medigan: a Python library of pretrained generative models for medical image synthesis

Richard Osuala[a,*] **Grzegorz Skorupko**[a] **Noussair Lazrak**[a] **Lidia Garrucho**[a] **Eloy García**[b] **Smriti Joshi**[a] **Socayna Jouide**[a] **Michael Rutherford**[c] **Fred Prior**[c] **Kaisar Kushibar**[a] **Oliver Díaz**[a] **and Karim Lekadir**[a]

[a]Universitat de Barcelona, Barcelona Artificial Intelligence in Medicine Lab (BCN-AIM), Facultat de Matemàtiques i Informàtica, Barcelona, Spain

[b]Universitat de Barcelona, Facultat de Matemàtiques i Informàtica, Barcelona, Spain

[c]University of Arkansas for Medical Sciences, Department of Biomedical Informatics, Little Rock, Arkansas, United States

Abstract

Purpose: Deep learning has shown great promise as the backbone of clinical decision support systems. Synthetic data generated by generative models can enhance the performance and capabilities of data-hungry deep learning models. However, there is (1) limited availability of (synthetic) datasets and (2) generative models are complex to train, which hinders their adoption in research and clinical applications. To reduce this entry barrier, we explore generative model sharing to allow more researchers to access, generate, and benefit from synthetic data.

Approach: We propose *medigan*, a one-stop shop for pretrained generative models implemented as an open-source framework-agnostic Python library. After gathering end-user requirements, design decisions based on usability, technical feasibility, and scalability are formulated. Subsequently, we implement *medigan* based on modular components for generative model (i) execution, (ii) visualization, (iii) search & ranking, and (iv) contribution. We integrate pretrained models with applications across modalities such as mammography, endoscopy, x-ray, and MRI.

Results: The scalability and design of the library are demonstrated by its growing number of integrated and readily-usable pretrained generative models, which include 21 models utilizing nine different generative adversarial network architectures trained on 11 different datasets. We further analyze three *medigan* applications, which include (a) enabling community-wide sharing of restricted data, (b) investigating generative model evaluation metrics, and (c) improving clinical downstream tasks. In (b), we extract Fréchet inception distances (FID) demonstrating FID variability based on image normalization and radiology-specific feature extractors.

Conclusion: *medigan* allows researchers and developers to create, increase, and domain-adapt their training data in just a few lines of code. Capable of enriching and accelerating the development of clinical machine learning models, we show *medigan*'s viability as platform for generative model sharing. Our multimodel synthetic data experiments uncover standards for assessing and reporting metrics, such as FID, in image synthesis studies.

 [DOI: 10.1117/1.JMI.10.6.061403]

Keywords: synthetic data; generative adversarial networks; Python; image synthesis; deep learning.

Paper 22262SSR received Oct. 5, 2022; accepted for publication Jan. 23, 2023; published online Feb. 20, 2023.

*Address all correspondence to Richard Osuala, Richard.Osuala@ub.edu

1 Introduction

1.1 *Deep Learning and the Benefits of Synthetic Data*

The use of deep learning has increased extensively in the last decade, thanks in part to advances in computing technology (e.g., data storage, graphics processing units) and the digitization of data. In medical imaging, deep learning algorithms have shown promising potential for clinical use due to their capability of extracting and learning meaningful patterns from imaging data and their high performance on clinically-relevant tasks. These include image-based disease diagnosis[1,2] and detection,[3] as well as medical image reconstruction,[4,5] segmentation,[6] and image-based treatment planning.[7–9]

However, deep learning models need vast amounts of well-annotated data to reliably learn to perform clinical tasks, whereas, at the same time, the availability of public medical imaging datasets remains limited due to legal, ethical, and technical patient data sharing constraints.[9,10] In the common scenario of limited imaging data, synthetic images, such as the ones illustrated in Fig. 1, are a useful tool to improve the learning of the artificial intelligence (AI) algorithm, e.g., by enlarging its training dataset.[7,11,12] Furthermore, synthetic data can be used to minimize problems associated with domain shift, data scarcity, class imbalance, and data privacy.[7] For instance, a dataset can be balanced by populating the less frequent classes with synthetic data during training (class imbalance). Further, as domain-adaptation technique, a dataset can be translated from one domain to another, e.g., from MRI to CT[13] (domain shift). Regarding data privacy, synthetic data can be shared instead of real patient data to improve privacy preservation.[7,14,15]

1.2 *The Need of Reusable Synthetic Data Generators*

Commonly, generative models are used to produce synthetic imaging data, with generative adversarial networks (GANs)[16] being popular models of choice. However, the adversarial training scheme required by GANs and related networks is known to pose challenges in regard to (i) achieving training stability, (ii) avoiding mode collapse, and (iii) reaching convergence.[17–19]

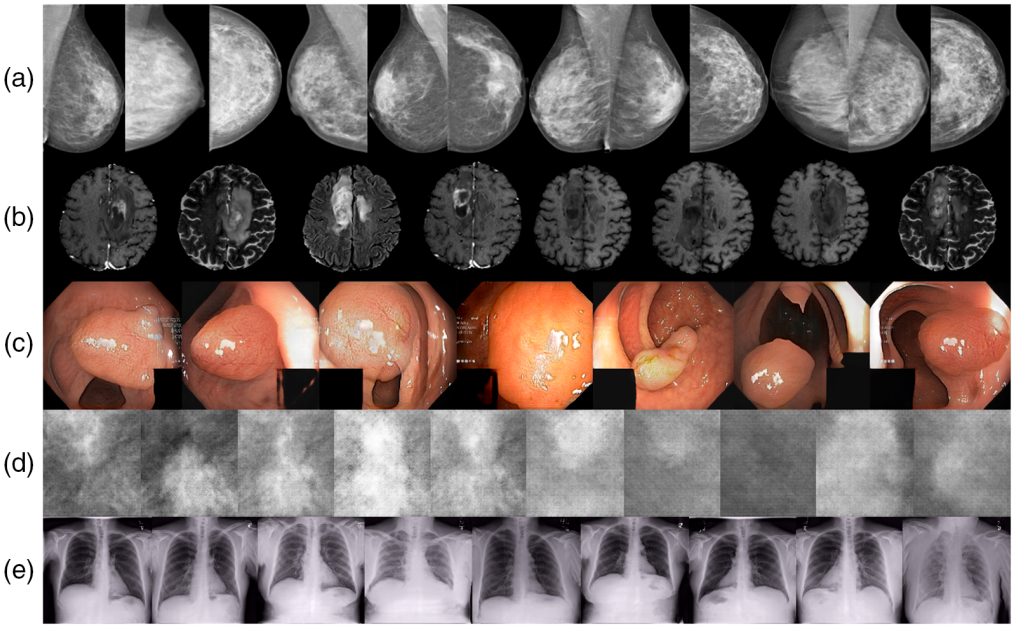

Fig. 1 Randomly sampled images generated by five *medigan* models ranging from (a) synthetic mammograms and (b) brain MRI to (c) endoscopy imaging of polyps, (d) mammogram mass patches, and (e) chest x-ray imaging. The models (a)–(e) correspond to the model IDs in Table 3, where (a) 3, (b) 7, (c) 10, (d) 12, and (e) 19.

Hence, the training process of GANs and generative models at large is nontrivial and requires a considerable time investment for each training iteration as well as specific hardware and a fair amount of knowledge and skills in the area of AI and generative modeling. Given these constraints, researchers and engineers often refrain from generating and integrating synthetic data into their AI training pipelines and experiments. This issue is further exacerbated by the prevailing need of training a new generative model for each new data distribution, which, in practice, often means that a new generative model has to be trained for each new application, use-case, and dataset.

1.3 *Community-Driven Model Sharing and Reuse*

We argue that a feasible solution to this problem is the community-wide sharing and reuse of pretrained generative models. Once successfully trained, such a model can be of value to multiple researchers and engineers with similar needs. For example, researchers can reuse the same model if they work on the same problem, conduct similar experiments, or evaluate their methods on the same dataset. We note that such reusing ideally is subject to previous inspection of generative model limitations with the model's output quality having qualified as suitable for the task at hand. The quality of a model's output data and annotations can commonly be measured via (a) expert assessment, (b) computation of image quality metrics, or (c) downstream task evaluation. In sum, the problem of synthetic data generation calls for a community-driven solution, where a generative model trained by one member of the community can be reused by other members of the community. Motivated by the absence of such a community-driven solution for synthetic medical data generation, we designed and developed *medigan* to bridge the gap between the need for synthetic data and complex generative model creation and training processes.

2 Background and Related Work

2.1 *Generative Models*

While discriminative models are able to distinguish between data instances of different kinds (label samples), generative models are able to generate new data instances (draw samples). In contrast to modeling decision boundaries in a data space, generative models model how data is distributed within that space. Deep generative models[20] are composed of multihidden layer neural networks to explicitly or implicitly estimate a probability density function (PDF) from a set of real data samples. After approximating the PDF from observed data points (i.e., learning the real data distribution), these models can then sample unobserved new data points from that distribution. In computer vision and medical imaging, synthetic images are generated by sampling such unobserved points from high-dimensional imaging data distributions. Popular deep generative models to create synthetic images in these fields include variational autoencoders,[21] normalizing flows,[22–24] diffusion models,[25–27] and GANs.[16] From these, the versatile GAN framework has seen the most widespread adoption in medical imaging to date.[7] We, hence, center our attention on GANs in the remainder of this work but emphasize that contributions of other types of generative models are equally welcome in the *medigan* library.

2.2 *Generative Adversarial Networks*

The training of GANs comprises two neural networks, the generator network (G) and the discriminator network (D), as illustrated by Fig. 2 for the example of mammography region-of-interest patch generation. G and D compete against each other in a two-player zero-sum game defined by the value function shown in Eq. (1). Subsequent studies extended the adversarial learning scheme by proposing innovations of the loss function, G and D network architectures, and GAN applications by introducing conditions into the image generation process

$$\min_G \max_D V(D,G) = \min_G \max_D [\mathbb{E}_{x\sim p_{\text{data}}}[\log\, D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]]. \quad (1)$$

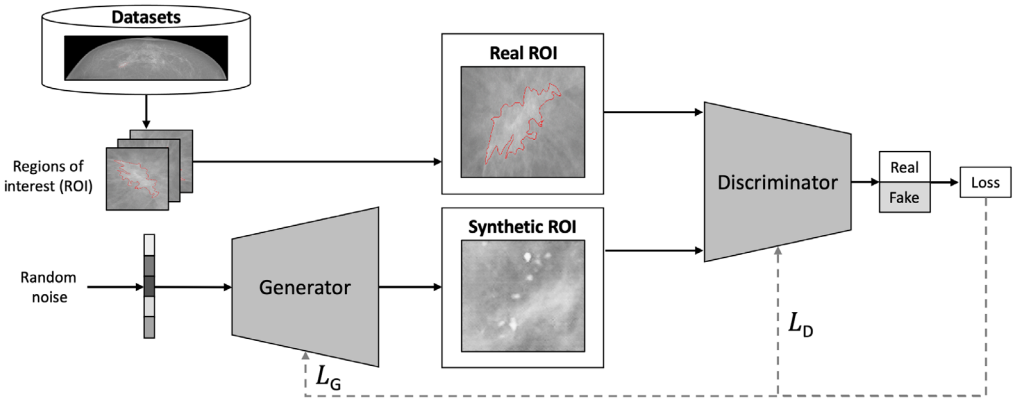

Fig. 2 The GAN framework. In this visual example, the generator network receives random noise vectors, which it learns to map to region-of-interest patches of full-field digital mammograms. During training, the adversarial loss is not only backpropagated to the discriminator as L_D but also to the generator as L_G. This particular architecture and loss function was used to train *medigan* models listed with IDs 1, 2, and 5 in Table 3.

