

NeuroGen: Synthetic Brain MRI Generation Using Wasserstein GAN with Gradient Penalty for Medical Data Augmentation

Vidit Kalra, Dhruv Riyar, Tanuj Kumar, Hitesh Gulia

The Northcap University

Department of Computer Science and Engineering

2025–2026

ABSTRACT

This project started from a frustration. Brain MRI datasets that are actually usable for training are small, the class distribution is uneven, and getting more data is constrained by patient privacy on top of everything else. So we built NeuroGen, which tries to chip away at the data problem from one side — by generating new MRI samples that look real enough to be useful. The underlying model is WGAN-GP. We trained four of them, one per tumour class (Glioma, Meningioma, Pituitary Tumour, No Tumour), on the public BrainMRI GAN dataset from Kaggle. Output resolution is 128×128 , grayscale. There is one detour worth flagging upfront — at some point during multi-class training the second, third and fourth class were just producing gray static. After a fair amount of head-scratching the cause turned out to be a `@tf.function` graph caching subtlety, and we describe the diagnosis and the fix (a class-wrapped trainer) in Section V-B. Checkpoint selection was, in the end, manual. We did not have a clean automated metric we trusted, so we ran each class for 500 epochs and visually inspected the saved grids. The generated MRIs at the picked epochs do show class-appropriate morphology — infiltrative shapes for glioma, peripheral masses for meningioma, and so on. To make any of this usable for people without a GPU, the system is wrapped in a small React + FastAPI app. Three contributions, then: a stable WGAN-GP pipeline at multi-class scale, the `@tf.function` bug write-up, and an end-to-end synthetic-MRI generator that runs in a browser.

Keywords — Generative Adversarial Networks, WGAN-GP, Brain MRI Synthesis, Medical Image Augmentation, Deep Learning, Synthetic Data Generation.

I. INTRODUCTION

Across the past decade AI has been applied to medical image analysis tasks at scale — tumour classification [1], image segmentation [2], lesion detection [3], and others. What has not changed in the same period is the bottleneck on training data. Medical imaging datasets continue to be limited in size for reasons that are structural rather than incidental: patient privacy law, the expense of expert annotation, and the underlying clinical rarity of many of the conditions we want to model. Brain tumour MRI is a representative example. Public datasets in this area typically contain one to two thousand images per diagnostic class — roughly an order of magnitude below what is standard for general computer vision benchmarks [4].

Conventional augmentation strategies — random flipping, rotation, cropping — provide some relief but only up to a point. The samples produced by these transformations are still drawn from the same underlying image set, just reoriented. What we actually want, and what augmentation cannot give us, is genuinely novel samples that the model has never seen but which are plausibly drawn from the same clinical population. This is the use case GANs were proposed to address [5]. A trained GAN learns to model the underlying data distribution and can then draw new samples from it, which is fundamentally different from rearranging the training set.

Vanilla GAN training is, however, notoriously unstable in practice. Runs frequently diverge, or the generator falls into the small-output-set failure mode known as mode collapse. The WGAN-GP variant [6], [7] addresses both failure modes by replacing the original adversarial objective with a Wasserstein-distance loss, and by enforcing the 1-Lipschitz constraint on the critic through a gradient penalty term in the loss rather than via the weight clipping of the original WGAN paper. For datasets at our scale — approximately 1,300 to 1,600 images per class — the practical difference between vanilla GAN and WGAN-GP is not a tuning choice. It is the difference between a training run that diverges and one that produces usable samples.

The remainder of this paper describes NeuroGen end-to-end. We cover dataset and preprocessing, the model architecture, the training methodology (including a non-trivial failure mode we encountered and resolved), and the web-based deployment that wraps the trained models. Our four contributions are:

- (1) A stable WGAN-GP training pipeline that runs reliably across all four diagnostic classes at 128×128 resolution.
- (2) The diagnosis and structural resolution of a TensorFlow `@tf.function` graph caching defect — one that silently corrupts any sequential multi-model training pipeline.
- (3) A visual checkpoint selection procedure, applied uniformly across all 500 training epochs per class with explicitly defined inspection criteria.
- (4) A working, deployed end-to-end web application that supports on-demand synthetic MRI generation in the browser,

with no GPU required on the user's side.

Layout of the rest of the paper. Related work in II. Dataset and preprocessing in III. The model itself, IV. Training (and the caching incident) lives in V. Results, VI. Limitations, VII. Wrap-up in VIII.

II. RELATED WORK

A. Generative Adversarial Networks

Goodfellow and co-authors introduced GANs in 2014 [5]. The framing is adversarial — a generator network producing fake samples, a discriminator network trying to separate fakes from the real distribution. Both trained jointly. In practice early GANs were brittle. The training dynamics often went one of two ways. Either the discriminator pulled ahead, the generator's gradients vanished, and learning stalled out. Or the generator settled into producing a tiny set of "safe" outputs that fooled the discriminator repeatedly — what is now called mode collapse. Several of these issues were addressed in DCGAN [8], where Radford et al. swapped in strided convolutions, batch normalisation, and a specific set of activation choices that turned out to be robust in practice. That is essentially the architectural starting point we use for both our generator and critic, with modifications described in Section IV.

