A. Theory Behind Generalized Exponentials

A.1. Generalized Exponential Function

The Generalized Exponential Function (GEF) is similar to the probability density function (PDF) of the Generalized Normal Distribution (GND) [14] with an additional amplitude parameter $A \in \mathbb{R}$. This function allows for a more flexible adaptation to various data shapes by adjusting the shape parameter $\beta \in (0, \infty)$. The GEF is given by the following.

$$f(x|\mu, \alpha, \beta, A) = A \exp\left(-\left(\frac{|x-\mu|}{\alpha}\right)^{\beta}\right) \tag{10}$$

where $\mu \in \mathbb{R}$ is the location parameter, $\alpha \in \mathbb{R}$ is the scale parameter, A defines the amplitude, and $\beta > 0$ is the shape parameter. For $\beta = 2$, the GEF becomes a scaled Gaussian distribution:

$$f(x|\mu, \alpha, \beta = 2, A) = \frac{A}{\alpha\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\alpha/\sqrt{2}}\right)^2\right) \tag{11}$$

And for $\beta = 1$, Eq. 10 reduces to a scaled Laplace distribution:

$$f(x|\mu, \alpha, \beta = 1, A) = \frac{A}{2\alpha} \exp\left(-\frac{|x-\mu|}{\alpha}\right) \tag{12}$$

The GEF, therefore, provides a versatile framework for modeling a wide range of data by varying β, unlike the Gaussian mixtures, which have a low-pass frequency domain. Many common signals, like the square or triangle, are band-unlimited, constituting a fundamental challenge to Gaussian-based methods (see Fig.12). In this paper, we try to *learn* a positive β for every component of the Gaussian splatting to allow for a generalized 3D representation.

A.2. Theoretical Results

Despite its generalizable capabilities, the GEF has no fixed behavior in terms of frequency domain. The error functions of the GEF and its Fourier domain cannot be studied analytically, as they involve complex integrals of exponentials without closed form that depend on the shape parameter β. For example, the Fourier of GEF is given by

$$\mathcal{F}(f)(\xi) = \int_{-\infty}^{\infty} A \exp\left(-\left(\frac{|x-\mu|}{\alpha}\right)^{\beta}\right) e^{-2\pi i x\xi}\, dx$$

which does not have a closed-form solution for a general β. We demonstrate that for specific cases, such as for a square signal, the GEF can achieve a smaller approximation error than the corresponding Gaussian function by properly choosing β. Theorem 1 provides a theoretical foundation for preferring the GEF over standard Gaussian functions in our GES representation instead of 3D Gaussian Splatting [27].

Theorem 1 (Superiority of GEF Approximation Over Gaussian for Square Wave Signals)**.** *Let $S(t)$ represent a square wave signal with amplitude $A > 0$ and width $L > 0$ centered at $t = 0$. Define two functions: a scaled Gaussian $G(t; \alpha, A) = Ae^{-\frac{t^2}{\alpha^2}}$, and a Generalized Exponential Function $GEF(t; \alpha, \beta, A) = Ae^{-(|t|/\alpha)^{\beta}}$. For any given scale parameter α, there exists a shape parameter β such that the approximation error $E_f = \int_{-\infty}^{\infty} |S(t) - f(t)|dt$. of the square signal $S(t)$ using GEF is strictly smaller than that using the Gaussian G.*

Proof. The error metric E_f for the square signal $S(t)$ approximation using f function as $E_f = \int_{-\infty}^{\infty} |S(t) - f(t)|dt$. Utilizing symmetry and definition of $S(t)$, and the fact that $S(t) > G(t; \alpha, A)$, the error for the Gaussian approximation simplifies to:

$$E_G = 2\int_0^{L/2} A(1 - e^{-\frac{t^2}{\alpha^2}})dt + 2\int_{L/2}^{\infty} Ae^{-\frac{t^2}{\alpha^2}}\, dt.$$

For the GEF approximation, the error is:

$$E_{GEF} = 2\int_0^{L/2} A(1-e^{-(t/\alpha)^{\beta}})dt + 2\int_{L/2}^{\infty} Ae^{-(t/\alpha)^{\beta}}\, dt.$$

The goal is to show the difference in errors $\Delta E = E_G - E_{GEF}$ to be strictly positive, by picking β appropriately. The error difference can be described as follows.

$$\Delta E = \Delta E_{middle} + \Delta E_{tail}$$

$$\Delta E_{middle} = 2\int_0^{L/2} A(1-e^{-\frac{t^2}{\alpha^2}})dt \; - \; 2\int_0^{L/2} A(1-e^{-(t/\alpha)^{\beta}})dt$$

$$\Delta E_{tail} = 2\int_{L/2}^{\infty} Ae^{-\frac{t^2}{\alpha^2}}\, dt - 2\int_{L/2}^{\infty} Ae^{-(t/\alpha)^{\beta}}\, dt$$

Let us Define $\text{err}(t)$ as the difference between the exponential terms:

$$\text{err}(t) = e^{-\frac{t^2}{\alpha^2}} - e^{-(t/\alpha)^{\beta}}.$$

The difference in the middle error terms for the Gaussian and GEF approximations, ΔE_{middle}, can be expressed using $\text{err}(t)$ as:

$$\Delta E_{middle} = 2A\int_0^{L/2} \text{err}(t)\, dt.$$

Using the trapezoidal approximation of the integral, this simplifies to:

$$\Delta E_{middle} \approx LA\, \text{err}(L/2) = LA\left(e^{-\frac{L^2}{4\alpha^2}} - e^{-(L/2\alpha)^{\beta}}\right).$$

Based on the fact that the negative exponential is monotonically decreasing and to ensure ΔE_{middle} is always positive, we choose β based on the relationship between $L/2$ and α :

- If $\frac{L}{2} > \alpha$ (i.e., $\frac{L}{2\alpha} > 1$), choosing $\beta > 2$ ensures $e^{-(L/2\alpha)^\beta} < e^{-\frac{L^2}{4\alpha^2}}$.

- If $\frac{L}{2} < \alpha$ (i.e., $\frac{L}{2\alpha} < 1$), choosing $0 < \beta < 2$ results in $e^{-(L/2\alpha)^\beta} < e^{-\frac{L^2}{4\alpha^2}}$.

Thus, ΔE_{middle} can always be made positive by choosing β appropriately, implying that the error in the GEF approximation in the interval $[-L/2, L/2]$ is always less than that of the Gaussian approximation. Similarly, the difference of tail errors ΔE_{tail} can be made positive by an appropriate choice of β, concluding that the total error E_{GEF} is strictly less than E_G. This concludes the proof. □

A.3. Numerical Simulation of Gradient-Based 1D Mixtures

Objective. The primary objective of this numerical simulation is to evaluate the effectiveness of the generalized exponential model in representing various one-dimensional (1D) signal types. This evaluation was conducted by fitting the model to synthetic signals generated to embody characteristics of square, triangle, parabolic, half sinusoidal, Gaussian, and exponential functions, which can constitute a non-exclusive list of basic topologies available in the real world.
Simulation Setup. The experimental framework was based on a series of parametric models implemented in PyTorch, designed to approximate 1D signals using mixtures of different functions such as Gaussian, Difference of Gaussians (DoG), Laplacian of Gaussian (LoG), and a Generalized mixture model. Each model comprised parameters for means, variances (or scales), and weights, with the generalized model incorporating an additional parameter, β, to control the exponentiation of the Gaussian function.
Models. Here, we describe the mixture models used to approximate the true signal forms.

- **Gaussian Mixture Model (GMM):** The GMM combines several Gaussian functions, each defined by its mean (μ_i), variance (σ_i^2), and weight (w_i). For a set of N Gaussian functions, the mixture model $g(x)$ can be expressed as:

$$g(x) = \sum_{i=1}^{N} w_i \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2 + \epsilon}\right), \tag{13}$$

  where ϵ is a small constant to avoid division by zero, with $\epsilon = 1e-8$.

- **Difference of Gaussians (DoG) Mixture Model:** The DoG mixture model is comprised of elements that represent the difference between two Gaussian functions with a fixed variance ratio ν. The model $d(x)$ for N components is given by:

$$d(x) = \sum_{i=1}^{N} w_i D_i$$
$$D_i = \left(\exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2 + \epsilon}\right) - \exp\left(-\frac{(x-\mu_i)^2}{2(\sigma_i^2/\nu) + \epsilon}\right)\right), \tag{14}$$

  where σ_i is a scale parameter, and the variance ratio ν is fixed to be 4.

- **Laplacian of Gaussian (LoG) Mixture Model:** The LoG mixture model is formed by a series of Laplacian of Gaussian functions, each defined by a mean (μ_i), scale (γ_i), and weight (w_i). The mixture model $l(x)$ is:

$$l(x) = \sum_{i=1}^{N} w_i \left(-\frac{(x-\mu_i)^2}{\gamma_i^2} + 1\right) \exp\left(-\frac{(x-\mu_i)^2}{2\gamma_i^2 + \epsilon}\right), \tag{15}$$

- **Generalized Mixture Model:** This model generalizes the Gaussian mixture by introducing a shape parameter β. Each component of the model $h(x)$ is expressed as:

$$h(x) = \sum_{i=1}^{N} w_i \exp\left(-\frac{|x-\mu_i|^\beta}{2\sigma_i^2 + \epsilon}\right), \tag{16}$$

  where β is a learnable parameter that is optimized alongside other parameters. When $\beta = 2$ is fixed, the equation in Eq.(16) reduces to the one in Eq.(13).

Model Configuration. The models were configured with a varying number of components N, with tests conducted using $N = \{2, 5, 8, 10, 15, 20, 50, 100\}$. The weights of the components could be either positive or unrestricted. For the generalized model, the β parameter was learnable.
Training Procedure. Each model was trained using the Adam optimizer with a mean squared error loss function. The input x was a linearly spaced tensor representing the domain of the synthetic signal, and the target y was the value of the signal at each point in x. Training proceeded for a predetermined number of epochs, and the loss was recorded at the end of training.
Data Generation. Synthetic 1D signals were generated for various signal types over a specified range, with a given data size and signal width. The signals were used as the ground truth for training the mixture models. The ground truth signals used in the experiment are one-dimensional (1D) functions that serve as benchmarks for evaluating signal processing algorithms. Each signal type is defined within a specified width around the origin, and the value outside this interval is zero (see Fig.12). The parameter `width` σ dictates the effective span of the non-zero portion of the signal. We define six distinct signal types as follows:

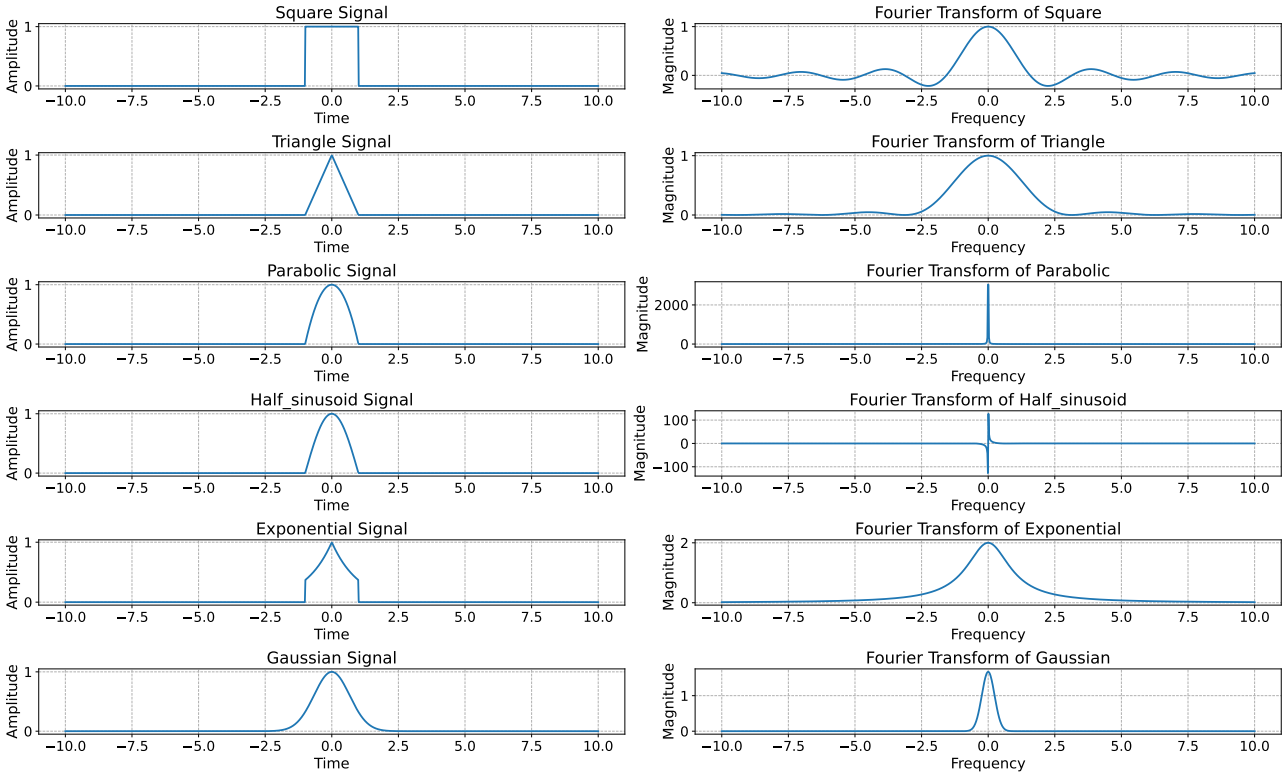


Figure 12. **Commong Signals Used and Their Fourier Transforms**. Note that the Gaussian function is low-pass bandwidth, while common signals like the square and triangle with sharp edges have infinite bandwidth, making them challenging to be fitted with mixtures that have low-pass frequency bandwidth (*e.g.* Gaussian mixtures, represented by Gaussian Splatting [27]).

