GIFTbench: Generative Image Fuzz Testing Benchmark

Maryam Maryam

University of Udine, Italy

Matteo Biagiola

Universität St. Gallen, Switzerland

Andrea Stocco

Technical University of Munich & fortiss, Germany

Vincenzo Riccio

University of Udine, Italy

Abstract

GIFTbench is a modular framework for testing Deep Learning image classifiers that combines Generative AI with genetic algorithms. Its architecture integrates pretrained generative models with a user-friendly Gradio interface, enabling automated, reproducible, and interpretable robustness testing. Supporting VAE, GAN, and Diffusion models, GIFTbench generates test inputs by perturbing latent representations to expose misbehaviors of the classifier under test. By automating test input generation and reducing the need for manual coding, GIFTbench accelerates experimentation and facilitates comparative evaluation of both classifiers and generative models. Designed for researchers and practitioners, it enables reproducible assessment of image classifiers, while supporting studies on classifier vulnerabilities, mutation strategies, and the role of generative models in robustness testing.

Keywords: Software Testing, Generative AI, Search-based Software Engineering

Metadata

Table 1: Code metadata

Code metadata description	**Please fill in this column**
Current code version	v1.2.0
Permanent link to code/repository used for this code version	`https://github.com/deeptestai/genai_tigs/tree/tool`
Permanent link to Reproducible Capsule	`https://hub.docker.com/r/maryam483/giftbench`
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	Python 3.10, PyTorch, diffusers, Gradio
Compilation requirements, operating environments and dependencies	CUDA-enabled, Docker, other required packages listed in `requirements.txt`
If available, link to developer documentation/manual	`https://github.com/deeptestai/genai_tigs/blob/tool/README.md`
Support email for questions	maryam@spes.uniud.it

1. Motivation and Significance

Deep Neural Network (DNN) based image classifiers have become integral components of software systems, also in safety-critical domains such as healthcare and autonomous driving. On standard benchmark datasets, these classifiers often outperform traditional approaches and even human experts [1, 2, 3]. However, these benchmarks do not fully capture the diversity and unpredictability of real-world conditions encountered during operation. As a result, DNNs struggle to generalize when exposed to new or slightly perturbed inputs, raising concerns about their robustness and reliability in practice [4, 5].

The gap between training data and real-world inputs highlights the need for systematic testing approaches. A major challenge for software testers is to generate test inputs that accurately reflect real-world operating conditions and trigger misclassifications, i.e., unexpected behaviors in which the predicted labels deviate from the expected ones.

To address this challenge, researchers have proposed several *Test Input Generators* (TIGs), i.e., tools that automatically produce synthetic images for assessing the quality of DNN classifiers [6, 7, 8, 9]. Recent advances in TIGs exploit the power and creativity of distribution-aware Generative AI (GenAI) models [10, 11, 12, 13, 9, 14, 15, 16, 17], which learn the input data distribution in the form of a latent space, a lower-dimensional representation of the input space that captures the key features of the problem domain [18]. By manipulating latent representations, GenAI models can generate novel inputs that are both diverse and semantically meaningful, providing more realistic test cases than traditional approaches [9].

GenAI-based TIGs adopted a variety of architectures, ranging from simpler models such as Variational AutoEncoders (VAEs) to more sophisticated Generative Adversarial Networks (GANs). More recently, diffusion models have emerged as state-of-the-art generative approaches, achieving impressive results but at the cost of increased complexity and computational demands for training. However, it is challenging to assess the specific contribution of different GenAI models, since existing TIGs are often influenced by confounding factors such as variations in testing algorithms, the absence of standardized training and hyperparameter tuning, and limited support for recent innovations like diffusion models.

To this end, we propose **GIFTbench** (Generative Image Fuzz Testing Benchmark), a framework that combines search-based test generation with different state-of-the-art GenAI models to enable automated testing of DNN classifiers through latent space manipulation. GIFTbench allows researchers and practitioners to analyze classifier behavior with inputs crafted to induce misclassifications. To make experimentation more accessible, GIFTbench integrates a lightweight web-based interface built with Gradio [19], a Python framework that allows users to interact with the tool, configure experiments, and visualize results without requiring extensive coding effort.