2.2.1 GAN loss functions

Goodfellow et al.[16] define the discriminator as a binary classifier classifying whether a sample x is either real or generated. The discriminator is, hence, trained via binary-cross entropy with the objective of minimizing the adversarial loss function shown in Eq. (2), which the generator, on the other hand, tries to maximize. In Wasserstein GAN (WGAN),[28] the adversarial loss function is replaced with a loss function based on the Wasserstein-1 distance between real and fake sample distributions estimated by D (alias "critic"). Gulrajani et al.[29] resolve the need to enforce a 1-Lipschitz constraint in WGAN via gradient penalty (WGAN-GP) instead of WGAN weight clipping. Equation (3) depicts the WGAN-GP discriminator loss with penalty coefficient λ and distribution $\mathbb{P}_{\hat{x}}$ based on sampled pairs from (a) the real data distribution $\mathbb{P}_{\text{data}}$ and (b) the generated data distribution $\mathbb{P}_g$

$$L_{D_{\text{GAN}}} = -\mathbb{E}_{x\sim p_{\text{data}}}[\log\, D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))], \tag{2}$$

$$L_{D_{\text{WGAN-GP}}} = E_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - E_{x\sim\mathbb{P}_{\text{data}}}[D(x)] + \lambda E_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]. \tag{3}$$

In addition to changes to the adversarial loss, further studies integrate additional loss terms into the GAN framework. For instance, FastGAN[30] uses an additional reconstruction loss in the discriminator, which, for improved regularisation, is trained as self-supervised feature-encoder.

2.2.2 *GAN network architectures and conditions*

A plethora of different GAN network architectures has been proposed[7,31] starting with a deep convolutional GAN (DCGAN)[32] neural network architecture of both D and G. Later approaches, e.g., include a ResNet-based architecture as backbone[29] and progressively-grow the generator and discriminator networks during training to enable high-resolution image synthesis (PGGAN).[33]

Another line of research has been focusing on conditioning the output of GANs based on discrete or continuous labels. For example, in cGAN this is achieved by feeding a label to both D and G,[34] whereas in the auxiliary classifier GAN (AC-GAN), the discriminator additionally predicts the label that is provided to the generator.[35]

Other models condition the generation process on input images[36–40] unlocking image-to-image translation and domain-adaptation GAN applications. A key difference in image-to-image translation methodology is the presence (paired translation) or absence (unpaired translation) of corresponding image pairs in the target and source domain. Using an L1 reconstruction loss between target and source domain alongside the adversarial loss from Eq. (2), pix2pix[36] defines a common baseline model for paired image-to-image translation. For unpaired translation,

cycleGAN[37] is a popular approach, which also consists of an L1 reconstruction (cycle-consistency) loss between a source (target) image and a source (target) image translated to target (source) and back to source (target) via two consecutive generators.

A further methodological innovation includes SinGAN,[41] which, based on only a single training image, learns to generate multiple synthetic images. This is accomplished via a multi-scale coarse-to-fine pipeline of generators, where a sample is passed sequentially through all generators, each of which also receives a random noise vector as input.

2.3 *Generative Model Evaluation*

One approach of evaluating generative models is by human expert assessment of their generated synthetic data. In medical imaging, such observer studies often enlist board-certified clinical experts such as radiologists or pathologists to examine the quality and/or realism of the synthetic medical images.[42,43] However, this approach is manual, laborious and costly, and, hence, research attention has been devoted to automating generative model evaluation,[44,45] including:

i. Metrics for automated analysis of the synthetic data and its distribution, such as the inception score (IS)[17] and Fréchet inception distance (FID).[46] Both metrics are popular in computer vision,[31] whereas the latter also has seen widespread adoption in medical imaging.[7]

FID is based on a pretrained Inception[47] model (e.g., v1,[48] v3[47]) to extract features from synthetic and real datasets, which are then fitted to multivariate Gaussians X (e.g., real) and Y (e.g., synthetic) with means μ_X and μ_Y and covariance matrices Σ_X and Σ_Y. Next, X and Y are compared via the Wasserstein-2 (Fréchet) distance (FD), as depicted as

$$\mathrm{FD}(X,Y) = \|\mu_X - \mu_Y\|_2^2 + \mathrm{tr}\left(\Sigma_X + \Sigma_Y - 2\left(\Sigma_X\Sigma_Y\right)^{\frac{1}{2}}\right). \tag{4}$$

ii. Metrics that compare a synthetic image with a real reference image such as mean squared error (MSE), peak signal-to-noise ratio (PSNR), and structural similarity index measure (SSIM).[49] Given the absence of corresponding reference images, such metrics are not readily applicable for unconditional noise-to-image generation models.

iii. Metrics that compare the performance of a model on a surrogate downstream task with and without generative model intervention.[7,14,50,51] For instance, training on additional synthetic data can increase a model's downstream task performance, thus, demonstrating the usefulness of the generative model that generated such data.

For the analysis of generative models in the present study, we discard (ii) due to its limitation of requiring specific reference images. We further deprioritize the IS from (i) due to its limited applicability to medical imagery stemming from it missing a comparison between real and synthetic data distributions combined with it having a strong bias on natural images via its ImageNet[52]-pretrained Inception classifier as backbone feature extractor. Therefore, we focus on FID from (i) and downstream task performance (iii) as potential evaluation measures for medical image synthesis models in the remainder of this work.

2.4 *Image Synthesis Tools and Libraries*

Related libraries, such as pygan,[53] torchGAN,[54] vegans,[55] imaginaire,[56] TF-GAN,[57] PyTorch-GAN,[58] keras-GAN,[59] mimicry,[60] and studioGAN,[31] have focused on facilitating the implementation, training, and comparative evaluation of GANs in computer vision (CV). Despite a strong focus on language models, the HuggingFace transformers library and model hub[61] also contain a few pretrained computer vision GAN models. The GAN Lab[62] provides an interactive visual experimentation tool to examine the training process and its data flows in GANs.

Specific to AI in medical imaging, Diaz et al.[63] provided a comprehensive survey of tools, libraries and platforms for privacy preservation, data curation, medical image storage, annotation, and repositories. Compared to CV, fewer GAN and AI libraries and tools exist in medical imaging. Furthermore, CV libraries are not always suited to address the unique challenges of

medical imaging data.[63–65] For instance, pretrained generative models from computer vision cannot be readily adapted to produce medical imaging-specific outputs. The TorchIO library[64] addresses the gap between CV and medical image data processing requirements providing functions for efficient loading, augmentation, preprocessing, and patch-based sampling of medical imagery. The medical open network for AI (MONAI)[66] is a PyTorch-based[67] framework that facilitates the development of diagnostic AI models with tutorials for classification, segmentation, and AI model deployment. Further efforts in this realm include NiftyNet,[68] the deep learning tool kit (DLTK),[69] MedicalZooPytorch,[70] and nnDetection.[71] The recent RadImageNet initiative[72] shares baseline image classification models pretrained on a dataset designed as the radiology medical imaging equivalent to ImageNet.[52]

To the best of our knowledge, no open-access software, tool, or library exists that targets reuse and sharing of pretrained generative models in medical imaging. To this end, we expect the contribution of our *medigan* library to be instrumental in enabling dissemination of generative models and increased adoption of synthetic data into AI training pipelines. As an open-access plug-and-play solution for generation of multipurpose synthetic data, *medigan* aims to benefit patients and clinicians by enhancing the performance and robustness of AI-based clinical decision support systems.

3 Method: The *medigan* Library

We contribute *medigan* as an open-source open-access MIT-licensed Python3 library distributed via the Python package index (Pypi) for synthetic medical dataset generation, e.g., via pretrained generative models. The metadata of *medigan* is summarized in Table 1. *medigan* accelerates research in medical imaging by flexibly providing (a) synthetic data augmentation and (b) preprocessing functionality, both readily integrable in machine learning training pipelines. It also allows contributors to add their generative models in a thought-through process and provides simplistic functions for end-users to search for, rank, and visualize models. The overview of *medigan* in Fig. 3 depicts the core functions demonstrating how end-users can (a) contribute a generative model, (b) find a suitable generative model inside the library, and (c) generate synthetic data with that model.

Table 1 Overview of *medigan* library information.

	Title	*medigan* metadata
1	Code version	v1.0.0
2	Code license	MIT
3	Code version control system	Git
4	Software languages	Python
5	Code repository	https://github.com/RichardObi/medigan.
6	Software package repository	Ref. 73
7	Developer documentation	Ref. 74
8	Tutorial	medigan quickstart (tutorial.ipynb)
9	Requirements for compilation	Python v3.6+
10	Operating system	OS independent. Tested on Linux, OSX, Windows.
11	Support email address	Richard.Osuala[at]gmail.com
12	Dependencies	tqdm, requests, torch, numpy, PyGithub, matplotlib (setup.py)

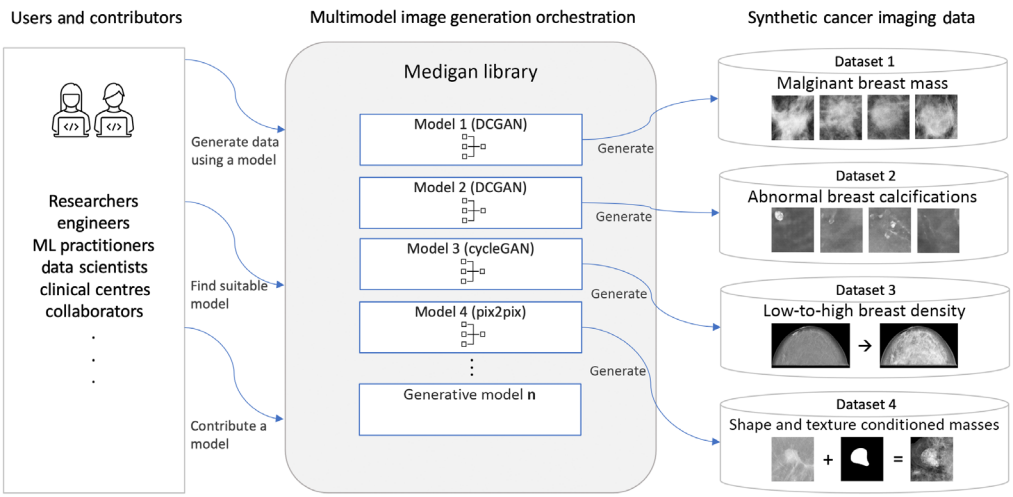

Fig. 3 Architectural overview of *medigan*. Users interact with the library by contributing, searching, and executing generative models, the latter shown here exemplified for mammography image generation with models with IDs 1 to 4 described in Table 3.

3.1 *User Requirements and Design Decisions*

End-user requirement gathering is recommended for the development of trustworthy AI solutions in medical imaging.[75] Therefore, we organized requirement gathering sessions with potential end-users, model contributors, and stakeholders from the EuCanImage Consortium, a large European H2020 project[76] building a cancer imaging platform for enhanced AI in oncology. Upon exploring the needs and preferences of medical imaging researchers and AI developers, respective requirements for the design of *medigan* were formulated to ensure usability and usefulness. For instance, the users articulated a clear preference for a user interface in the format of an importable package as opposed to a graphical user interface (GUI), web application, database system, or API. Table 2 summarizes key requirements and the corresponding design decisions.

3.2 *Software Design and Architecture*

medigan is built with a focus on simplicity and usability. The integration of pretrained models is designed as internal Python package import and offers simultaneously (a) high flexibility to and (b) low code dependency on these generative models. The latter allows the reuse of the same orchestration functions in *medigan* for all model packages.

Using object-oriented programming, the same model_executor class is used to implement, instantiate, and run all different types of generative model packages. To keep the library maintainable and lightweight, and to avoid limiting interdependencies between library code and generative model code, *medigan*'s models are hosted outside the library (on Zenodo) as independent Python modules. To avoid long initialization times upon library import, lazy loading is applied. A model is only loaded and its `model_executor` instance is only initialized if a user specifically requests synthetic data generation for that model. To achieve high cohesion,[79] i.e., keeping the library and its functions specific, manageable, and understandable, the library is structured into several modular components. These include the loosely-coupled `model_executor`, `model_selector`, and `model_contributor` modules.

The `generators` module is inspired by the facade design pattern[80] and acts as a single point of access to all of *medigan*'s functionalities. As single interface layer between users and library, it reduces interaction complexity and provides users with a clear set of readily extendable library functions. Also, the `generators` module increases internal code reusability and allows for combination of functions from other modules. For instance, a single function call can run the generation of samples by the model with the highest FID score of all models found in a keyword search.

