ARTICLE OPEN

Synthetic electronic health records generated with variational graph autoencoders

Giannis Nikolentzos[1] ✉, Michalis Vazirgiannis[1,2], Christos Xypolopoulos[1], Markus Lingman[3,4] and Erik G. Brandt[5]

Data-driven medical care delivery must always respect patient privacy—a requirement that is not easily met. This issue has impeded improvements to healthcare software and has delayed the long-predicted prevalence of artificial intelligence in healthcare. Until now, it has been very difficult to share data between healthcare organizations, resulting in poor statistical models due to unrepresentative patient cohorts. Synthetic data, i.e., artificial but realistic electronic health records, could overcome the drought that is troubling the healthcare sector. Deep neural network architectures, in particular, have shown an incredible ability to learn from complex data sets and generate large amounts of unseen data points with the same statistical properties as the training data. Here, we present a generative neural network model that can create synthetic health records with realistic timelines. These clinical trajectories are generated on a per-patient basis and are represented as linear-sequence graphs of clinical events over time. We use a variational graph autoencoder (VGAE) to generate synthetic samples from real-world electronic health records. Our approach generates health records not seen in the training data. We show that these artificial patient trajectories are realistic and preserve patient privacy and can therefore support the safe sharing of data across organizations.

npj Digital Medicine (2023)6:83 ; https://doi.org/10.1038/s41746-023-00822-x

INTRODUCTION

Access to real-world health data is often restricted by privacy-protecting regulations like Health Insurance Portability and Accountability Act (HIPAA) and General Data Protection Regulation (GDPR), but also due to technical limitations or simply lacking incentives for data-sharing. Even when pseudo-anonymized (by leaving out personal identifiers such as social security numbers, residence, age, etc), a malicious agent with sufficient knowledge could re-identify patients by connecting patient attributes, conditions, medical prescriptions, etc. for an individual. Techniques like federated learning[1], differential privacy[2], and homo-morphic encryption[3] are actively researched to overcome these barriers.

Carefully created synthetic data could reduce data scarcity by exempting data from privacy-preserving regulations. By design, synthetic data mimics real data but is decoupled from real individuals and can be safely shared among healthcare providers, academics, and private stakeholders without leaking sensitive or personally identifiable information. High-quality synthetic data enables exploration and hypothesis generation but could also be used to pre-train AI models and thus decrease the need for vast amounts of original data. A synthetic data set that mirrors the original data well could also help focus efforts on more probable hypotheses before seeking confirmation in the source data. Therefore, synthetic data could meet both privacy concerns and re-balance the effort of data access in relation to the chance of relevant findings and also explore data patterns before investing too much in new research routes.

Healthcare data sets are complex in both space (heterogeneous and strongly connected) and time (cause and effect of symptoms, diagnoses, medications, etc.). Understanding the relationships between the different parts of information about a patient created along a patient trajectory is essential in clinical medicine. It is relatively straightforward to mimic the static properties of a given data distribution but far more difficult to mimic diverse and coupled time series with non-equidistant time steps[4,5]. To the best of our knowledge, this remains to be done in the context of electronic health records (EHRs).

Deep learning (DL) models have revolutionized a wide range of real-world applications, from autonomous vehicles[6] to machine translation[7] and molecule generation in drug discovery[8]. Nevertheless, even the most successful DL model is at the mercy of the amount and quality of its training data set. DL algorithms are very data-hungry, and the training samples must adequately reflect the full population that is to be learned. When such conditions can be met, DL algorithms have a canning ability to capture complex data patterns, and they also generalize well to unseen data. In practice, available data is often insufficient to train DL models with millions of parameters in any meaningful way. As a consequence, DL models are especially sensitive to limited data availability, as manifested in healthcare.

Machine learning algorithms have already been successfully introduced in the healthcare informatics domain[9–15]. Variational Autoencoders (VAEs)[16,17]—and their graph off-springs[18–20]—and Generative Adversarial Networks (GANs)[21,22] are recent deep learning architectures of particular promise. These models learn a "hidden" underlying data distribution from the training data. VAEs consist of an encoder–decoder pair. The encoder maps the input data to a latent (hidden) distribution, which is randomly sampled by the decoder with the objective of reconstructing the original input data. The latent distribution is usually chosen as a multivariate normal distribution characterized by its mean value and standard deviation. Once the model is trained, an arbitrary number of new samples can be generated by feeding the decoder random samples from the normal distribution. GANs, on the other hand, use two neural networks that are trained together but in adverse. The two networks are known as the generator and the

[1]LIX, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France. [2]Department of Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden. [3]Department of Molecular and Clinical Medicine/Cardiology, Institute of Medicine, Sahlgrenska Academy, University of Gothenburg, Gothenburg, Sweden. [4]Center for Applied Intelligent Systems Research, Halmstad University, Halmstad, Sweden. [5]SHAARPEC, Stockholm, Sweden. ✉email: nikolentzos@lix.polytechnique.fr

discriminator. The generator learns to create samples as realistically as possible, while the discriminator learns to distinguish synthetic samples from real ones. Once both networks are fully trained, the generator can create unseen data samples with high similarity to the real data.

Earlier efforts to simulate EHR data have used Bayesian network learning[23], and deterministic differential modeling, e.g., as implemented in the popular open-source software Synthea[24]. This open-source software package is designed to simulate the lifespans of synthetic patients but is based on fixed demographic properties extracted from public data and does not learn through a training procedure. GANs have so far been the primary deep-learning method to generate synthetic EHRs[9,25–27]. Notably, Choi et al. proposed medGAN[9], a neural network model that generates high-dimensional discrete variables to represent EHR events. Baowaly et al.[25] derived two enhanced versions of medGAN with a more complex (Wasserstein) architecture, and Yale et al.[26] identified limitations to medGAN and proposed HealthGAN, another Wasserstein-based method. They also developed improved metrics for synthetic health data quality. Chin-Cheong et al.[28] created synthetic EHR data with GANs trained on patient data from intensive care units. The final results were combined with federated learning[1] to mimic a real-world scenario with data sets from different organizations isolated in silos. Finally, Esteban et al.[11] generated a synthetic medical time series with a GAN using recurrent neural networks for both the generator and the discriminator. While GANs have achieved promising results, they also have drawbacks. GAN-generated data is continuous by default and must be paired to an autoencoder[9] or LSTM generator to produce discrete results[11]. Also, the adversial style used to train GANs increases the risk of getting stuck in local minima during learning[29]. Such minima are very challenging to escape with more training alone and may limit the use of GANs on certain data structures (such as graphs).

Interestingly, variational graph autoencoders (VGAEs) have not yet been used to generate synthetic EHRs. VGAEs present a promising route in this area because they are easy to train, have been applied successfully to other graph learning problems, and can accurately model the underlying data distribution. We discuss the strengths and weaknesses of GANs vs VGAEs in more detail in the “Results” section.

In this paper, we develop a machine-learning algorithm for generating electronic healthcare records represented as sequential graphs (patient trajectories). A patient trajectory is a time sequence of encounters (visits) at healthcare organizations (e.g., hospitals or other providers). Each encounter links to patient interventions such as identified medical conditions and requested medications. Analyzing such patient trajectories is key to delivering data-driven insights to healthcare organizations. Creating synthetic EHRs with graph deep learning is, to the best of our knowledge, a new concept. Synthetic graphs are already trending in drug design[30–33], but patient trajectories require much larger (e.g., hundreds of nodes) graph representations than their drug molecule counterparts. This poses a significant challenge to generation algorithms. Here, we propose a VGAE tailored to patient trajectories that can generate novel large-scale samples.

RESULTS

EHR data source

The Medical Information Mart for Intensive Care (MIMIC-IV) database was the source of all our numerical experiments[34]. MIMIC-IV provides critical care data for thousands of patients admitted to the intensive care units at the Beth Israel Deaconess Medical Center. The patient, visit (encounter), diagnosis (condition), and medication (medication request) data tables were migrated to a labeled property graph (LPG) database. These data tables cover healthcare events at the emergency unit and the following in-patient stays. They are a subset of a complete graph model for healthcare data developed by one of the authors[35], which follows the FHIR standard for healthcare data[36]. The LPG database was constructed by connecting all visits (encounters) for a patient in chronological order. Conditions and medications for the patient are then attached to the encounters. Descriptive nodes for the types of conditions and medications are shared among the patients in the database. Labels are the only node and edge attributes included in this work. Our results are easily extended to include more metadata stored as key-value pairs on nodes and edges.

A patient graph (trajectory) is defined as the subgraph constructed with its origin at a patient node, following edges outwards until reaching terminal nodes (type-describing nodes that have no outgoing edges). Patient graphs defined in this way are directed acyclic graphs (DAGs), i.e., they do not contain directed cycles. We extracted a subset of patients whose trajectories contain any of the ICD-10-CM diagnosis codes I48.0, I48.1, I48.2, and I48.9. This group corresponds to a real-world cohort of patients diagnosed with atrial fibrillation.

