GLUT: 3D Gaussian Lookup Table for Continuous Color Transformation

Danna Xue[1], **David Serrano-Lozano**[1,2], **Shaolin Su**[1*], **Javier Vazquez-Corral**[1,2]
[1]Computer Vision Center, [2]Universitat Autònoma de Barcelona
{dxue, dserrano, shaolin, javier.vazquez}@cvc.uab.cat

Abstract

3D Lookup Tables (3D LUTs) are widely used for color mapping, but their grid-based representation discretizes the RGB space, leading to a capacity-memory trade-off that becomes prohibitive when storing large numbers of LUTs. Recent approaches use implicit neural representations to improve scalability, but their black-box nature limits interpretability and localized editing. In this paper, we propose Gaussian LUT (GLUT), a continuous and explicit representation of color transformations using learnable 3D Gaussian primitives. By eliminating fixed-resolution grids, GLUT provides flexible representational capacity with a compact memory footprint. Its spatially localized primitives also facilitate interpretation and targeted editing. Building on GLUT, we introduce Conditional Gaussian LUT (CGLUT), a compact generator that predicts GLUT parameters for multiple LUTs within a shared framework, enabling smooth blending between color styles. GLUT further supports efficient local editing by allowing targeted adjustments to specific color regions without retraining the model. Experiments show that GLUT outperforms existing neural LUT representations in both accuracy and efficiency, while supporting controllable localized color editing. CGLUT further improves compact modeling and smooth blending of multiple color styles. Code is available at https://github.com/CVC-Color/glut.

1 Introduction

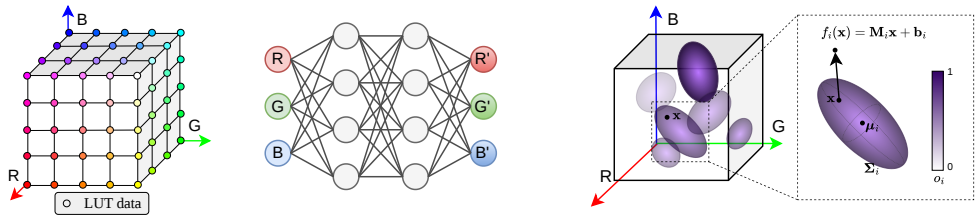
3D Lookup Tables (3D LUTs) provide an efficient way to implement deterministic color transformations by storing output colors at sampled points in the color space. They are widely used in real-time image signal processing (ISP) on consumer devices [16, 19, 34], film color grading [30, 33, 7], and computational photography [48, 41, 47, 43, 21, 44]. A 3D LUT partitions the RGB space into a regular grid, where each vertex stores a transformed color value, as illustrated in Figure 1a. At inference time, output colors are obtained via table lookup followed by interpolation among neighboring grid points. This non-parametric design captures complex nonlinear mappings with high fidelity, and its lookup-based inference offers low computational cost and predictable memory requirements, making 3D LUTs suitable for resource-constrained deployment.

Despite these advantages, traditional 3D LUTs face a fundamental trade-off between representational capacity and memory consumption. Their representational power is tightly coupled to grid resolution: dense sampling enables fine-grained color control but substantially increases storage requirements. In contrast, sparse sampling saves memory but can introduce interpolation artifacts and limit the representation of sharp transitions in color mappings, which may be important for high-contrast cinematic looks and effects such as bleach bypass.

Recent works represent color transformations using implicit neural representations (INRs) [8, 10, 46], where a compact network parameterizes a continuous neural LUT (see Figure 1b). While

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Traditional LUT (b) INR-based LUT (c) Gaussian LUT (Ours)

Figure 1: Comparison between different color LUT representations. (a) Explicit Traditional LUT *vs.* (b) Implicit INR-based LUT *vs.* (c) Explicit Gaussian LUT (Ours).

these methods avoid explicit grids and support efficient inference on edge devices, they sacrifice interpretability and local controllability. Since color transformations are encoded across shared network parameters, adjusting a specific color range requires reoptimizing the network weights, limiting practical use in interactive or color-specific editing.