Table 2 Overview of the key requirements gathered together with potential end-user alongside the respective design decisions taken toward fulfilling these requirements with *medigan*.

No	End-user requirement	Respective design decision
1	Instead of a GUI tool, *medigan* should be implemented as a platform-independent library importable into users' code.	Implementation of *medigan* as publicly accessible Python package distributed via PyPI.
2	It should support common frameworks for building generative models, e.g., PyTorch,[67] TensorFlow,[77] Keras.[78]	*medigan* is built framework-agnostic treating each model as separate Python package with freedom of choice of framework and dependencies.
3	The library should allow different types of generative models and generation processes.	*medigan* supports any type of data generation model including GANs,[16] VAEs,[21] flow-based,[22–24] diffusion,[25–27] and nondeep learning models.
4	The library should support different types of synthetic data.	*medigan* supports any type of synthetic data ranging from 2D and 3D images to image pairs, masks, and tabular data.
5	Sample generation functions should be easily integrable into diverse user code, pipelines, and workflows.	*medigan*'s generate function can (i) return samples, (ii) generate folders with samples, or (iii) return a model's generate function as callable.
6	User should be able to integrate *medigan* data in AI training via a dataloader.	For each model, *medigan* supports returning a torch dataloader readily integrable in AI training pipelines, combinable with other dataloaders.
7	Despite using large deep learning models, the library should be as lightweight as possible.	Only the user-requested models are downloaded and locally imported. Thus, model dependencies are not part of *medigan*'s dependencies.
8	It should be possible to locally review and adjust a generative model of the library.	After download, a model's code and config are available for end-users to explore and adjust. *medigan* can also load models from local file systems.
9	The library should support both CPU and GPU usage depending on a user's hardware.	Contributed *medigan* models are reviewed and, if need be, enhanced to run on both GPU and CPU.
10	Version and source of the models that the library load should be transparent to the end-user.	Convention of storing *medigan* models on Zenodo, where each model's source code and version history is available.
11	There should be no need to update the version of the *medigan* package each time a new model is contributed.	*medigan* is designed independently of its model packages separately stored on Zenodo. Config updates do not require new *medigan* versions.

Table 2 (*Continued*).

No	End-user requirement	Respective design decision
12	Following,[75] models are contributed in transparently and traceably, allowing quality and reproducibility checks.	Model contribution is traceable via version control. Adding models to *medigan* requires a config change via pull request.
13	The risk that the library downloads models that contain malicious code should be minimized.	Zenodo model uploads receive static DOIs. After verification, unsolicited uploads/changes do not affect *medigan*, which points to specific DOI.
14	License and authorship of generative model contributors should be clearly stated and acknowledged.	Separation of models and library allows freedom of choice of model license and transparent authorship reported for each model.
15	Each generative model in the library should be documented.	Each available model is listed and described in *medigan*'s documentation, in the readme, and also separately in its Zenodo entry.
16	The library should have minimal dependencies on the user side and should run on common end-user systems.	*medigan* has a minimal set of Python dependencies, is OS-independent, and avoids system and third-party dependencies.
17	Contributing models should be simple and at least partially automated.	*medigan*'s contribution workflow automates local model configuration, testing, packaging, Zenodo upload, and issue creation on GitHub.
18	If different models have the same dependency but with different versions, this should not cause a conflict.	Model dependency versions are specified in the config. *medigan*'s generate method can install unsatisfied dependencies, avoiding conflicts.
19	Any model in the library should be automatically tested and results reported to make sure all models work as designed.	On each commit to main, a CI pipeline automatically builds, formats, and lints *medigan* before testing all models and core functions.
20	The library should make the results of the models visible with minimal code required by end-users.	*medigan*'s simple visualization feature allows users to adjust a model's input latent vector for intuitive exploration of output diversity and fidelity.
21	The library should support large synthetic dataset generation on user machines with limited random-access memory.	For large synthetic dataset generation, *medigan* iteratively generates samples via small batches to avoid exceeding users' in-memory storage limits.
22	Users can specify model weights, model inputs, number, and storage location of the synthetic samples.	Diverging from defaults, users can specify (i) weights, (ii) number of samples (iii) return or store, (iv) store location, (v) optional inputs.

3.3 *Model Metadata*

The FID score and all other model information such as dependencies, modality, type, zenodo link, associated publications, and generate function parameters are stored in a single comprehensive model metadata json file. Alongside its searchability, readability, and flexibility, the choice of json as file format is motivated by its extendability to a nonrelational database. As a single source of model information, the *global.json* file consists of an array of model IDs, where under each model id the respective model metadata is stored. Toward ensuring model traceability as recommended by the FUTURE-AI consensus guidelines,[75] each model (on Zenodo) and its global.json metadata (on GitHub) are version-controlled with the latter being structured into the following objects.

i. *execution*: contains the information needed to download, package, and run the model resources.
ii. *selection*: contains model evaluation metrics and further information used to search, compare, and rank models.
iii. *description*: contains general information and main details about the model such as title, training dataset, license, date, and related publications.

This *global.json* metadata file is retrieved, provided, and handled by the `config_manager` module once a user imports the `generators` module. This facilitates rapid access to a model's metadata given its *model_id* and allows one to add new models or model versions to *medigan* via pull request without requiring a new release of the library.

3.4 *Model Search and Ranking*

The number of models in *medigan* is expected to grow over time. Potentially this will lead to the foreseeable issue where users of *medigan* have a large number of models to choose from. Users likely will be uncertain which model best fits their needs depending on their data, modality, use-case, and research problem at hand and would have to go through each model's metadata to find the most suitable model in *medigan*. Hence, to facilitate model selection, the `model_selector` module implements model search and ranking functionalities. This search workflow is shown in Fig. 4 and triggered by running Code Snippet 1.

The `model_selector` module contains a search method that takes search operator (i.e OR, AND, or XOR) and a keyword search values list as parameters and recursively searches through the models' metadata. The latter is provided by the `config_manager` module.

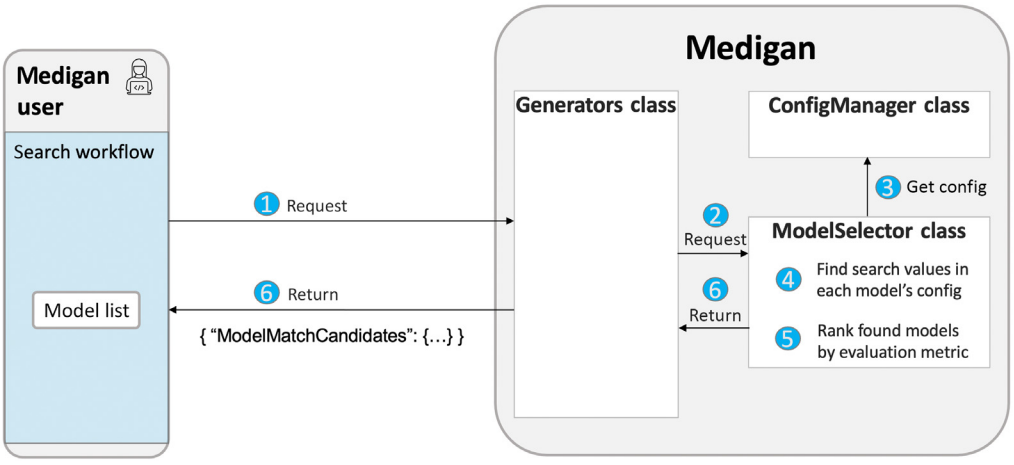

Fig. 4 The search workflow. A user sends a search query (1) to the generators class, which triggers a search (2) via the ModelSelector class. The latter retrieves the *global.json* model metadata/config dict (3), in which it searches for query values finding matching models (4). Next, the matched models are optionally also ranked based on a user-defined performance indicator (5) before being returned as list to the user.

Code Snippet 1: Searching for a model in *medigan*.

```
1. from medigan import Generators # import
2. generators = Generators() # init
3. values=['patches', 'mammography'] # keywords of search query
4. operator='AND' # all keywords are needed for match
5. results = generators.find_model(values, operator)
```

The `model_selector` populates a `modelMatchCandidates` object with `matchedEntry` instances each of which represents a potential model match to the search query. The `modelMatchCandidates` class evaluates which of it is associated model matches should be flagged as true match given the search values and search operator. The method `rank_models_by_performance` compares either all or specified models in medigan by a performance indicator such as FID. This indicator commonly is a metric that correlates with diversity, fidelity, or condition adherence to estimate the quality of generative models and/or the data they generate.[7] The `model_selector` looks up the value for the specified performance indicator in the model metadata and returns a descendingly or ascendingly ranked list of models to the user.

3.5 *Synthetic Data Generation*

Synthetic data generation is *medigan*'s core functionality toward overcoming scarcity of (a) training data and (b) reusable generative model in medical imaging. Posing a low entry barrier for nonexpert users, *medigan*'s `generate` method is both simple and scalable. While a user can run it with only one line of code, it flexibly supports any type of generative model and synthetic data generation process, as illustrated in Table 3 and Fig. 1.

3.5.1 *Generate workflow*

An example of the usage of the `generate` method is shown in Code Snippet 2, which triggers the model execution workflow illustrated in Fig. 5. Further parameters of the generate method allow users to specify the number of samples to be generated (`num_samples`), if samples are returned as a list or stored on a disk (`save_images`), where they are stored (`output_path`), and whether model dependencies are automatically installed (`install_dependencies`). Optional model-specific inputs can be provided via the `**kwargs` parameter. These include for example, (i) a nondefault path to the model weights, (ii) a path to an input image folder for image-to-image translation models, (iii) a conditional input for class-conditional generative models, or (iv) the `input_latent_vector` as commonly used as model input in GANs.

Running the `generate` method triggers the `generators` module to initialize a `model_executor` instance for the user-specified generative model. The model is identified via its *model_id* as unique key in the *global.json* model metadata database, parsed and managed by the `config_manager` module. Using the latter, the `model_executor` checks if the required Python package dependencies are installed, retrieves the Zenodo URL and downloads, unzips, and imports the model package. It further retrieves the name of the internal data generation function inside the model's `__init__.py` script. As final step before calling this function, its parameters and their default values are retrieved from the metadata and combined with user-provided arguments. These user-provided arguments customize the generation process, which enables handling of multiple image generation scenarios. For instance, the aforementioned provision of the input image folder allows users to point to their own images to transform them using *medigan* models that are, e.g., pretrained for cross-modality translation. In the case of large dataset generation, the number of samples indicated by `num_samples` are chunked into smaller-sized batches and iteratively generated to avoid overloading the random-access memory available on the user's machine.

Table 3 Models currently available in *medigan*. Also, computed FID scores for each model in *medigan* are shown. The number of real samples used for FID calculation is indicated by #imgs. The *lower bound* FID_{rr} is computed between a pair of randomly sampled sets of real data (real-real), whereas the *model* FID_{rs} is computed between two randomly sampled sets of real and synthetic data (real-syn). The results for model 7 (Flair, T1, T1c, T2) and 21 (T1, T2) are averaged across modalities.