A trajectory graph was constructed for each of those 6535 patients. The node and edge labels are described by the functions ℓ_V and ℓ_E, which assign elements from the sets Σ_V and Σ_E (i.e., $\ell_V : V \rightarrow \Sigma_V$ and $\ell_E : E \rightarrow \Sigma_E$). Here, V and E denote the set of nodes

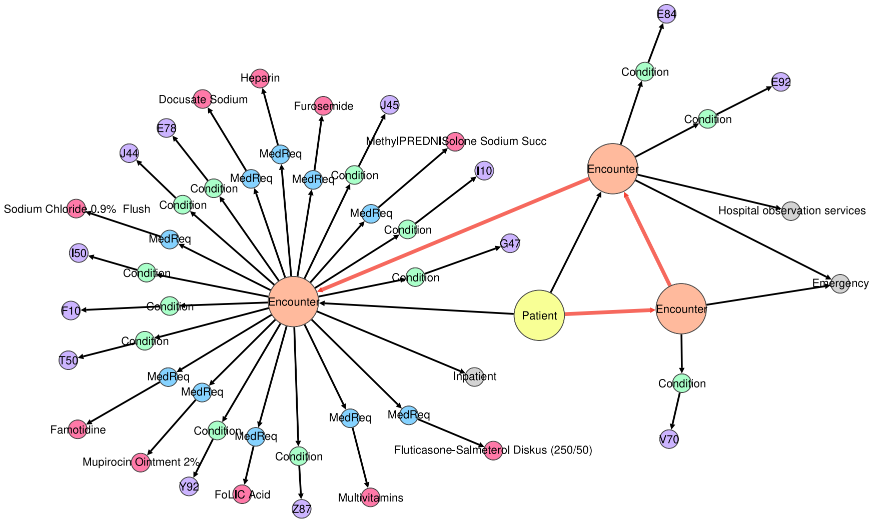


Fig. 1 Visualization of a patient trajectory. Each trajectory is represented as a directed acyclic graph. The patient timeline is represented by the edges between Encounters (highlighted in red).

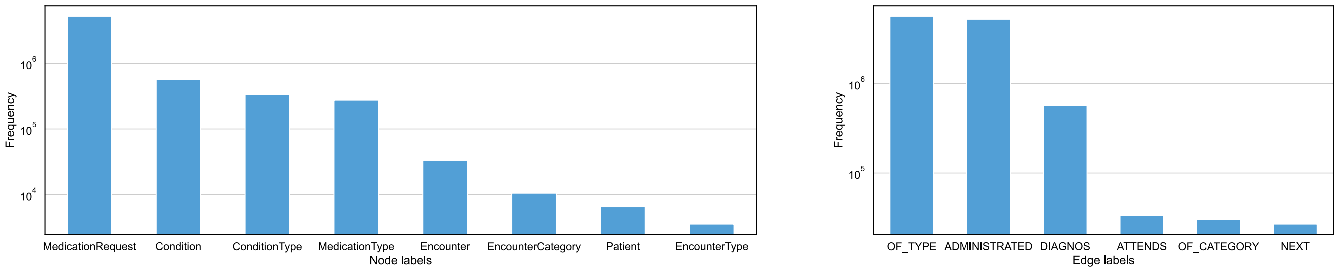


Fig. 2 Histogram of the number of (higher level) node labels and edge labels in the training data. There exist eight different (higher level) node labels and six different edge labels in total. Note that some higher-level node labels, such as MedicationType, and ConditionType, aggregate hundreds or even thousands of lower-level node labels.

and edges of all graphs, respectively. There are, in total $|\Sigma_V| = 13{,}980$ different node labels and $|\Sigma_E| = 6$ different edge labels. Figure 1 shows an example of a patient trajectory. Each trajectory contains one Patient node and a number of Encounter nodes that form the patient timeline. Each Encounter is described by an EncounterCategory and is shaped like a star graph with Condition/ConditionType and MedicationRequests/MedicationType pairs for the diagnosis and medication events. The edge labels are ATTENDS between Patient and each Encounter, NEXT between neighboring Encounters, OF_CATEGORY to describe the Encounter, DIAGNOS between Encounter and Condition, ADMINISTRATED between Encounter and Medication, and OF_TYPE to describe the Condition/ConditionType and MedicationRequests/MedicationType pairs. The edge labels are uniquely determined by the label pair of the ancestor and successor nodes. Note that edge labels are omitted from Fig. 1 to simplify the presentation. Figure 2 shows the frequency of the node and edge labels in the training data (see Table 1 for more details on the source data). ConditionType and MedicationType labels are higher-level labels that contain all diagnosis and medication events, respectively (i.e., labels I10, E92, Heparin, Mupirocin Ointment 2%, etc., in Fig. 1).

There is a large number of distinct ICD-10-CM diagnostic codes (and analog ATC codes for medications) in the MIMIC-IV data. Since the atrial fibrillation cohort is limited to 6535 patients, pre-processing is needed to reduce the number of node labels for the learning algorithm. The data was processed by: (1) Dropping all Condition nodes, which correspond to earlier versions than ICD-10-CM (ICD-9 and a few ICD-8 codes). (2) Only keeping the chapter (the three first characters) of the ICD-10-CM codes and merging nodes that ended up as identical. (3) Dropping rare events (condition and medication nodes that occurred less than 50 times). After these steps, 944 node labels remained. The largest graph had 143 Encounters, and the largest Encounter had 180 successors. More details about the pre-processed trajectories are given in Table 1.

Table 1. Statistics on the patient trajectories calculated from the atrial fibrillation cohort extracted from the MIMIC-IV database.

	Raw	After Pre-processing
Max # nodes	18,947	2772
Min # nodes	10	10
Average # nodes	1044.1	221.2
Max # edges	36,811	5162
Min # edges	9	9
Average # edges	1867.3	294.7
# node labels ($|\Sigma_V|$)	13,980	944
# edge labels ($|\Sigma_E|$)	6	6
# graphs	6535	6535

Statistics are provided for both raw data and processed data.

Generating model

Graph learning algorithms are usually permutation invariant, i.e., invariant to the ordering of nodes. Since nodes are added one at a time, node order becomes important. By modeling patient graphs (patient trajectories) as DAGs, graph generation is significantly simplified because every DAG has at least one linear ordering of the nodes such that for every directed edge (u, v), node u comes before node v. This is known as a topological ordering and can be computed in linear time.

We found that standard recurrent neural networks were unable to learn realistic patient trajectories (see the discussion in the Methods section). We, therefore, designed a model tailored to the structure of our patient graphs. These trajectories are built up of linear sequences of Encounter nodes, where each Encounter node is the center of a star graph. These two substructures (linear sequence and star) can be modeled separately.

GANs and VAEs are two of the most common architectures for generative models. They both learn from training data and can generate new and previously unseen data samples. GANs can generate highly realistic images, audio, and text but can be difficult to train, as the generator and discriminator networks may get stuck in a local equilibrium[29]. GANs struggle to provide meaningful latent data representations, while VAEs can interpolate smoothly between data points to find and generate new representative patient trajectories. Importantly, GANs work best with continuous data. We formulate graph generation as a sequential task where nodes and edges are added to the graph in iterations. This is natural for DAGs but poses a challenge for backpropagation because the gradients are zero. The usual workarounds can not be applied to graphs (e.g., Gumbel softmax[37]). One option is to avoid sequential generation and generate the adjacency matrix in one shot[38]. However, this approach is only feasible for small-graph applications, e.g., molecule generation (the adjacency matrix is an $(n \times n)$-matrix for a graph of n nodes).

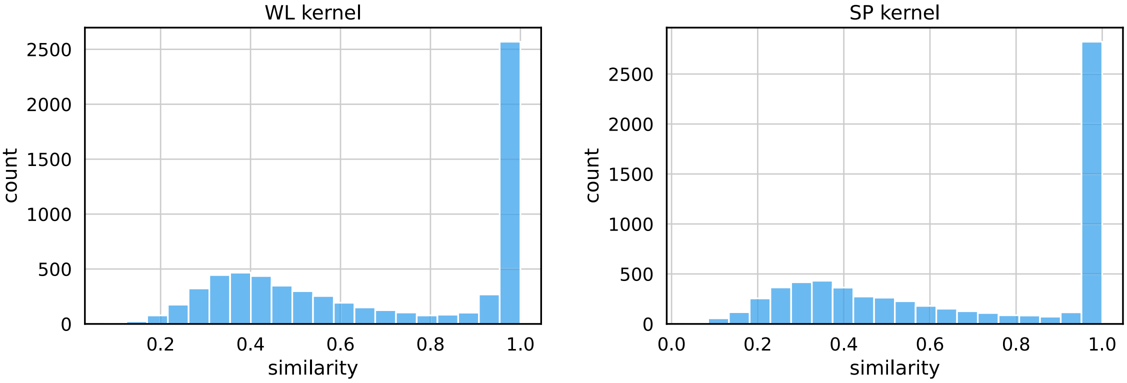
In light of the above discussion, we used a variational autoencoder as the basis of our synthetic data generator. The encoder maps patient trajectories into a parameterized multivariate Gaussian distribution (i.e., the encoder predicts the mean vector and covariance matrix of this distribution). A random sample is drawn from the distribution (the "hidden" representation of the input patient trajectory) and fed into the decoder to reconstruct the original patient trajectory. Once trained, new trajectories can be generated at scale by drawing random samples from the normal distribution and using the decoder to output a synthetic trajectory. Further details on the model architecture and training details are found in the "Methods" section.

