018; p. 146.
29. Khider, H.; Hammoudi, S.; Meziane, A. Business Process Model Recommendation as a Transformation Process in MDE: Conceptualization and First Experiments. In Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development, Valletta, Malta, 25–27 February 2020; SciTePress: Setúbal, Portugal, 2020; pp. 65–75.
30. Fayoumi, A. Toward an Adaptive Enterprise Modelling Platform. In *The Practice of Enterprise Modeling, Proceedings of the 11th IFIP WG 8.1. Working Conference, PoEM 2018, Vienna, Austria, 31 October–2 November 2018*; Lecture Notes in Business Information Processing; Springer: Cham, Switzerland, 2018; Volume 335, pp. 362–371. [\[CrossRef\]](#)
31. Wang, J.; Gui, S.; Cao, B. A Process Recommendation Method Using Bag-of-Fragments. *Int. J. Intell. Internet Things Comput.* **2019**, *1*, 32. [\[CrossRef\]](#)
32. Jangda, A.; Polisetty, S.; Guha, A.; Serafini, M. Accelerating Graph Sampling for Graph Machine Learning Using GPUs. In Proceedings of the Sixteenth European Conference on Computer Systems, Edinburgh, UK, 26–28 April 2021; ACM: New York, NY, USA, 2021; pp. 311–326.
33. Valera, M.; Guo, Z.; Kelly, P.; Matz, S.; Cantu, V.A.; Percus, A.G.; Hyman, J.D.; Srinivasan, G.; Viswanathan, H.S. Machine Learning for Graph-Based Representations of Three-Dimensional Discrete Fracture Networks. *Comput. Geosci.* **2018**, *22*, 695–710. [\[CrossRef\]](#)
34. Chen, C.; Ye, W.; Zuo, Y.; Zheng, C.; Ong, S.P. Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals. *Chem. Mater.* **2019**, *31*, 3564–3572. [\[CrossRef\]](#)
35. Na, G.S.; Chang, H.; Kim, H.W. Machine-Guided Representation for Accurate Graph-Based Molecular Machine Learning. *Phys. Chem. Chem. Phys.* **2020**, *22*, 18526–18535. [\[CrossRef\]](#) [\[PubMed\]](#)
36. Nielsen, R.F.; Nazemzadeh, N.; Sillesen, L.W.; Andersson, M.P.; Gernaey, K.V.; Mansouri, S.S. Hybrid Machine Learning Assisted Modelling Framework for Particle Processes. *Comput. Chem. Eng.* **2020**, *140*, 106916. [\[CrossRef\]](#)
37. Wang, H.; Wang, J.; Wang, J.; Zhao, M.; Zhang, W.; Zhang, F.; Xie, X.; Guo, M. GraphGAN: Graph Representation Learning With Generative Adversarial Nets. *Proc. AAAI Conf. Artif. Intell.* **2018**, *32*, 2508–2515. [\[CrossRef\]](#)
38. Wang, H.; Wang, J.; Wang, J.; Zhao, M.; Zhang, W.; Zhang, F.; Li, W.; Xie, X.; Guo, M. Learning Graph Representation with Generative Adversarial Nets. *IEEE Trans. Knowl. Data Eng.* **2021**, *33*, 3090–3103. [\[CrossRef\]](#)
39. Zhao, M.; Zhang, Y. GAN-based Deep Neural Networks for Graph Representation Learning. *Eng. Rep.* **2022**, *4*, e12517. [\[CrossRef\]](#)
40. Fan, S.; Huang, B. Labeled Graph Generative Adversarial Networks. *arXiv* **2019**, arXiv:1906.03220.
41. Fan, S.; Tech, V.; Huang, B. Deep Generative Models for Labelled Graphs. In Proceedings of the Deep Generative Models for Highly Structured Data (ICLR 2019 Workshop), New Orleans, LA, USA, 6–9 May 2019; pp. 1–10.

42. Zhou, D.; Zheng, L.; Xu, J.; He, J. Misc-GAN: A Multi-Scale Generative Model for Graphs. *Front. Big Data* **2019**, *2*, 3. [\[CrossRef\]](#)
43. Jia, N.; Tian, X.; Gao, W.; Jiao, L. Deep Graph-Convolutional Generative Adversarial Network for Semi-Supervised Learning on Graphs. *Remote Sens.* **2023**, *15*, 3172. [\[CrossRef\]](#)
44. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv* **2016**, arXiv:1609.02907. [\[CrossRef\]](#)
45. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. *arXiv* **2017**, arXiv:1710.10903.
46. De Cao, N.; Kipf, T. MolGAN: An Implicit Generative Model for Small Molecular Graphs. *arXiv* **2018**, arXiv:1805.11973. [\[CrossRef\]](#)
47. Macedo, B.; Ribeiro Vaz, I.; Taveira Gomes, T. MedGAN: Optimized Generative Adversarial Network with Graph Convolutional Networks for Novel Molecule Design. *Sci. Rep.* **2024**, *14*, 1212. [\[CrossRef\]](#) [\[PubMed\]](#)
48. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. *arXiv* **2014**, arXiv:1406.2661. [\[CrossRef\]](#)
49. Liello, L.D.; Ardino, P.; Gobbi, J.; Morettin, P.; Teso, S.; Passerini, A.; Di Liello, L.; Ardino, P.; Gobbi, J.; Morettin, P.; et al. Efficient Generation of Structured Objects with Constrained Adversarial Networks. In Proceedings of the Advances in Neural Information Processing Systems 33 (NeurIPS 2020), Virtual, 6–12 December 2020; Volume 2020-Decem, p. 12.
50. Gui, J.; Sun, Z.; Wen, Y.; Tao, D.; Ye, J. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 3313–3332. [\[CrossRef\]](#)
51. Jabbar, A.; Li, X.; Omar, B. A Survey on Generative Adversarial Networks: Variants, Applications, and Training. *ACM Comput. Surv.* **2022**, *54*, 157. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.