						$FID_{ImageNet}$[47,52]			
ID	Output	Modality	Model	Size	Training dataset	#imgs	Real-real	Real-syn	r_{FID}
1	Breast calcifications	Mammography	DCGAN	128 × 128	INbreast[81]	1000	33.61	67.60	0.497
2	Breast masses	Mammography	DCGAN[11]	128 × 128	OPTIMAM[82]	1000	28.85	80.51	0.358
3	High/low density breasts	Mammography	CycleGAN[50]	1332 × 800	BCDR[83]	74	65.94	150.16	0.439
4	Breast masses with masks	Mammography	pix2pix	256 × 256	BCDR[83]	199	68.22	161.17	0.423
5	Breast masses	Mammography	DCGAN[14]	128 × 128	BCDR[83]	199	68.22	180.04	0.379
6	Breast masses	Mammography	WGAN-GP[14]	128 × 128	BCDR[83]	199	68.22	221.30	0.308
7	Brain tumors with masks	Cranial MRI	Inpaint GAN [84]	256 × 256	BRATS 2018[85]	1000	30.73	140.02	0.219
8	Breast masses (mal/benign)	Mammography	C-DCGAN	128 × 128	CBIS-DDSM[86]	379	37.56	137.75	0.272
9	Polyps with masks	Endoscopy	PGGAN[51]	256 × 256	HyperKvasir[87]	1000	43.31	225.85	0.192
10	Polyps with masks	Endoscopy	FastGAN[51]	256 × 256	HyperKvasir[87]	1000	43.31	63.99	0.677
11	Polyps with masks	Endoscopy	SinGAN[51]	≈250 × 250	HyperKvasir[87]	1000	43.31	171.15	0.253
12	Breast masses (mal/benign)	Mammography	C-DCGAN	128 × 128	BCDR[83]	199	68.22	205.29	0.332
13	High/low density breasts MLO	Mammography	CycleGAN[50]	1332 × 800	OPTIMAM[82]	358	65.75	101.09	0.650
14	High/low density breasts CC	Mammography	CycleGAN[50]	1332 × 800	OPTIMAM[82]	350	41.61	73.77	0.564
15	High/low density breasts MLO	Mammography	CycleGAN[50]	1332 × 800	CSAW[88]	192	74.96	162.67	0.461
16	High/low density breasts CC	Mammography	CycleGAN[50]	1332 × 800	CSAW[88]	202	42.68	98.38	0.434
17	Lung nodules	Chest x-ray	DCGAN	128 × 128	NODE21[89]	1476	24.34	126.78	0.192
18	Lung nodules	Chest x-ray	WGAN-GP	128 × 128	NODE21[89]	1476	24.34	211.47	0.115
19	Full chest radiograph	Chest x-ray	PGGAN	1024 × 1024	ChestX-ray14[90]	1000	28.74	96.74	0.297
20	Full chest radiograph	Chest x-ray	PGGAN[91]	1024 × 1024	ChestX-ray14[90]	1000	28.33	52.17	0.543
21	Brain scans (T1/T2)	Cranial MRI	CycleGAN[92]	224 × 192	CrossMoDA 2021[93]	1000	24.41	59.49	0.410

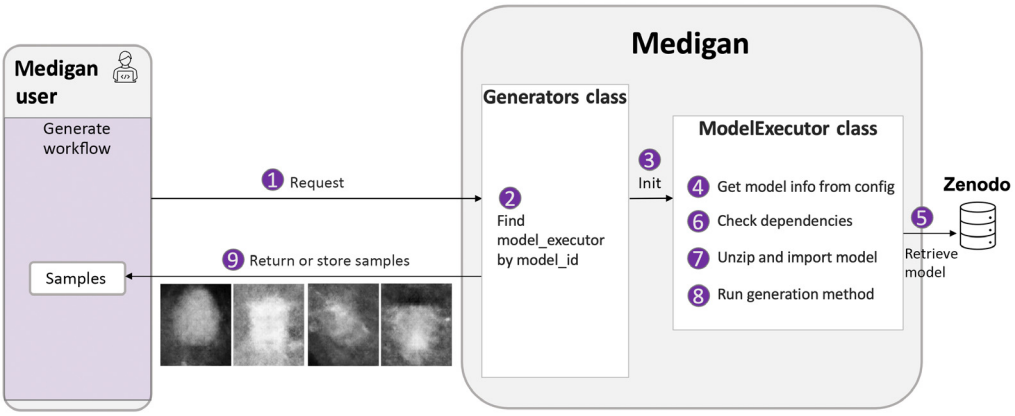

Fig. 5 The generated workflow. A user specifies a *model_id* in a request (1) to the generators class, which checks (2) if the model's ModelExecutor class instance is already initialized. If not, a new one is created (3), which (4) gets the model's config from the *global.json* dict, (5) loads the model (e.g., from *Zenodo*), (6) checks its dependencies, and (7) unzips and imports it, before running its internal generate function (8). Finally, the generated samples are returned to the user.

Code Snippet 2: Executing a *medigan* model for synthetic data generation.

```
1. from medigan import Generators

2. generators = Generators()

# create 100 polyps with masks using model 10 (FASTGAN)

generators.generate(model_id=10, num_samples=100)
```

3.5.2 *Generate workflow extensions*

Apart from storing or returning samples, a callable of the model's internal generate function can be returned to the user by setting `is_gen_function_returned`. This function with prepared but adjustable default arguments enables integration of the generate method into other workflows within *medigan* (e.g., model visualization) or outside of *medigan* (e.g., a user's AI model training). As a further alternative, a torch[67] dataset or dataloader can be returned for any model in medigan running `get_as_torch_dataset` or `get_as_torch_dataloader`, respectively. This further increases the versatility with which users can introduce *medigan*'s data synthesis capabilities into their AI model training and data preprocessing pipelines.

Instead of a user manually selecting a model via *model_id*, a model can also be automatically selected based on the recommendation from the model search and/or ranking methods. For instance, as triggered by Code Snippet 3, the models found in a search for *mammography* are ranked in ascending order based on FID, with the highest ranking model being selected and executed to generate the synthetic dataset.

3.6 *Model Visualization*

To allow users to explore the generative models in *medigan*, a novel model visualization module has been integrated into the library. It allows users to examine how changing inputs like the latent variable z and/or the class conditional label y (e.g., malignant/benign) can affect the generation process. Also, the correlation between multiple model outputs, such as the image and corresponding segmentation mask, can be observed and explored. Figure 6 illustrates an example showing an image-mask sample pair from medigan's polyp generating FastGAN model.[51]

Code Snippet 3: Sequential searching, ranking, and data generation with highest ranked model.

```
1. from medigan import Generators

2. generators = Generators()

3. values = ['mammography'] # keywords for searching

4. metric = 'FID' # metric for ranking

5. generators.find_models_rank_and_generate
   (values=values, metric=metric)
```

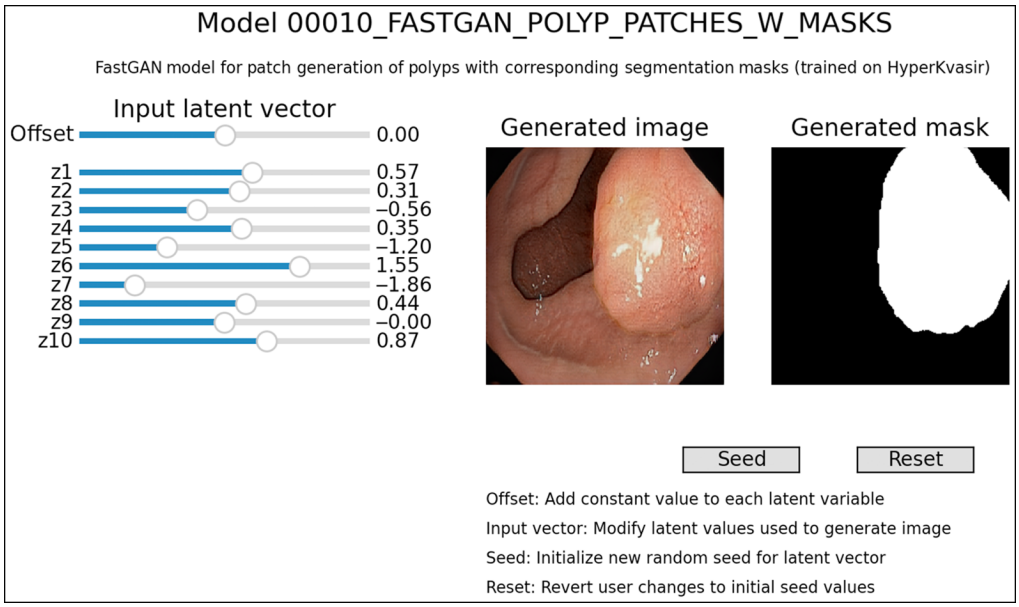

Fig. 6 Graphical user interface of *medigan*'s model visualization tool on the example of model 10, a FastGAN that synthesizes endoscopic polyp images with respective masks.[51] The latent input vector can be adjusted via the sliders, reset via the *Reset* button, and sampled randomly via the *Seed* button.

This depiction of the graphical user interface (GUI) of the model visualization tool can be recreated by running Code Snippet 4.

Internally, the `model_visualizer` module retrieves a model's internal generate method as callable from the `model_executor` and adjusts the input parameters based on user interaction input from the GUI. This interaction further provides insight into a model's performance and capabilities. On one hand, it allows one to assess the fidelity of the generated samples. On the other hand, it also shows the model's captured sample diversity, i.e., as observed output variation over all possible input latent vectors. We leave the automation of manual visual analysis of this output variation to future work. For instance, such future work can use the `model_visualizer` to measure the variance of a reconstruction/perceptual error computed between pairs of images sampled from fixed-distance pairs of latent space vectors $\mathbf{z}$. The slider controls on the left of the interface allow one to change the latent variable, which for this specific model affects, for instance, polyp size, position, and background. As the size of the latent vector $\mathbf{z}$ commonly is relatively large, each n (e.g., 10) variables are grouped into one indexed slider resulting in z_m adjustable latent input variables. The seed button on the right allows one to initialize a new set of latent variables, which results in a new generated image. The reset buttons allows one to revert user's modifications to previous random values.

Code Snippet 4: Visualization of a model in *medigan*.

```
1. from medigan import Generators

2. generators = Generators()

3. n = 10  # grouping latent vector z dimensions by dividing them by 10

4. generators.visualize(model_id=10, slider_grouper=n)
# polyp with mask
```

3.7 *Model Contribution*

A core idea of *medigan* is to provide a platform where researchers can share and access trained models via a standardized interface. We provide in-depth instructions on how to contribute a model to medigan complemented by implementations automating parts of the model contribution process for users. In general, a pretrained model in medigan consists of a Python `__init__.py` and, in case the generation process is based on a machine learning model, a respective checkpoint or weights file. The former needs to contain a synthetic data storage method and a data generation method with a set of standardized parameters described in Sec. 3.5.1. Ideally, a model package further contains a license file, a *metadata.json* and/or a *requirements.txt* file, and a *test.sh* script to quickly verify the model's functionalities. To facilitate creation of these files, *medigan*'s GitHub repository provides model contributors with reusable templates for each of these files.

Keeping the effort of pretrained model inclusion to a minimum, the `generators` module contains a `contribute` function that initializes a `ModelContributor` class instance dedicated to automating the remainder of the model contribution process. This includes automated (i) validation of the user-provided *model_id*; (ii) validation of the path to the model's `__init__.py`; (iii) test of `importlib` import of the model as package; (iv) creation of the model's metadata dictionary; (v) adding the model metadata to *medigan*'s *global.json* metadata; (vi) end-to-end test of model with sample generation via `generators.test_model()`; (vii) upload of zipped model package to Zenodo via API; and (viii) creation of a GitHub issue, which contains the Zenodo link and model metadata, in the *medigan* repository. Being assigned to this GitHub issue, the *medigan* development team is notified about the new model, which can then be added via pull request. Code Snippet 5 shows how a user can run the `contribute` method illustrated in Fig. 7.

3.8 *Model Testing Pipeline*

Each new model contribution is being systematically tested before becoming part of *medigan*. For instance, on each submitted pull request to medigan's GitHub repository, a CI pipeline automatically builds, formats, lints, and tests *medigan*'s codebase. This includes the automatic verification of each model's package, dependencies, compatibility with the interface, and correct functioning of its generation workflow. This allows one to ensure that all models and their metadata in the *global.json* file are available and working in a reproducible and standardized manner.