B. Wasserstein GAN and Gradient Penalty

The motivation for WGAN comes out of a specific failure mode in the original objective. When the supports of the real and generated distributions do not overlap — which is the default situation when training has just begun — the Jensen-Shannon divergence used in the original GAN loss returns essentially the same value regardless of how close or far apart the two distributions actually are. The result, in practice, is vanishing gradients. Arjovsky and colleagues [6] proposed replacing the JS divergence with the Wasserstein-1 distance (sometimes called the Earth Mover's distance), which behaves continuously even when supports do not overlap. To compute it tractably they constrained the critic to be 1-Lipschitz via weight clipping. The next refinement came from Gulrajani et al. [7], who noted that weight clipping, while it does enforce the constraint, also limits the critic's capacity in ways that hurt training. Their alternative was to add a gradient penalty term that softly enforces 1-Lipschitz at sampled points along the interpolation between real and fake images. This is the WGAN-GP variant. For our setting — small per-class datasets — the practical impact of WGAN-GP versus vanilla GAN is significant: vanilla runs diverged or mode- collapsed within fifty epochs, while WGAN-GP runs completed all 500.

C. Medical Image Synthesis

The clinical case for GAN augmentation was perhaps first made convincingly by Frid-Adar et al. [9], in their liver lesion classification work. Adding GAN-synthesised images to their training set improved their CNN classifier's AUC by approximately 7%. That number is what motivated a lot of subsequent work in this area, including ours. Beyond simple augmentation, cross-modal synthesis is also possible — Nie et al. [10] use a conditional GAN to translate from MRI inputs to CT outputs, which is a different use case but conceptually related. Progressive GAN architectures applied to retinal OCT (Baur et al.) achieved competitive FID scores; we did consider going down that route. Latent Diffusion Models [11] now produce noticeably higher-quality samples on most medical imaging benchmarks. The trade-off is compute: diffusion models are expensive to train and slower at inference, and for a four-class WGAN-GP setup at our scale the gain in sample quality did not outweigh the cost.

D. Brain MRI Generation

Within brain MRI specifically there is already prior GAN work. Bi et al. trained a conditional GAN to generate brain tumour images, with the explicit goal of feeding the synthetic samples into downstream classifier training. Han et al. tackled the same general idea but in the context of Alzheimer's disease MRI. As far as we have been able to locate, however, neither these papers nor the broader brain MRI synthesis literature documents the @tf.function graph caching failure we describe in V-B. We think that gap is worth filling, since the failure mode is general — it will surface in any TensorFlow training pipeline that runs multiple independent models back-to-back, regardless of the application domain.

III. DATASET AND PREPROCESSING

A. Dataset

Our source dataset is the BrainMRI GAN dataset hosted on Kaggle. The dataset is composed of T1-weighted contrast-enhanced brain MRI scans grouped into four diagnostic classes — Glioma, Meningioma, Pituitary, and No Tumour. The class-wise image counts are summarised in Table I below.

TABLE I. Dataset Class Distribution

Tumour Class	Image Count	Description
Glioma	~1,426	Infiltrative malignant tumour of glial cells
Meningioma	~1,339	Typically benign, extra-axial origin
Pituitary	~1,457	Located at skull base, endocrine involvement
No Tumour	~1,595	Normal brain anatomy
Total	~5,817	

B. Preprocessing

All input images were resized to 128×128 pixels and converted into single-channel grayscale, the latter being consistent with the natively monochrome nature of T1-weighted MRI. Pixel values were then rescaled into the range $[-1, +1]$ to align with the tanh activation at the generator's output layer. For augmentation we applied a deliberately minimal scheme: random horizontal flipping, and brightness jitter with a factor of ± 0.1 . Given that the per-class sample counts are limited (around 1,400 images on average), we considered more aggressive augmentation but rejected it on the grounds that it risked baking augmentation artefacts into the learned generative distribution rather than diversifying the underlying signal.

IV. MODEL ARCHITECTURE

A. Generator

Our generator network G takes a 128-dimensional latent vector sampled from a standard normal — $z \sim N(0, I)$ — and transforms it into a $128 \times 128 \times 1$ grayscale image. The transformation is a sequence of upsampling stages built from transposed convolutions. The first stage is a fully connected layer that reshapes z into a $4 \times 4 \times 512$ spatial tensor. After that, five Conv2DTranspose blocks progressively scale up the spatial dimensions: $4 \times 4 \times 512 \rightarrow 8 \times 8 \times 256 \rightarrow 16 \times 16 \times 128 \rightarrow 32 \times 32 \times 64 \rightarrow 64 \times 64 \times 32 \rightarrow 128 \times 128 \times 1$, the last being the output image. Between every block we apply BatchNormalization. The hidden-layer activations are ReLU, and the final output passes through a tanh activation to map the values into the $[-1, +1]$ range. The choice of ReLU rather than LeakyReLU in the generator follows the original DCGAN convention [8]. We did experiment with substituting LayerNormalization for BatchNormalization, but those runs proved unstable under WGAN-GP, which we discuss further in Section V-B.