Experiments

We first investigated whether the proposed model can accurately reconstruct its input graphs. To this end, we used graph kernels, which are symmetric positive semi-definite functions on the set of graphs $\mathcal{G}$[39]. Roughly speaking, a graph kernel measures the similarity of graphs. Once we define a function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ on the set $\mathcal{G}$, there exists a map $\phi : \mathcal{G} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$, such that $k(G, G') = \langle \phi(G), \phi(G') \rangle_{\mathcal{H}}$ for all $G, G' \in \mathcal{G}$ where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$. Graph kernels are grouped into major families that focus on different structural aspects of graphs. We primarily relied on the Weisfeiler-Lehman subtree (WL) kernel[40] and on the shortest path (SP) kernel[41] to compare input graphs against reconstructed graphs. WL and SP are among the most successful graph kernels and account for both graph structure and node label information. We computed the histogram of $k_i = k(G_i, \hat{G}_i)$, where G_i is an input graph, $\hat{G}_i$ its corresponding reconstructed graph, and $k(\cdot, \cdot)$ is a graph kernel (i.e., WL or SP) with $i \in \{1, 2, \ldots, 6535\}$. Here, $k_i = 0$ means that reconstructed graph i is completely different from its input, and vice versa $k_i = 1$ implies high similarity or even identity up to isomorphism. For the reconstruction task, ideally, we would like the model to output graphs isomorphic to those given as input. Thus, we would like most kernel values to be large (close to 1). The histograms in Fig. 3 show very high similarity ($k > 0.9$ for 3/4 of the graph distribution) for most graphs. This indicates that the proposed model yields very good performance in reconstructing the input graphs, even when some of them are large and consist of many Encounter nodes.

We have established that the model successfully reconstructs the patterns from the input graphs. Can the model also generate novel synthetic graphs that are realistic but not found in the training data? To investigate this, we generated 10,000 synthetic patient graphs by feeding random samples drawn from the multivariate normal distribution to the decoder. We used the WL and SP graph kernels to compare the generated synthetic graphs to the input graphs from the training data. We computed the histogram of the maximum similarity $k_j^{\max} = \max k_{ij}$ for the two kernels, where now G_i is an input graph, $\hat{G}_j$ is a ^{i}graph generated from a random sample with $j \in \{1, 2, \ldots, 10000\}$, and $k_{ij} = k(G_i, \hat{G}_j)$ is the (i, j):th element of a $(6535 \times 10{,}000)$-similarity matrix with $k(\cdot, \cdot)$ being a graph kernel. We end up with 10,000 kernel values. Figure 4 shows that both kernels' maximum similarity distributions are centered around $k^{\max} \sim 0.55$. There is a small fraction of samples for which $k^{\max} \approx 1$ holds. Such graphs correspond to near-identical replicas of input graphs and could lead to patient privacy leaking from the training set. These graphs must be eliminated from the generated data set to reduce the risk of privacy leak.

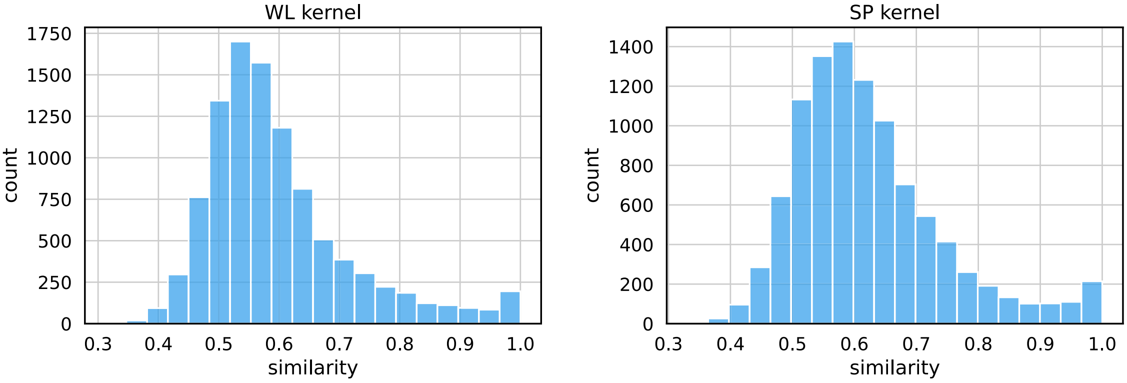
A simple method to avoid these replica graphs is to add Gaussian noise to the decoder-generated Encounter node



(a)

(b)

Fig. 3 Histograms of similarities between input and reconstructed trajectories. Each input trajectory was compared against its corresponding reconstructed trajectory using a graph kernel. **a** Histogram of similarities between input graphs and reconstructed graphs using the Weisfeiler-Lehman subtree (WL) kernel. **b** Histogram of similarities between input graphs and reconstructed graphs using the shortest path (SP) kernel.



(a)

(b)

Fig. 4 Histograms of similarities between input and generated trajectories. The model was used to generate 10,000 synthetic trajectories, and for each synthetic trajectory, the most similar input trajectory was found. **a** Histograms of similarities between input and generated trajectories using the Weisfeiler–Lehman subtree (WL) kernel. **b** Histograms of similarities between input and generated trajectories using the shortest path (SP) kernel.

representations. We can guarantee that all generated graphs are novel samples by tuning the variance of the Gaussian. Figure 5 shows the proportion of novel samples as a function of the standard deviation of the Gaussian distribution. We generated 20,000 samples for each value of σ. As expected, the number of novel samples is an increasing function of σ, and all samples are novel when $\sigma \geq 0.5$. More rigorous methods to ensure privacy could also be used. For example, there are VAE algorithms that perturb the latent vectors that represent the samples in a differential privacy-consistent way[42]. The fraction of replicas is small, even without added noise. More than 99% of the generated samples are novel. In addition, Fig. 4 shows that there are no graphs at low $k^{\max}$. Contributions near $k^{\max} = 0$ would have indicated a very low similarity to the inputs and generated patient trajectories that are unrealistic.

Further, we must determine to what extent these novel-generated samples are realistic representations of electronic health records. In what follows, we remove samples that are very similar to input graphs (those graphs $\hat{G}_j$ for which $k_j^{\max}$>0.9 according to the WL or SP kernel). We use the Graph Kernel-Maximum Mean Discrepancy (GK-MMD) metric to evaluate the quality of the generated graphs. The GK-MMD accounts for both graph topology and node labels[43]. The MMD statistical test determines whether sample sets from two distributions P and Q were in fact derived from the same distribution (i.e., whether distributions P and Q are identical). The square of the MMD is:

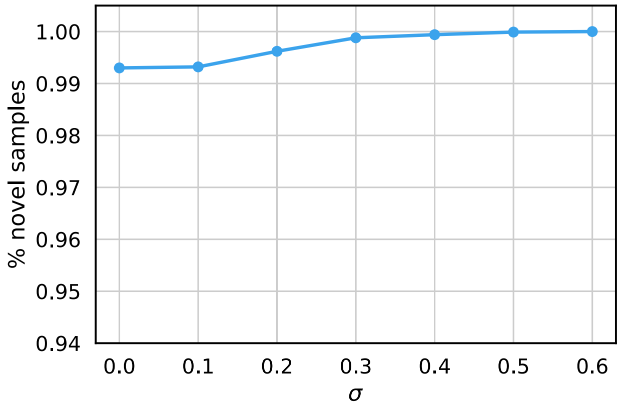


Fig. 5 Percentage of novel trajectories as a function of the standard deviation of the Gaussian noise. The mean of the Gaussian was chosen to be the zero vector. The noise was added to the representations of the Encounter nodes. For each value of $\sigma \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, 20000 trajectories were generated in total, which was compared against the real trajectories.

$$\mathrm{MMD}^2(P, Q) = \mathbb{E}_{x,x' \sim P}[k(x, x')] - 2\,\mathbb{E}_{x \sim P, y \sim Q}[k(x, y)] + \mathbb{E}_{y,y' \sim Q}[k(y, y')] \tag{1}$$

where $k(\cdot, \cdot)$ is the associated kernel function[44]. A lower MMD score indicates a better approximation in terms of the employed kernel (0 means that they are identical). We computed the squared WL-MMD and SP-MMD scores between the generated trajectories and their training data. The scores were 0.0017 and 0.0020, respectively.