1. **Square Signal:** The square signal is a binary function where the value is 1 within the interval $(-\frac{\sigma}{2}, \frac{\sigma}{2})$ and 0 elsewhere. Mathematically, it is represented as

$$f_{\text{square}}(x) = \begin{cases} 1 & \text{if } -\frac{\sigma}{2} < x < \frac{\sigma}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

   Its Fourier Transform is given by

$$\text{FT}\{\text{Square Wave}\}(f) = \text{sinc}\left(\frac{f \cdot \sigma}{\pi}\right) \tag{18}$$

2. **Triangle Signal:** This signal increases linearly from the left edge of the interval to the center and decreases symmetrically to the right edge, forming a triangular shape. It is defined as

$$f_{\text{triangle}}(x) = \begin{cases} \frac{\sigma}{2} - |x| & \text{if } -\frac{\sigma}{2} < x < \frac{\sigma}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

   Its Fourier Transform is

$$\text{FT}\{\text{Triangle Wave}\}(f) = \left(\text{sinc}\left(\frac{f \cdot \sigma}{2\pi}\right)\right)^2 \tag{20}$$

3. **Parabolic Signal:** This signal forms a downward-facing parabola within the interval, and its expression is

$$f_{\text{parabolic}}(x) = \begin{cases} (\frac{\sigma}{2})^2 - x^2 & \text{if } -\frac{\sigma}{2} < x < \frac{\sigma}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

   The Fourier Transform of the parabolic signal is

$$\text{FT}\{\text{Parabolic Wave}\}(f) = \frac{3 \cdot \left(\text{sinc}\left(\frac{f \cdot \sigma}{2\pi}\right)\right)^2}{\pi^2 \cdot f^2} \tag{22}$$

4. **Half Sinusoid Signal:** A half-cycle of a sine wave is contained within the interval, starting and ending with zero amplitude. Its formula is

$$f_{\text{half\_sinusoid}}(x) = \begin{cases} \sin\left((x + \frac{\sigma}{2})\frac{\pi}{\sigma}\right) & \text{if } -\frac{\sigma}{2} < x < \frac{\sigma}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

   Its Fourier Transform is described by

$$\text{FT}\{\text{Half Sinusoid}\}(f) = \begin{cases} \frac{\sigma}{2} & \text{if } f = 0 \\ \frac{\sigma \cdot \sin(\pi \cdot f \cdot \sigma)}{\pi^2 \cdot f^2} & \text{otherwise} \end{cases} \tag{24}$$

5. **Exponential Signal:** Exhibiting an exponential decay centered at the origin, this signal is represented by

$$f_{\text{exponential}}(x) = \begin{cases} \exp(-|x|) & \text{if } -\frac{\sigma}{2} < x < \frac{\sigma}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$

The Fourier Transform for the exponential signal is

$$\text{FT}\{\text{Exponential}\}(f) = \frac{\sigma}{f^2 + \left(\frac{\sigma}{2}\right)^2} \tag{26}$$

6. **Gaussian Signal:** Unlike the others, the Gaussian signal is not bounded within a specific interval but instead extends over the entire range of x, with its amplitude governed by a Gaussian distribution. It is given by

$$f_{\text{Gaussian}}(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right). \tag{27}$$

The Fourier Transform of the Gaussian signal is also a Gaussian, which in the context of standard deviation σ is represented as

$$\text{FT}\{\text{Gaussian}\}(f) = \sqrt{2\pi}\cdot\sigma\cdot\exp\left(-2\pi^2\sigma^2 f^2\right) \tag{28}$$

As shown in Fig.12, the Gaussian function has a low-pass band, while signals like the square and triangle with sharp edges have infinite bandwidth, making them challenging for mixtures that have low-pass frequency bandwidth (*e.g.* Gaussian mixtures, represented by Gaussian Splatting [27]).

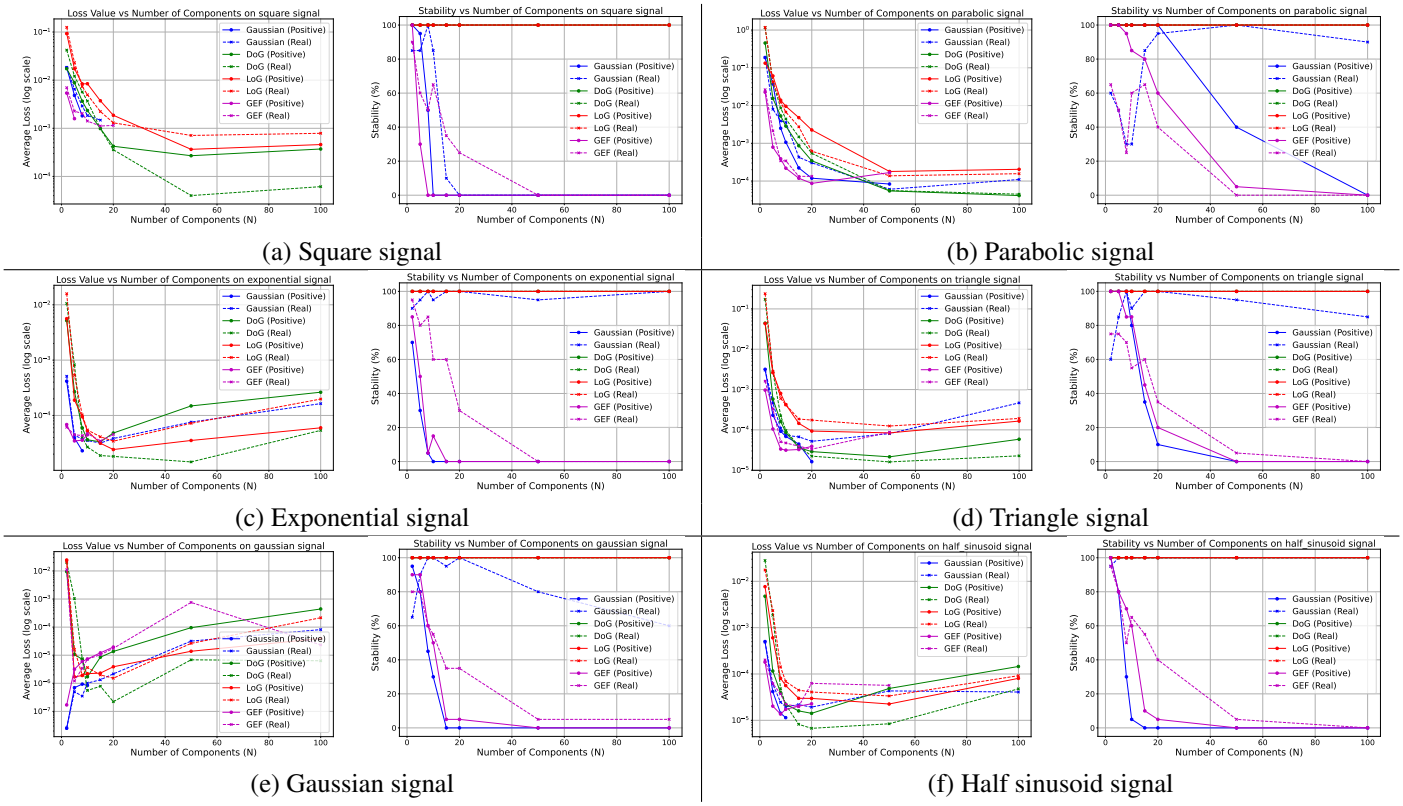
Each signal is sampled at discrete points using a PyTorch tensor to facilitate computational manipulation and analysis within the experiment's framework. We show in Fig.14,15,16,17,18,19,20,21,22,23,24, and 25 examples of fitting all the mixture on all different signal types of interest when positive weighting is used in the mixture *vs.* when allowing real weighting in the combinations in the above equations. Note how sharp edges constitute a challenge for Gaussians that have low pass bandwidth while a square signal has an infinite bandwidth known by the sinc function [25].

Loss Evaluation. The models' performance was evaluated based on the loss value after training. Additionally, the model's ability to represent the input signal was visually inspected through generated plots. Multiple runs per configuration were executed to account for variance in the results.

Stability Evaluation. Model stability and performance were assessed using a series of experiments involving various signal types and mixture models. Each model was trained on a 1D signal generated according to predefined signal types (square, triangle, parabolic, half sinusoid, Gaussian, and exponential), with the goal of minimizing the mean squared error (MSE) loss between the model output and the ground truth signal. The number of components in the mixture models (N) varied among a set of values, and models were also differentiated based on whether they were constrained to positive weights. For a comprehensive evaluation, each configuration was run multiple times (20 runs per configuration) to account for variability in the training process. During these runs, the number of instances where the training resulted in a `NaN` loss was recorded as an indicator of stability issues. The stability of each model was quantified by the percentage of successful training runs ($\frac{\text{Total Runs}-\text{NaN Loss Counts}}{\text{Total Runs}} \times 100\%$). The experiments that failed failed because the loss has diverged to `NaN`. This typical numerical instability in optimization is the result of learning the variance which can go close to zero, resulting in the exponential formula (in Eq.(10)) to divide by an extremely small number.

The average MSE loss from successful runs was calculated to provide a measure of model performance. The results of these experiments were plotted, showing the relationship between the number of components and the stability and loss of the models for each signal type.

Simulation Results. In the conducted analysis, both the loss and stability of various mixture models with positive and non-positive weights were evaluated on signals with different shapes. As depicted in Figure 13, the Gaussian Mixture Model with positive weights consistently yielded the lowest loss across the number of components, indicating its effective approximation of the square signal. Conversely, non-positive weights in the Gaussian and General models showed a higher loss, emphasizing the importance of weight sign-on model performance. These findings highlight the intricate balance between model complexity and weight constraints in achieving both low loss and high stability. Note that GEF is very efficient in fitting the square with few components, while LoG and DoG are more stable for a larger number of components. Also, note that positive weight mixtures tend to achieve lower loss with a smaller number of components but are less stable for a larger number of components.



(a) Square signal (b) Parabolic signal

(c) Exponential signal (d) Triangle signal

(e) Gaussian signal (f) Half sinusoid signal

Figure 13. **Numerical Simulation Results of Different Mixtures.** We show a comparison of average loss and stability (percentage of successful runs) for different mixture models optimized with gradient-based optimizers across varying numbers of components and weight configurations (positive *vs*. real weights) on various signal types (a-f).