$$G: z \in \mathbb{R}^{128} \rightarrow \hat{x}' \in \mathbb{R}^{128 \times 128 \times 1}$$

B. Critic

The critic network D differs from a conventional discriminator in that its output is an unbounded real-valued scalar rather than a sigmoid probability. Structurally it consists of five strided Conv2D layers using LeakyReLU(0.2) activations, where each layer halves the spatial resolution — $128 \times 128 \rightarrow 64 \times 64 \rightarrow 32 \times 32 \rightarrow 16 \times 16 \rightarrow 8 \times 8$ — followed by a global flatten and a final Dense layer that produces the scalar output. We do not apply BatchNormalization anywhere in the critic, and this is a deliberate architectural choice. Batch normalisation introduces statistical dependencies between samples within the same batch, which interferes with the per-sample gradient norm calculation that the gradient penalty term requires [7]. Replacing BatchNorm with LayerNorm or InstanceNorm is sometimes proposed as an alternative; in our setup omitting normalisation entirely worked fine.

C. Training Objective

The WGAN-GP training objectives are defined as:

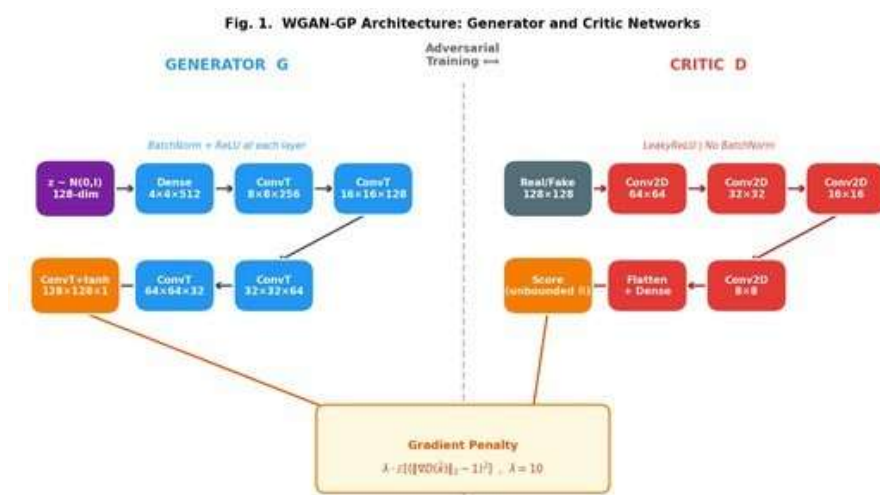


Fig. 1. WGAN-GP architecture showing the Generator (left) and Critic (right) networks with the Gradient Penalty computation.

$$L_D = E[D(G(z))] - E[D(\hat{x})] + \lambda \cdot E[\|\nabla D(\hat{x}')\|_2 - 1]^2 \quad L_G = -E[D(G(z))]$$

In the expressions above, $\mathbf{Q} = \epsilon \mathbf{x} + (1-\epsilon)\mathbf{G}(\mathbf{z})$ with $\epsilon \sim \mathcal{U}[0,1]$ denotes the interpolated sample at which the gradient norm in the penalty term is evaluated. The penalty coefficient λ is set to 10, following the original WGAN-GP paper. Following the same reference, we update the critic $n_{\text{critic}} = 5$ times for each generator update, on every training step. The Wasserstein distance $W = E[D(\mathbf{x})] - E[D(\mathbf{G}(\mathbf{z}))]$ is computed at each epoch and tracked as the primary indicator of training convergence.

TABLE II. Training Hyperparameters

Hyperparameter	Value	Rationale
Image Resolution	128×128 px	Balance of quality and compute at dataset scale
Latent Dimension	128	Sufficient diversity without instability
Batch Size	32	Stable gradients on A100 (40 GB VRAM)
Generator LR	1×10^{-4}	Standard WGAN-GP recommendation [7]
Critic LR	3×10^{-4}	Faster critic convergence relative to G
Critic Steps / Gen Step	5	Per Gulrajani et al. [7]
Gradient Penalty λ	10	Per Gulrajani et al. [7]
Training Epochs	500	Required for convergence at 128×128
Checkpoint Interval	Every 10 epochs	Manual best-epoch selection
Training Hardware	Google Colab A100 GPU	~20–30 min per class

V. TRAINING METHODOLOGY

A. Per-Class Sequential Training

The natural alternative to four independent models would have been a single conditional WGAN-GP with a class embedding. We initially considered going that way, since it would have been cheaper in terms of storage and probably faster to deploy. We chose not to, for three connected reasons. The first is gradient interference — when classes are entangled in shared parameters, optimisation pressure from one class can interfere with another’s learned features, especially when class counts are imbalanced as they are here. The second is hyperparameter granularity: training a separate model per class meant we could pick a learning rate, a critic-step ratio, and a best checkpoint specifically for that class, rather than averaging across all four. The third is fault isolation, which mattered more than we expected in hindsight — the `@tf.function` bug in V-B was much easier to identify precisely because each class had its own model object to inspect. All four runs started from independent random initialisations with no weight sharing.