To overcome (1) the capacity-memory trade-off of grid-based LUTs, and (2) the limited interpretability and local controllability in INR-based LUTs, in this paper, we propose a continuous and explicit LUT representation through the lens of explicit volumetric primitives. Inspired by 3D Gaussian Splatting (3DGS) [18], which represents a scene as a set of optimizable, spatially localized Gaussians, we model the color transformation in RGB space using a collection of learnable 3D Gaussian primitives (see Figure 1c). Specifically, we introduce Gaussian LUT (GLUT), which formulates color transformation as a soft, adaptive partition of the color space using explicit Gaussian anchors. Building on GLUT, we further introduce Conditional Gaussian LUT (CGLUT), a compact generator that predicts GLUT parameters for multiple LUT instances, enabling unified modeling and smooth blending of diverse color styles.

GLUT offers three main advantages. First, its continuous representation decouples representational capacity from a fixed grid resolution, allowing a flexible trade-off between model size and mapping fidelity. Second, its explicit Gaussian primitives provide spatially localized representations of color transformations, facilitating interpretation and targeted editing. Third, localized editing can be performed by optimizing only a subset of relevant primitives, avoiding global model retraining. CGLUT further enables compact modeling and controllable blending of multiple color styles within a shared framework.

Our primary contributions are as follows:

1. We propose Gaussian LUT (GLUT), an explicit and continuous LUT representation of color transformations based on a compact set of learnable 3D Gaussian primitives.
2. We introduce Conditional Gaussian LUT (CGLUT), a compact generator that predicts the parameters of multiple GLUT instances, enabling unified modeling and smooth, controllable blending of diverse color styles.
3. We develop a localized editing mechanism based on GLUT that adjusts selected Gaussian primitives to modify specific color ranges without retraining the entire model, enabling fine-grained color control with limited interference in unrelated regions.
4. Extensive experiments demonstrate that GLUT achieves favorable accuracy-efficiency trade-offs compared with existing neural LUT representations. Its explicit, spatially localized formulation also supports interpretable color mapping and efficient local editing.

2 Related Works

Color Look-Up Table Representations. 3D LUTs provide an efficient mechanism for mapping input colors to desired outputs via a function $(R', G', B') = f(R, G, B)$. A standard 3D LUT discretizes the RGB space into a cubic grid (typically 17^3 or 33^3 vertices), where the outputs are computed through trilinear interpolation or tetrahedral interpolation [17]. Higher-order schemes such as tricubic splines [12] yield smoother mappings at a higher computational cost.

While computationally efficient, grid LUTs face a trade-off between representational capacity and memory consumption. Existing compression methods reduce storage through hierarchical and differential encoding [2, 32, 36], keypoint-based reconstruction via multi-scale anisotropic diffusion [37, 38], and low-rank tensor factorization [26]. While substantially reducing storage, these approaches retain a discrete grid representation or reconstruct one from a more compact encoding, so their spatial resolution and editing behavior remain tied to the underlying grid. Compact parametric alternatives have also been studied for color correction and device characterization. Root-polynomial regression [11] models color mappings using global polynomial functions, while radial basis function (RBF) interpolation [29] uses localized kernels. These methods are primarily designed for smooth device characterization and color correction, whereas stylized LUTs can contain substantially more complex and localized nonlinear transformations.

More recently, implicit neural representations (INRs) have been introduced for continuous LUT modeling. NILUT [8] uses a lightweight MLP for efficient color transformation on mobile deployment and supports LUT blending through embedding interpolation. INRetouch [10] incorporates convolutional features for content-aware local color transformation, while ENNELUT [46] uses an invertible network to represent hundreds of LUTs within a single model. However, neural implicit functions exhibit a spectral bias toward smooth, low-frequency mappings [31, 35], which can make sharp local transitions difficult to represent accurately. Moreover, their fully implicit parameterization provides no explicit spatial decomposition of the RGB space, making localized modification less direct. In contrast, GLUT represents color transformations using spatially localized Gaussian primitives, providing an explicit structure for color mapping and local editing.

Gaussian-based Representations. Representing a function as a Gaussian-weighted combination of local models has a long history. Radial basis function (RBF) networks [27] approximate functions with localized basis functions, and RBF interpolation has been applied to color characterization [29]. Mixtures of local experts [15] combine specialized models through input-dependent gating, while GLLiM [9] couples Gaussian gating with locally affine regressors and fits its parameters through expectation-maximization (EM). These methods demonstrate the effectiveness of localized function approximation, but differ in their parameterizations and optimization objectives. They do not specifically target a compact, continuous, and directly editable representation of complex color transformations.