We also investigate if paths of length n occurred with the same frequencies in the generated graphs as in the input graphs. Such n-paths can be thought of as Condition and Medication node pairs separated in time by $(n - 1)$ Encounters. In this terminology, 1-path corresponds to node labels, 2-paths to nearest neighbors, and 3-paths to next-nearest-neighbors. The first is a static (time-independent) property, but the other two are dynamic properties through the timeline implicit by the NEXT relation between neighboring Encounters. Examples of 2- and 3-paths are highlighted in Fig. 6. The number of such paths increases exponentially, which has a significant impact on the time complexity to compute correlation coefficients for larger n. We calculated Pearson's r as a function of n (Table 2 and Fig. 7). The static 1-paths (node labels) are perfectly retained in the generated graphs ($r = 0.995$). This shows that static properties like patient attributes and the number of diagnoses and medications are indistinguishable in the generated graphs compared to the training data. The dynamic 2- and 3-paths are almost equally well preserved in the novel training samples ($r > 0.93$). This shows without a doubt that the model learns time-dependencies between conditions and medications that occur in consecutive Encounters.

We can show the generated data quality more concretely. First, the patient visit length is the number of consecutive Encounters in the patient's trajectory. Figure 8 shows the frequency of visit lengths for real and synthetic patient cohorts, respectively. The two distributions are very similar, and both real and synthetic cohorts are dominated by short visit lengths (less than 20 Encounters, even though the input data contains a 144-length outlier). We computed an MMD of 0.025 between the two distributions. This very low value further verifies that the generated trajectories are realistic.

Next, we calculated comorbidities, defined as two or more medical conditions that exist simultaneously in the patient trajectory regardless of causal relationships. Following ref. [45], we define the comorbidities for atrial fibrillation as Heart failure (ICD-10-CM code I50), Hypertension (I10–I15), Diabetes mellitus

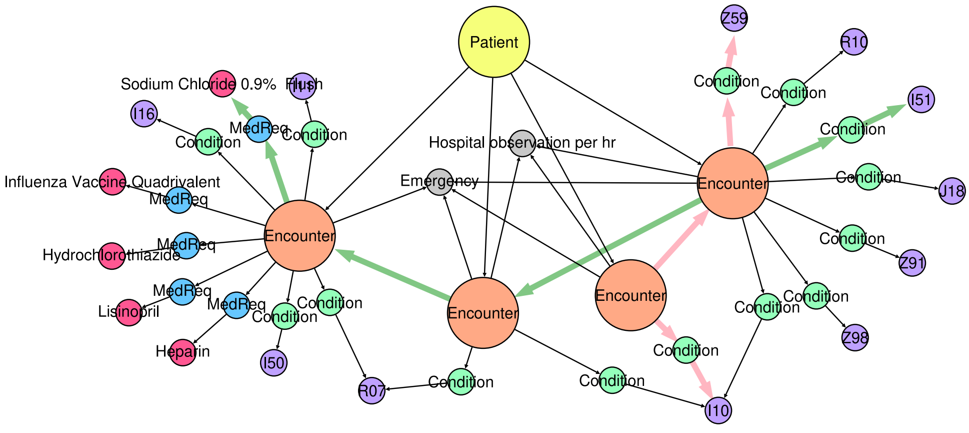


Fig. 6 Examples of 2- and 3-paths in a patient trajectory. A 2-path from the ConditionType node I10 of the first Encounter node to ConditionType node Z59 of the second Encounter node is highlighted in pink. A 3-path from the ConditionType node I51 of the second Encounter node to the MedicationType node Sodium Chloride 0.9% of the fourth Encounter node is highlighted in green.

Table 2. Pearson's *r* between the frequency of different structures in the collection of training trajectories and the collection of generated trajectories.

n	*r*	Description
1	0.995	Node types
2	0.963	Sequences of condition/medication codes of length two
3	0.934	Sequences of condition/medication codes of length three

Structures correspond to 1-paths (node types), 2-paths, and 3-paths, as illustrated in Fig. 6.

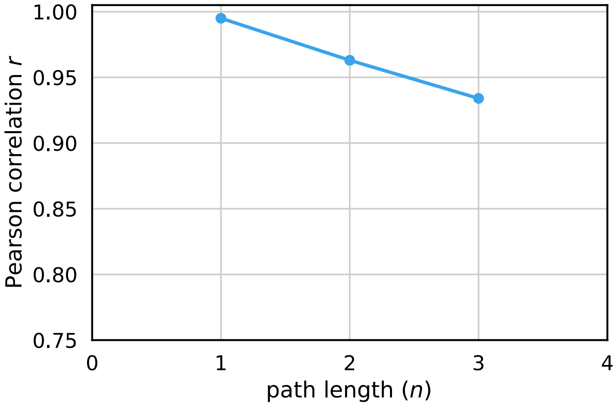


Fig. 7 Pearson's *r* between the frequency of different structures in the collection of training trajectories and the collection of generated trajectories. Structures correspond to 1-paths (node labels), 2-paths, and 3-paths, as illustrated in Fig. 6.

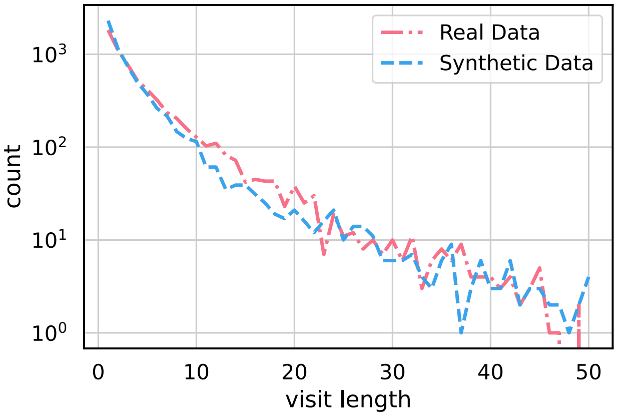


Fig. 8 Plot of visit lengths for real and synthetic data. The visit length refers to the number of Encounter nodes that exist in a patient trajectory. Only trajectories consisting of, at most, 50 Encounter nodes are considered.

(E10–E14), and Stroke/TIA (G45, I63, and I74). Figure 9 shows comorbidities for the real and synthetic atrial fibrillation cohorts. The synthetic data faithfully replicates the trends in the original data. The only difference is that patients with combined heart failure and hypertension are slightly more common with diabetes in the real data than in the synthetic data.

We now investigate whether synthetic data samples can stand in for real-world data in downstream analytics tasks. As mentioned, heart failure is a common and potentially fatal comorbidity to atrial fibrillation. It is the leading cause of hospitalization and readmission in older adults[46]. In fact, about 40 million people worldwide suffered from heart failure in 2015[47]. Machine learning methods have already improved predictions of hospital readmissions in heart failure patients[48], but the onset of heart failure is more difficult to predict. Here, we designed a classification task where the objective is to predict whether heart failure is the cause of the next patient encounter. We developed a classifier by pairing the encoder module of our VGAE with a multi-layer perceptron (MLP). This model was trained from scratch to make predictions in two different scenarios: "train on real, test on real" and "train on synthetic, test on real". These scenarios demonstrate whether a model trained on synthetic data generalizes to real data with similar performance to a model trained on real data. We used identical test sets in the two scenarios and the same number of training samples. All data sets were balanced with an equal number of input samples from each class. For each scenario, we designed two experiments. In Task 1, the last Encounter node was removed from each trajectory. The sample was assigned to class 1 if that Encounter node linked to a Condition node with ICD-10-CM code I50 (heart failure) and to class 0 otherwise. In Task 2, we removed all trajectories with either no I50 code or where the I50 code was connected to the first or second Encounter. By definition, the first I50 code of the trajectory is connected to the *i*th encounter ($i > 2$). The trajectories were then assigned to class 0 (heart failure did not occur) or class 1 (heart failure did occur) with equal probability. For each class 0-trajectory, we generated a random number $k = \{1, \ldots, i-2\}$. The trajectory's first *k* Encounters and their Conditions and MedicationRequests were kept. For each class 1-trajectory, the first $i-1$ Encounters and their Conditions and MedicationRequests were kept. We trained the two tasks as binary classifications. Training and analysis were repeated 10 times. The average test accuracy and the standard deviation are reported in Table 3. We find that the classifier's performance is independent of whether real or synthetic training data is used. In both tasks, training on synthetic data does not impact the performance of the classifier. Thus, for these downstream tasks, the synthetic data provides the same

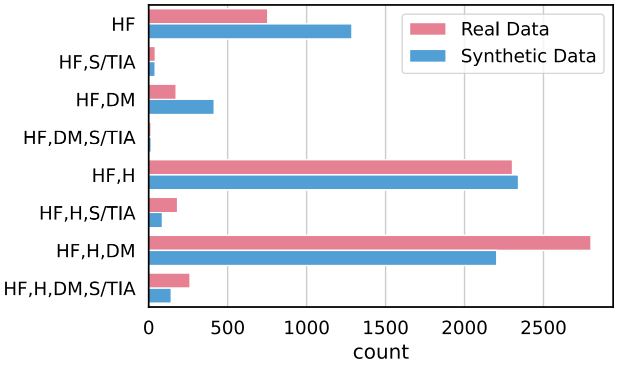


Fig. 9 Comorbidities for the real and synthetic atrial fibrillation cohorts. We define the comorbidities for atrial fibrillation as follows: HF heart failure, S/TIA stroke/TIA, DM diabetes mellitus, H hypertension.

Table 3. Classification accuracy of the two experimental scenarios in two separate downstream analytics tasks.