GIFTbench has already enabled a fair comparison of test generation capabilities across different GenAI models, performed in our prior work [17]. In a large-scale empirical study, we evaluated three representative architectures (VAEs, GANs, and diffusion models) across four datasets of increasing complexity. This study revealed several key insights into GenAI-based test generation. In particular, we observed that diffusion models achieve superior performance on complex tasks, but at the cost of substantially higher inference time (up to 10$\times$ slower than alternative models). We also found that applying larger perturbations can accelerate test generation without compromising input validity.

By providing an intuitive and modular integration of search-based testing techniques with GenAI models, GIFTbench supports the advancement of research on the quality assurance of DNNs, while also helping practitioners identify the generative approach best suited to their classification tasks.

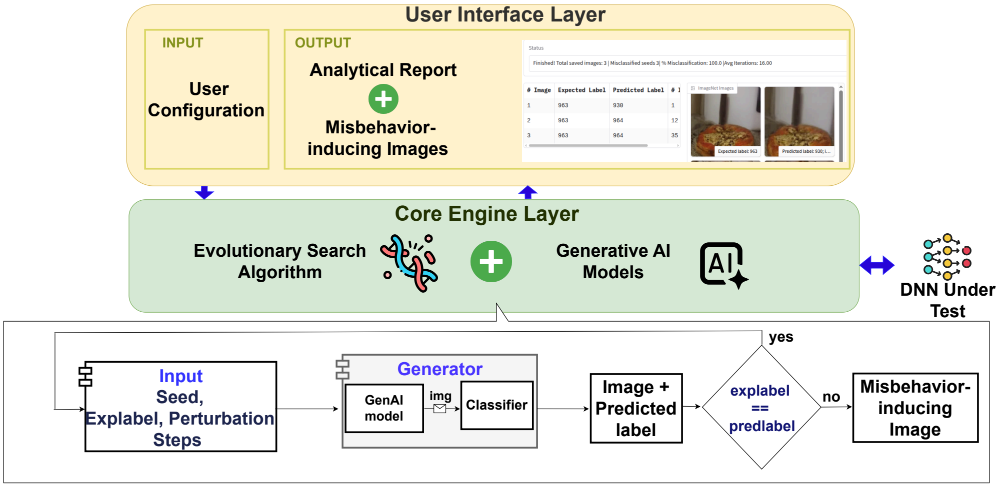


Figure 1: Layered architecture of GIFTbench, showing the user interface, core engine, and the iterative search process in the latent space.

2. The GIFTbench Framework

GIFTbench is a modular and user-friendly framework for testing image classifiers using GenAI models. Its architecture integrates (1) input generation through GenAI models, (2) search-based optimization for latent space manipulation, and (3) evaluation and visualization.

At its core, GIFTbench automatically generates diverse and perturbed inputs to probe and evaluate the robustness of classifiers. It currently supports three state-of-the-art GenAI architectures: VAEs [20], GANs [21] and Diffusion Models [22]. These are combined with an evolutionary search mechanism based on a Genetic Algorithm, which systematically explores input variations to produce misbehavior-inducing test cases. Our framework is highly configurable, allowing users to control key testing parameters such as the testing budget (i.e., the number of evolutionary iterations) and the magnitude of perturbations applied to latent vectors. To improve accessibility, GIFTbench provides an interactive interface built with Gradio [19], enabling experimentation and visualization of results without requiring programming expertise.

2.1. Software Architecture

GIFTbench follows a layered architecture that separates the user interface from the core engine, as depicted in Figure 1. The modular design promotes extensibility and a clear functional separation, providing a scalable foundation for testing DNN-based image classifiers.

The **User Interface Layer** serves as the system's front end, implemented with Gradio [19]. It abstracts backend complexity and allows users to interact with the platform via a simple web interface [23]. Through this layer, users can: (1) select datasets (e.g., MNIST [24], ImageNet [25]); (2) configure pretrained GenAI models (e.g., VAEs, diffusion models) and classifiers (default or user-supplied); (3) adjust test parameters using sliders and toggles; and (4) launch and monitor test campaigns. The interface automatically verifies dataset–model compatibility (ensuring, for instance, that the chosen generative model was trained on the selected dataset), and enforces classifier-specific constraints such as input dimensions or file formats (e.g., `.jit` TorchScript models [26]). Once a test session is executed, results are visualized directly in the interface, including generated images, performance metrics, and summary reports.