4 Applications

4.1 *Community-Wide Data Access: Sharing the Essence of Restricted Datasets*

medigan facilitates sharing and reusing trained generative models with the medical research community. On one hand, this reduces the need for researchers to retrain their own similar generative models, which can reduce the extensive carbon footprint[94] of deep learning in medical imaging. On the other hand, this provides a platform for researchers and data owners to share their dataset distribution without sharing the real data points of the dataset. Put differently,

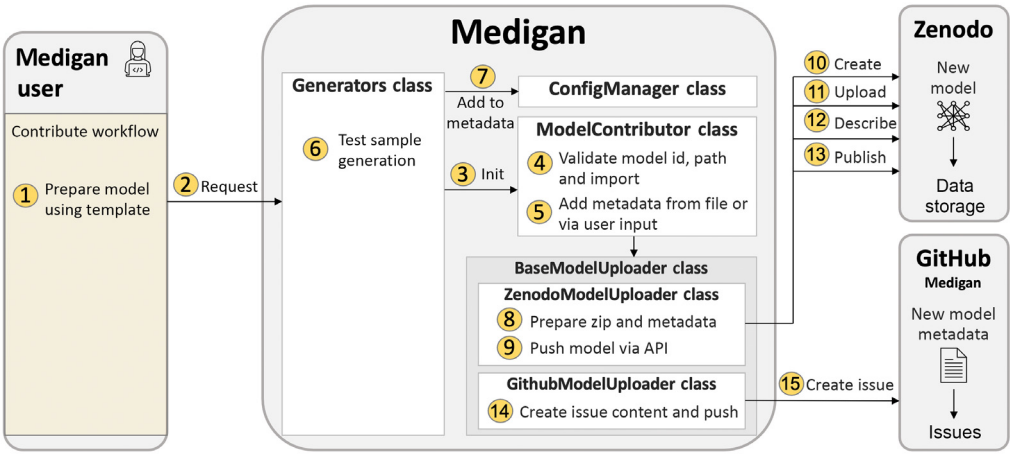

Fig. 7 Model contribution workflow. After model preparation (1), a user provides the model's id and metadata (2) to the generators class to (3) initialize a ModelContributor instance, which (4) validates and (5) extends the metadata. Next, (6) the model's sample generation capability is tested after (7) integration into *medigan*'s *global.json* model metadata. If successful, (8) the model package is prepared and (9–13) pushed to Zenodo via API. Lastly, (14 and 15) a GitHub issue containing the model metadata is created, assigned, and pushed to the *medigan* repository.

sharing generative models trained on (and instead of) patient datasets not only is beneficial as data curation step,[14] but also minimizes the need to share images and personal data directly attributable to a patient. In particular, the latter can be quantifiably achieved when the generative model is trained using a differential privacy guarantee[7,95] before being added to *medigan*. By reducing the barriers posed by data sharing restrictions and necessary patient privacy protection regulation, *medigan* unlocks a new paradigm of medical data sharing via generative models. This places *medigan* at the center toward solving the well-known issue of data scarcity[7,9] in medical imaging.

Apart from that, *medigan*'s generative model contributors benefit from an increased exposure, dissemination, and impact of their work, as their generative models become readily usable by other researchers. As Table 3 illustrates, to date, *medigan* consists of 21 pretrained deep generative models contributed to the community. Among others, these include two conditional DCGAN models, six domain translation CycleGAN models and one mask-to-image pix2pix model. The training data comes from 10 different medical imaging datasets. Various of the

Code Snippet 5: Contribution of a model to *medigan*.

```
1. from medigan import Generators
2. generators = Generators()
3. generators.contribute(
4.     model_id = "00100_YOUR_MODEL", # assign ID
5.     init_py_path ="path/ending/with/__init__.py", # model package root
6.     generate_method_name = "generate", # method inside __init__.py
7.     model_weights_name = "10000",
8.     model_weights_extension = ".pt",
9.     dependencies = ["numpy", "torch"],
10.    zenodo_access_token = "TOKEN", #zenodo.org/account/settings/applications
11.    github_access_token = "TOKEN") #github.com/settings/tokens
```

models were trained on breast cancer datasets including INbreast,[81] OPTIMAM,[82] BCDR,[83] CBIS-DDSM,[86] and CSAW.[88] Models allow one to generate samples of different pixel resolutions ranging from regions-of-interest patches of size 128×128 and 256×256 to full images of 1024×1024 and 1332×800 pixels.

4.2 *Investigating Synthetic Data Evaluation Methods*

A further application of *medigan* is testing the properties of medical synthetic data. For instance, evaluation metrics for generative models can be readily tested in *medigan*'s multiorgan, multimodality, and multimodel synthetic data setting.

Compared to generative modeling, synthetic data evaluation is a less explored research area.[7] In particular, in medical imaging the existing evaluation frameworks, such as the FID[46] or the IS,[17] are often limited in their applicability, as mentioned in Sec. 2.3. The models in *medigan* allow one to compare existing and new synthetic data evaluation metrics and their validation in the field of medical imaging. Multimodel synthetic data evaluation allows one to measure the correlation and statistical significance between synthetic data evaluation metrics and downstream task performance metrics. This enables the assessment of clinical usefulness of generative models on one hand and of synthetic data evaluation metrics on the other hand. In that sense, the metric itself can be evaluated including its variations when measured under different settings, datasets, or preprocessing techniques.

4.2.1 *FID of medigan Models*

We compute the FID to assess the models in *medigan* and report the results in Table 3. We further note that the FID can be computed not only between a synthetic and a real dataset (*rs*) but also between two sets of samples of the real dataset (*rr*). As the FID_{rr} describes the distance within two randomly sampled sets of the real data distribution, it can be used as an estimate of the real data variation and optimal lower bound for the FID_{rs} as shown in Table 3. Given the above, it follows that a high FID_{rr} likely also results in a higher FID_{rs}, which highlights the importance of accounting for the FID_{rr} when discussing the FID_{rs}. To do so, we propose the reporting of a FID ratio r_{FID} to describe the FID_{rs} in terms of the FID_{rr}.

$$r_{\text{FID}}(\text{FID}_{\text{rs}}, \text{FID}_{\text{rr}}) = 1 - \frac{\text{FID}_{\text{rs}} - \text{FID}_{\text{rr}}}{\text{FID}_{\text{rs}}}, \quad r_{\text{FID}} \in [0,1] \subset \mathbb{R}. \tag{5}$$

Assuming $\text{FID}_{\text{rs}} \geq \text{FID}_{\text{rr}}$ bounds r_{FID} between 0 and 1, r_{FID} the simplifies the comparison of FIDs computed using different models and datasets. A r_{FID} close to 1 indicates that much of the FID_{rs} can be explained by the general variation in the real dataset. The code used to compute the FID scores is available at https://github.com/RichardObi/medigan/blob/main/tests/fid.py.

The models in Table 3 yielding the highest ImageNet-based r_{FID} score are the ones with ID 10 (0.677, endoscopy, 256×256, FastGAN), ID 13 (0.650, mammography, 1332×800, CycleGAN), 14 (0.564, mammography, 1332×800, CycleGAN), 20 (0.543, chest x-ray, 1024×1024, PGGAN) and 1 (0.497, mammography, DCGAN, 128×128). This indicates that the r_{FID} does not depend on the modality, nor on the pixel resolution of the synthetic images. Further, neither image-to-image translation (e.g. CycleGAN) nor noise-to-image models (e.g., PGGAN, DCGAN, FastGAN) seem to have a particular advantage for achieving higher r_{FID} results.

The flow chart in Fig. 8 provides further insight into the comparison between the lower bound FID_{rr} and the model FID_{rs}. The red trend line shows a positive correlation between the FID_{rr} and FID_{rs}, which corroborates our previous assumption that a higher model FID_{rs} is to be expected given a higher lower bound FID_{rr}. Hence, for increased transparency, we motivate further studies to routinely report the lower bound FID_{rr} and the FID ratio r_{FID} apart from the model FID_{rs}. The three-channel RGB endoscopic images represented by orange dots have an FID_{rr} comparable with their grayscale radiologic counterparts. However, both chest x-ray datasets ChestX-ray14[90] and Node21[89] represented by green dots show a slightly lower FID_{rr} than other modalities. The model FID_{rs} shows a high variation across models without readily observable dependence on modality, generative model, or image size.

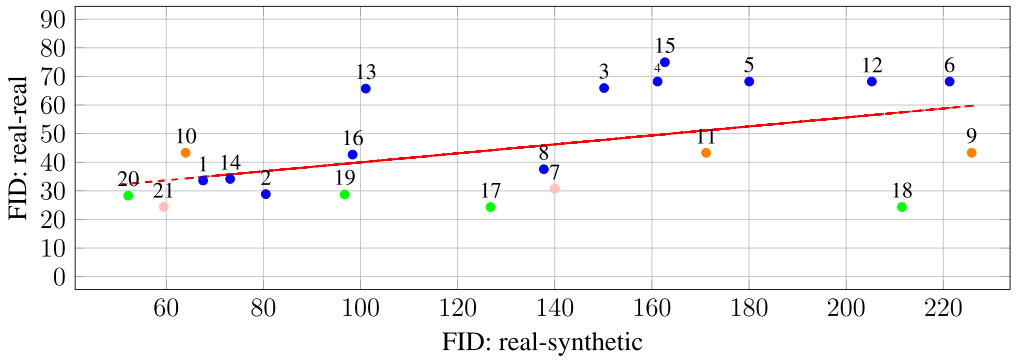

Fig. 8 Scatter plot illustrating the FID_{rs} of *medigan*'s models (real-synthetic) compared to the lower bound FID_{rr} between two sets of the model's respective training dataset (real-real). The lower bound can represent an optimally achievable model and, as such, facilitates interpretation. Each model is represented by a dot below its model ID. The dots' color encoding depicts model modality, where blue: mammography, orange: endoscopy, green: chest x-ray, and pink: brain MRI. The red regression line illustrates the trend across all data points/models.

4.2.2 *Analysing potential sources of bias in FID*

The popular FID metric is computed based on the features of an Inception classifier (e.g., v1,[48] v3[47]) trained on ImageNet[52]—a database of natural images inherently different from the domain of medical images. This potentially limits the applicability of the FID to medical imaging data. Furthermore, the FID has been observed to vary based on the input image resizing methods and ImageNet backbone feature extraction model types.[31] Based on this, we further hypothesize a susceptibility of the FID to variation due to (a) different backbone feature extractor weights and random seed initializations, (b) different medical and nonmedical backbone model pretraining datasets, (c) different image normalization procedures for real and synthetic dataset, (d) nuances between different frameworks and libraries used for FID calculation, and (f) the dataset sizes used to compute the FID.

Such variations can obstruct a reliable comparison of synthetic images generated by different generative models. Illustrating the potential of *medigan* to analyze such variations, we report and experiment with the FID. In particular, we subject the FID to variations in (i) the pretraining dataset of its backbone feature extractor and by (ii) testing the effects of image normalization across a set of *medigan* models. We experiment with the Inception v3 model trained on the recent RadImageNet dataset[72] released as radiology-specific alternative to the ImageNet database.[52] The RadImageNet-pretrained Inception v3 model weights we used are available at https://github.com/BMEII-AI/RadImageNet. We further compute the FID_{rs} and FID_{rr} with and without normalization to analyze the respective impact on results.

In Table 4, the FID results are summarized allowing for cross-analysis between variations due to image normalization and/or due to the pretraining dataset of the FID feature extraction model. We observe generally lower FID values (1.15 to 7.32) for RadImageNet compared to ImageNet as FID model pretraining datasets (52.17 to 225.85). To increase FID comparability, we compute, as before, the FID ratio r_{FID}. The RadImageNet-based model results in notably lower r_{FID} values for both normalized and non-normalized images. Notably, an exception to this are models with ID 5 (mammography, 128×128, DCGAN) and 6 (mammography, 128×128, WGAN-GP) achieving respective RadImageNet-based r_{FID} scores of 0.593 and 0.550. In general, the RadImageNet-based model seems more robust at detecting if two sets of data originate from the same distribution resulting in low FID_{rr} values. Overall, for most models, the FID is explained only by a limited amount by the variation in the real dataset and $r_{FID} < 0.7$ for all ImageNet and RadImageNet-based FIDs. The scatter plot in Fig. 9 further compares the RadImagnet-based FID with the ImageNet-FID for the models from Table 4. Noticeably, the difference between non-normalized and normalized images is surprisingly high for several models for both ImageNet and RadImageNet FIDs (e.g., models with IDs 6 and 8) while negligible for others (e.g., models with ID 1, 10, 13-16, and 19-21). Another observation is the relatively modest correlation between RadImageNet and ImageNet FID indicated by the slope of the red

Table 4 Normalized (left) and non-normalized (right) FID scores. This table measures the normalization impact on FID scores based on a promising set of *medigan*'s deep generative models. Synthetic samples were randomly-drawn for each model matching the number of available real samples. The *lower bound* FID_{rr} is computed between a pair of randomly sampled sets of real data (real-real), whereas the *model* FID_{rs} is computed between two randomly sampled sets of real and synthetic data (real-syn). The results for model 7 (Flair, T1, T1c, T2) and 21 (T1, T2) are averaged across modalities.

