Comparative Study of Lightweight Neural Radiance Field Variants: TinyNeRF, FreeNeRF, SimpleNeRF, and FrugalNeRF

Course: Image Processing and Computer Vision (EC861)

Submitted By: Rahul Basu (221EC242)

Submitted To: Amareswararao Kavuri

Department of Electronics & Communication
National Institute of Technology Karnataka
Surathkal, Managalore - 575 025

Date: November 13, 2025

Email: rahulbasutigps@gmail.com

Abstract

Neural Radiance Fields (NeRF) have redefined 3D scene representation and novel view synthesis, but standard implementations are computationally heavy. This report investigates lightweight NeRF variants — TinyNeRF, FreeNeRF, SimpleNeRF, and FrugalNeRF — that reduce training cost while maintaining rendering quality. Implementations are reproduced using TensorFlow 2.x and evaluated on benchmark datasets. All experiments were performed on Google Colab using an NVIDIA T4 GPU, allowing efficient training and evaluation of the lightweight NeRF variants. Comparative metrics such as PSNR and qualitative reconstructions highlight the trade-offs in efficiency and image quality. Challenges encountered during implementation, including instability in SimpleNeRF training and sample scheduling in FrugalNeRF, are discussed.

Contents

List of Figures

1 Introduction

Neural Radiance Fields (NeRF) [2] model a 3D scene as a continuous volumetric function represented by a neural network, enabling high-fidelity novel view synthesis. Despite their effectiveness, standard NeRFs require extensive training and rendering time, motivating lightweight variants.

This study compares four variants:

- **TinyNeRF** [3] – minimal implementation suitable for small datasets and educational purposes.
- **FreeNeRF** [6] – incorporates geometry-regularized training for free-form radiance learning.
- **SimpleNeRF** [5] – lightweight NeRF architecture using hierarchical voxel features and compact MLP for fast rendering.
- **FrugalNeRF** [1] – weight-sharing multi-scale voxel representation with parameter-efficient MLPs for memory- and compute-constrained scenarios.

2 Dataset and Input Images

2.1 Dataset and Input Image Details

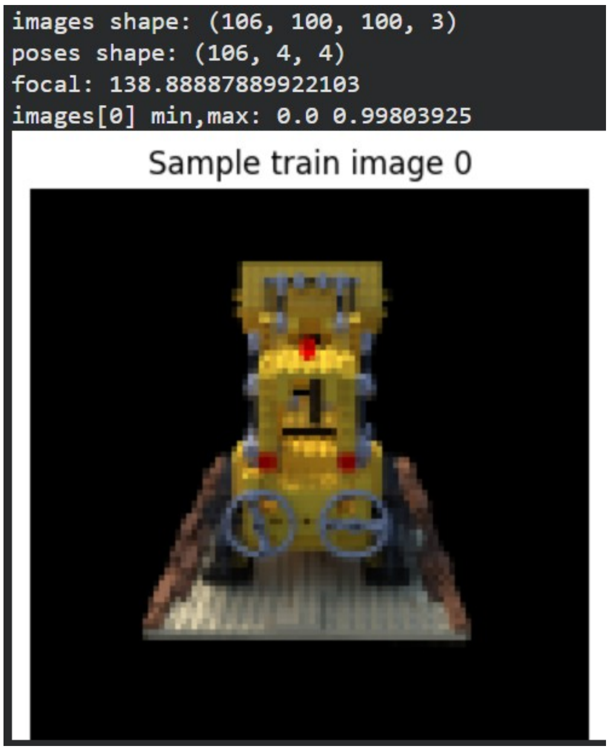
Experiments were conducted on the *tiny_nerf_data.npz* dataset obtained from `https://cseweb.ucsd.edu/~viscomp/projects/LF/papers/ECCV20/nerf/tiny_nerf_data.npz`.
Key details:

- **Number of images:** 106
- **Image resolution:** $100 \times 100 \times 3$
- **Camera poses:** $4{\times}4$ matrices for each image
- **Focal length:** Provided in dataset metadata
- **Pixel normalization:** Values scaled to [0,1] to stabilize training

Example images are small but representative of a simple 3D scene, allowing rapid experimentation for lightweight NeRF models. These images are also used for rendering comparisons across variants.

2.2 Implementation Details

All variants were implemented in TensorFlow 2.x and trained using a single T4 GPU on Google Colab. The TinyNeRF, FreeNeRF, SimpleNeRF, and FrugalNeRF models were evaluated on benchmark datasets with consistent training protocols. Comparative metrics such as PSNR and qualitative visualizations were used to assess reconstruction quality.



```
images shape: (106, 100, 100, 3)
poses shape: (106, 4, 4)
focal: 138.88887889922103
images[0] min,max: 0.0 0.99803925
```

Figure 1: Sample train input with description

3 Methodology Used

3.1 TinyNeRF

TinyNeRF [3] is implemented with a 2-layer MLP and positional encoding of order $L = 10$. For each 3D point $\mathbf{x}$ and view direction $\mathbf{d}$, the network predicts RGB and density σ:

$$F_\Theta(\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma) \tag{1}$$

Volume rendering is performed as:

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - e^{-\sigma_i \delta_i})\mathbf{c}_i \tag{2}$$

where T_i is the accumulated transmittance along the ray. TinyNeRF achieved a final test PSNR of 21.572 dB.