The **Core Engine Layer** is the heart of GIFTbench's test generation process. It integrates a genetic algorithm with the selected GenAI model to test the target classifier. In particular, this layer generates and evolves candidate inputs in the latent space with the goal of inducing misclassifications. Although it interacts closely with the User Interface, the Core Engine is designed to remain independent and easily extensible to new datasets, generative models, and classifiers, thus supporting reuse and adaptation. The lower part of Figure 1 illustrates its iterative test generation process: starting from a latent seed, an expected label, and a perturbation budget, the test generator progressively modifies the latent representation until it produces an input that causes the classifier's prediction to differ from the expected label, thereby exposing a robustness weakness.

2.2. Software Functionality

Figure 2 illustrates the end-to-end workflow of GIFTbench. Executing `./run.sh` sets up the Docker-based Gradio app and launches the user interface. From there, the user proceeds with the selection of a dataset, either *MNIST*, *SVHN*, *CIFAR-10*, or two classes of *ImageNet* (i.e., pizza and teddy bear). Once a dataset is selected, the user must choose one of the three generative models (*VAE*, *GAN*, or *Diffusion Model*). Then, GIFTbench initialize

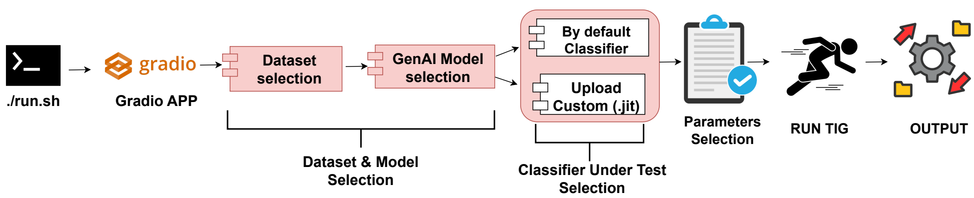


Figure 2: GIFTbench workflow.

the testing process with one of the default configurations from our empirical study [17], but sliders allow for customization, which is particularly useful when testing custom classifiers. Once the configuration is completed, GIFTbench allows users to configure the testing process. The number of test images can be specified via the `Images_to_Sample` parameter. The underlying Genetic Algorithm is also customizable: population size (default 25), number of iterations (default 250), and perturbation size (low/high). This algorithm manipulates the latent vectors in two steps: first, an initial perturbation to generate the base population, and then iterative adjustments to trigger misclassifications guided by the classifier's responses. Users can toggle perturbation size (low/high) to observe its effects on model robustness and TIG efficiency. Each perturbed image is evaluated by the classifier, and its misclassification likelihood is used as fitness score. The loop continues until a misclassification is detected or the iteration budget is exhausted.

The Gradio interface provides real-time monitoring and results visualization. Users can inspect side-by-side galleries of original and perturbed images, compare expected and predicted labels, and track metrics through a status bar summarizing misclassification rate and average iteration count. A detailed tabular report logs image IDs, expected/predicted labels, and iteration counts, while all generated images can be exported as a `.zip` archive with metadata-enriched filenames for downstream analysis. This interactive execution and reporting flow makes GIFTbench a practical and efficient tool for evaluating classifier robustness in real time.

3. Using GIFTbench

3.1. Installation and Setup

GIFTbench is packaged as a dockerized Python application for easy deployment across different environments. There are two ways to run GIFTbench: (1) using the pre-built Docker image from Docker Hub or (2) building and running from source code (recommended for developers).

3.1.1. Running from Docker Hub

The pre-built container image provides the fastest way to reproduce our results without installing any dependencies locally.

Listing 1: Running GIFTbench from Docker Hub

```
docker pull maryam483/giftbench:v1.2.0
docker run --name giftbench-running --gpus all -p 7860:7860 \
 maryam483/giftbench:v1.2.0
```

These commands pull the versioned image and launch GIFTbench. On the first run, the container automatically downloads the required pretrained GenAI models. Internet access is therefore required only once, while subsequent runs reuse the cached models.

3.1.2. Building from Source

The repository can be cloned from GitHub, and the application can be launched with a single command. To run the tool, users must have Docker already installed.