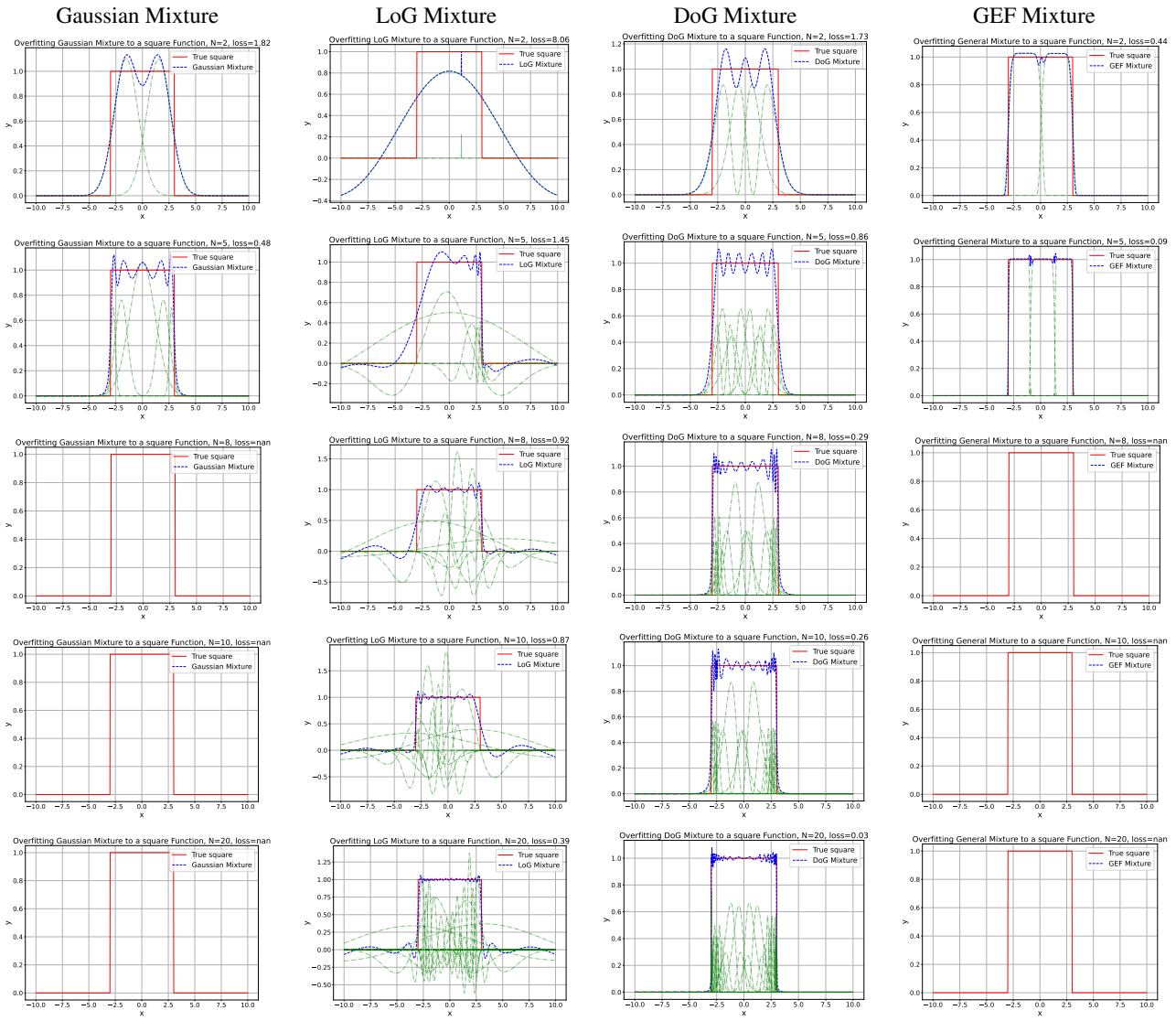


Figure 14. **Numerical Simulation Examples of Fitting Squares with Positive Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for Square signals with positive weights mixtures. The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the Square with few components while LoG and DoG are more stable for a larger number of components.

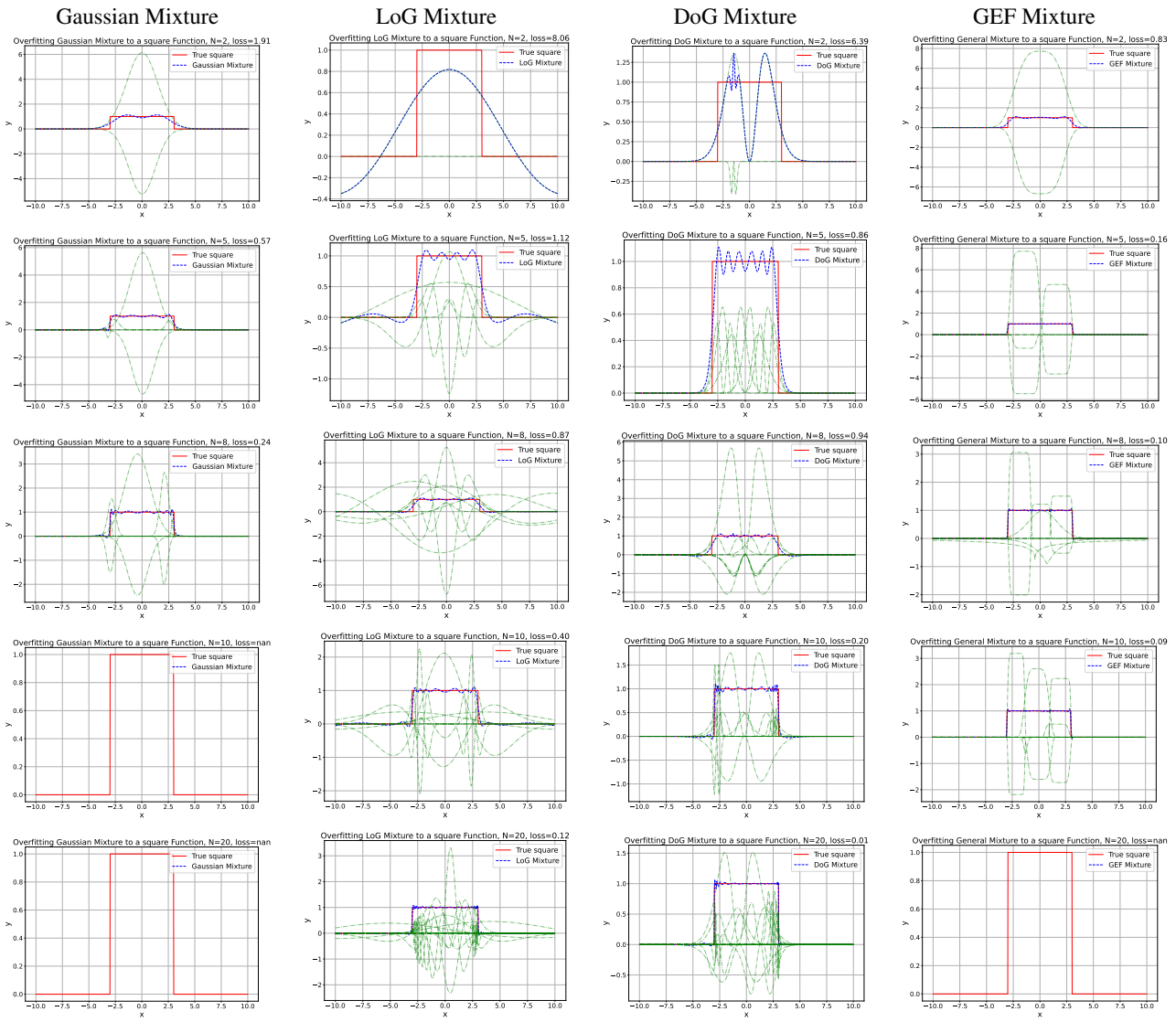


Figure 15. **Numerical Simulation Examples of Fitting Squares with Real Weights Mixtures (N= 2, 5, 8, and 10).** We show some fitting examples for Square signals with Real weights mixtures (can be negative). The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the Square with few components while LoG and DoG are more stable for a larger number of components.

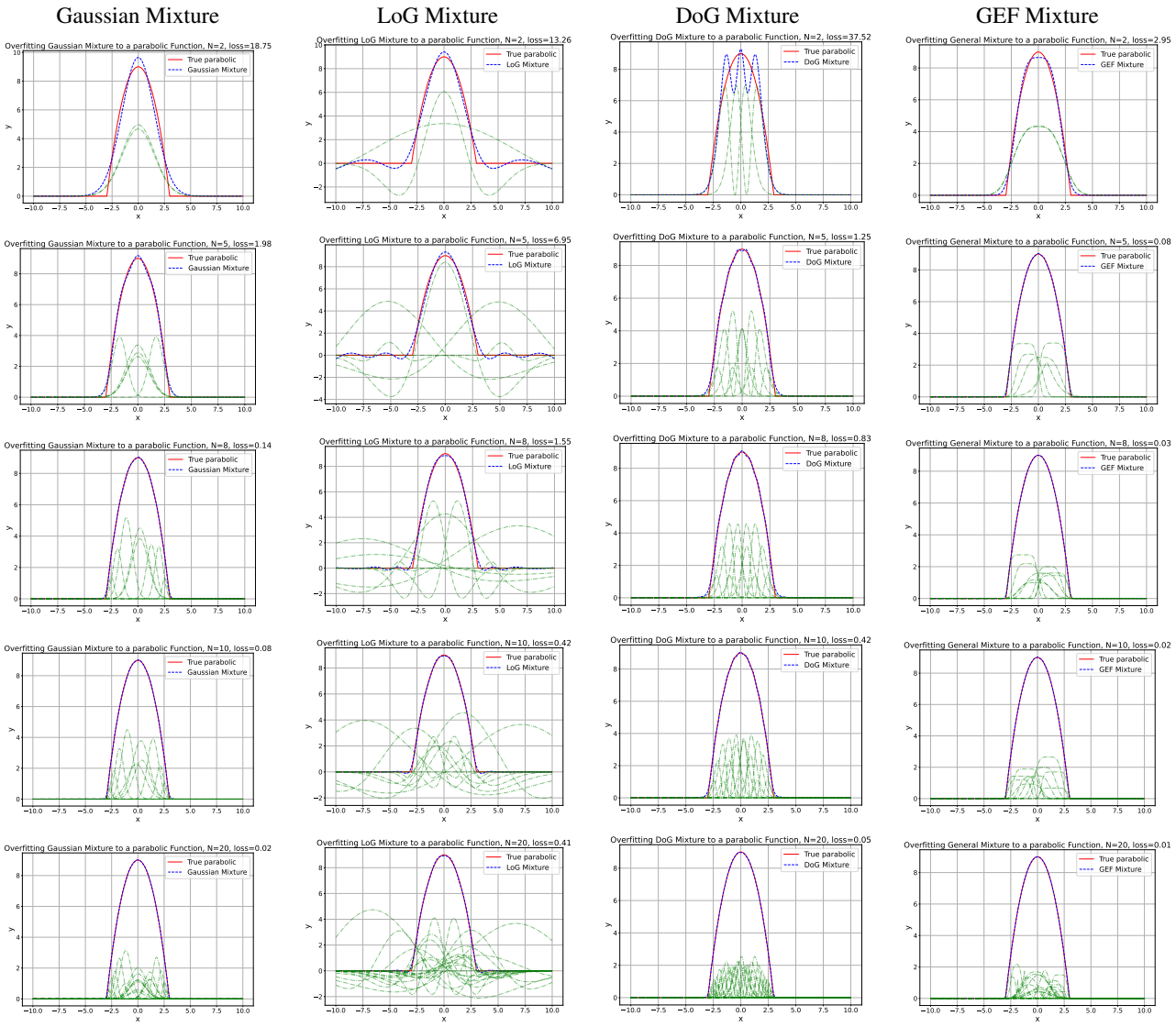


Figure 16. **Numerical Simulation Examples of Fitting parabolics with Positive Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for parabolic signals with positive weights mixtures. The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the parabolic with few components while LoG and DoG are more stable for a larger number of components.