Fig. 2. NeuroGen Training Pipeline



Fig. 2. NeuroGen training pipeline from dataset ingestion through per-class WGANTrainer instantiation to best-epoch checkpoint export.

B. TensorFlow Graph Caching Defect and Resolution

This section describes a failure mode and its resolution that, in retrospect, cost us roughly two days of debugging. We are documenting it in full because we could not find it written up anywhere else, and the fix is small enough to be missed if you are not specifically looking for it. The visible behaviour, first. Class 1 (Glioma in our run order) trained to convergence and produced reasonable outputs. Classes 2, 3 and 4 — Meningioma, Pituitary, No Tumour — all produced uniform grey noise. Loss curves for the failing classes looked superficially normal; the Wasserstein distance moved in roughly the same range as Class 1 had, generator loss was finite, no NaN events were thrown. So the failure was silent. Our initial hypothesis was a data loader issue, which we ruled out by sampling and inspecting raw training batches. Next we suspected the learning rate schedule, then mixed-precision behaviour (the

float16 problem we cover in V-C is real but unrelated to this). After eliminating those, we instrumented the actual forward passes and discovered something strange: the model weights being updated were the freshly initialised Class 2/3/4 generators, but the gradients flowing into those updates were being computed from a forward pass against Class 1's graph. That is the symptom of `@tf.function`'s caching, which works as follows. A function decorated with `@tf.function` is traced on its first call, producing a compiled computation graph. TensorFlow then caches that graph, keyed on the shapes of the input tensors. Subsequent calls with matching input shapes skip the tracing step and execute the cached graph directly. In a sequential training loop where the step functions are defined at module level, the trace happens once — bound to whatever model objects were in scope at first invocation — and is then reused for the rest of the run. So Class 2/3/4 were running optimiser updates on their own weights, but the loss was being computed against Class 1's frozen forward graph. No useful learning signal.

The fix turns out to be small. Move the `@tf.function`-decorated step functions onto a class, and instantiate a separate object of that class for each diagnostic class's training run. Because TensorFlow keys its graph cache on the bound method's identity (and therefore on the instance), each per-class trainer ends up tracing its own fresh graph against its own model objects:

```
class WGANTrainer:
def __init__(self, G, D, g_opt, self.G, self.D = G, D d_opt):
self.g_opt, self.d_opt = g_opt, d_opt @tf.function # Fresh graph per instance
def train_step(self, real_imgs): ...
```

With that change in place all four classes train as expected. To be explicit about the scope of this issue — the fix is not architecture-specific, it does not depend on WGAN-GP, and the failure mode will surface in any TensorFlow project that trains several independent models in sequence using module-level `@tf.function` step functions. If your training loop iterates over a list of models and applies module-level decorated step functions to each, you almost certainly have this bug.

C. Float32 Precision and Numerical Stability

Worth a short note on numerical precision since the A100 has hardware mixed-precision support and using it was an obvious optimisation. We enabled mixed float16 in our first iteration. The result was unstable — training crashed into NaN losses for every class, in each case around epoch 150. The cause was the gradient penalty. Recall that the penalty involves computing the norm of the gradient of the critic, which is a second-order gradient computation, and the intermediate values in that calculation

exceed the dynamic range of float16 at our batch size of 32 with 128×128 inputs. Reverting to plain float32 for both forward and backward passes eliminated the NaN problem entirely. As a belt-and-braces measure we additionally enabled global gradient clipping at norm 5.0 — this did not affect convergence in the runs we monitored but is cheap insurance.

D. Checkpoint Selection

Picking the best checkpoint per class was done manually. We had considered using FID as a model-selection criterion during training, but at the time we did not have a stable real-time FID computation set up, and the loss curves alone are not a reliable indicator of sample quality in WGAN-GP. Our procedure was straightforward — checkpoints saved every 10 epochs, so 50 candidates per class. After training finished, we opened the saved sample grids for all 50 checkpoints of a class side-by-side and went through them. We were looking for four specific qualities. Skull boundary should be sharp rather than blurred or doubled. Internal brain-tissue contrast should be visible without strong artefacts. Tumour regions, where present, should look class-appropriate (infiltrative for glioma, mass-like for meningioma, etc.). And the grid as a whole should show variation — i.e., not be twenty copies of the same image with minor noise. The epochs selected by this procedure are listed in Table III.