Listing 2: Cloning and running the GIFTbench tool from the `tool` branch

```
git clone https://github.com/deeptestai/genai_tigs.git
cd genai_tigs
git checkout tool
./run.sh
```

3.1.3. Accessing the Interface

When launched, Gradio provides two possible endpoints:

- **Local URL:** `http://localhost:7860`, accessible directly from the user's browser on the same machine.
- **Public URL:** a temporary link of the form `https://abcdef12345.gradio.live`, automatically generated by Gradio to share sessions. This link expires after 72 hours or when the container is stopped.

3.2. Test Generation

The test generation process can be configured and executed through the user interface shown in Figure 3. After selecting the dataset, GenAI model, classifier, and configuring the genetic algorithm parameters, the user can start the test generation by clicking the `Run TIG` button.

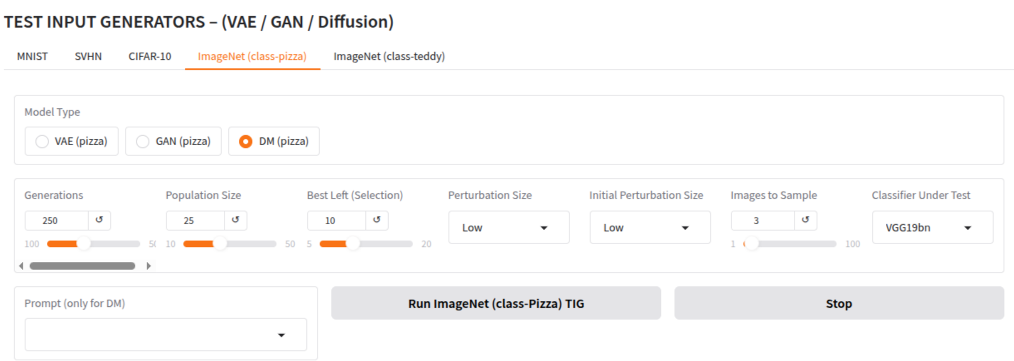
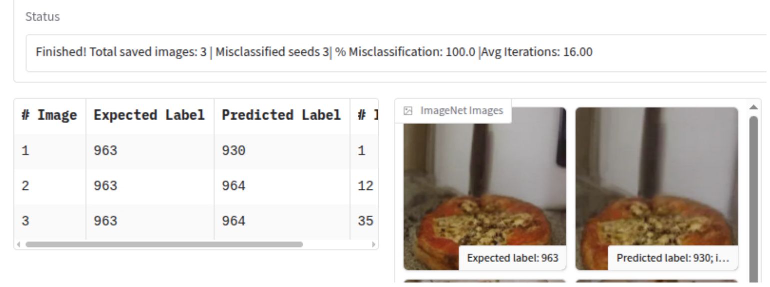
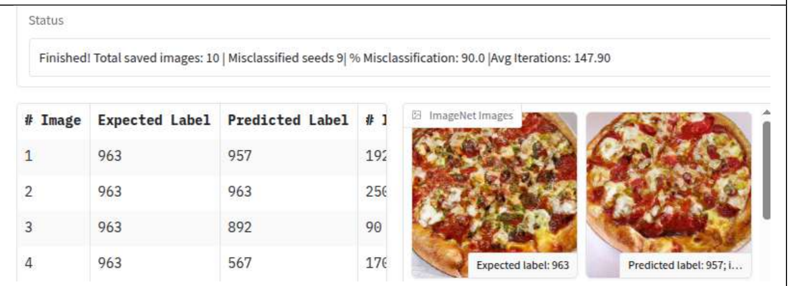
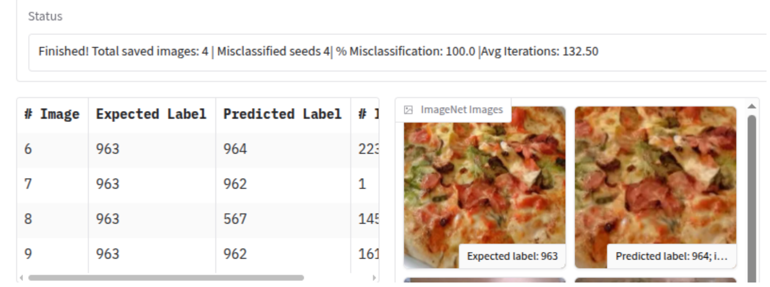


Figure 3: Configuration of GIFTbench parameters.