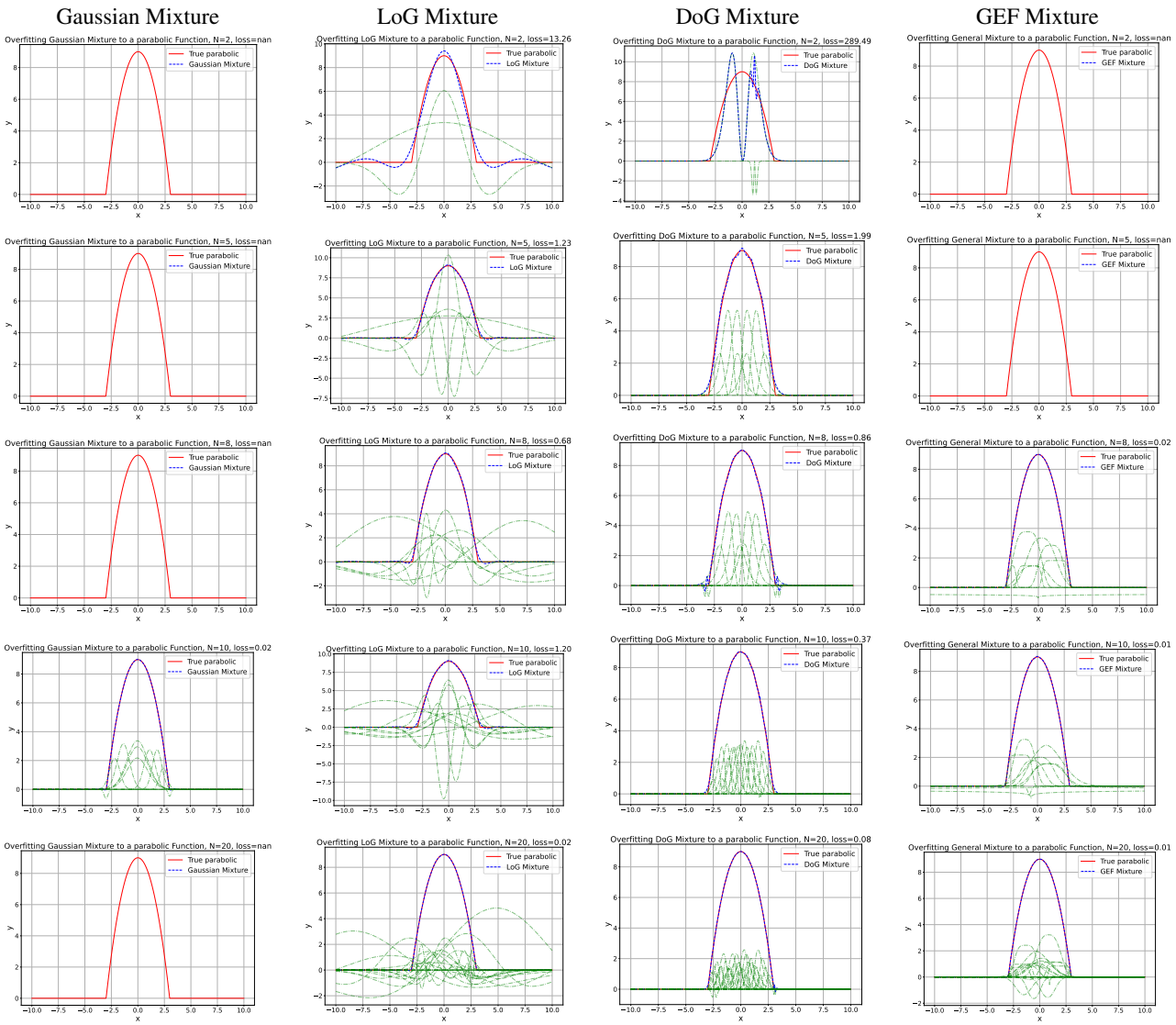


Figure 17. **Numerical Simulation Examples of Fitting Parabolics with Real Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for parabolic signals with Real weights mixtures (can be negative). The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the parabolic with few components while LoG and DoG are more stable for a larger number of components.

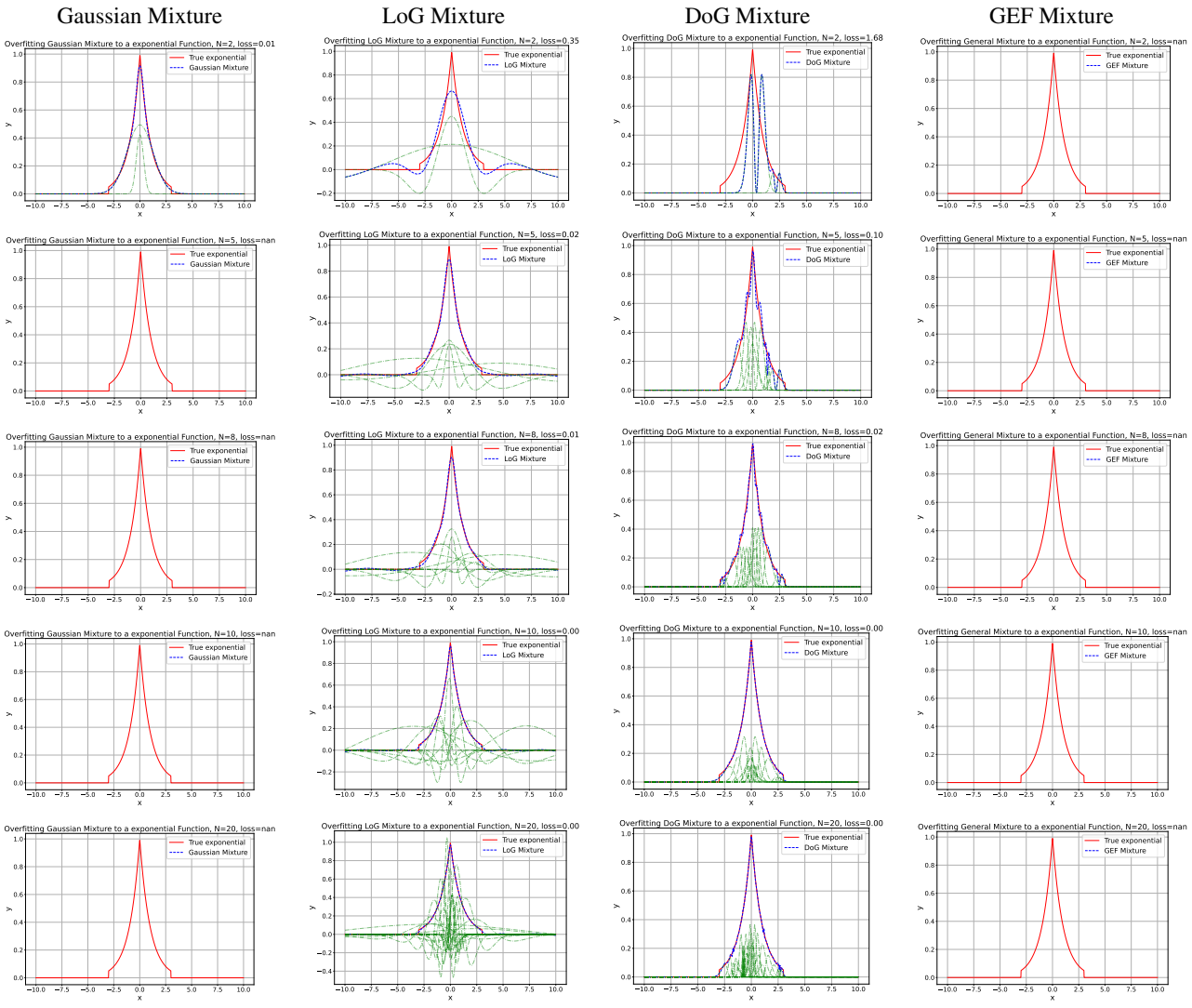


Figure 18. **Numerical Simulation Examples of Fitting Exponentials with Positive Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for exponential signals with positive weight mixtures. The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the exponential with few components while LoG and DoG are more stable for a larger number of components.

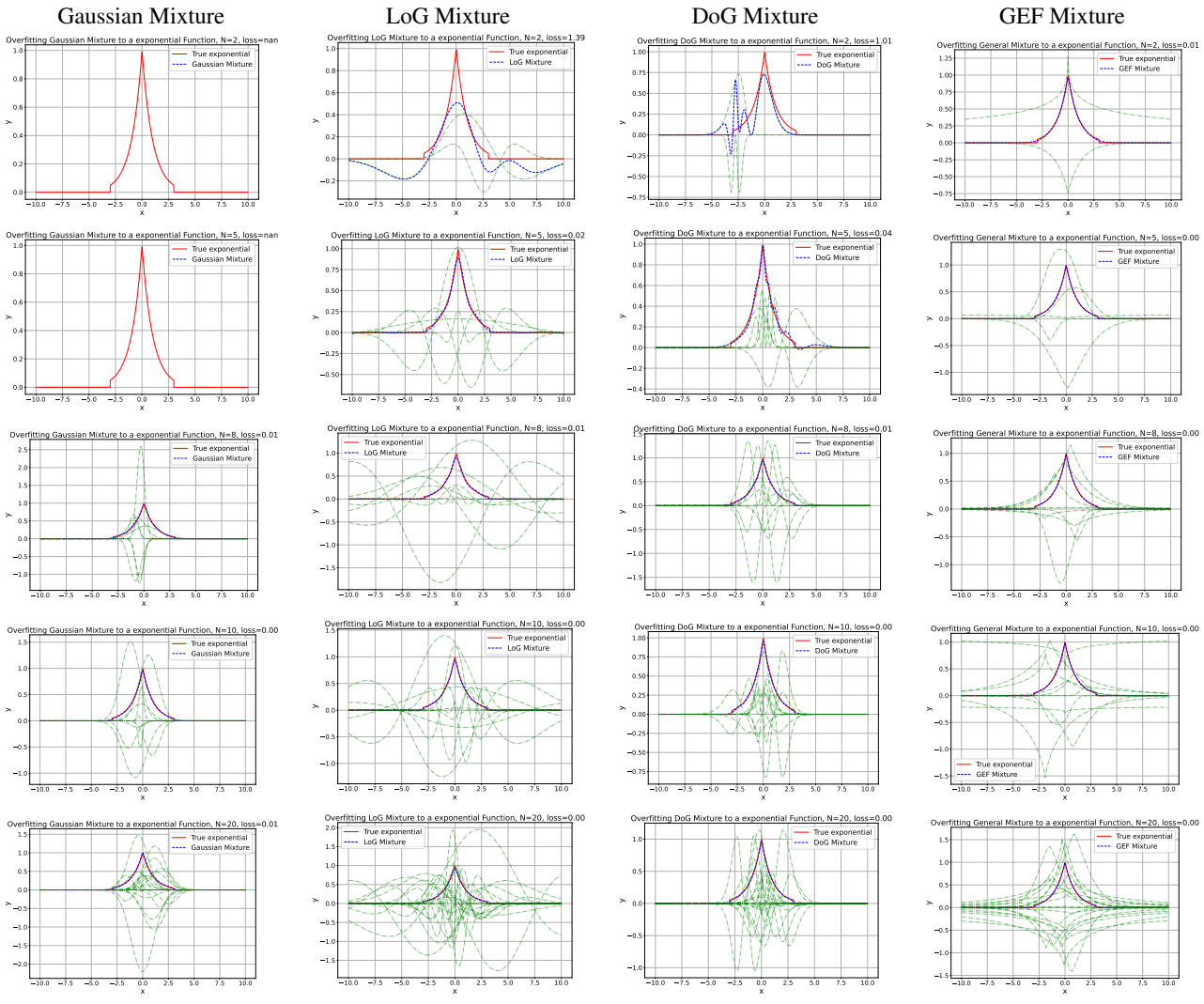


Figure 19. **Numerical Simulation Examples of Fitting Exponentials with Real Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for exponential signals with Real weights mixtures (can be negative). The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the exponential with few components while LoG and DoG are more stable for a larger number of components.