Scenario	Task 1	Task 2
Train on real, test on real	64.02% ± 2.93	73.04% ± 1.35
Train on synthetic, test on real	63.85% ± 2.57	73.32% ± 2.90

Both tasks are variants of predicting the onset of heart failure in patients. In the first scenario, the classifier is trained on real data and is evaluated on real data, while in the second scenario, the classifier is trained on synthetic data and is evaluated on real data.

Table 4. Percentage of real and synthetic samples that were identified as outliers by the one-class SVM algorithm.

Kernel	Outliers	
	Synthetic data	Real data
WL	7.8% ± 1.2	10.7% ± 0.9
SP	6.9% ± 1.0	11.3% ± 0.8

Two graph kernels are employed to compute the kernel values (i.e., similarities) between trajectories, namely the Weisfeiler-Lehman subtree (WL) kernel and the shortest path (SP) kernel.

predictive performance as the real data. The data patterns needed for these predictions are learned by the synthetic generator.

Finally, we validated our model with outlier identification using a one-class Support Vector Machine (SVM), which is an unsupervised variant of the standard SVM[49]. SVMs can be applied to graph data using graph kernels. We generated 6535 synthetic patient trajectories and checked that no samples were isomorphic to the real training set. We randomly selected 10% of the synthetic trajectories and added them to the real trajectories (this is the "inlier" set). Then, we sampled 10% of the remaining synthetic trajectories and used the trained one-class SVM on each candidate to decide whether it was an inlier or not ("outlier"). We used the SVM parameter $v = 0.1$, meaning that no more than 10% of the training samples can be considered outliers by the decision boundary. We repeated the prediction experiment 10 times by sampling different subsets of synthetic trajectories. For comparison, we split the real data into training and test sets at a 90:10 ratio. As before, we enriched the training set with 10% of the synthetic trajectories and let the SVM classifier identify outliers in real data. Table 4 shows the results of these numerical experiments. The one-class SVM algorithm predicts that a large majority (92–93%) of the synthetic trajectories are inliers independent of the graph kernel. The algorithm performance is similar (90–91%) to the real data samples. This shows that the generated trajectories are of high quality and indistinguishable from real data. Figure 10 shows examples of how similar trajectories from the original data set are to novel samples generated with our VGAE-based model.

We also trained a support vector machine (SVM) classifier to learn to distinguish between real trajectories and the synthetic trajectories generated by the model. However, the classifier could only discern that synthetic trajectories were slightly more similar to each other than to the real trajectories. Those results are provided in Table 5 in the Supplementary Materials.

DISCUSSION

Well-generated synthetic healthcare data could provide an opportunity to improve the value of analytics by allowing easier access to data in order to pre-train AI models, generate novel hypotheses, and explore data patterns without jeopardizing patient integrity. In this paper, we present a deep-learning model for generating synthetic patient trajectories from electronic health records. We show that the model can be effectively trained on real graphs and generate novel ones that are not in the training set. These patient trajectories are clinically realistic while sufficiently different from the trajectories in training set to preserve patient privacy.

Our model is a variational graph autoencoder (VGAE) tailored to patient trajectories represented as directed acyclic graphs. Previous generative models have failed to produce large graphs or to learn long-range time correlations. The model proposed here solves these issues by decoupling the sequential patient timeline from the clinical interventions. The model is well suited for the complex time dependencies found in electronic health records. Our numerical results show that the model generates novel synthetic patient trajectories, not found in the training data, that are sufficiently different to preserve patient privacy yet retains the characteristics of the real-world data. Arguably the most significant feature is that the model is powerful enough to learn long-range correlations between trajectory nodes.

An interesting question arising from this work is to what extent synthetic data can replace real-world data in downstream analysis. Given our experimental results, and the model's ability to learn paths in patient trajectories, we expect analysis based on either real or synthetic to lead to similar conclusions. This could be tested in practice by comparing the output of more machine learning classifiers trained on synthetic trajectories against those trained on real-world data. Are the data-driven insights identical?

As we have already emphasized, the success of any deep learning model rests on the quality and amount of training data. The model can capture general trends already from limited training data but ultimately requires large amounts of training data to generate long and accurate patient trajectories. In short, as always, the more data, the better results. In general, outliers (patient trajectory groups that rarely occur in the training set) are also difficult to generate with accuracy. The generative model should always take measures to ensure that all trajectories of interest are well-represented in the training set.

We next discuss the technical limitations of the proposed model. First, node and edge labels are the only metadata included in our model. There is a lot of additional metadata in EHR systems (for example, lab data including values and units) that is of interest for analysis. Such data is represented as key/value pairs on nodes and edges in our graph model. Our generator is easily extended to node and edge attributes by coupling a multi-layer perceptron (MLP) to the model once the node type has been determined. Second, the present version of the model assumes that Encounter nodes are connected to only one type of Condition or Medication node. In practice, there can be more than one on the same node (if, for example, the patient is administered the same medication multiple times in the same encounter). This limitation is due to the model's binary classifier, which decides whether (or not) a single node of each type should be added to the encounter. A future iteration of the model could replace the binary classifier with a module that accounts for multiplicity. Third, the model has a number of hyper-parameters (see the Methods section) that could be investigated for further sensitivity analysis and optimization.

Another aspect of the work that is worth discussing is data bias. Data bias (i.e., skewed representations of certain data categories based on e.g., ethnicity, gender, or something else) in healthcare AI systems has been debated because of its potential to harm misrepresented population groups[50,51]. Although data bias is a real concern to the systematic use of healthcare data in general, it is important to separate the discussion regarding bias in the underlying training data from any eventual bias introduced by the AI model itself. We have identified no such model bias.

Synthetic data is not meant to straight-off replace original patient data. Instead, synthetic data can first increase data efficiency by providing a safe and accessible environment for analytics and pre-training AI models. Tentative conclusions from analytics will still need to be verified on the original data, as before. But crucially, synthetic data avoids the need to access sensitive real-world data when the chances for meaningful conclusions are lower.

With regard to privacy risks in the context of synthetic data, a major threat is when a malicious agent can use synthetic patient trajectories to re-identify real patients from the training data. This is called privacy leaking. That risk is magnified when the agent is in possession of additional information about the real individuals (medical conditions, prescriptions, etc.) that can be combined with the synthetic data to form recognizable patterns that can be used

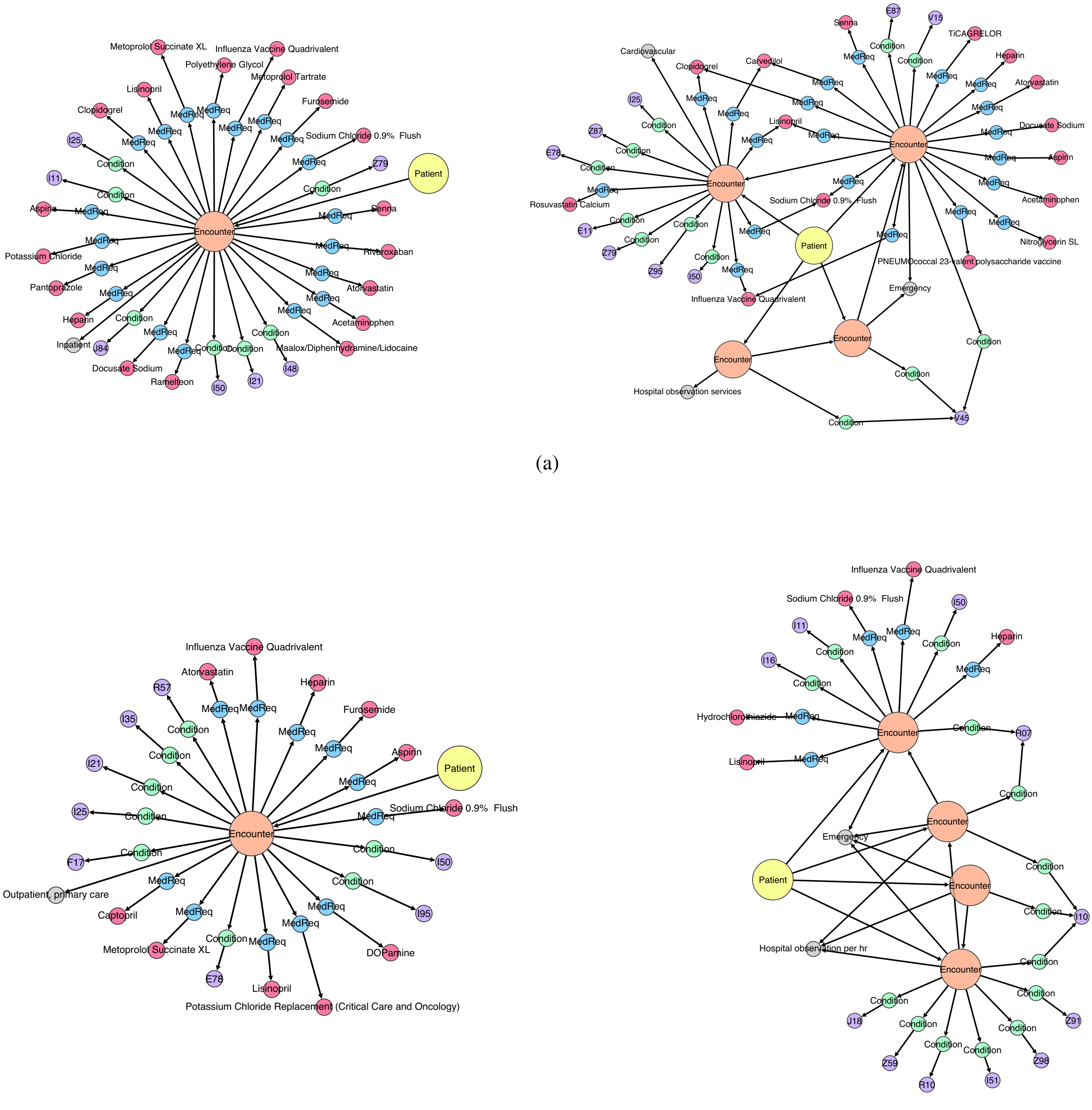


Fig. 10 Visualizations of patient trajectories. a Two examples of trajectories extracted from the training data. **b** Two examples of synthetic trajectories that were generated with the proposed model.