Table 2: Misclassified images generated by different models for the ImageNet Pizza class

Model	Gradio Results
VAE	
GAN	
DM	

3.3. Illustrative Example: Testing an ImageNet Classifier

A complete evaluation of the GenAI models integrated in GIFTbench is presented in our earlier work [17], where VAE, GAN, and Diffusion Models were benchmarked for their effectiveness in generating valid, misclassification-inducing test inputs across four datasets.

In this paper, we provide a demonstration revisiting a small subset of that setup to illustrate how GIFTbench facilitates test generation and model evaluation. We configured GIFTbench to test a default VGG19bn ImageNet classifier, focusing on the Pizza class. As shown in Figure 3, the Gradio interface is set up for generating 10 seeds, perform 250 iterations of the genetic algorithm with a population of size 25, and adopting a small perturbation extent in the latent space. By applying the same configuration to all three GenAI model types, we ensure a fair comparison of their effectiveness.

Table 2 demonstrates how GIFTbench operationalizes hypotheses about model robustness, misclassification sensitivity, and the impact of input perturbation extent. After each run, the status bar reports the number and percentage of misclassified seeds, together with the average number of perturbation iterations required. In our illustrative runs (not statistically significant due to the small number of seeds), the GAN model achieved the highest misclassification rate, with 9 out of 10 seeds triggering a misclassification within the test budget.

4. Expected Impact and Significance

GIFTbench provides a novel experimental environment for studying the interplay between generative AI models, latent space perturbations, and classifier vulnerabilities under controlled conditions. It supports systematic comparisons of classifier robustness and generative model effectiveness across diverse architectures and mutation settings.

Compared to prior robustness testing frameworks, GIFTbench offers the most comprehensive integration of evolutionary test generation with GenAI-based input synthesis. Its automated workflows minimize manual coding effort while ensuring reproducibility and extensibility. We anticipate that the tool will serve as an accelerator for researchers, students, and practitioners, enabling them to conduct reproducible robustness evaluations through an intuitive interface that lowers entry barriers and facilitates both experimentation and analysis.

5. Conclusions and Future Work

We presented GIFTbench, a framework that combines generative AI models with evolutionary algorithms to systematically assess the quality of DNN-based image classifiers. By integrating pretrained GenAI models with an intuitive interface, the tool enables automated, reproducible, and visually interpretable robustness testing across multiple datasets.

Future work will focus on enhancing the flexibility and extensibility of the framework, enabling seamless integration of custom classifiers, datasets, and generative models, thus broadening its applicability for both research and practice. Moreover, extending GIFTbench with additional fitness functions and exploration strategies will open avenues for mutation analysis [27, 28], targeted test generation [5, 15], deeper input space and behavioral boundary exploration [29, 30], and increased input diversity [31].

References

[1] D. Sarwinda, R. H. Paradisa, A. Bustamam, P. Anggia, Deep Learning in Image Classification using Residual Network (ResNet) Variants for Detection of Colorectal Cancer, Procedia Computer Science 179 (2021) 423–431. doi:10.1016/j.procs.2021.01.025.

[2] G. Lou, Y. Deng, X. Zheng, M. Zhang, T. Zhang, Testing of autonomous driving systems: where are we and where should we go?, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, 2022, p. 31–43. doi:10.1145/3540250.3549111.
URL https://doi.org/10.1145/3540250.3549111

[3] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. Guo, S. Liu, S. Guo, Y.-F. Li, L. Ma, Y. Xue, Y. Liu, A survey on automated driving system testing: Landscapes and trends (2023). arXiv:2206.05961.
URL https://arxiv.org/abs/2206.05961

[4] A. Guerriero, R. Pietrantuono, S. Russo, Operation is the hardest teacher: estimating dnn accuracy looking for mispredictions, in: Proceedings of the 43rd International Conference on Software Engineering, ICSE '21, IEEE Press, 2021, p. 348–358. doi:10.1109/ICSE43902.2021.00042.
URL https://doi.org/10.1109/ICSE43902.2021.00042

[5] T. Zohdinasab, V. Riccio, P. Tonella, DeepAtash: Focused Test Generation for Deep Learning Systems, in: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA, 2023, p. 954–966. doi:10.1145/3597926.3598109.