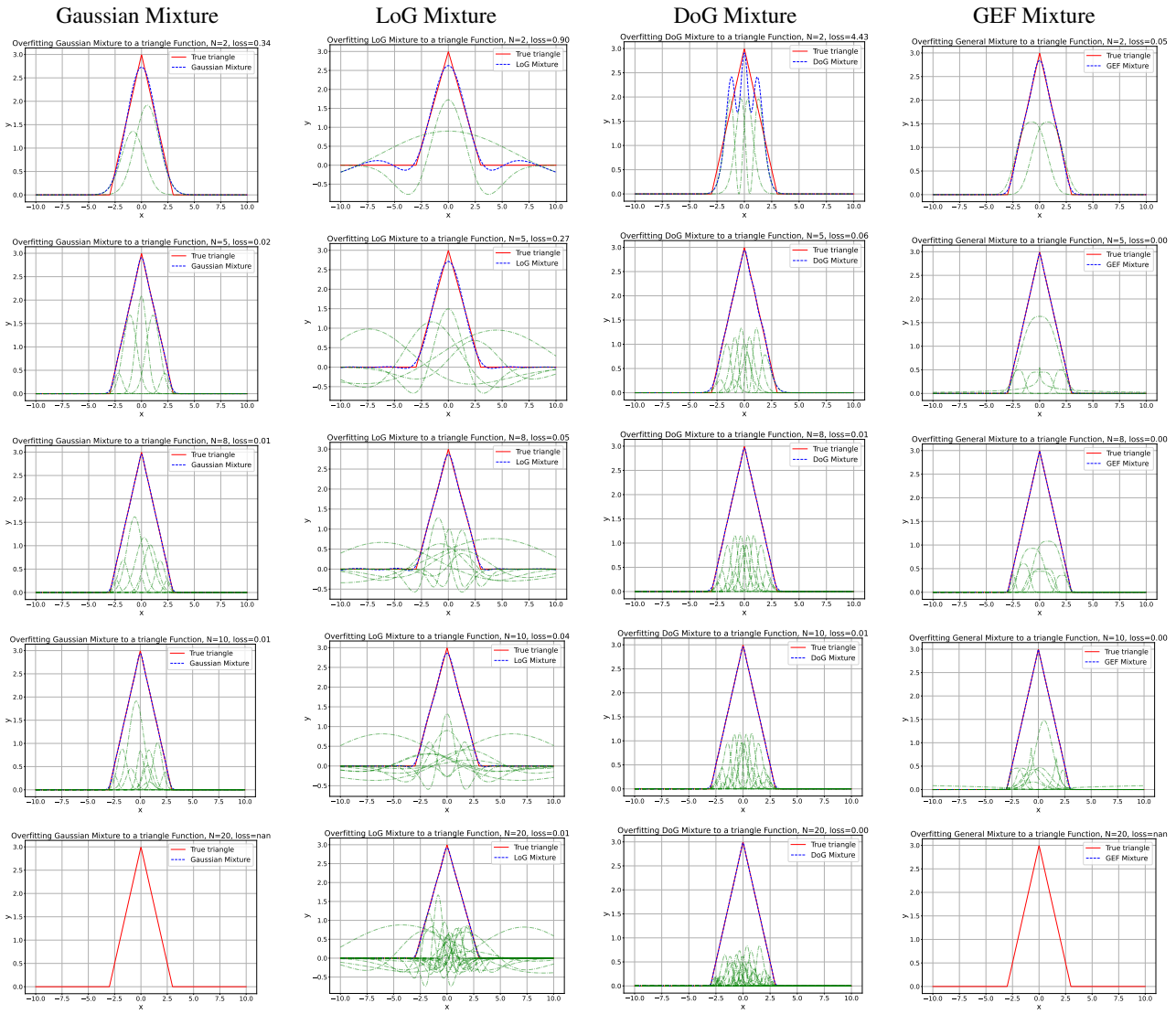


Figure 20. **Numerical Simulation Examples of Fitting Triangles with Positive Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for triangle signals with positive weight mixtures. The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the triangle with few components while LoG and DoG are more stable for a larger number of components.

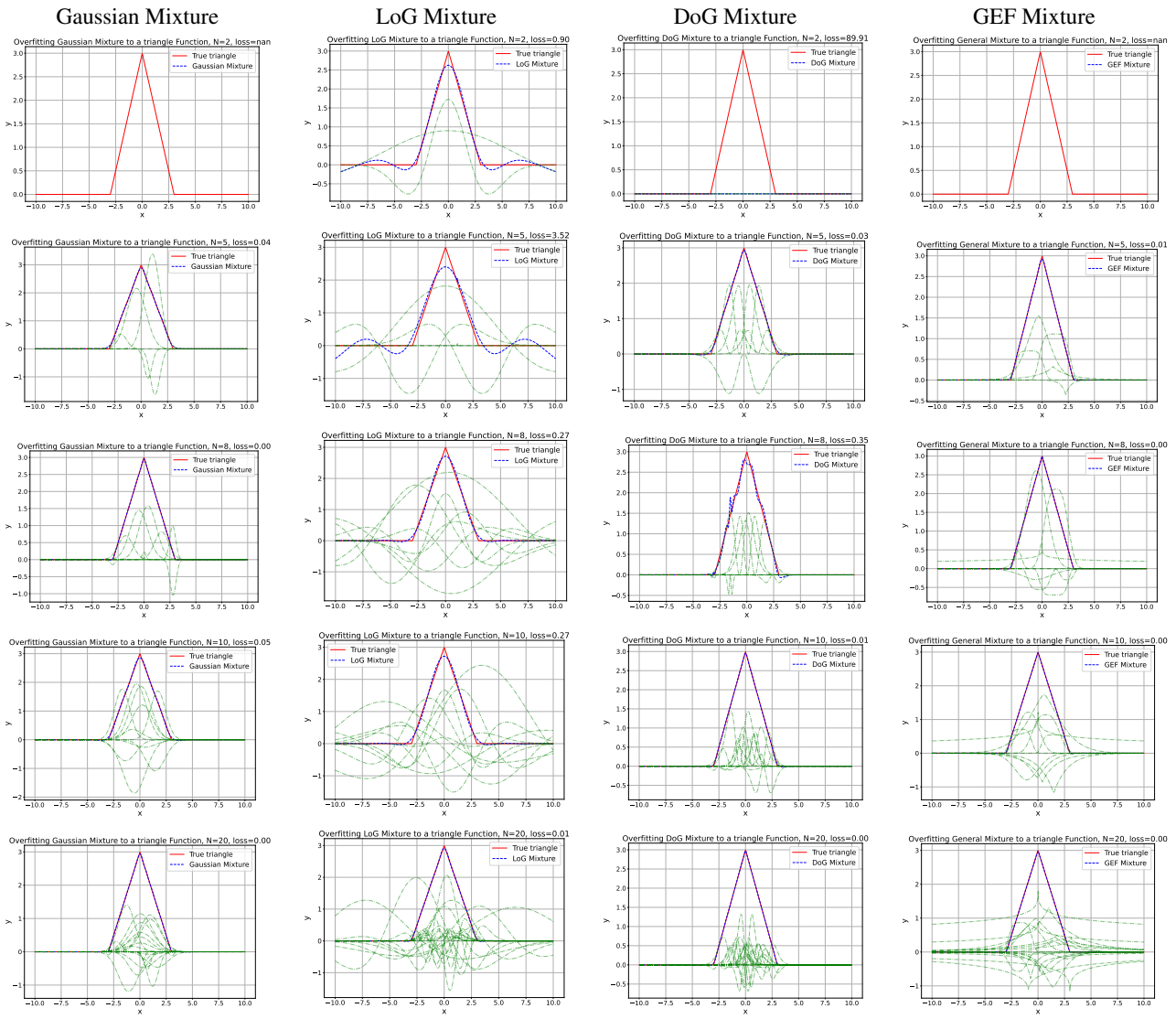


Figure 21. **Numerical Simulation Examples of Fitting Triangles with Real Weights Mixtures (N= 2, 5, 8, and 10).** We show some fitting examples for triangle signals with Real weights mixtures (can be negative). The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the triangle with few components while LoG and DoG are more stable for a larger number of components.

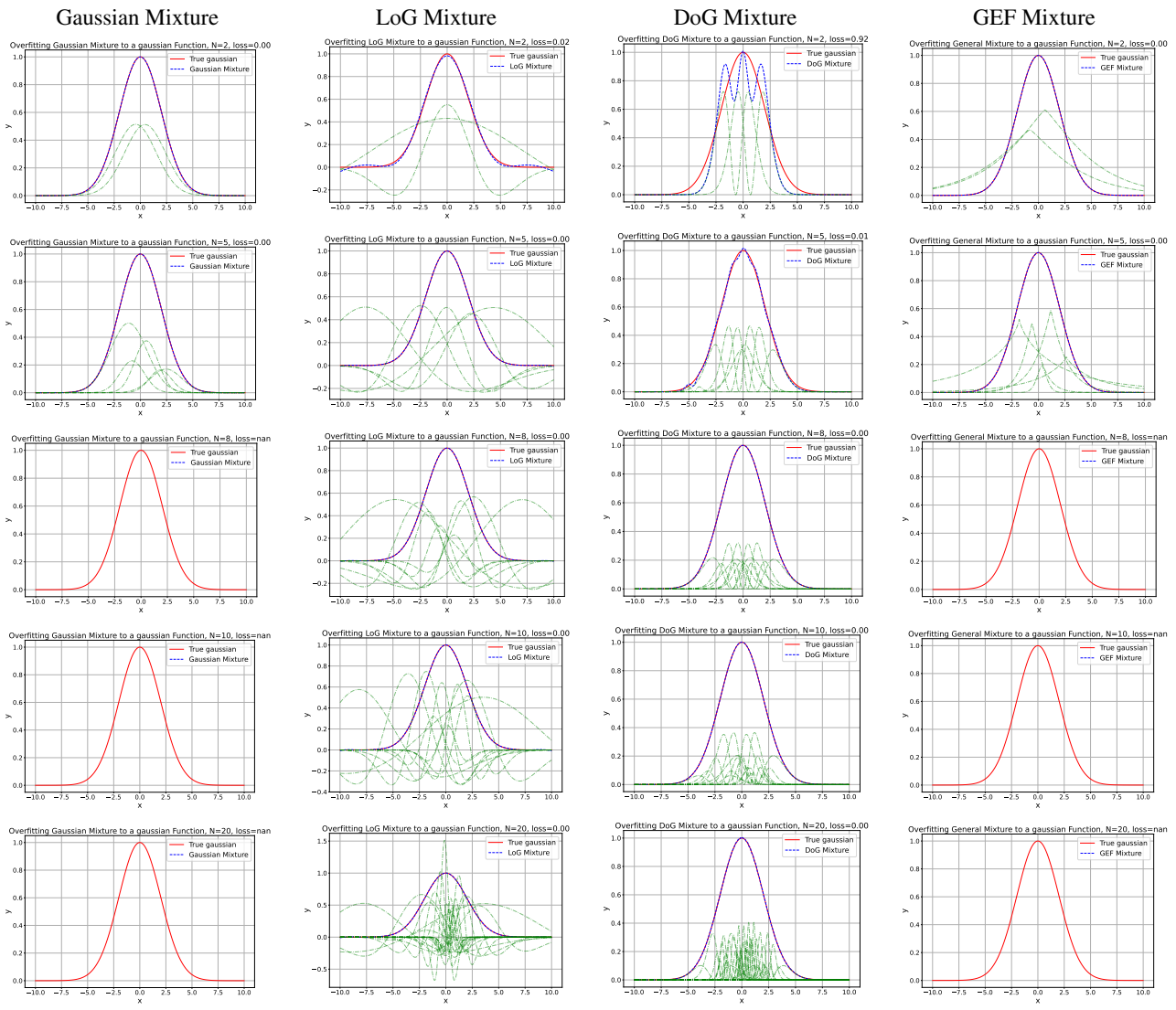


Figure 22. **Numerical Simulation Examples of Fitting Gaussians with Positive Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for Gaussian signals with positive weight mixtures. The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the Gaussian with few components while LoG and DoG are more stable for a larger number of components.