for re-identification. In our model, the amount of similarity between the synthetic and real trajectories is adjustable by the amplitude of the noise injected into the sampled latent space. Synthetic data should undergo a careful evaluation with respect to identity disclosure risks prior to distribution[52]. A number of different approaches for reducing the risk of information disclosure[53,54] have been proposed since disclosure control methods have a significant impact on data utility.

To conclude, graph deep learning is a powerful tool for learning complex data patterns. Here, a variant of a variational graph autoencoder (VGAE) tailored to patient trajectories represented as large directed acyclic graphs created privacy-preserving and highly accurate synthetic EHRs with long-range time correlations. This approach could reduce the problem of restricted access to health data, thus enabling explorative analyses, algorithm pre-training, hypothesis generation, and data expansion without jeopardizing privacy.

METHODS

Notation

Let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n \geq 1$ and $G = (V, E)$ be a directed graph where V is the node set, and E is the edge set, such that n is the number of nodes and m is the number of edges in the graph.

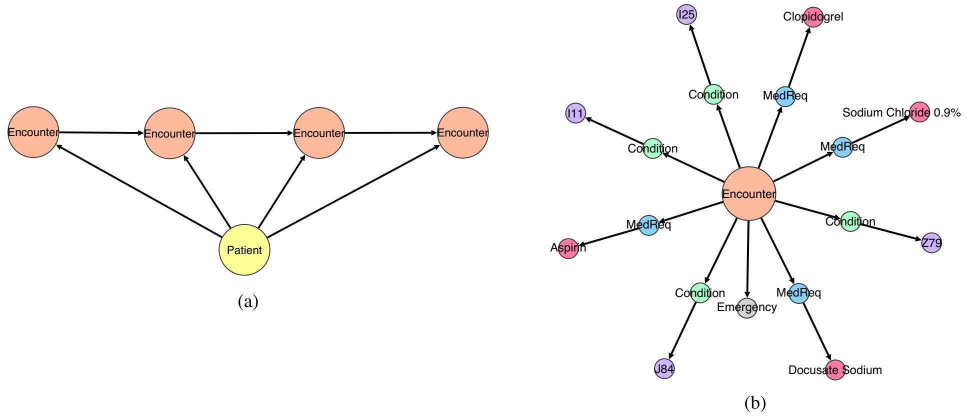


Fig. 11 The two main structures into which a patient trajectory can be decomposed. A patient trajectory can be decomposed into a patient node followed by a sequence of Encounter nodes and a set of Encounter nodes, each connected to several Condition and MedicationRequest nodes, which in turn are terminated with ConditionType and MedicationType nodes. **a** Example of a sequence of 4 Encounter nodes. **b** Example of an Encounter node.

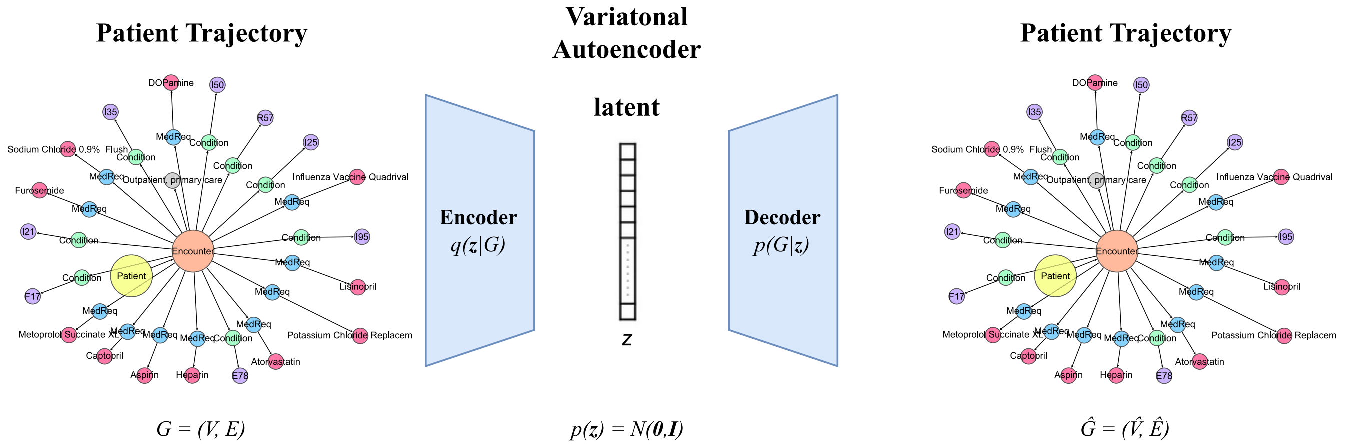


Fig. 12 An overview of the proposed variational autoencoder. To train the model, a patient trajectory is fed to the encoder, which projects the trajectory to a distribution parameterized as a multivariate Gaussian. Then a latent vector **z** is sampled from that distribution and is passed on to the decoder, which reconstructs the input trajectory. To generate a synthetic sample, a vector is sampled from the normal distribution and is then fed directly to the decoder, which constructs the synthetic trajectory.

The neighborhood $\mathcal{N}(v)$ of a node v is the set of all nodes adjacent to v. For a directed graph, we use $\mathcal{N}^+(v) = \{u \,|\, (v,u) \in E\}$ to indicate the set of out-neighbors of v where (v, u) is an edge between nodes v and u of V, and $\mathcal{N}^-(v) = \{u \,|\, (u,v) \in E\}$ to indicate the set of in-neighbors of v. The out-degree of node v is $d^+(v) = |\mathcal{N}^+(v)|$ and its in-degree is $d^-(v) = |\mathcal{N}^-(v)|$. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ of a graph G is a symmetric (and typically sparse) matrix used to encode edge information in the graph. Element (i, j) is the weight of the edge between nodes v_i and v_j if the edge exists and 0 otherwise. For graphs with node labels and edge labels, nodes and edges are associated with discrete labels, expressed by two functions $\ell_V: V \to \Sigma_V$ and $\ell_E: E \to \Sigma_E$ that map nodes and edges to labels from the sets of labels Σ_V and Σ_E, respectively.

Architectural details

We designed a model tailored to patient trajectories where each graph corresponds to the following:

- A patient node is followed by a sequence of Encounter nodes (Fig. 11a).
- Each Encounter node is connected to Condition and MedicationRequest nodes, which in turn are terminated with ConditionType and MedicationType nodes. An Encounter node could also be connected to EncounterType and/or EncounterCategory nodes (Fig. 11b).

Clearly, the graph generation can be carried out in two steps: (1) Generate the Encounter node sequence. (2) Generate the successors of each Encounter node. For the first task, we use the topological order of the patient trajectory subgraph obtained only from the Patient and Encounter nodes. This topological order is important because it keeps the trajectory timeline by enforcing Encounter node u to precede Encounter v chronologically. Since Patient and Encounter nodes are only a small fraction of the nodes in the patient trajectory, a recurrent neural network (RNN) can capture the relationships between consecutive encounters in the sequence. For the second task, we could generate successors of the Encounter nodes by imposing any topological ordering and letting another RNN learn that structure. That is possible since Encounter nodes do not have too many successors. In this work, we used an alternative approach where we consider the Encounter successor nodes as a set, and then we simply generate a set that contains those nodes.

We use an encoder–decoder architecture. The encoder maps input DAGs to a distribution parameterized as a multivariate Gaussian. In other words, the encoder predicts the mean and standard deviation of this Gaussian distribution. A random sample is then drawn from the distribution and serves as the latent representation of the input graph. The decoder tries to reconstruct the input DAGs given their vector representations. The decoder is a variational approximation, $p_\theta(G|\mathbf{z})$, which takes an embedding $\mathbf{z}$ as input. An overview of the proposed model is given in Fig. 12.

Two pre-processing steps were applied to the patient trajectories before encoding. First, we merged Condition/ConditionType and MedicationRequest/Medication type node pairs. Second, for each graph, an End node was added via a directed edge to the last Encounter node. This allows the model to decide when to terminate the generation of nodes in a new graph.