[6] J. M. Zhang, M. Harman, L. Ma, Y. Liu, Machine learning testing: Survey, landscapes and horizons, IEEE Transactions on Software Engineering 48 (1) (2020) 1–36.

[7] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, P. Tonella, Testing machine learning based systems: a systematic mapping, Empirical Software Engineering 25 (6) (2020) 5193–5254.

[8] H. B. Braiek, F. Khomh, On testing machine learning programs, Journal of Systems and Software 164 (2020) 110542.

[9] V. Riccio, P. Tonella, When and why test generators for deep learning produce invalid inputs: an empirical study, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, 2023, pp. 1161–1173.

[10] S. Dola, M. B. Dwyer, M. L. Soffa, Distribution-aware testing of neural networks using generative models, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021.

[11] S. Kang, R. Feldt, S. Yoo, Sinvad: Search-based image space navigation for dnn image classifier test input generation, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, 2020, pp. 521–528.

[12] I. Dunn, H. Pouget, D. Kroening, T. Melham, Exposing previously undetectable faults in deep neural networks, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 56–66.

[13] A. Aleti, Software testing of generative ai systems: Challenges and opportunities, in: 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), 2023.

[14] S. Dola, R. McDaniel, M. B. Dwyer, M. L. Soffa, Cit4dnn: Generating diverse and rare inputs for neural networks using latent space combinatorial testing, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–13.

[15] O. Weißl, A. Abdellatif, X. Chen, G. Merabishvili, V. Riccio, S. Kacianka, A. Stocco, Targeted deep learning system boundary testing (2025). arXiv:2408.06258.
URL https://arxiv.org/abs/2408.06258

[16] L. Baresi, D. Y. X. Hu, A. Stocco, P. Tonella, Efficient domain augmentation for autonomous driving testing using diffusion models, in: Proceedings of 47th International Conference on Software Engineering, ICSE '25, IEEE, 2025.

[17] M. Maryam, M. Biagiola, A. Stocco, V. Riccio, Benchmarking generative ai models for deep learning test input generation, in: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE, 2025, pp. 174–185.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Communications of the ACM 63 (11) (2020).

[19] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, J. Zou, Gradio: Hassle-free sharing and testing of ml models in the wild, arXiv preprint arXiv:1906.02569 (2019).

[20] D. P. Kingma, M. Welling, Auto-encoding variational bayes (2022). arXiv:1312.6114.
URL https://arxiv.org/abs/1312.6114

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Commun. ACM 63 (11) (2020) 139–144. doi:10.1145/3422622.
URL https://doi.org/10.1145/3422622

[22] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, S. Ganguli, Deep unsupervised learning using nonequilibrium thermodynamics (2015). arXiv:1503.03585.
URL https://arxiv.org/abs/1503.03585

[23] Gradio, Gradio: Build machine learning demos in python, https://www.gradio.app, accessed: 2025-05-30 (2025).

[24] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE Signal Processing Magazine 29 (6) (2012) 141–142. doi:10.1109/MSP.2012.2211477.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255. doi:10.1109/CVPR.2009.5206848.

[26] PyTorch Team, Torchscript: A way to create serializable and optimized models from pytorch code, https://pytorch.org/docs/stable/jit.html, accessed: 2025-05-30 (2018).

[27] V. Riccio, N. Humbatova, G. Jahangirova, P. Tonella, Deepmetis: Augmenting a deep learning test set to increase its mutation score, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2021, pp. 355–367.

[28] A. Abbasishahkoo, M. Dadkhah, L. Briand, D. Lin, Teasma: A practical methodology for test adequacy assessment of deep neural networks, IEEE Transactions on Software Engineering (2024).

[29] V. Riccio, P. Tonella, Model-based exploration of the frontier of behaviours for deep learning system testing, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, 2020, p. 876–888. doi:10.1145/3368089.3409730.
URL https://doi.org/10.1145/3368089.3409730

[30] T. Zohdinasab, V. Riccio, A. Gambi, P. Tonella, Deephyperion: exploring the feature space of deep learning-based systems through illumination search, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 79–90.

[31] Z. Aghababaeyan, M. Abdellatif, L. Briand, M. Bagherzadeh, et al., Black-box testing of deep neural networks through test case diversity, IEEE Transactions on Software Engineering 49 (5) (2023) 3182–3204.