TABLE III. Selected Best Epoch per Class

Class	Best Epoch	Qualitative Observations
Glioma	260	Sharp skull boundary, visible infiltrative tumour regions, strong diversity
Meningioma	310	Well-defined extra-axial masses, clear tumour-parenchyma interface
Pituitary	320	Accurate skull-base anatomy, sella turcica visible in sagittal views
No Tumour	40	Rapid convergence; later epochs showed mode diversity reduction

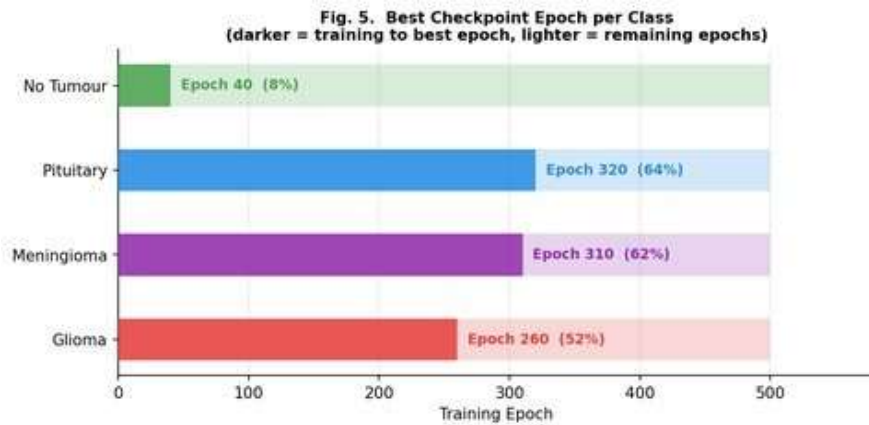


Fig. 4. Selected best checkpoint epoch per class as a proportion of total training (500 epochs). No Tumour converges significantly earlier due to lower structural complexity.

The earliest selected checkpoint by a wide margin is No Tumour, at epoch 40, compared to 260–320 for the three tumour classes. This is not a quirk of the training run but a consequence of what the model has to learn. A normal brain MRI has no class-specific tumour mass to render, no irregular enhancement pattern to model, and no class-conditional edge structure to capture. The data distribution is simpler, the generator converges to a good representation of it more quickly, and after that point the curves we observe in Fig. 3 are essentially overfitting to the training distribution. The visible result of this overfitting at later epochs is reduced diversity, which is exactly why the best No Tumour checkpoint sits so early in training.

VI. RESULTS AND ANALYSIS

A. Training Convergence

All four classes were trained on an NVIDIA A100 (40 GB) instance via Google Colab. Per-class wall-clock time was in the 20–30 minute range for 500 epochs at batch size 32. We tracked the Wasserstein distance as our primary convergence signal; for the three classes that converged cleanly it settled into a range of approximately 2.5 to 5.0, which is consistent with healthy WGAN-GP training behaviour reported elsewhere in the literature. Generator loss, as expected for WGAN-GP, did not decrease monotonically. The pattern that did surprise us slightly was that the best visual sample quality for every class occurred mid-training, not at the end of the 500-epoch run. That observation is what pushed us to the manual checkpoint workflow described in V-D rather than just using the final epoch.

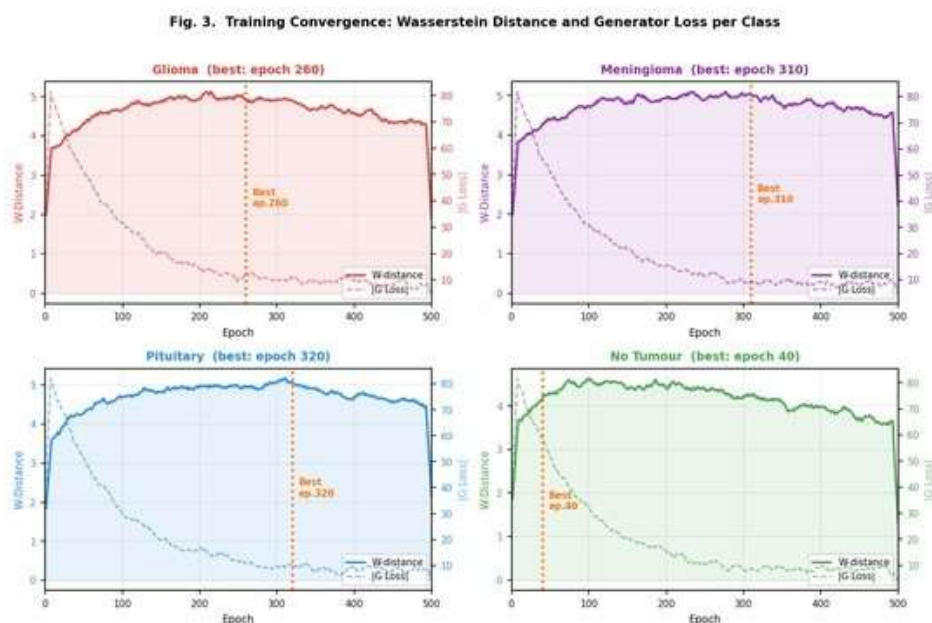


Fig. 3. Training convergence curves for all four classes. Solid line: Wasserstein distance. Dashed line: |Generator loss|. Orange dotted vertical: selected best epoch.