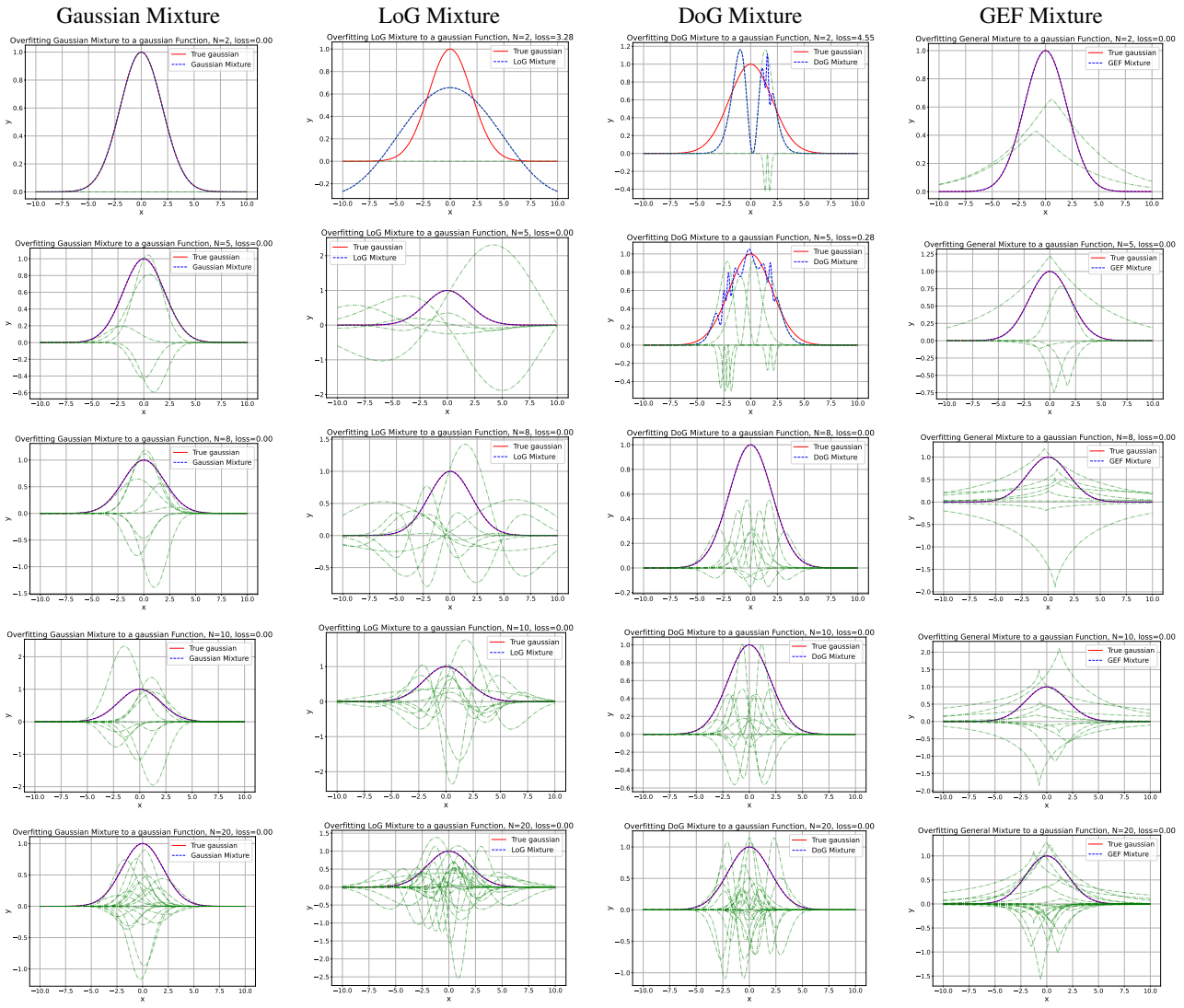


Figure 23. **Numerical Simulation Examples of Fitting Gaussians with Real Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for Gaussian signals with Real weights mixtures (can be negative). The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the Gaussian with few components while LoG and DoG are more stable for a larger number of components.

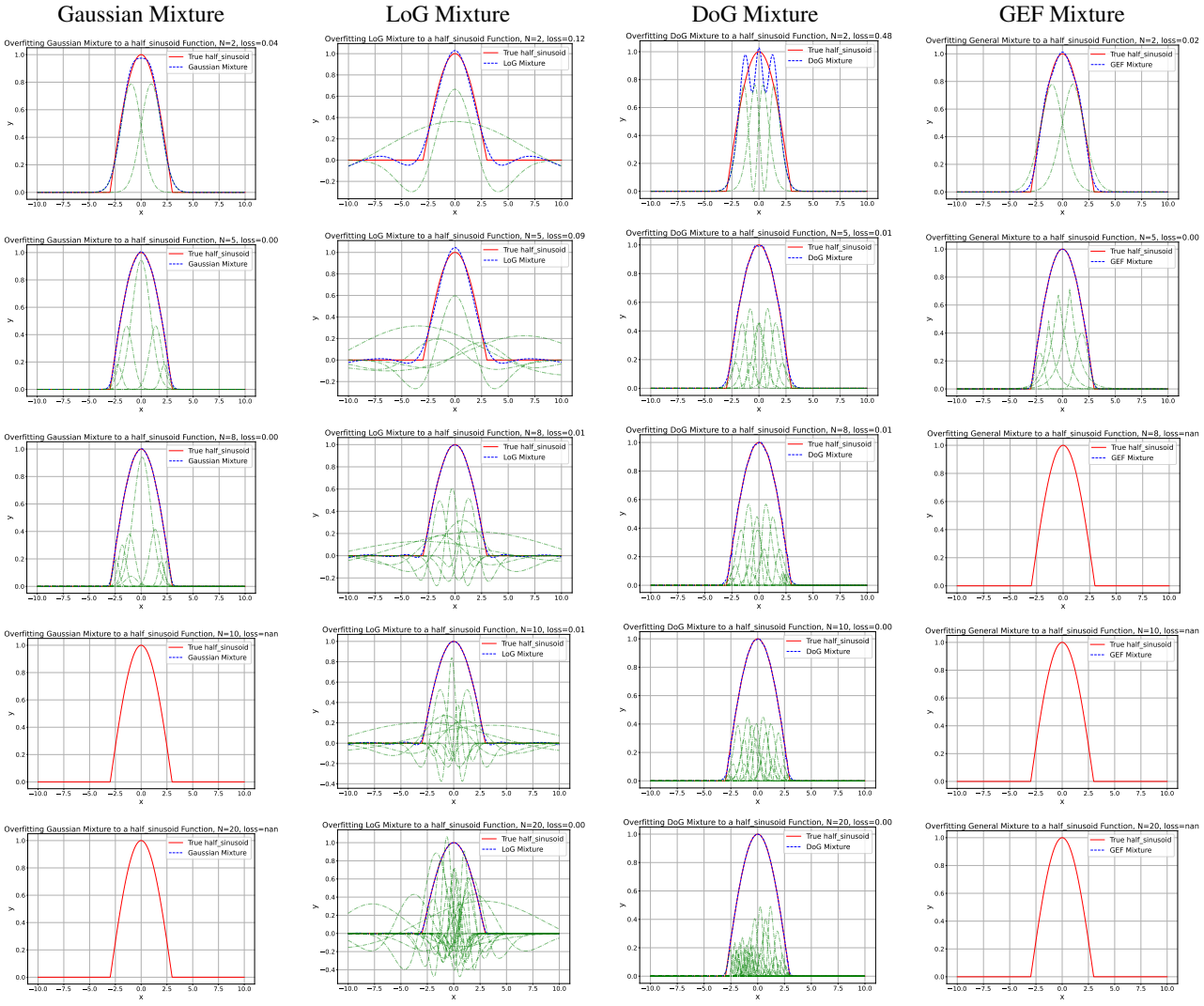


Figure 24. **Numerical Simulation Examples of Fitting Half sinusoids with Positive Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for half sinusoid signals with positive weights mixtures. The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the half sinusoid with few components while LoG and DoG are more stable for a larger number of components.

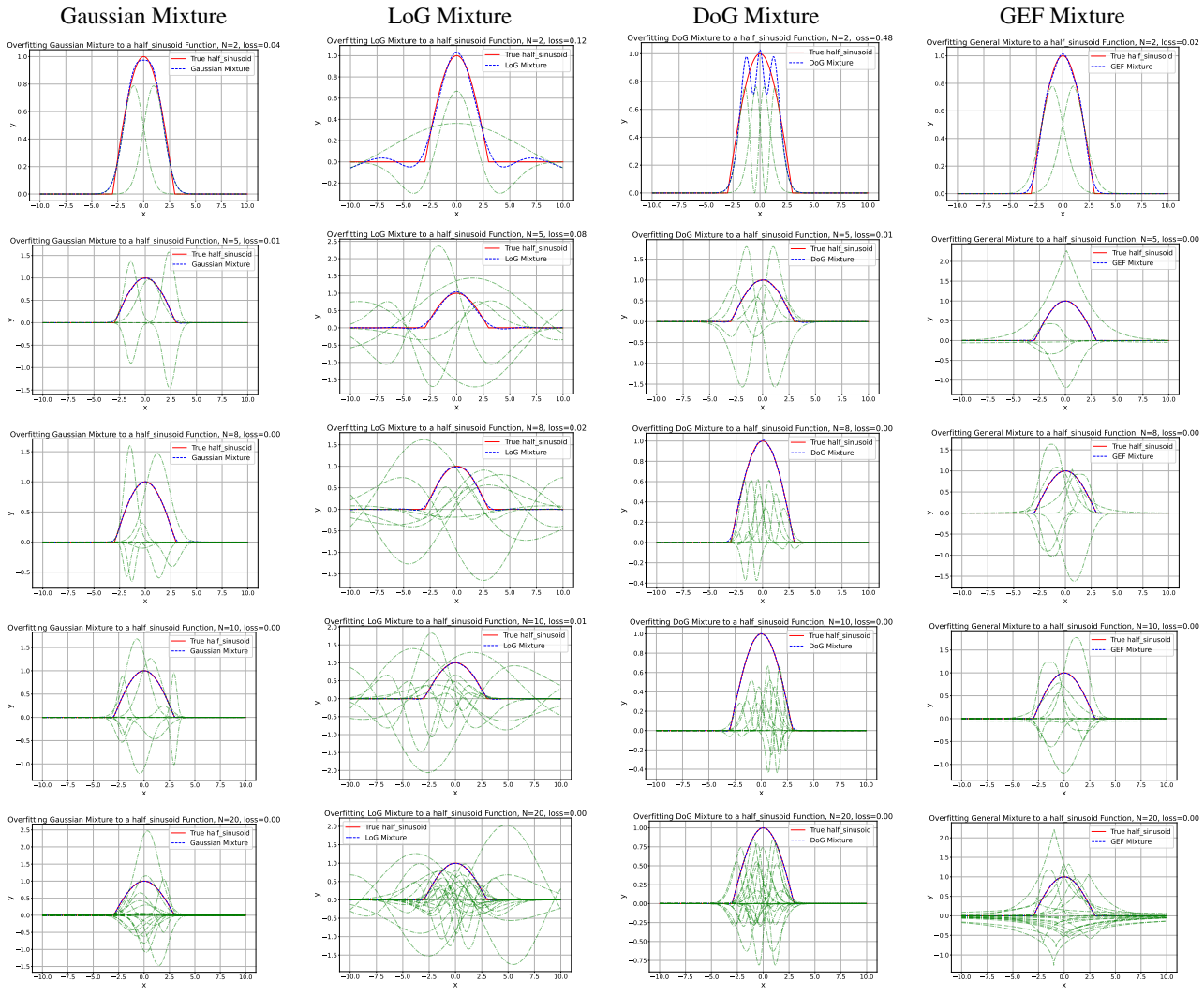


Figure 25. **Numerical Simulation Examples of Fitting Half sinusoids with Real Weights Mixtures (N= 2, 5, 8, and 10)**. We show some fitting examples for half sinusoid signals with Real weights mixtures (can be negative). The four mixtures used from left to right are Gaussians, LoG, DoG, and General mixtures. From top to bottom: N = 2, 8, and 10 components. The optimized individual components are shown in green. Some examples fail to optimize due to numerical instability in both Gaussians and GEF mixtures. Note that GEF is very efficient in fitting the half sinusoid with few components while LoG and DoG are more stable for a larger number of components.

B. Genealized Exponential Splatting Details

B.1. Upper Bound on the Boundary View-Dependant Error in the Approximate GES Rasterization

Given the Generalized Exponential Splatting (GES) function defined in Eq.(2) and our approximate rasterization given by Eq.(3),4, and 5, we seek to establish an upper bound on the error of our approximation in GES rendering. Since it is very difficult to estimate the error accumulated in each individual pixel from Eq.(3), we seek to estimate the error directly on each splatting component affecting the energy of all passing rays.