		Normalized images						Non-Normalized images					
		$FID_{ImageNet}$[47,52]			$FID_{RadImageNet}$[72]			$FID_{ImageNet}$[47,52]			$FID_{RadImageNet}$[72]		
ID	Dataset	real-real	real-syn	r_{FID}	real-real	real-syn	r_{FID}	real-real	real-syn	r_{FID}	real-real	real-syn	r_{FID}
1	Inbreast	33.61	67.60	0.497	0.25	1.27	0.197	28.59	66.76	0.428	0.29	1.15	0.252
2	Optimam	28.85	80.51	0.358	0.22	6.19	0.036	28.75	77.95	0.369	0.33	4.11	0.080
3	BCDR	65.94	150.16	0.439	0.80	3.00	0.265	66.25	149.33	0.444	0.80	3.10	0.259
5	BCDR	68.22	180.04	0.379	0.99	1.67	0.593	64.45	174.38	0.370	0.87	4.04	0.215
6	BCDR	68.22	221.30	0.308	0.99	1.80	0.550	64.45	206.57	0.312	0.87	2.95	0.295
7	BRATS 2018	30.73	140.02	0.219	0.07	5.31	0.012	30.73	144.00	0.215	0.07	6.53	0.010
8	CBIS-DDSM	37.56	137.75	0.272	0.46	3.05	0.151	32.06	91.09	0.352	0.36	6.58	0.055
10	HyperKvasir	43.31	63.99	0.677	0.11	7.32	0.015	43.31	64.01	0.677	0.11	7.33	0.015
12	BCDR	68.22	205.29	0.332	0.99	5.69	0.080	64.45	199.50	0.323	0.87	4.25	0.205
13	OPTIMAM	65.75	101.01	0.650	0.17	1.14	0.153	65.83	101.15	0.651	0.18	1.10	0.163
14	OPTIMAM	41.61	73.77	0.564	0.16	0.83	0.190	41.71	74.03	0.563	0.15	0.81	0.184
15	CSAW	74.96	162.67	0.461	0.31	4.07	0.076	73.62	165.53	0.445	0.19	3.71	0.051
16	CSAW	42.68	98.38	0.439	0.38	2.71	0.142	42.50	99.81	0.426	0.22	2.82	0.077
19	ChestX-ray14	28.75	96.74	0.297	0.19	0.77	0.243	28.75	96.78	0.297	0.19	0.66	0.286
20	ChestX-ray14	28.33	52.17	0.543	0.20	2.83	0.071	28.33	52.38	0.541	0.20	2.59	0.077
21	CrossMoDA	24.41	59.49	0.410	0.02	1.45	0.014	24.41	60.11	0.406	0.02	1.40	0.014

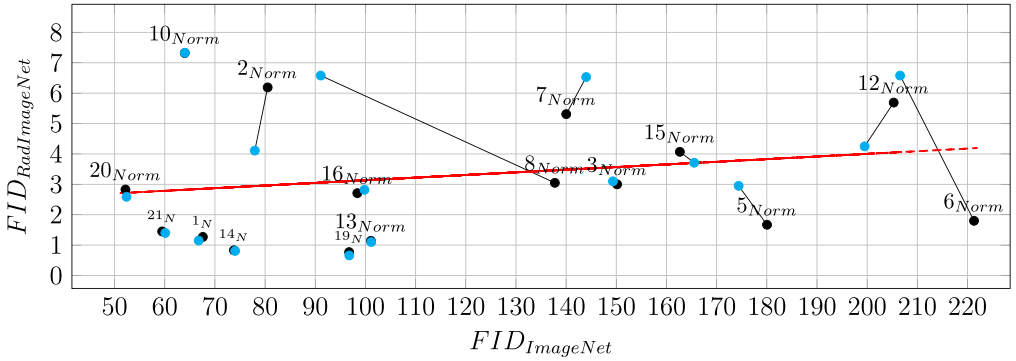

Fig. 9 Scatter plot demonstrating the FID_{rs} (real-synthetic) of *medigan* models from Table 4. The FID_{rs} is based on the features of two different inception classifiers,[47] one trained on ImageNet[52] (*x*-axis) and the other trained on RadImageNet[72] (*y*-axis). Each model is represented by a dot below its model ID. A black dot indicates an FID calculated from normalized (Norm/*N*) images, e.g., with pixel values scaled between 0 and 1, as opposed to a blue dot indicating an FID calculated from images without previous normalization. The dots that correspond to the same model IDs (normalized and non-normalized) are connected via black lines. The red regression line illustrates the trend across all data points.

regression line. Counterexamples for this correlation include model 2 (normalized), which has a low ImageNet-based FID (80.51) compared to a high RadImageNet-based FID (6.19), and model 6 (normalized), which, in contrast, has a high ImageNet-based FID (221.30) and a low RadImageNet-based FID (1.80). With a low ImageNet-based FID (63.99), but surprisingly high RadImageNet-based FID (7.32), model 10 (both normalized and non-normalized) is a further counterexample. The example of model 10 is of particular interest, as it indicates limited applicability of the Radiology-specific RadImageNet-based FID for out-of-domain data, such as three-channel endoscopic images.

Given the demonstrated high impact of backbone model training set and image normalization on FID, it is to be recommended that studies specify the exact model used for FID calculation and any applied data preprocessing and normalization steps. Further, where possible, reporting the RadImageNet-based FID allows for reporting a radiology domain-specific FID. The latter is seemingly less susceptible to variation in the real datasets than the ImageNet-based FID while also being capable of capturing other, potentially complementary, patterns in the data.

4.3 *Improving Clinical Medical Image Analysis*

A high-impact clinical application of synthetic data is the improvement of clinical downstream task performance such as classification, detection, segmentation, or treatment response estimation. This can be achieved by using image synthesis for data augmentation, domain adaptation, and data curation (e.g., artifact removal, noise reduction, super-resolution)[7,63] to enhance the performance of clinical decision support systems such as computer-aided diagnosis (CADx) and detection (CADe) software.

In Table 5, the capability of improving the clinical downstream task performance is demonstrated for various *medigan* models and modalities. Downstream task models trained on a combination of real and synthetic imaging data achieve promising results surpassing the alternative results achieved from training only on real data. The results are taken from the respective publications[11,14,50,84] and indicate that image synthesis can further improve the promising performance demonstrated by deep learning-based CADx and CADe systems, e.g., in mammography[96] and brain MRI.[85] For downstream task evaluation, we generally note the importance of avoiding data leakage between training, validation, and test sets by training the

Table 5 Examples of the impact of synthetic data generated by *medigan* models on downstream task performance. Based on real test data, we compare the performance metrics of a model trained *only on real* data with a model trained on real data *augmented with synthetic* data. The metrics are taken from the respective publications describing the models.

ID	Test dataset	Task	Metric	Trained on real	Real + synthetic
2	OPTIMAM	Mammogram patch classification[11]	F1	0.90	0.96
3	BCDR	Mammogram mass detection[50]	FROC AUC	0.83	0.89
5	BCDR	Mammogram patch classification[14]	F1	0.891	0.920
5	BCDR	Mammogram patch classification[14]	AUROC	0.928	0.959
5	BCDR	Mammogram patch classification[14]	AUPRC	0.986	0.992
6	BCDR	Mammogram patch classification[14]	F1	0.891	0.969
6	BCDR	Mammogram patch classification[14]	AUROC	0.928	0.978
6	BCDR	Mammogram patch classification[14]	AUPRC	0.986	0.996
7	BRATS 2018	Brain tumor segmentation[84]	Dice	0.796	0.814
14	OPTIMAM	Mammogram mass detection[50]	FROC AUC	0.83	0.85
15	OPTIMAM	Mammogram mass detection[50]	FROC AUC	0.83	0.85

Table 6 Examples of the impact of synthetic data generated by *medigan* models on downstream task performance. Based on real test data, we compare the performance metrics of a model trained *only on real* data with a model trained *only on synthetic* data. The metrics are taken from the respective publications describing the models. *n.a.* refers to the case where only synthetic data can be used, as no annotated real training data is available.

ID	Test dataset	Task	Metric	Trained on real	Trained on synthetic
4	BCDR	Mammogram mass segmentation	Dice	0.865	0.737
11	HyperKvasir	Polyp segmentation[51]	Dice loss	0.112	0.137
11	HyperKvasir	Polyp segmentation[51]	IoU	0.827	0.798
11	HyperKvasir	Polyp segmentation[51]	F-Score	0.888	0.863
20	ChestX-ray14	Lung disease classification[91]	AUROC	0.947	0.878
21	CrossMoDA	Brain tumor segmentation[92]	Dice	n.a.	0.712
21	CrossMoDA	Cochlea segmentation[92]	Dice	n.a.	0.478

generative model either on only the dataset partition used to train the respective downstream task model (e.g., IDs 2, 3, 7, 14, 15) or to train the generative models on an entirely different dataset (e.g., IDs 5, 6).

The approaches displayed in Table 6 represent the application, where synthetic data is used instead of real data to train downstream task models. Despite an observable performance decrease when training on synthetic data only, the results[51,91,92] demonstrate the usefulness of synthetic data if none or only limited real training data is available or shareable. For example, if labels or annotations in a target domain are scarce but present in a source domain, a generative model can translate annotated data from the source domain to the target domain to enable supervised training of downstream task models.[92,93]

5 Discussion and Future Work

In this work, we introduced *medigan*, an open-source Python library, which allows one to share pretrained generative models for synthetic medical image generation. The package is easily integrable into other packages and tools, including commercial ones. Synthetic data can enhance the performance, capabilities, and robustness of data-hungry deep learning models as well as to mitigate common issues such as domain shift, data scarcity, class imbalance, and data privacy restrictions. Training one's own generative network can be complex and expensive since it requires a considerable amount of time, effort, specific dedicated hardware, carbon emissions, as well as knowledge and applied skills in generative AI. An alternative and complementary solution is the distribution of pretrained generative models to allow their reuse by AI researchers and engineers worldwide.

medigan can help to reduce the time to run synthetic data experiments and can readily be added as a component, e.g., as a dataloader as discussed in Sec. 3.5.2, in AI training pipelines. As such, the generated data can be used to improve supervised learning models as described in Sec. 4.3 via training or fine-tuning but can also serve as plug-and-play data source for self/semisupervised learning, e.g., to pretrain clinical downstream task models.

Furthermore, studies that use additional synthetic training data for training deep learning models often do not report all the specifics about their underlying generative model.[7,75] Within *medigan*, each generative model is documented, openly accessible, and reusable. This increases the reproducibility of studies that use synthetic data and makes it more transparent where the data or parts thereof originated from. This can help to achieve the traceability objectives outlined in the FUTURE-AI consensus guiding principles toward AI trustworthiness in medical imaging.[75] *medigan*'s currently 21 generative models are illustrated in Table 3 and developed and validated by AI researchers and/or specialized medical doctors. Furthermore, each model contains traceable[75] and version-controlled metadata in medigan's *global.json* file, as outlined in

Sec. 3.3. The searchable (see Sec. 3.4) metadata allows one to choose a suitable model for a user's task at hand and includes, among others, the dataset used during the training process, the trained date, publication, modality, input arguments, model types, and comparable evaluation metrics.