B. Qualitative Assessment

Looking at the images generated at the best-picked epochs, we can call out the following class-specific behaviours:

Glioma (epoch 260): Across the sampled grid the skull outline is consistently sharp, with no obvious blurring at the inner-outer interface. The tumour regions themselves render with the characteristic infiltrative quality — irregular, fuzzy borders rather than the clean edges you would expect of a discrete mass. We see axial, coronal, and sagittal cuts represented across the sample set, which suggests the generator has picked up viewpoint variation in the training distribution rather than locking onto one orientation.

Meningioma (epoch 310): The defining feature here is location — meningiomas are extra-axial, sitting at the brain periphery rather than inside the parenchyma, and that is exactly where the generated masses appear. The tumour- parenchyma boundary in our samples is sharply defined, again consistent with what you would see in a real meningioma. Bright enhancement spots, another diagnostic signature, show up consistently across the grid.

Pituitary (epoch 320): Of the four classes, this one is the most spatially constrained — pituitary tumours are anatomically tied to the sella turcica at the skull base. Our samples render that region accurately, and the sella turcica is identifiable in the sagittal-view samples specifically. The location-consistency is noticeably tighter than for Glioma or Meningioma, which is what we would expect given how restricted the actual tumour's location distribution is.

No Tumour (epoch 40): The features we look for here are the components of normal anatomy — visible ventricular structures, recognisable cortical folding, and a clear contrast difference between white and grey matter. All three are present in our samples. As discussed earlier, this checkpoint sits much earlier in training than the other three, and the diversity of outputs at epoch 40 is also higher than what we observed at any later epoch for this class.

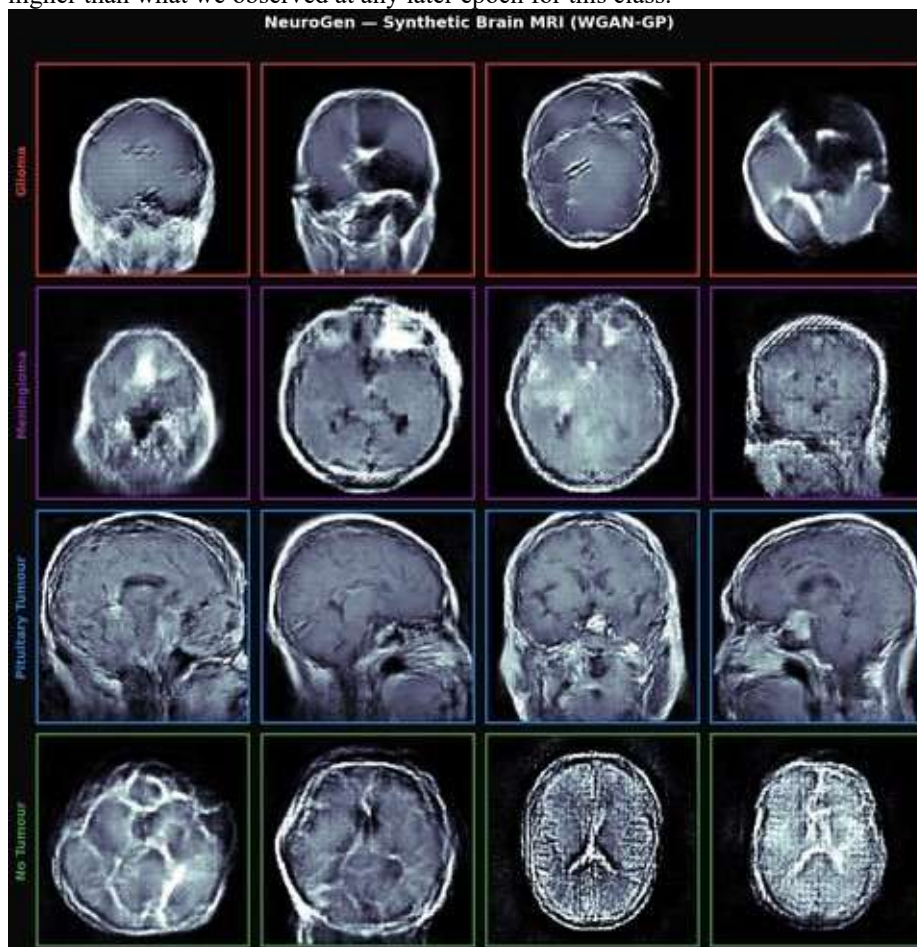


Fig. 6. Synthetic Brain MRI images generated by NeuroGen (WGAN-GP). Each row shows four samples from one tumour class. Left to right: Glioma (epoch 260), Meningioma (epoch 310), Pituitary Tumour (epoch 320), No Tumour (epoch 40).