More recently, Gaussian Splatting [18, 14] has emerged as a powerful alternative to implicit neural fields like NeRF [25], offering superior rendering fidelity and speed. By representing complex 3D scenes with optimizable Gaussian primitives, these methods enable efficient rendering and direct manipulation of explicit scene elements. This paradigm has also been extended beyond 3D reconstruction and editing [18, 14, 40, 3] to low-level vision tasks such as image restoration [6], image compression [50, 51], and super-resolution [5, 45].

Inspired by these explicit, localized representations, we transfer the Gaussian-primitive paradigm from the spatial domain to the RGB color space. GLUT represents a color transformation as a weighted combination of Gaussian-localized affine transformations, providing a continuous representation with explicit local structure that supports targeted color editing.

Color Manipulation with LUTs. Conventional LUTs apply a global color mapping independently of image content. To support context-dependent transformations, prior work has explored image-adaptive LUTs by combining multiple basis LUTs [47]. Subsequent methods improve adaptation through adaptive sampling [41], context-aware weighting [23], spatially aware bilateral grids [20], and text-guided control [22]. Other approaches focus on accelerating LUT-based processing [43, 42], or generating LUTs directly using diffusion models or vision-language models (VLMs) [33, 24]. These methods primarily address how LUTs are predicted, conditioned, or adapted to image content. Conversely, our work focuses on the LUT representation itself. Given a target color transformation, GLUT seeks to encode its mapping in a compact, continuous, and explicitly structured form while supporting localized editing.

3 Method

We propose 3D Gaussian LUT (GLUT), an explicit representation for continuous color transformations using a compact set of learnable 3D Gaussian primitives. Section 3.1 introduces the GLUT formulation. Section 3.2 presents a conditional generator that predicts GLUT parameters for multiple

LUT styles within a shared framework. Finally, in Section 3.3, we present an efficient LUT editing mechanism enabled by the localized structure of Gaussian primitives.

3.1 3D Gaussian LUT

GLUT represents a color mapping using a set of learnable 3D Gaussian primitives in RGB space, as illustrated in Figure 1c. Unlike INR-based LUTs, which encode color transformations in shared network parameters, GLUT explicitly associates each primitive with a spatially localized affine transformation.

Model Parameters. The representation consists of N 3D Gaussian primitives, where each primitive i is defined by:

- *Mean* $\boldsymbol{\mu}_i \in \mathbb{R}^3$, representing its center in RGB space.
- *Covariance matrix* $\boldsymbol{\Sigma}_i \in \mathbb{R}^{3\times3}$, controlling spatial extent and orientation. To ensure positive definiteness of each covariance matrix, we use the Cholesky parameterization $\boldsymbol{\Sigma}_i = \mathbf{L}_i\mathbf{L}_i^\top$, where $\mathbf{L}_i$ is a lower-triangular matrix whose diagonal entries are constrained to be positive using the Softplus function. We parameterize each covariance matrix using six learnable parameters per Gaussian.
- *Opacity* $o_i \in [0, 1]$, determining the contribution weight of the Gaussian.
- *Local color transformation*: parameterized by a transformation matrix $\mathbf{M}_i \in \mathbb{R}^{3\times3}$ and a bias term $\mathbf{b}_i \in \mathbb{R}^3$.

In addition, we introduce a *global base transformation* for each GLUT, parameterized by a matrix $\mathbf{G} \in \mathbb{R}^{3\times3}$ and bias $\mathbf{g} \in \mathbb{R}^3$, which captures coarse global color shifts. All the GLUT parameters $\mathbf{p} = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, o_i, \mathbf{M}_i, \mathbf{b}_i\}_{i=1}^N \bigcup \{\mathbf{G}, \mathbf{g}\}$ are learnable.

Color Transformation via GLUT. Given an input color $\mathbf{x} \in [0, 1]^3$, we compute the continuous color mapping $f(\mathbf{x})$ by combining the global transformation with a set of weighted, localized Gaussian transformations.

Gaussian weighting. The influence of each Gaussian transformation is defined by the density of its corresponding Gaussian primitive in RGB space:

$$p_i(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^3|\boldsymbol{\Sigma}_i|}} \exp\left(-\frac{1}{2}d_i(\mathbf{x})^2\right), \tag{1}$$

where $d_i(\mathbf{x}) = \sqrt{(\mathbf{x}-\boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)}$ denotes the Mahalanobis distance from $\mathbf{x}$ to Gaussian i.