Let us consider a simple 2D case with symmetrical components as in Fig.6. The error between the scaled Gaussian component and the original GES component is related to the energy loss of rays and can be represented by simply estimating the *ratio* η between the area difference and the area of the scaled Gaussian. Here we will show we can estimate an upper bound on η relative to the area of each component.

For the worst-case scenario when $\beta \to \infty$, we consider two non-overlapping conditions for the approximation: one where the square is the outer shape and one where the circle covers the square. The side length of the square is $2r$ for the former case and $2r/\sqrt{2}$ for the latter case. The radius r of the circle is determined by the effective projected variance α from Eq.(4). For a square with side length $2r$ and a circle with radius r, we have: $A_{\text{square}} = 4r^2, A_{\text{circle}} = \pi r^2$. For a square with side length $2r/\sqrt{2}$, the area is:$A_{\text{square, covered}} = 2r^2$.

The area difference ΔA is:

$$\Delta A_{\text{square larger}} = A_{\text{square}} - A_{\text{circle}} = 4r^2 - \pi r^2, \tag{29}$$

$$\Delta A_{\text{circle larger}} = A_{\text{circle}} - A_{\text{square, covered}} = \pi r^2 - 2r^2. \tag{30}$$

The ratio of the difference in areas to the area of the inner shape, denoted as η, is bounded by:

$$\eta_{\text{square larger}} = \frac{\Delta A_{\text{square larger}}}{A_{\text{circle}}} = \frac{4r^2 - \pi r^2}{\pi r^2} \approx 0.2732, \tag{31}$$

$$\eta_{\text{circle larger}} = \frac{\Delta A_{\text{circle larger}}}{A_{\text{circle}}} = \frac{\pi r^2 - 2r^2}{\pi r^2} \approx 0.3634. \tag{32}$$

Due to the PDF normalization constraint in GND [14], the approximation followed in Eq.(4), and 5 will always ensure $\eta_{\text{square larger}} \leq \eta \leq \eta_{\text{circle larger}}$. Thus, our target ratio η when using our approximate scaling of variance based on β should be within the range $0.2732 \leq \eta \leq 0.3634$. This implies in the worst case, our GES approximation will result in 36.34% energy error in the lost energy of all rays passing through *all* the splatting components. In practice, the error will be much smaller due to the large number of components and the small scale of all the splatting components.

B.2. Implementation Details

Note that the DoG in Eq.(7) will be very large when σ_2 is large, so we downsample the ground truth image by a factor ‘$\text{scale}_{\text{im,freq}}$‘ and upsample the mask M_ω similarly before calculating the loss in Eq.(8). In the implementation of our Generalized Exponential Splatting (GES) approach, we fine-tuned several hyperparameters to optimize the performance. The following list details the specific values and purposes of each parameter in our implementation:

- Iterations: The algorithm ran for a total of 40,000 iterations.
- Learning Rates:
  - Initial position learning rate ($\text{lr}_{\text{pos, init}}$) was set to 0.00016.
  - Final position learning rate ($\text{lr}_{\text{pos, final}}$) was reduced to 0.0000016.
  - Learning rate delay multiplier ($\text{lr}_{\text{delay mult}}$) was set to 0.01.
  - Maximum steps for position learning rate ($\text{lr}_{\text{pos, max steps}}$) were set to 30,000.
- Other Learning Rates:
  - Feature learning rate ($\text{lr}_{\text{feature}}$) was 0.0025.
  - Opacity learning rate ($\text{lr}_{\text{opacity}}$) was 0.05.
  - Shape and rotation learning rates (lr_{shape} and $\text{lr}_{\text{rotation}}$) were both set to 0.001.
  - Scaling learning rate ($\text{lr}_{\text{scaling}}$) was 0.005.
- Density and Pruning Parameters:
  - Percentage of dense points ($\text{percent}_{\text{dense}}$) was 0.01.
  - Opacity and shape pruning thresholds were set to 0.005.
- Loss Weights and Intervals:
  - SSIM loss weight (λ_{ssim}) was 0.2.
  - Densification, opacity reset, shape reset, and shape pruning intervals were set to 100, 3000, 1000, and 100 iterations, respectively.
- Densification Details:
  - Densification started from iteration 500 and continued until iteration 15,000.
  - Gradient threshold for densification was set to 0.0003.
- Image Laplacian Parameters:

- Image Laplacian scale factor ($\text{scale}_{\text{im,freq}}$) was 0.2.
- Weight for image Laplacian loss (λ_ω) was 0.5.

- Miscellaneous:
  - Strength of shape ρ was set to 0.1.

These parameters were carefully chosen to balance the trade-off between computational efficiency and the fidelity of the synthesized views. The above hyperparameter configuration played a crucial role in the effective implementation of our GES approach. For implementation purposes, the modification functions have been shifted by -2 and the β initialization is set to 0 instead of 2 (which should not have any effect on the optimization).

C. Additional Results and Analysis

C.1. Additional Results

We show in Fig.32 additional GES results (test views) and comparisons to the ground truth and baselines. In Fig.33, show PSNR, LPIPS, SSIM, and file size results for every single scene in MIPNeRF 360 dataset [5, 6] of our GES and re-running the Gaussian Splatting [27] baseline with the *exact same* hyperparameters of our GES and on different number of iterations.

C.2. Applying GES in Fast 3D Generation

GES is adapted to modern 3D generation pipelines using score distillation sampling from a 2D text-to-image model [50], replacing Gaussian Splatting for improved efficiency. We employ the same setup as DreamGaussian [68], altering only the 3D representation to GES . This change demonstrates GES 's capability for real-time representation applications and memory efficiency.

For evaluation, we use datasets NeRF4 and RealFusion15 with metrics PSNR, LPIPS [87], and CLIP-similarity [53] following the benchmarks in Realfusion [41] and Magic123 [52]. Our GES exhibits swift optimization with an average runtime of 2 minutes, maintaining quality, as shown in Table 3 and Fig.26.

C.3. Shape Parameters

In Table 5, we explore the effect of all hyperparameters associated with the new shape parameter on novel view synthesis performance. We find that the optimization process is relatively robust to these changes, as it retains relatively strong performance and yields results with similar sizes.

Density Gradient Threshold. In Fig.27, we visualize the impact of modifying the density gradient threshold for splitting, using both GES and the standard Gaussian Splatting (after modifying the setup for a fair comparison to GES). We see that the threshold has a significant impact on the trade-off between performance and size, with a higher threshold decreasing size at the expense of performance. Notably, we see that GES outperforms GS across the range of density gradient thresholds, yielding similar performance while using less memory.

Dataset	Metrics\Methods	Point-E	DreamGaussian	GES (Ours)
NeRF4	CLIP-Similarity↑	0.48	0.56	0.58
	PSNR↑	0.70	13.48	13.33
RealFusion15	CLIP-Similarity↑	0.53	0.70	0.70
	PSNR↑	0.98	12.83	12.91
-	Average Runtime ↓	78 secs	2 mins	2 mins

Table 3. **GES Application: Fast Image-to-3D Generation pipeline** We show quantitative results in terms of CLIP-Similarity↑ / PSNR↑ , and Runtime↓, compared to fast methods: Point-E [46] and DreamGaussian [68] . GES offers a good option for a fast and effective image-to-3D solution.

C.4. Analysing the Frequency-Modulated Image Loss

We study the effect of the frequency-modulated loss $\mathcal{L}_\omega$ on the performance by varying λ_ω and show the results in Table 4 and Table 2. Note that increasing λ_ω in GES indeed reduces the size of the file, but can affect the performance. We chose $\lambda_\omega = 0.5$ as a middle ground between improved performance and reduced file size.

C.5. Visualizing the Distribution of Parameters

We visualize the distribution of shape parameters β in Fig.28 and the sizes of the splatting components in Fig.29. They clearly show a smooth distribution of the components in the scene, which indicates the importance of initialization. This hints a possible future direction in this line of research.

C.6. Typical Convergence Plots

We show in Fig.30 examples of the convergence plots of both GES and Gaussians if the training continues up to 50K iterations to inspect the diminishing returns of more training. Despite requiring more iterations to converge, GES trains faster than Gaussians due to its smaller number of splatting components.

Figure 26. **Visulization for 3D generation**. We show selected generated examples by GES from Realfusion15 (*left*) and NeRF4 datasets (*middle*). Additionally, we pick two text prompts: *"a car made out of sushi" and "Michelangelo style statue of an astronaut"*, and then use StableDiffusion-XL [49] to generate the reference images before using GES on them(*right*).

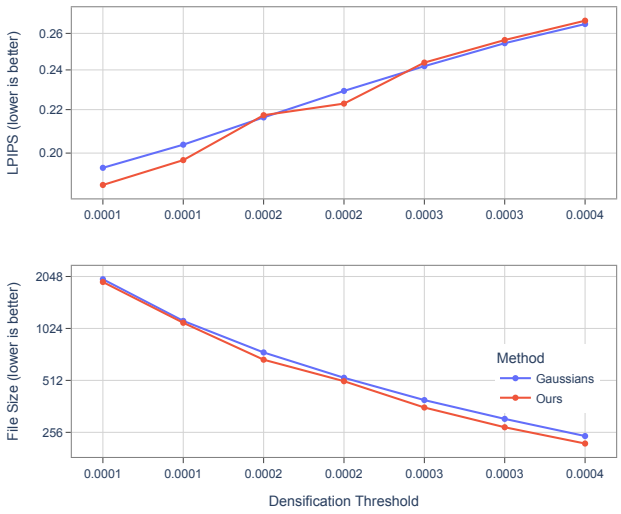


Figure 27. **Ablation Study of Densification Threshold on Novel View Synthesis.** Impact of the densification threshold on reconstruction quality (LPIPS) and file size (MB) for our method and Gaussian Splatting [27], averaged across all scenes in the Mip-NeRF dataset. We see that the densification threshold has a significant impact on both file size and quality. Across the board, our method produces smaller scenes than Gaussian Splatting with similar or even slightly improved performance.

λ_{freq}	Method	PSNR	LPIPS	SSIM	Size
	Deep Blending				
0.05	GES	29.58	0.252	0.900	431
	GES (fixed $\beta = 2$)	29.53	0.251	0.901	433
0.10	GES	29.54	0.252	0.901	428
	GES (fixed $\beta = 2$)	29.61	0.252	0.901	435
0.50	GES	29.66	0.251	0.901	397
	GES (fixed $\beta = 2$)	29.61	0.252	0.901	437
0.90	GES	27.21	0.259	0.899	366
	GES (fixed $\beta = 2$)	29.62	0.252	0.901	434
	MipNeRF				
0.05	GES	27.08	0.250	0.796	405
	GES (fixed $\beta = 2$)	27.05	0.250	0.795	411
0.10	GES	27.05	0.250	0.795	403
	GES (fixed $\beta = 2$)	27.05	0.250	0.796	412
0.50	GES	26.97	0.252	0.794	376
	GES (fixed $\beta = 2$)	27.09	0.250	0.796	415
0.90	GES	25.82	0.255	0.792	364
	GES (fixed $\beta = 2$)	27.08	0.250	0.795	413
	Tanks and Temples				
0.05	GES	23.49	0.196	0.837	251
	GES (fixed $\beta = 2$)	23.55	0.196	0.836	255
0.10	GES	23.54	0.196	0.837	247
	GES (fixed $\beta = 2$)	23.53	0.196	0.837	255
0.50	GES	23.35	0.197	0.836	221
	GES (fixed $\beta = 2$)	23.65	0.196	0.837	256
0.90	GES	22.65	0.200	0.834	210
	GES (fixed $\beta = 2$)	23.50	0.197	0.836	256

Table 4. **Ablation of λ_{freq}.** We show a comparison of performance (PSNR, LPIPS, SSIM) for various values of λ_{freq}. Note that increasing λ_{freq} in GES indeed reduces the size of the file, but can affect the performance. We chose $\lambda_{\text{freq}} = 0.5$ as a middle ground between improved performance and reduced file size.