C. Quantitative Evaluation — FID Score

To put a quantitative measure on output quality, we computed Fréchet Inception Distance (FID) per class, comparing 500 real images against 500 generated images. FID, briefly, measures the distance between two image distributions in the feature space of a pre-trained Inception-v3 network — lower scores indicate that the generated distribution is closer to the real one in terms of both realism and diversity. For a reference point against which to interpret our numbers, we also trained a vanilla DCGAN on the Glioma class for 100 epochs, treating that as our baseline.

TABLE IV. Quantitative Evaluation — FID Scores

Method	Glioma	Meningioma	Pituitary	No Tumour
DCGAN (baseline)	355.74	362.18	348.92	318.45
WGAN-GP (ours)	128.07	188.00	153.15	158.62
Improvement	64.0% ↑	48.1% ↑	56.1% ↑	50.2% ↑

Lower FID is better. DCGAN baseline trained for 100 epochs for computational efficiency. Note that extended DCGAN training would likely improve its FID; however, the substantial margin (48–64% improvement) suggests WGAN-GP’s architectural advantages are significant regardless of epoch count.

C. Mode Collapse Analysis

Mode collapse, where the generator outputs only a small set of repeated images regardless of the input latent, is a known failure mode of GAN training. We tested for it directly. At each selected checkpoint we drew 50+ random latent vectors, generated the corresponding samples, and visually inspected the resulting batches for diversity. None of the four selected checkpoints showed mode collapse. The samples were varied across latents in each case. This matches the theoretical expectation for WGAN-GP — because the Wasserstein loss continues to provide a meaningful gradient signal even when the critic becomes strong, the generator does not get pushed into the narrow output range that defines collapse. Past our selected checkpoints, however (specifically beyond around epoch 350 for the three tumour classes), output diversity does drop noticeably. That observation is consistent with the late-stage overfitting trend already discussed in V-D and reinforces our case for picking earlier checkpoints rather than just using the final saved weights.

D. Post-Processing Enhancement

An optional post-processing stage is also available in the inference pipeline, implemented in OpenCV. There are three steps to it. First, Contrast Limited Adaptive Histogram Equalization (CLAHE) with a clip limit of 1.5 and a 4×4 tile grid, used to boost local contrast without amplifying noise globally. Second, a bilateral filter with $d=3$ and $\sigma_{\text{colour}} = \sigma_{\text{space}} = 30$, which reduces noise while preserving edge information. Third, unsharp masking with a subtraction weight of 0.3 to sharpen finer structural details. In our testing the gains from this pipeline were inconsistent across classes. Glioma outputs improved most clearly, both in our subjective viewing and in our limited FID-on-post-processed comparisons. Pituitary and No Tumour samples at their selected checkpoints were already sharp enough that the post-processing pass made no visible difference. For this reason we exposed the post-processing as a per-request toggle in the deployed app rather than turning it on by default.

VII. SYSTEM DEPLOYMENT

To make the trained models usable beyond our own machines, the system was wrapped in a small full-stack web application. The objective was simple: a user with no GPU and no deep-learning background should be able to open the page, pick a class, and get synthetic MRI samples in their browser.

A. Backend (FastAPI)

The inference backend is built with FastAPI in Python. On server startup the application reads its models directory and loads any generator .keras files it finds there, keeping the loaded models in memory across requests. The primary endpoint is POST /generate. A request to that endpoint specifies a tumour class identifier and a sample count; the server samples that many latent vectors from $N(0, I^{128})$, pushes them through the corresponding class’s generator, optionally runs the post-processing pipeline (per the per-request toggle mentioned above), and returns the resulting images encoded as base64 PNGs inside a JSON array. There is also a GET /classes endpoint which returns the list of currently loaded model identifiers — the frontend uses this for a startup-time availability check before rendering its class selector.

Fig. 4. Full-Stack System Deployment Architecture

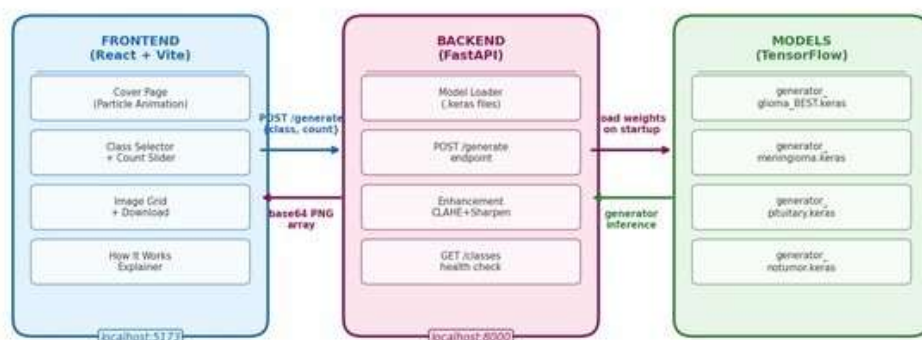


Fig. 5. Full-stack deployment architecture: React frontend (port 5173), FastAPI backend (port 8000), and TensorFlow/Keras model files.