To assess model suitability, users are recommended to first (i) ensure the compatibility between their planned downstream task (e.g., mammogram region-of-interest classification) and a candidate medigan model (e.g., mammogram region-of-interest generator). Second, (ii) a user's real (test) data and the model's synthetic data should be compatible corresponding, for instance, in domain, organ, or disease manifestation. If the awareness of the domain shifts between real and synthetic data remains limited after this qualitative analysis, (iii) a quantitative assessment (e.g., via FID) is recommended. Finally, (iv) it is to be assessed if a downstream task improvement is plausible. This depends, among others, on the tested scenario and the task at hand, but also on the amount, domain, task specificity and quality of the available real data, and the generative model's capabilities as indicated by its reported evaluation metrics from previous studies. If a positive impact of synthetic data on downstream task performance is plausible, users are recommended to proceed toward empirical verification.

The exploration and multimodel evaluation of the properties of generative models and synthetic data is a further application of *medigan*. *medigan*'s visualization tool (see Sec. 3.6) intuitively allows the user to explore and adjust the input latent vector of generative models to visually evaluate, e.g., its inherent diversity and condition adherence[7] (i.e., how well does a given mask or label fit the generated image). The evaluation of synthetic data by human experts, such as radiologists, is a costly and time-consuming task, which motivates the usage of automated metric-based evaluation such as the FID. Our multimodel analysis reveals sources of bias in FID reporting. We show the susceptibility of FID to vary substantially based on changes in input image normalization or in the choice of the pretraining dataset of the FID feature extractor. This finding highlights the need to report the specific models, preprocessing, and implementations used to compute the FID alongside the FID ratio r_{FID} proposed in Sec. 4.2.1 to account for the variation immanent in the real dataset. With *medigan* model experiments demonstrably leading to insights in synthetic data evaluation, future research can use *medigan* as a tool to accelerate, test, analyze, and compare new synthetic data and generative model evaluation and exploration techniques.

5.1 *Legal Frameworks for Sharing of Synthetic and Real Patient Data*

Many countries have enacted regulations that govern the use and sharing of data related to individuals. The two most recognized legal frameworks are the Health Insurance Portability and Accountability Act (HIPAA)[97] from the United States (U.S.) and the General Data Protection Regulation (GDPR)[98] from the European Union (E.U.). These regulations govern the use and disclosure of individuals' protected health information (PHI) and assures individuals' data is protected while allowing use for providing quality patient care.[99–102]

Conceptually, synthetic data is not real data about any particular individual and conversely to real data, synthetic data can be generated at high volumes and potentially shared without restriction. In this sense, under both GDPR and HIPAA regulation, the rules govern the use of real data for the generation and evaluation of synthetic datasets, as well as the sharing of the original dataset. However, once fully synthetic data is generated, this new dataset falls outside the scope of the current regulations based on the argument that there is no direct correlation between the original subjects and the synthetic subjects. A common interpretation is that as long as the real data remains in a secure environment during the generation of synthetic data, there is little to no risk to the original subjects.[103]

As a consequence, the use of synthetic data can help prevent researchers from inadvertently using and possibly exposing patients identifiable data. Synthetic data can also lessen the controls imposed by Institutional Review Boards (IRBs) and based on international regulations by ensuring data is never mapped to real individuals.[104] There are multiple methods of generating synthetic data, some of which include building models from real data, which can create a set statistically similar to real data. How similar the synthetic data is to real-world data often defines its "utility," which will vary depending on the synthesis methods used and the needs of the study at hand. If the utility of the synthetic data is high enough then evaluation results are expected to

be similar to those that use real data.[103] Being built based on real data, a common concern is patient reidentification and leaking of patient-specific features in generative models.[7,15] Despite the arguably permissive aforementioned regulations, deidentification[63] of the training data prior to generative model training is to be recommended. This can minimize the possibility of generative models leaking sensitive patient data during inference and after sharing. A further recommended and mathematically-proven tool for privacy preservation is differential privacy (DP).[95] DP can be included in the training of deep generative model, among other setups, by adding DP noise to the gradients.

5.2 *Expansion of Available Models*

In the future, further generative models across medical imaging disciplines, modalities, and organs can be integrated into medigan. The capabilities of additional models can range from privatising or translating the user's data from one domain to another, balancing or debiasing imbalanced datasets, reconstructing, denoising or removing artifacts in medical images, or resizing images, e.g., using image super-resolution techniques. Despite *medigan*'s current focus on models based on GANs,[16] the inclusion of different additional types of generative models is desirable and will enable insightful comparisons. In particular, this is to be further emphasized considering the recent successes of diffusion models,[25–27] variational autoencoders,[21] and normalizing flows[22–24] in the computer vision and medical imaging[105–107] domains. Before integrating and testing a new model via the pipeline described in Sec. 3.8, we assess whether a model is to become a candidate for inclusion into medigan. This threefold assessment is based on the SynTRUST framework[7] and reviews whether (1) the model is well-documented (e.g., in a respective publication), (2) the model or its synthetic data is applicable to a task of clinical relevance, and (3) whether the model has been methodically validated.

5.3 *Synthetic DICOM Generation*

Since the dominant data format used for medical imaging is Digital Imaging and Communications in Medicine (DICOM), we plan to enhance medigan by integrating the generation of DICOM compliant files. DICOM consists of two main components, pixel data and the DICOM header. The latter can be described as an embedded dataset rich with information related to the pixel data such as the image sequence, patient, physicians, institutions, treatments, observations, and equipment.[63] Future work will explore combining our GAN generated images with synthetic DICOM headers. The latter will be created from the same training images from which the *medigan* models are trained to create synthetic DICOM data with high statistical similarity to real-world data. In this regard, a key research focus will be the creation of an appropriate and DICOM-compliant description of the image acquisition protocol for a synthetic image. The design and development of an open-source software package for generating DICOM files based on synthesized DICOM headers associated to (synthetic) images will extend prior work[108] that demonstrated the generation of synthetic headers for the purpose of evaluating deidentification methods.

6 Conclusion

We presented the open-source *medigan* package, which helps research in medical imaging to rapidly create synthetic datasets for a multitude of purposes such as AI model training and benchmarking, data augmentation, domain adaptation, and intercentre data sharing. *medigan* provides simple functions and interfaces for users, allowing one to automate generative model search, ranking, synthetic data generation, and model contribution. By reuse and dissemination of existing generative models in the medical imaging community, *medigan* allows researchers to speed up their experiments with synthetic data in a reproducible and transparent manner.

We discuss three key applications of *medigan*, which include (i) sharing of restricted datasets, (ii) improving clinical downstream task performance, and (iii) analyzing the properties of generative models, synthetic data, and associated evaluation metrics. Ultimately, the aim of *medigan* is to contribute to benefiting patients and clinicians, e.g., by increasing the performance and robustness of AI models in clinical decision support systems.

Disclosures

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Acknowledgments

We would like to thank all model contributors, such as Alyafi et al.,[11] Szafranowska et al.,[14] Thambawita et al.,[51] Kim et al.,[84] Segal et al.,[91] Joshi et al.,[92] and Garrucho et al.[50] This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements No. 952103 and No. 101057699. Eloy García and Kaisar Kushibar hold the Juan de la Cierva fellowship from the Ministry of Science and Innovation of Spain with reference numbers FJC2019-040039-I and FJC2021-047659-I, respectively.

Data, Materials, and Code Availability

medigan is a free Python (v3.6+) package published under the MIT license and distributed via the Python Package Index (https://pypi.org/project/medigan/). The package is open-source and invites the community to contribute on GitHub (https://github.com/RichardObi/medigan). A detailed documentation of *medigan* is available (https://medigan.readthedocs.io/en/latest/) that contains installation instructions, the API reference, a general description, code examples, a testing guide, a model contribution user guide, and documentation of the generative models available in medigan.

References

1. C. Martin-Isla et al., "Image-based cardiac diagnosis with machine learning: a review," *Front. Cardiovasc. Med.* **7**, 1 (2020).
2. R. Aggarwal et al., "Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis," *NPJ Digital Med.* **4**(1), 1–23 (2021).
3. X. Liu et al., "A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis," *Lancet Digital Health* **1**(6), e271–e297 (2019).
4. J. Schlemper et al., "A deep cascade of convolutional neural networks for MR image reconstruction," *Lect. Notes Comput. Sci.* **10265**, 647–658 (2017).
5. E. Ahishakiye et al., "A survey on deep learning in medical image reconstruction," *Intell. Med.* **1**(03), 118–127 (2021).
6. N. Tajbakhsh et al., "Embracing imperfect datasets: a review of deep learning solutions for medical image segmentation," *Med. Image Anal.* **63**, 101693 (2020).
7. R. Osuala et al., "Data synthesis and adversarial networks: a review and meta-analysis in cancer imaging," *Med. Image Anal.* **84**, 102704 (2023).
8. C. Jin et al., "Predicting treatment response from longitudinal images using multi-task deep learning," *Nat. Commun.* **12**, 1851 (2021).
9. W. L. Bi et al., "Artificial intelligence in cancer imaging: clinical challenges and applications," *CA: Cancer J. Clin.* **69**(2), 127–157 (2019).
10. F. Prior et al., "Open access image repositories: high-quality data to enable machine learning research," *Clin. Radiol.* **75**(1), 7–12 (2020).
11. B. Alyafi, O. Diaz, and R. Marti, "DCGANs for realistic breast mass augmentation in x-ray mammography," *Proc. SPIE* **11314**, 1131420 (2020).
12. X. Yi, E. Walia, and P. Babyn, "Generative adversarial network in medical imaging: a review," *Med. Image Anal.* **58**, 101552 (2019).
13. J. M. Wolterink et al., "Deep MR to CT synthesis using unpaired data," *Lect. Notes Comput. Sci.* **10557**, 14–23 (2017).
14. Z. Szafranowska et al., "Sharing generative models instead of private data: a simulation study on mammography patch classification," *Proc. SPIE* **12286**, 122860Q (2022).
15. T. Stadler, B. Oprisanu, and C. Troncoso, "Synthetic data–anonymisation groundhog day," in *31st USENIX Secur. Symp. (USENIX Security 22)*, pp. 1451–1468 (2022).

16. I. Goodfellow et al., "Generative adversarial nets," in *Adv. Neural Inf. Process. Syst.*, pp. 2672–2680 (2014).
17. T. Salimans et al., "Improved techniques for training GANs," in *Adv. Neural Inf. Process. Syst. 29*, pp. 2234–2242 (2016).
18. L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?," in *Int. Conf. Mach. Learn.*, PMLR, pp. 3481–3490 (2018).
19. S. Arora, A. Risteski, and Y. Zhang, "Do GANs learn the distribution? Some theory and empirics," in *Int. Conf. Learn. Represent.* (2018).
20. L. Ruthotto and E. Haber, "An introduction to deep generative modeling," *GAMM-Mitteilungen* **44**(2), e202100008 (2021).
21. D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," arXiv:1312.6114 (2013).
22. D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Int. Conf. Mach. Learn.*, PMLR, pp. 1530–1538 (2015).
23. L. Dinh, D. Krueger, and Y. Bengio, "Nice: non-linear independent components estimation," arXiv:1410.8516 (2014).
24. L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," arXiv:1605.08803 (2016).
25. J. Sohl-Dickstein et al., "Deep unsupervised learning using nonequilibrium thermodynamics," in *Int. Conf. Mach. Learn.*, PMLR, pp. 2256–2265 (2015).
26. Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *Adv. Neural Inf. Process. Syst. 32* (2019).
27. J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Adv. Neural Inf. Process. Syst. 33*, pp. 6840–6851 (2020).
28. M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Int. Conf. Mach. Learn.*, PMLR, pp. 214–223 (2017).
29. I. Gulrajani et al., "Improved training of wasserstein gans," arXiv:1704.00028 (2017).
30. B. Liu et al., "Towards faster and stabilized gan training for high-fidelity few-shot image synthesis," in *Int. Conf. Learn. Represent.* (2020).
31. M. Kang, J. Shin, and J. Park, "StudioGAN: a taxonomy and benchmark of GANs for image synthesis," arXiv:2206.09479 (2022).
32. A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv:1511.06434 (2015).
33. T. Karras et al., "Progressive growing of gans for improved quality, stability, and variation," arXiv:1710.10196 (2017).
34. M. Mirza and S. Osindero, "Conditional generative adversarial nets," arXiv:1411.1784 (2014).
35. A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Int. Conf. Mach. Learn.*, PMLR, pp. 2642–2651 (2017).
36. P. Isola et al., "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 1125–1134 (2017).
37. J.-Y. Zhu et al., "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 2223–2232 (2017).
38. Y. Choi et al., "Stargan: unified generative adversarial networks for multi-domain image-to-image translation," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 8789–8797 (2018).
39. T. Park et al., "Semantic image synthesis with spatially-adaptive normalization," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 2337–2346 (2019).
40. V. Sushko et al., "OASIS: only adversarial supervision for semantic image synthesis," *Int. J. Comput. Vision* **130**(12), 2903–2923 (2022).
41. T. R. Shaham, T. Dekel, and T. Michaeli, "Singan: learning a generative model from a single natural image," in *Proc. IEEE/CVF Int. Conf. Comput. Vision*, pp. 4570–4580 (2019).
42. D. Korkinof et al., "Perceived realism of high resolution generative adversarial network derived synthetic mammograms," *Radiol.: Artif. Intell.* **3**, e190181 (2020).
43. B. Alyafi et al., "Quality analysis of DCGAN-generated mammography lesions," *Proc. SPIE* **11513**, 115130B (2020).
44. A. Borji, "Pros and cons of GAN evaluation measures," *Comput. Vision Image Understanding* **179**, 41–65 (2019).