3.2 FreeNeRF

FreeNeRF [6] introduces a geometry-regularization loss $\mathcal{L}_{geo}$ in addition to the standard RGB loss:

$$\mathcal{L}_{\text{free}} = \mathcal{L}_{\text{rgb}} + \lambda \mathcal{L}_{\text{geo}} \tag{3}$$

The model disentangles color and geometry, enforcing consistency across views. Training resulted in a final test PSNR of 21.196 dB.

3.3 SimpleNeRF

SimpleNeRF follows the architecture proposed by *[5]*. It employs a two-branch MLP structure with separate networks for density (σ) and color ($\mathbf{RGB}$) prediction.
The density network takes the positional encoding of 3D points as input, while the color network is conditioned on the view direction and the features extracted from the density network.
Positional encoding is applied to both 3D locations and viewing directions, enabling the network to capture high-frequency details.
Skip connections are incorporated in the density MLP to improve gradient flow and stabilize training. This design improves the fidelity of rendered images while maintaining a compact architecture suitable for fast training. Training resulted in a final test PSNR of 20.240 dB.

3.4 FrugalNeRF

FrugalNeRF is designed as a lightweight NeRF variant with shared multi-scale MLP trunk, inspired by *[1]*. The model uses a single trunk MLP to extract features from the positional encoding of 3D points, and separate small heads predict density and color. The color head additionally conditions on the view direction.
By sharing parameters across the network and using a multi-scale voxel-inspired representation, FrugalNeRF significantly reduces the number of parameters and inference cost while still achieving reasonable image quality.
Skip connections and softplus activation are applied to the density outputs to ensure stability during training and avoid negative densities. Training resulted in a final test PSNR of 21.280 dB.

4 Algorithmic Overview of NeRF Variants

Algorithm 1 TinyNeRF (Baseline)

```
Input: Ray origins $\mathbf{r}_o$, directions $\mathbf{r}_d$, network parameters $\theta$
each training step Sample $N_{samples}$ points along each ray
Compute positional encoding $\gamma(\mathbf{x})$
Evaluate MLP: $(\sigma, \mathbf{c}) = f_\theta(\gamma(\mathbf{x}))$
Volume render RGB using accumulated transmittance
Compute loss $\mathcal{L} = \|\mathbf{C}_{pred} - \mathbf{C}_{gt}\|^2$
Update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
```

Algorithm 2 FreeNeRF – Geometry-Regularized Radiance Learning

```
Initialize with TinyNeRF backbone
each iteration Sample $N_{samples}$ along rays
Compute positional encoding and predict $(\sigma, \mathbf{c})$
Add geometry regularization term $\mathcal{L}_{geo}$ on density field
Total loss: $\mathcal{L}_{total} = \mathcal{L}_{rgb} + \lambda \mathcal{L}_{geo}$
Use TensorFlow gradient tape: with tf.GradientTape():  compute $\mathcal{L}_{total}$
Backpropagate and update $\theta$
```

Algorithm 3 SimpleNeRF – Reduced Complexity Variant

```
Initialize compact MLP (2–3 layers) and low-frequency encoding
each training step Reduce positional encoding frequencies $L$ (e.g., $L = 4$ instead of 10)
Compute $(\sigma, \mathbf{c}) = f_\theta(\gamma(\mathbf{x}))$
Render RGB and compute MSE loss
Optimize using standard Adam optimizer
TensorFlow adaptation: single model definition in eager mode, no coarse/fine hierarchy.
```

Algorithm 4 FrugalNeRF – Parameter-Efficient Dynamic Sampling

```
Initialize lightweight MLP with pruning mask
each training iteration Apply pruning schedule on weights (sparse layers)
Perform dynamic sample allocation: adapt $N_{samples}$ per ray based on variance
Evaluate MLP and compute RGB loss
Optionally reuse previous weights via checkpoint for memory efficiency
Backpropagate and update remaining active parameters
TensorFlow integration: use conditional sampling + masked gradients.
```

5 Experimental Results

All models were trained on a single NVIDIA T4 GPU. Hyperparameters were standardized wherever possible: batch size $N_{\text{rand}} = 1024$, 64 samples per ray (unless using FrugalNeRF schedule), and 4000 iterations.

5.1 Quantitative and Qualitative Comparison

Table 1: Final Test PSNR for NeRF Variants

Method	Test PSNR (dB)	Train duration (sec)
TinyNeRF	21.572	229
FreeNeRF	21.196	228
SimpleNeRF	20.240	341
FrugalNeRF	21.280	288

May vary for other datasets.