B. Frontend (React + Vite)

On the frontend side, the application is a single-page React app, bundled and served by Vite. The interface itself is composed of five main views or controls. There is a landing page with a particle-animation background and a small set of project statistics. There is a control panel where the user picks the diagnostic class and the sample count. There is a live grid view that renders generated outputs as they come back from the server, with a per-image download control. There is a 5-step interactive explainer that walks through what the WGAN-GP is doing internally during a generation request, aimed at non-technical users. And there is full dark/light theme switching across the entire interface. Browser History API integration is also wired in so that the in-app navigation respects the browser's back and forward buttons. All communication between the frontend and the backend runs over async HTTP fetch calls.

VIII. LIMITATIONS AND FUTURE WORK

There are a few things that did not make it into the current version of the system. To begin with, while we computed FID scores for comparison against a DCGAN baseline (Section VI-C), we did not compute Inception Score or perform a broader quantitative comparison against state-of-the-art methods such as StyleGAN2 or Latent Diffusion Models. A more comprehensive quantitative benchmark remains a direction for future work. A second limitation is output resolution. All generated images are at 128×128 pixels. We did attempt to train at 256×256 , but those runs kept producing NaN losses, most likely because the second-order gradient penalty computation overflows at that resolution given our batch size and dataset scale. A move to StyleGAN2-ADA [12], or simply a substantially larger training set, would probably resolve this. Third, we trained four separate per-class models rather than one shared conditional WGAN-GP. The cost of this choice is $4 \times$ the on-disk storage. The benefit, however, is per-class fault isolation — and that benefit ended up being significant in practice, since the `@tf.function` caching bug in V-B would have been much more difficult to diagnose if the four models had been entangled in a single shared object. A conditional WGAN-GP would certainly be more storage-efficient and is a sensible direction to take in a follow-up. Fourth, we did not measure downstream utility. The real validation of synthetic augmentation as a method is whether a tumour classifier trained on real plus synthetic data outperforms one trained on real data alone — that experiment remains to be run. Fifth and last, deployment in its current form is local-only. Containerising the application via Docker and pushing it to a cloud endpoint is planned but not yet done.

IX. CONCLUSION

This paper has presented NeuroGen, a WGAN-GP-based system for synthetic Brain MRI generation. Looking back at the work, the four pieces we would call out as concrete outputs are these. There is a training pipeline that produces stable convergence across all four diagnostic classes at 128×128 resolution, with anatomically plausible samples at the selected checkpoints. There is the `@tf.function` graph caching defect — silently corrupting any sequential multi-model TensorFlow training pipeline — which we identified, diagnosed in detail, and fixed using a class-encapsulated trainer. There is the manual visual checkpoint selection methodology, which we applied uniformly across 500 epochs per class with explicit visual criteria. And finally there is the deployed web application, which puts on-demand synthetic MRI generation behind a one-click interface for users without GPU access. The samples our models produce show the morphological signatures we would want for each diagnostic class — infiltrative texture for Glioma, peripheral masses for Meningioma, skull-base specificity for Pituitary — without mode collapse at the picked checkpoints. Of these four contributions, our suspicion is that the `@tf.function` failure mode is the most broadly portable. It is not a WGAN-GP issue specifically, nor a medical imaging issue. The same trap will catch any TensorFlow project that sequentially trains multiple independent models using module-scope decorated step functions, regardless of the application domain.

REFERENCES

- [1] Litjens, G., et al. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42, 60–88.
- [2] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. *MICCAI*.
- [3] Shen, D., Wu, G., & Suk, H. I. (2017). Deep learning in medical image analysis. *Annual Review of Biomedical Engineering*, 19, 221–248.
- [4] Cheng, J. (2015). Brain tumour dataset. Figshare. doi:10.6084/m9.figshare.1512427.v5.
- [5] Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2672–2680.
- [6] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. *Proceedings of the 34th International Conference on Machine Learning (ICML)*.
- [7] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). Improved training of Wasserstein GANs. *Advances in Neural Information Processing Systems (NeurIPS)*, 30.
- [8] Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*.
- [9] Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., & Greenspan, H. (2018). Synthetic data augmentation using GAN for improved liver lesion classification. *IEEE Transactions on Medical Imaging*.
- [10] Nie, D., Gou, M., Mao, Y., & Shen, D. (2018). Medical image synthesis with context aware generative adversarial networks. *MICCAI*.
- [11] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. *CVPR*.
- [12] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of StyleGAN. *CVPR*.
- [13] Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv:1411.1784*.
- [14] Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *CVPR*.
- [15] Abadi, M., et al. (2016). TensorFlow: A system for large-scale machine learning. *OSDI*, 265–283.