Combining density with opacity o_i, we obtain a normalized influence weight of Gaussian i:

$$w_i(\mathbf{x}) = \frac{p_i(\mathbf{x})\, o_i}{\sum_{j=1}^N p_j(\mathbf{x})\, o_j + \epsilon}, \tag{2}$$

where ϵ is a small constant to ensure numerical stability. This formulation yields a soft partition of the RGB space.

Final color output. Each Gaussian applies a local affine transformation:

$$f_i(\mathbf{x}) = \mathbf{M}_i\mathbf{x} + \mathbf{b}_i. \tag{3}$$

In addition, the global affine transform provides a coarse base mapping:

$$f_{\text{global}}(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{g}. \tag{4}$$

The final color output is obtained via a mixture of all local and global transformations:

$$f(\mathbf{x}) = \sum_{i=1}^N w_i(\mathbf{x}) f_i(\mathbf{x}) + f_{\text{global}}(\mathbf{x}). \tag{5}$$

The value of $\hat{\mathbf{y}} = f(\mathbf{x})$ is clamped to $[0, 1]^3$ to ensure valid RGB values.

Training Objective. Since GLUT is an explicit representation, all the parameters are learnable and can be directly optimized using a loss function consisting of three terms:

Reconstruction loss. We use the ℓ_1 reconstruction loss between the predicted output $\hat{\mathbf{y}} = f(\mathbf{x})$ and the target color $\mathbf{y}$:

$$\mathcal{L}_{\text{rec}} = \|\hat{\mathbf{y}} - \mathbf{y}\|_1 . \tag{6}$$

Hue-Chroma loss. We further introduce a hue-chroma loss in the perceptually uniform CIELAB color space. Given the predicted color $\hat{\mathbf{y}}$ and the target color $\mathbf{y}$, we first transform both into the CIELAB space. For a color with CIELAB components $(L^*a^*b^*)$, its chroma is defined as $C = \sqrt{a^2 + b^2}$ and the hue is represented as a 2D unit vector $\mathbf{h} = (\frac{a}{C}, \frac{b}{C})$. We define the loss as the cosine distance between the predicted and target hue vectors, weighted by the target chroma:

$$\mathcal{L}_{\text{hc}} = C \cdot (1 - \langle \hat{\mathbf{h}}, \mathbf{h} \rangle). \tag{7}$$

This formulation prioritizes hue accuracy in highly saturated regions while maintaining tolerance in low-chroma regions (*e.g.*, neutrals), where hue is less perceptually meaningful and more susceptible to noise.

Sparsity regularization. To encourage specialization of Gaussian primitives, we regularize their opacity parameters using a binary entropy term:

$$\mathcal{R}_{\text{sparse}} = -\frac{1}{N} \sum_{i=1}^{N} \left[o_i \log(o_i + \epsilon) + (1 - o_i) \log(1 - o_i + \epsilon) \right], \tag{8}$$

where ϵ is a small constant for numerical stability.

Total loss. The overall training objective is $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{hc}}\mathcal{L}_{\text{hc}} + \lambda_{\text{sparse}}\mathcal{R}_{\text{sparse}}$, where λ_{hc} and λ_{sparse} control the contributions of the hue-chroma loss and sparsity regularization, respectively.

In summary, GLUT represents a color transformation as a continuous mapping that does not depend on a fixed sampling grid. It can model complex nonlinear transformations with fewer parameters than implicit neural networks. Each Gaussian primarily governs a local region of the color space, which enables targeted editing. The Gaussian parameters also have clear geometric meanings: the mean identifies the center of the affected color region, and the covariance describes the shape and extent of the region of influence. The entire pipeline is differentiable, so the GLUT parameters $\mathbf{p}$ can be optimized end-to-end, either from color tables or directly from natural images.

3.2 Conditional Gaussian LUT

Building upon the single Gaussian LUT formulation, we extend the model to represent multiple color transformations within a unified conditional framework, termed Conditional Gaussian LUT (CGLUT). Instead of representing each LUT independently using a one-hot condition vector, as in [8, 46], we condition the transformation on a learnable embedding that encodes each color style (see Figure 2). This enables a single model to represent multiple transformation functions while sharing structural priors across conditions.

Condition Embedding. Let $l \in \{1, \ldots, L\}$ denote the condition index of the lth LUT. We introduce a learnable embedding matrix $\mathbf{E} \in \mathbb{R}^{L \times D}$, where D is the embedding dimension. For a given condition l, the embedding vector $\mathbf{e}_l \in \mathbb{R}^D$ serves as a compact latent representation of the corresponding color style.