45. A. Borji, "Pros and cons of GAN evaluation measures: new developments," *Comput. Vision Image Understanding* **215**, 103329 (2022).
46. M. Heusel et al., "GANs trained by a two time-scale update rule converge to a local nash equilibrium," *Adv. Neural Inf. Process. Syst.* **30** (2017).
47. C. Szegedy et al., "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 2818–2826 (2016).
48. C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 1–9 (2015).
49. Z. Wang et al., "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.* **13**(4), 600–612 (2004).
50. L. Garrucho et al., "High-resolution synthesis of high-density breast mammograms: application to improved fairness in deep learning based mass detection," *Front. Oncol.* **12**, 1044496 (2022).
51. V. Thambawita et al., "SinGAN-Seg: synthetic training data generation for medical image segmentation," *PLoS One* **17**(5), e0267976 (2022).
52. J. Deng et al., "Imagenet: a large-scale hierarchical image database," in *IEEE Conf. Comput. Vision and Pattern Recognit.*, IEEE, pp. 248–255 (2009).
53. accel brain, "Generative adversarial networks library: Pygan," 2021, https://github.com/accel-brain/accel-brain-code/tree/master/Generative-Adversarial-Networks/.
54. A. Pal and A. Das, "TorchGAN: a flexible framework for gan training and evaluation," *J. Open Source Software* **6**(66), 2606 (2021).
55. J. Herzer and T. NeuerRadu, "vegans," 2021, https://github.com/unit8co/vegans/.
56. NVIDIA, "Imaginaire," 2021, https://github.com/NVlabs/imaginaire.
57. J. Shor, "Tensorflow-GAN (TF-GAN): tooling for gans in tensorflow," 2022, https://github.com/tensorflow/gan.
58. E. Linder-Norén, "Keras-GAN: Pytorch implementations of generative adversarial networks," 2021, https://github.com/eriklindernoren/PyTorch-GAN.
59. E. Linder-Norén, "Keras-GAN: Keras implementations of generative adversarial networks," 2022, https://github.com/eriklindernoren/Keras-GAN.
60. K. S. Lee and C. Town, "Mimicry: towards the reproducibility of GAN research," arXiv:2005.02494 (2020).
61. T. Wolf et al., "Transformers: state-of-the-art natural language processing," in *Proc. 2020 Conf. Empirical Methods in Nat. Language Process.: Syst. Demonstrations*, pp. 38–45 (2020, October).
62. M. Kahng et al., "GAN lab: understanding complex deep generative models using interactive visual experimentation," *IEEE Trans. Vision Comput. Graphics* **25**(1), 310–320 (2018).
63. O. Diaz et al., "Data preparation for artificial intelligence in medical imaging: a comprehensive guide to open-access platforms and tools," *Phys. Med.* **83**, 25–37 (2021).
64. F. Pérez-Garca, R. Sparks, and S. Ourselin, "TorchIO: a Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning," *Comput. Methods Prog. Biomed.* **208**, 106236 (2021).
65. C. M. Moore et al., "CleanX: a Python library for data cleaning of large sets of radiology images," *J. Open Source Software* **7**(76), 3632 (2022).
66. M. J. Cardoso et al., "MONAI: an open-source framework for deep learning in healthcare," arXiv:2211.02701 (2022).
67. A. Paszke et al., "Pytorch: an imperative style, high-performance deep learning library," in *Adv. Neural Inf. Process. Syst. 32*, H. Wallach et al., Eds., pp. 8024–8035, Curran Associates, Inc. (2019).
68. E. Gibson et al., "Niftynet: a deep-learning platform for medical imaging," *Comput. Methods Prog. Biomed.* **158**, 113–122 (2018).
69. N. Pawlowski et al., "DLTK: state of the art reference implementations for deep learning on medical images," arXiv:1711.06853 (2017).
70. A. Nikolaos, "Deep learning in medical image analysis: a comparative analysis of multimodal brain-MRI segmentation with 3D deep neural networks," Master's thesis, University of Patras (2019).

71. M. Baumgartner et al., "nndetection: a self-configuring method for medical object detection," *Lect. Notes Comput. Sci.* **12905**, 530–539 (2021).
72. X. Mei et al., "RadImageNet: an open radiologic deep learning research dataset for effective transfer learning," *Radiol.: Artif. Intell.* **4**, e210315 (2022).
73. The Python Package Index, "medigan 1.0.0," 2022,https://pypi.org/project/medigan/ (accessed 5 February 2023).
74. R. Osuala, G. Skorupko, and N. Lazrak, "medigan getting started," 2022, https://medigan.readthedocs.io/en/latest (accessed 5 February 2023).
75. K. Lekadir et al., "FUTURE-AI: guiding principles and consensus recommendations for trustworthy artificial intelligence in medical imaging," arXiv:2109.09658 (2021).
76. EuCanImage Consortium, "EuCanImage towards a European cancer imaging platform for enhanced artificial intelligence in oncology," 2020, https://eucanimage.eu/ (accessed 5 February 2023).
77. M. Abadi et al., "TensorFlow: large-scale machine learning on heterogeneous systems," 2015, tensorflow.org.
78. F. Chollet et al., "Keras," 2015, https://github.com/fchollet/keras.
79. C. Larman, *Applying UML and Pattern: An Introduction to Object Oriented Analysis and Design and the Unified Process*, Prentice Hall PTR (2001).
80. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Pearson Deutschland GmbH (1995).
81. I. C. Moreira et al., "INbreast: toward a full-field digital mammographic database," *Acad. Radiol.* **19**(2), 236–248 (2012).
82. M. D. Halling-Brown et al., "Optimam mammography image database: a large-scale resource of mammography images and clinical data," *Radiol.: Artif. Intell.* **3**, e200103 (2020).
83. M. G. Lopez et al., "BCDR: a breast cancer digital repository," in *15th Int. Conf. Exp. Mech.*, Vol. 1215 (2012).
84. S. Kim, B. Kim, and H. Park, "Synthesis of brain tumor multicontrast MR images for improved data augmentation," *Med. Phys.* **48**(5), 2185–2198 (2021).
85. B. H. Menze et al., "The multimodal brain tumor image segmentation benchmark (BRATS)," *IEEE Trans. Med. Imaging* **34**(10), 1993–2024 (2014).
86. R. S. Lee et al., "A curated mammography data set for use in computer-aided detection and diagnosis research," *Sci. Data* **4**(1), 170177 (2017).
87. H. Borgli et al., "Hyperkvasir, a comprehensive multi-class image and video dataset for gastrointestinal endoscopy," *Sci. Data* **7**(1), 283 (2020).
88. K. Dembrower, P. Lindholm, and F. Strand, "A multi-million mammography image dataset and population-based screening cohort for the training and evaluation of deep neural networks-the cohort of screen-aged women (CSAW)," *J. Digital Imaging* **33**(2), 408–413 (2020).
89. E. Sogancioglu, K. Murphy, and B. van Ginneken, "NODE21 (v-5) [data set]," *Zenodo* (2021).
90. X. Wang et al., "Chestx-ray8: hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 2097–2106 (2017).
91. B. Segal et al., "Evaluating the clinical realism of synthetic chest x-rays generated using progressively growing GANs," *SN Comput. Sci.* **2**(4), 1–17 (2021).
92. S. Joshi et al., "nn-UNet training on CycleGAN-translated images for cross-modal domain adaptation in biomedical imaging," *Lect. Notes Comput. Sci.* **12963**, 540–551 (2022).
93. R. Dorent et al., "CrossMoDA 2021 challenge: benchmark of cross-modality domain adaptation techniques for vestibular schwannoma and cochlea segmentation," *Med. Image Anal.* **83**, 102628 (2022).
94. R. Selvan et al., "Carbon footprint of selecting and training deep learning models for medical image analysis," in *Med. Image Comput. and Comput. Assist. Intervention–MICCAI 2022: 25th Int. Conf.*, 18–22 September 2022, Singapore, Proceedings, Part V, pp. 506–516, Springer Nature, Cham, Switzerland (2022).
95. C. Dwork et al., "The algorithmic foundations of differential privacy," *Found. Trends® Theor. Comput. Sci.* **9**(3–4), 211–407 (2014).

96. L. Abdelrahman et al., "Convolutional neural networks for breast cancer detection in mammography: a survey," *Comput. Biol. Med.* **131**(Jan.), 104248 (2021).
97. Centers for Medicare & Medicaid Services, "The Health Insurance Portability and Accountability Act of 1996 (HIPAA)," 1996, http://www.cms.hhs.gov/hipaa/.
98. European Parliament and Council of European Union, "Council regulation (EU) no 2016/679," 2018, https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679/.
99. Committee on Health Research and the Privacy of Health Information: The HIPAA Privacy Rule, Board on Health Sciences Policy, Board on Health Care Serviceset al., *Beyond the HIPAA Privacy Rule: Enhancing Privacy, Improving Health Through Research*, p. 12458, National Academies Press, Washington, D.C. (2009).
100. U.S. Dept. of Health and Human Services, "Summary of the HIPAA privacy rule: HIPAA compliance assistance," 2003, http://purl.fdlp.gov/GPO/gpo9756.
101. S. M. Shah and R. A. Khan, "Secondary use of electronic health record: opportunities and challenges," *IEEE Access* **8**, 136947–136965 (2020).
102. C. F. Mondschein and C. Monda, "The EU's general data protection regulation (GDPR) in a research context," in *Fundamentals of Clinical Data Science*, P. Kubben, M. Dumontier, and A. Dekker, Eds., pp. 55–71, Springer International Publishing, Cham (2019).
103. K. El Emam, L. Mosquera, and R. Hoptroff, *Practical Synthetic Data Generation: Balancing Privacy and the Broad Availability of Data*, 1st ed., O'Reilly Media, Inc, Sebastopol, California (2020).
104. F. K. Dankar and M. Ibrahim, "Fake it till you make it: guidelines for effective synthetic data generation," *Applied Sciences* **11**, 2158 (2021).
105. W. H. L. Pinaya et al., "Brain imaging generation with latent diffusion models," in *Deep Generative Models: Second MICCAI Workshop, DGM4MICCAI 2022, Held in Conjunction with MICCAI 2022, Singapore, September 22, 2022, Proceedings*, pp. 117–126, Springer Nature, Cham, Switzerland (2022, October).
106. W. H. L. Pinaya et al., "Unsupervised brain imaging 3D anomaly detection and segmentation with transformers," *Med. Image Anal.* **79**, 102475 (2022).
107. N. Pawlowski, D. Coelho de Castro, and B. Glocker, "Deep structural causal models for tractable counterfactual inference," in *Adv. Neural Inf. Process. Syst. 33*, pp. 857–869 (2020).
108. M. Rutherford et al., "A DICOM dataset for evaluation of medical image de-identification," *Sci. Data* **8**, 183 (2021).

Biographies of the authors are not available.