The encoder of the model is a message-passing graph neural network. Its first part is an embedding layer that creates representations for the nodes in each patient DAG. Each node v has a trainable node embedding $\mathbf{x}_v$, and there is a single node embedding for each node type. These node embeddings are updated during training with a combination of synchronous and asynchronous message-passing schemes.

First, the Encounter node embeddings are updated by aggregating the embeddings of their successors, excluding Encounter and End nodes:

$$\begin{aligned} \mathbf{m}_v &= \sum_{u \in \mathcal{N}^+(v)} f(\mathbf{x}_u) \\ \mathbf{h}_v &= \mathrm{GRU}(\mathbf{x}_v, \mathbf{m}_v) \end{aligned} \tag{2}$$

where $\mathcal{N}^+(v)$ is the set of successors of Encounter node v (again, excluding Encounter and End nodes), f is a neural network (MLP), $\mathbf{x}_v$ is the embedding of node v, and GRU is a gated recurrent unit.

An asynchronous message passing scheme is then applied where we sequentially perform message passing according to the topological sorting obtained from the patient subgraph of Encounter and End nodes. This differs from the standard message-passing scheme in graph neural networks, where all node embeddings are updated at each algorithm step. In our algorithm, the node embeddings are updated in this step according to the following:

$$\begin{aligned} \mathbf{m}_v &= \sum_{u \in \mathcal{N}^-(v)} f(\mathbf{h}_u) \\ \mathbf{h}_v &= \mathrm{GRU}(\mathbf{h}_v, \mathbf{m}_v) \end{aligned} \tag{3}$$

where $\mathcal{N}^-(v)$ is the set of incoming neighbors of v for Encounter nodes.

Once all node embeddings of the DAG have been computed, we use the end node embedding (i.e., the node without any successors) as the output of the encoder. Thus, $\mathbf{h}_G = \mathbf{h}_e$ where e denotes the End node of G. This vector is passed to two fully-connected layers to get the mean and variance parameters of the posterior approximation $q(\mathbf{z}|G)$:

$$\begin{aligned} \boldsymbol{\mu} &= \mathbf{W}_{\boldsymbol{\mu}} \mathbf{h}_G + \mathbf{b}_{\boldsymbol{\mu}} \\ \log \boldsymbol{\sigma}^2 &= \mathbf{W}_{\boldsymbol{\sigma}^2} \mathbf{h}_G + \mathbf{b}_{\boldsymbol{\sigma}^2} \end{aligned} \tag{4}$$

The decoder of the model also applies an asynchronous message-passing scheme to generate node representations. The decoder uses a GRU to update node embeddings when generating the graph.

A fully-connected layer is used to map the input latent vector $\mathbf{z}$ to the initial (hidden) state vector $\mathbf{h}_0$. The state vector is passed to the GRU, which constructs a DAG node-by-node. So far, all are Encounter (or End) nodes. The embedding of the first (Patient) node is $\mathbf{h}_{v_1} = \mathrm{GRU}(\mathbf{x}_{v_1}, \mathbf{h}_0)$. The following steps are performed to generate node v_i:

- Compute the label distribution of v_i with an MLP based on the current graph state $\mathbf{h}_G = \mathbf{h}_{v_{i-1}}$.
- Sample the label of v_i. If this is the end label, stop the decoding, connect the last Encounter node to v_i, and return the DAG. If not, continue the generation.
- Connect the last added Encounter node and the Patient node to v_i. Update $\mathbf{h}_{v_i}$ according to:

$$\begin{aligned} \mathbf{m}_{v_i} &= \sum_{u \in \mathcal{N}^-(v_i)} f(\mathbf{h}_u) \\ \mathbf{h}_{v_i} &= \mathrm{GRU}(\mathbf{x}_{v_i}, \mathbf{m}_{v_i}) \end{aligned} \tag{5}$$

- Produce a vector $\mathbf{s} \in \mathbb{R}^c$ (c denotes the different types of successors of Encounter nodes excluding Encounter and End nodes):

$$\mathbf{s} = \mathrm{MLP}(\mathbf{h}_{v_i}) \tag{6}$$

The sigmoid function is applied point-wise to the MLP output, and then the model decides whether to add a node of each type of successor to the graph. When a new node is added, so is a directed edge from v_i. The decision to add a successor to the graph is a binary classification problem. We, therefore, use the binary cross entropy loss to train the model.

Loss function

The loss function of our variational autoencoder has two terms,

$$\mathcal{L} = \mathcal{L}_{\text{reconstruction}} + \beta \mathcal{L}_{KL} \tag{7}$$

The first term is the reconstruction loss, i.e., the variational lower bound, and measures how well the model reconstructs the input data. The reconstruction loss is high if the reconstructed DAG is very different from its input. This term can be split into two contributions, $\mathcal{L}_{\text{reconstruction}} = \mathcal{L}_{\text{encounter}} + \mathcal{L}_{\text{other}}$. One contribution measures how well the model can reconstruct the sequence of Encounter nodes. It is equal to the binary cross-entropy between the predicted types of Encounter or End nodes and their actual types:

$$\mathcal{L}_{\text{encounter}} = -\sum_{i=1}^{k} \ell(v_i) \log(\hat{y}_i) + (1 - \ell(v_i)) \log(1 - \hat{y}_i) \tag{8}$$

Remember, $\{v_1, \ldots, v_k\}$ are only the Encounter and End nodes in the DAG. The other contribution measures how well the model can reconstruct the successors of the Encounter nodes. It is equal to the binary cross-entropy between the predicted and the actual successors of each Encounter node:

$$\mathcal{L}_{\text{other}} = -\sum_{i=1}^{k} \sum_{j=1}^{r} f(v_i, \sigma_j) \log(\hat{y}_{ij}) + (1 - f(v_i, \sigma_j)) \log(1 - \hat{y}_{ij}) \tag{9}$$

Here the $\{\sigma_1, \ldots, \sigma_r\}$ set includes all nodes except Encounter and End nodes for node v:

$$f(v, \sigma) = \begin{cases} 1, & \left|\{u | u \in \mathcal{N}^+(v) \wedge \ell(u) = \sigma\}\right| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The second term of the loss function is a regularization term. It is equal to the Kullback-Leibler (KL) divergence of the approximate $q(\mathbf{z}|G) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ from the true posterior $p(\mathbf{z})$, where $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{0}$ and $\mathbf{I}$ are the all-zeros vector and the identity matrix, respectively. The KL divergence measures how closely the output distribution $q(\mathbf{z}|G)$ matches $p(\mathbf{z})$:

$$\mathcal{L}_{KL} = -\mathrm{KL}[q(\mathbf{z}|G) || p(\mathbf{z})] \tag{11}$$

Training

To train the model, we used teacher forcing[55], i.e., a strategy for training RNNs that feeds the observed sequence values as input to the model instead of feeding the model's output from a prior time step, thus forcing the model to stay close to the ground-truth sequence. Given the topological ordering of the Encounter nodes, in each decoding step, we force the model to generate the ground-truth node type (Encounter node or End node) and the ground-truth condition and medication nodes connected to the Encounter nodes. Note that during the generation of new samples, teacher forcing cannot be applied since there is no ground-truth information, and thus we sample node types according to the decoding distributions. We used a cyclical scheme for the annealing parameter in Eq. (7), which increases β multiple times[56]. This is done to balance the two terms in the loss function such that the model learns the underlying distribution while maximizing the use of the available latent space.

Experimental setup

We used the following values for the model's hyper-parameters. The hidden-dimension size of the embedding layer and the GRU layers was 512. The hidden-dimension size of the fully connected layer that transforms the sampled vector representation of the graphs was set to 512 and followed by a tanh-activation. We used an MLP with hidden-dimension size 1024 to decide whether a new node type was to be added to the graph (and also to determine its type). We used an MLP with a hidden-dimension size of 2048 to compute the successors of Encounter nodes. The hidden layers in both MLPs were followed by ReLU activation functions. The dimension of the multivariate Gaussian distribution was set to 256. The batch size was 256, and the number of learning epochs was 5000. We used the Adam optimizer with an initial learning rate of 10^{-3} and decayed the learning rate by 0.1 every 1000 epochs to a minimum of 10^{-5}. The model with the lowest training loss was stored on disk and retrieved at the end of training. The best model was then used to generate new graphs for the numerical experiments.

Reporting summary

Further information on research design is available in the Nature Research Reporting Summary linked to this article.

DATA AVAILABILITY

MIMIC-IV data is available on the PhysioNet repository (https://physionet.org/), and access is authorized to users through a data use agreement with the providers.

CODE AVAILABILITY

The code of the presented model, together with the code for the statistical analyses, is available upon request to the corresponding author.