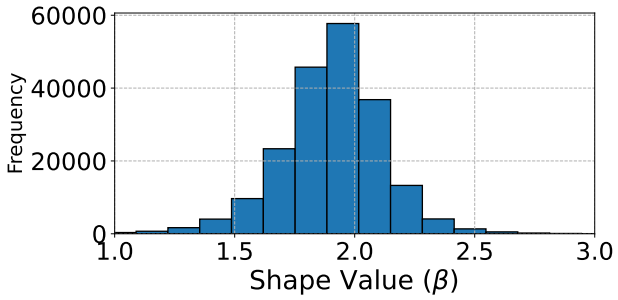


Figure 28. **Distribution of Shape Values**. We show a distribution of β values of a converged GES initialized with $\beta = 2$. It shows a slight bias to β smaller than 2.

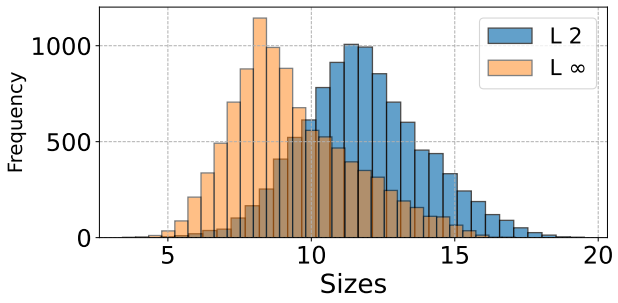


Figure 29. **Distribution of Sizes**. We show a distribution of sizes (L_2 and L_∞) of the GES components of a converged scene.

Shape Learning Rate	PSNR	SSIM	LPIPS	File Size (MB)
0.0005	26.83	0.845	**0.141**	659
0.0010	26.85	0.845	**0.141**	658
0.0015	**26.89**	**0.846**	**0.141**	**651**
0.0020	26.82	0.844	0.142	658
Shape Reset Interval	**PSNR**	**SSIM**	**LPIPS**	**File Size (MB)**
200	26.87	0.845	**0.141**	656
500	26.86	0.845	**0.141**	658
1000	**26.89**	**0.846**	**0.141**	**651**
2000	26.84	0.845	**0.141**	657
5000	26.84	0.845	**0.141**	661
Shape Strength	**PSNR**	**SSIM**	**LPIPS**	**File Size (MB)**
0.010	26.87	0.845	**0.141**	661
0.050	26.84	0.845	**0.141**	653
0.100	**26.89**	**0.846**	**0.141**	**651**
0.150	26.83	0.844	0.142	656

Table 5. **Ablation Study on Novel View Synthesis.** Impact of the shape parameter's learning rate, reset interval, and strength on reconstruction quality and file size for the garden scene from the Mip-NeRF dataset.

Ablation Setup	$PSNR^{\uparrow}$	$SSIM^{\uparrow}$	$LPIPS^{\downarrow}$	**Size (MB)**$^{\downarrow}$
Gaussians	23.14	0.841	0.183	411
GES w/o $\mathcal{L}_\omega$	23.54	0.836	0.197	254
GES w/ random β init.	23.37	0.836	0.198	223
GES w/ $\beta = 2$ init.	23.35	0.836	0.198	**222**

Table 6. **Ablation Study on Novel View Synthesis.** We study the impact of several components in GES on the reconstruction quality and file size in the Tanks & Temples dataset.

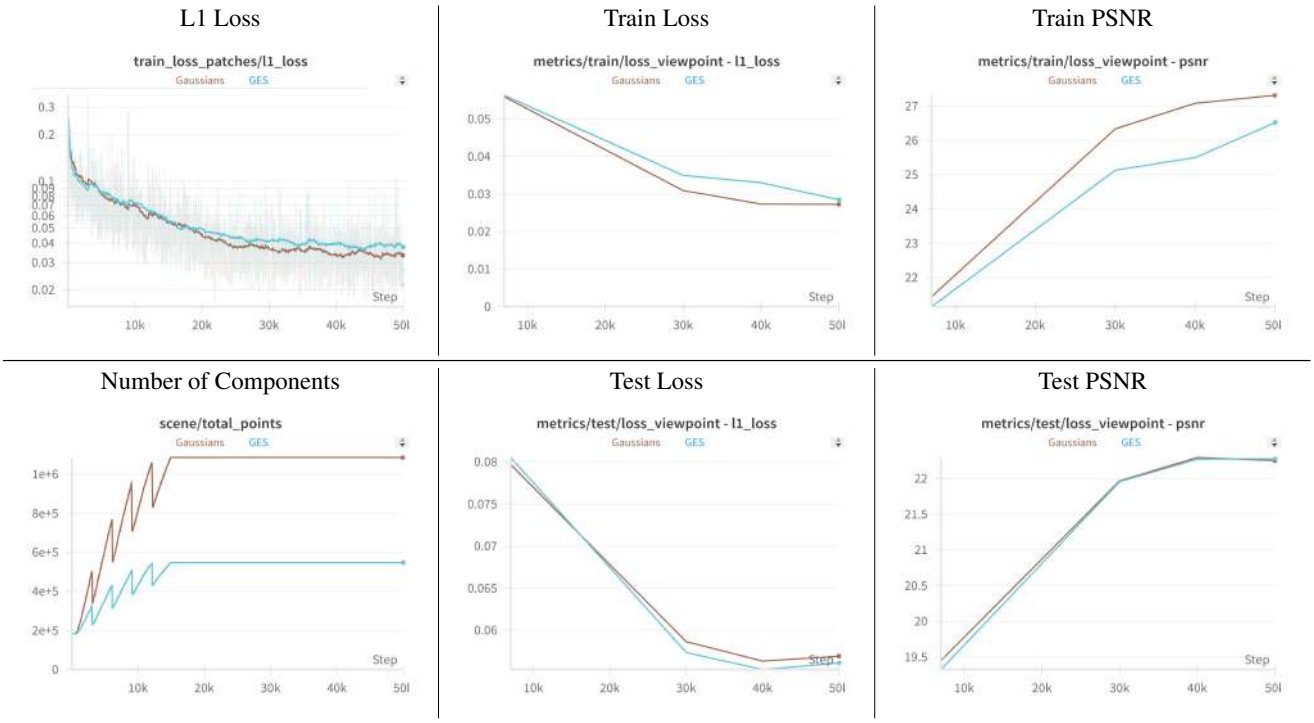


Figure 30. **Convergence Plots of Gaussians *vs*. GES** . We show an example of the convergence plots of both GES and Gaussians if the training continues up to 50K iterations to inspect the diminishing returns of more training. Despite requiring more iterations to converge, GES trains faster than Gaussians due to its smaller number of splatting components.

Figure 31. **Frequency-Modulated Image Masks.** For the input example image on the top left, We show examples of the frequency loss masks M_ω used in Sec.4.3 for different numbers of target normalized frequencies ω ($\omega = 0\%$ for low frequencies to $\omega = 100\%$ for high frequencies). This masked loss helps our GES learn specific bands of frequencies. Note that due to Laplacian filter sensitivity for high-frequencies, the mask for $0 < \omega \leq 50\%$ is defined as $1 - M_\omega$ for $50 < \omega \leq 100\%$. This ensures that all parts of the image will be covered by one of the masks M_ω, while focusing on the details more as the optimization progresses.

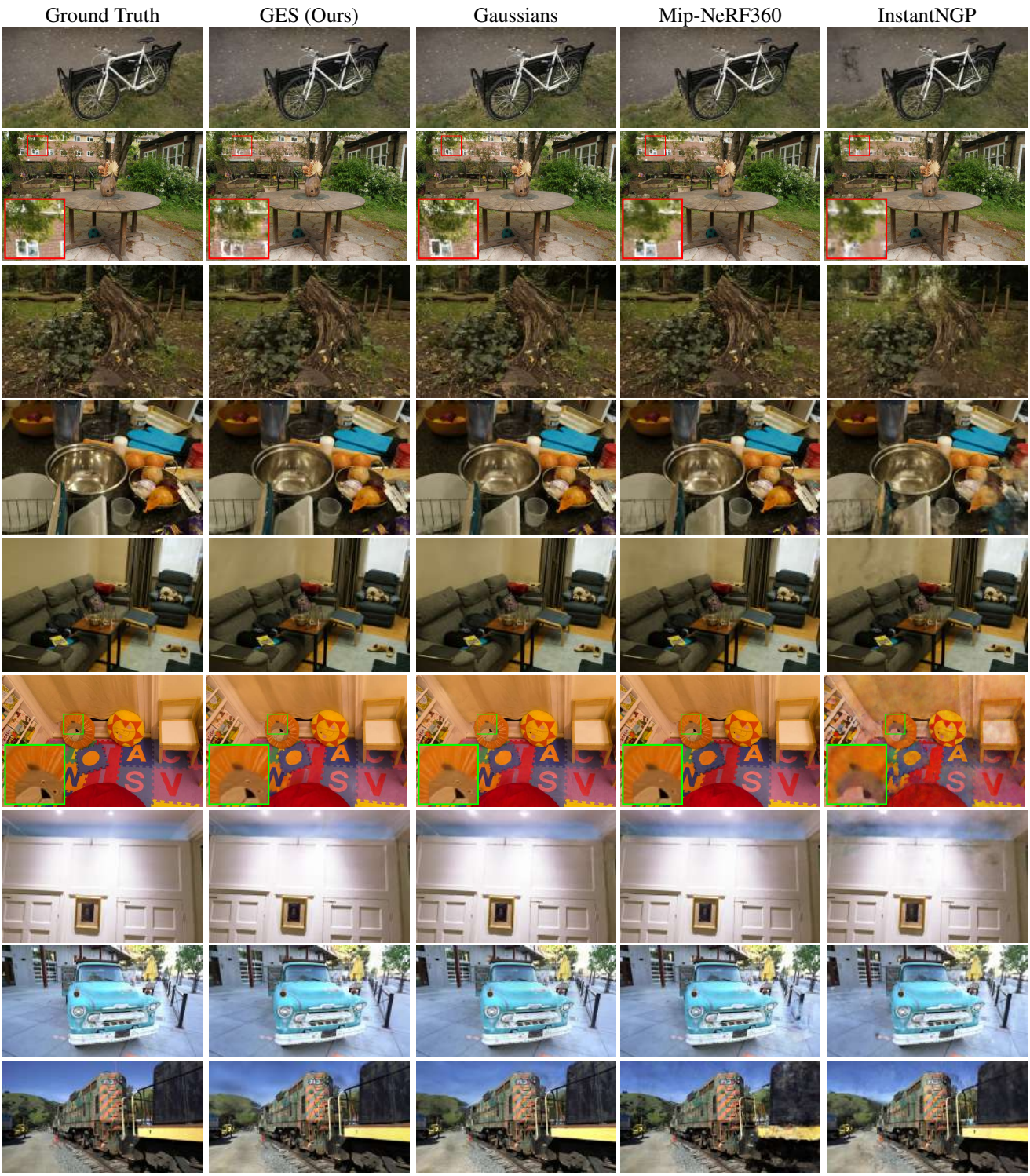


Figure 32. **Comparative Visualization Across Methods.** Displayed are side-by-side comparisons between our proposed method and established techniques alongside their respective ground truth imagery. The depicted scenes are ordered as follows: BICYCLE, GARDEN, STUMP, COUNTER, and ROOM from the Mip-NeRF360 dataset; PLAYROOM and DRJOHNSON from the Deep Blending dataset, and TRUCK and TRAIN from Tanks&Temples. Subtle variances in rendering quality are accentuated through zoomed-in details. It might be difficult to see differences between GES and Gaussians because they have almost the same PSNR (despite GES requiring 50% less memory).

Figure 33. **Detailed Per Scene Results On MipNeRF 360 for Different Iteration Numbers.** We show PSNR, LPIPS, SSIM, and file size results for every single scene in MIPNeRF 360 dataset [5] of our GES and re-running the Gaussian Splatting [27] baseline with the *exact same* hyperparameters of our GES and on different number of iterations.

Figure 34. **Frequency-Modulated Loss Effect.** We show the effect of the frequency-modulated image loss $\mathcal{L}_\omega$ on the performance on novel views synthesis. Note how adding this $\mathcal{L}_\omega$ improves the optimization in areas where large contrast exists or where a smooth background is rendered.