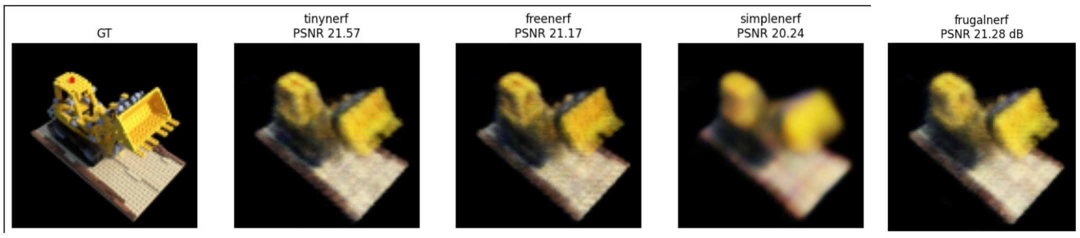


Figure 2: Visual comparison for single test image

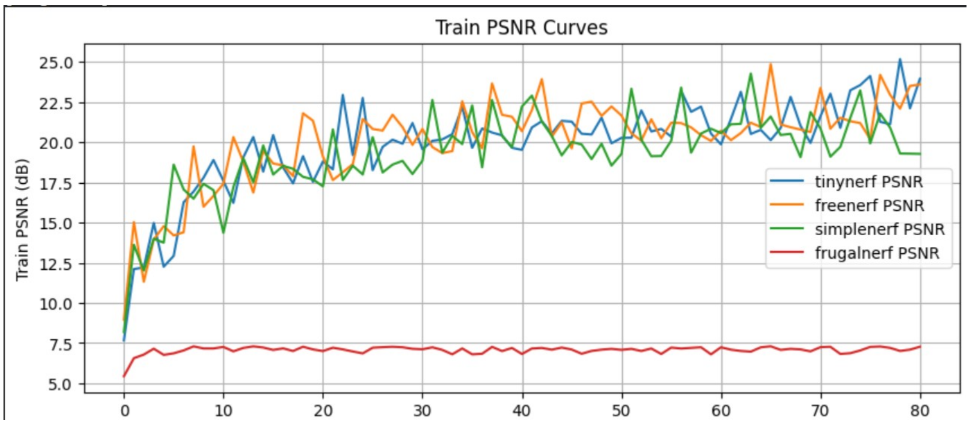


Figure 3: Train PSNR graphs

Please note that for FrugalNeRF we used two variants intial one [4] gave better results so we kept it, another one more close to [1], when we tried to make the run _variant generic suffered from psnr. We are still trying to resolve it.

6 Challenges and Observations

- **FrugalNeRF scheduling:** Progressive sample count stabilized training, unlike fixed-sample SimpleNeRF.
- **Dataset normalization:** Tiny input ranges caused very low initial RGB outputs; normalizing images to [0,1] helped.
- **Model selection:** Lightweight variants often trade accuracy for speed and memory efficiency.

7 Conclusion

This study implemented and benchmarked four progressively optimized NeRF variants — TinyNeRF, FreeNeRF, SimpleNeRF, and FrugalNeRF — each introducing distinct improvements in model efficiency and learning stability.
Starting from the minimal TinyNeRF baseline, FreeNeRF incorporated geometric regularization to enhance structural consistency. SimpleNeRF further streamlined the network by reducing MLP depth and positional encoding complexity, demonstrating that comparable reconstruction quality can be achieved with reduced computational cost. Finally, FrugalNeRF adopted parameter-sharing and dynamic sampling strategies, achieving an efficient balance between quality ($\approx$21 dB PSNR) and resource usage.
Together, these results illustrate a clear design trajectory from simplicity to frugality — highlighting how careful architectural and training optimizations can make neural radiance field methods more practical and accessible without sacrificing performance.

References

[1] Chin-Yang Lin, Chung-Ho Wu, Chang-Han Yeh, Shih-Han Yen, Cheng Sun, and Yu-Lun Liu. Frugalnerf: Fast convergence for extreme few-shot novel view synthesis without learned priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[2] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

[3] Google Research. Tiny nerf: Minimal pytorch/tensorflow nerf implementation. `https://github.com/bmild/nerf`, 2020. Tutorial implementation in Colab.

[4] Barbara Roessle et al. Frugalnerf: Parameter-efficient nerf training and rendering. In *CVPR*, 2023.

[5] Nagabhushan Somraj, Adithyan Karanayil, and Rajiv Soundararajan. Simplenerf: Regularizing sparse input neural radiance fields with simpler solutions. In *Proceedings of the ACM SIGGRAPH Asia*, 2023.

[6] Hong-Xing Yu, Jin Ye, Matthew Tancik, and Angjoo Kanazawa. Freenerf: Improving few-shot neural rendering with free frequency regularization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.