Received: 12 October 2022; Accepted: 5 April 2023;
Published online: 29 April 2023

REFERENCES

1. Rieke, N. et al. The future of digital health with federated learning. *npj Digit. Med.* **3**, 1–7 (2020).
2. Abadi, M. et al. in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318 (2016).
3. Acar, A., Aksu, H., Uluagac, A. S. & Conti, M. A survey on homomorphic encryption schemes: theory and implementation. *ACM Comput. Surv.* **51**, 1–35 (2018).
4. Yoon, J., Jarrett, D. & Van der Schaar, M. in *Advances in Neural Information Processing Systems* (2019).
5. Ramponi, G., Protopapas, P., Brambilla, M. & Janssen, R. T-cgan: conditional generative adversarial network for data augmentation in noisy time series with irregular sampling. Preprint at *arXiv* https://doi.org/10.48550/arXiv.1811.08295 (2018).
6. Kuutti, S., Bowden, R., Jin, Y., Barber, P. & Fallah, S. A survey of deep learning applications to autonomous vehicle control. *IEEE Trans. Intell. Transp. Syst.* **22**, 712–733 (2020).
7. Popel, M. et al. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nat. Commun.* **11**, 1–15 (2020).
8. Walters, W. P. & Barzilay, R. Applications of deep learning in molecule generation and molecular property prediction. *Acc. Chem. Res.* **54**, 263–270 (2020).
9. Choi, E. et al. in *Proceedings of Machine Learning for Healthcare 2017*, pp. 286–305 (2017).
10. Jordon, J., Yoon, J. & Van Der Schaar, M. in *7th International Conference on Learning Representations* (2019).
11. Esteban, C., Hyland, S. L. & Rätsch, G. Real-valued (medical) time series generation with recurrent conditional gans. Preprint at *arXiv* https://doi.org/10.48550/arXiv.1706.02633 (2017).
12. Wendland, P. et al. Generation of realistic synthetic data using multimodal neural ordinary differential equations. *npj Digit. Med.* **5**, 1–10 (2022).
13. Tucker, A., Wang, Z., Rotalinti, Y. & Myles, P. Generating high-fidelity synthetic patient data for assessing machine learning healthcare software. *npj Digit. Med.* **3**, 1–13 (2020).
14. Chen, R. J., Lu, M. Y., Chen, T. Y., Williamson, D. F. & Mahmood, F. Synthetic data in machine learning for medicine and healthcare. *Nat. Biomed. Eng.* **5**, 493–497 (2021).
15. Goncalves, A. et al. Generation and evaluation of synthetic patient data. *BMC Med. Res. Methodol.* **20**, 1–40 (2020).
16. Kingma, D. P. & Welling, M. in *2nd International Conference on Learning Representations* (2014).
17. Kingma, D. P. & Welling, M. An introduction to variational autoencoders. *Found. Trends® Mach. Learn.* **12**, 307–392 (2019).
18. Simonovsky, M. & Komodakis, N. in *Proceedings of the 27th International Conference on Artificial Neural Networks*, pp. 412–422 (2018).
19. Salha, G., Limnios, S., Hennequin, R., Tran, V. A. & Vazirgiannis, M. in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 589–598 (2019).
20. Chatzianastasis, M., Dasoulas, G., Siolas, G. & Vazirgiannis, M. in *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision Workshops*, pp. 393–402 (2021).
21. Creswell, A. et al. Generative adversarial networks: an overview. *IEEE Signal Process. Mag.* **35**, 53–65 (2018).
22. 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.*, 3313–3332 (2021).
23. Kaur, D. et al. Application of Bayesian networks to generate synthetic health data. *J. Am. Med. Inform. Assoc.* **28**, 801–811 (2021).
24. Walonoski, J. et al. Synthea: an approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *J. Am. Med. Inform. Assoc.* **25**, 230–238 (2018).
25. Baowaly, M. K., Lin, C. C., Liu, C. L. & Chen, K. T. Synthesizing electronic health records using improved generative adversarial networks. *J. Am. Med. Inform. Assoc.* **26**, 228–241 (2019).
26. Yale, A. et al. Generation and evaluation of privacy preserving synthetic health data. *Neurocomputing* **416**, 244–255 (2020).
27. Arvanitis, T.N., White, S., Harrison, S., Chaplin, R. & Despotou, G. A method for machine learning generation of realistic synthetic datasets for validating healthcare applications. *Health Inform. J.* **28**, 1–16 (2022).
28. Chin-Cheong, K., Sutter, T. & Vogt, J. E. in *Workshop on Machine Learning for Health (ML4H) at the 33rd Conference on Neural Information Processing Systems* (2019).
29. Saxena, D. & Cao, J. Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Comput. Surv.* **54**, 1–42 (2021).
30. You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. in *Proceedings of the 35th International Conference on Machine Learning*, pp. 5708–5717 (2018).
31. Jin, W., Barzilay, R. & Jaakkola, T. in *Proceedings of the 35th International Conference on Machine Learning*, pp. 2323–2332 (2018).
32. Li, Y., Vinyals, O., Dyer, C., Pascanu, R. & Battaglia, P. in *Proceedings of the 35th International Conference on Machine Learning* (2018).
33. Bongini, P., Bianchini, M. & Scarselli, F. Molecular generative graph neural networks for drug discovery. *Neurocomputing* **450**, 242–252 (2021).
34. Johnson, A. et al. *Mimic-iv* https://physionet.org/content/mimiciv/1.0/ (2021).
35. Implemented in the SHAARPEC Analytics platform. https://www.shaarpec.com.
36. Bender, D. & Sartipi, K. in *Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems*, pp. 326–331 (2013).
37. Jang, E., Gu, S. & Poole, B. in *5th International Conference on Learning Representations* (2017).
38. De Cao, N. & Kipf, T. Molgan: an implicit generative model for small molecular graphs. Preprint at *arXiv* https://doi.org/10.48550/arXiv.1805.11973 (2018)
39. Nikolentzos, G., Siglidis, G. & Vazirgiannis, M. Graph kernels: a survey. *J. Artif. Intell. Res.* **72**, 943–1027 (2021).

40. Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K. & Borgwardt, K. M. Weisfeiler-Lehman graph kernels. *J. Mach. Learn. Res.* **12**, 2539–2561 (2011).
41. Borgwardt, K. M. & Kriegel, H. P. in *Proceedings of the 5th IEEE International Conference on Data Mining* (2005).
42. Weggenmann, B., Rublack, V., Andrejczuk, M., Mattern, J. & Kerschbaum, F. in *Proceedings of the ACM Web Conference 2022*, pp. 721–731 (2022).
43. Kawai, W., Mukuta, Y. & Harada, T. Scalable generative models for graphs with graph attention mechanism. Preprint at *arXiv* https://arxiv.org/pdf/1906.01861.pdf (2019).
44. Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B. & Smola, A. A Kernel two-sample test. *J. Mach. Learn. Res.* **13**, 723–773 (2012).
45. Engdahl, J., Holmén, A., Rosenqvist, M. & Strömberg, U. Uptake of atrial fibrillation screening aiming at stroke prevention: geo-mapping of target population and non-participation. *BMC Public Health* **13**, 715–724 (2013).
46. Members, W. G. et al. Heart disease and stroke statistics—2012 update: a report from the American heart association. *Circulation* **125**, e2–e220 (2012).
47. Vos, T. et al. Global, regional, and national incidence, prevalence, and years lived with disability for 310 diseases and injuries, 1990–2015: a systematic analysis for the global burden of disease study 2015. *The Lancet* **388**, 1545–1602 (2016).
48. Mortazavi, B. J. et al. Analysis of machine learning techniques for heart failure readmissions. *Circulation* **9**, 629–640 (2016).
49. Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J. & Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural Comput.* **13**, 1443–1471 (2001).
50. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K. & Galstyan, A. A survey on bias and fairness in machine learning. *ACM Comput. Surv.* **54**, 1–35 (2021).
51. Parikh, R. B., Teeple, S. & Navathe, A. S. Addressing bias in artificial intelligence in health care. *JAMA* **322**, 2377–2378 (2019).
52. Reiter, J. P. & Mitra, R. Estimating risks of identification disclosure in partially synthetic data. *J. Privacy Confid.* **1**, 99–110 (2009).
53. Reiter, J. P. Satisfying disclosure restrictions with synthetic data sets. *J. Off. Stat.* **18**, 531 (2002).
54. Park, N. et al. Data synthesis based on generative adversarial networks. *Proc. VLDB Endow.* **11**, 1071–1083 (2018).
55. Williams, R. J. & Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.* **1**, 270–280 (1989).
56. Fu, H. et al. in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 240–250 (2019).

ACKNOWLEDGEMENTS

M.V. is partially supported by the "Wallenberg AI, Autonomous Systems and Software Program" (WASP). M.L. is partially supported by AIR Lund (Artificially Intelligent use of Registers at Lund University) research environment and received funding from the Swedish Research Council (VR; grant No. 2019-00198). G.N. is supported by the French National research agency via the AML-HELAS (ANR-19-CHIA-0020) project.

AUTHOR CONTRIBUTIONS

E.G.B. designed and directed this study. G.N. and M.V. contributed to the methodology design. G.N. conducted all the experiments. G.N., M.V., M.L., and E.G.B. analyzed and discussed the results. All authors wrote, reviewed, and revised the paper.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41746-023-00822-x.

Correspondence and requests for materials should be addressed to Giannis Nikolentzos.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.