Gaussian Parameter Generation. We adopt a Gaussian parameter generator $\mathcal{G}$ that maps the embedding $\mathbf{e}_l$ to the parameters of all Gaussian primitives in a Gaussian LUT: $\mathbf{p}_l = \mathcal{G}(\mathbf{e}_l)$. The generator consists of a shared encoder that extracts common features, followed by multiple prediction heads, where each head predicts one type of parameter of the Gaussians. The generator is implemented using MLPs. Given a condition index l, the generator first predicts the corresponding GLUT parameters $\mathbf{p}_l = \mathcal{G}(\mathbf{e}_l)$. The input color $\mathbf{x}$ is then transformed using the predicted parameters, following the same GLUT formulation as in the single-LUT case. The training objectives are the same as those of GLUT (Section 3.1), while the learnable parameters in CGLUT comprise the embeddings $\mathbf{e}_l$ and the parameters of the generator $\mathcal{G}$.

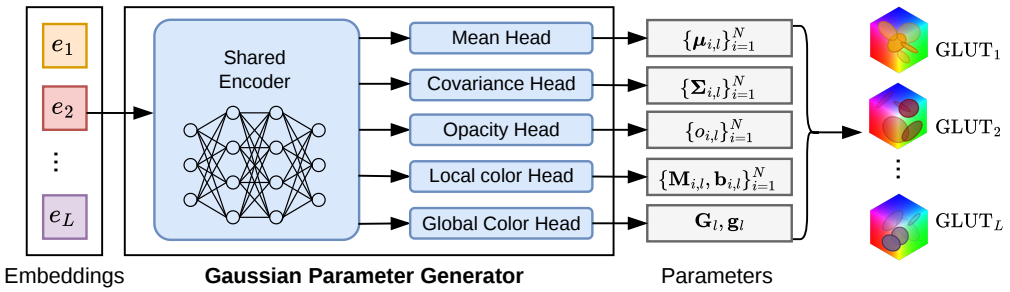


Figure 2: Structure of Conditional Gaussian LUT (CGLUT). The Gaussian parameter generator maps the learned embedding to the parameters of all Gaussian primitives in a GLUT using a shared MLP encoder and parameter-specific prediction heads.

LUT Blending. Since each LUT is represented by a continuous embedding vector in CGLUT, we blend two LUTs, indexed by l_1 and l_2, by linearly interpolating their embeddings. The interpolated embedding $\mathbf{e}^{\alpha}_{l_1 l_2} = (1-\alpha)\mathbf{e}_{l_1} + \alpha \mathbf{e}_{l_2}$, $\alpha \in [0,1]$ is fed into the generator to obtain the corresponding Gaussian parameters, yielding smooth transitions between color transformations.

In addition to generating all GLUT parameters independently for each condition (*Full Generation*), we also consider an alternative configuration, termed *Shared Geometry*. In this setup, the spatial attributes, Gaussian positions and shapes $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^N$, are shared across all styles, while the opacities and global color transformation parameters $\{o_{i,l}, \mathbf{M}_{i,l}, \mathbf{b}_{i,l}, \mathbf{G}_l, \mathbf{g}_l\}_{i=1}^N$ are generated conditionally. While the *Full Generation* setup offers greater flexibility for better reconstruction quality, *Shared Geometry* facilitates smoother color style transitions through embedding interpolation, as it maintains a consistent spatial partition of the color manifold.

Overall, CGLUT preserves the continuous and localized nature of GLUT, while enabling compact multi-style modeling with a single network.

3.3 GLUT Editing

A key advantage of GLUT over INR-based LUTs [8, 46] is that its parameters carry explicit geometric interpretations in RGB space, enabling direct and interpretable editing without retraining. Although users could manipulate Gaussian parameters directly, we found this approach unintuitive in practice, as multiple primitives must be adjusted simultaneously to achieve a coherent result. Instead, we propose a user-friendly editing interface in which the user specifies a *color constraint* $\mathbf{c} = (\mathbf{c}_{\text{in}}, \mathbf{c}_{\text{out}})$, where $\mathbf{c}_{\text{in}} \in [0,1]^3$ is an input color to modify and $\mathbf{c}_{\text{out}} \in [0,1]^3$ is the desired output color. The goal is to update a minimal subset of GLUT parameters such that $f(\mathbf{c}_{\text{in}}) \approx \mathbf{c}_{\text{out}}$, while minimizing unintended changes elsewhere in the color space. This formulation naturally supports iterative, fine-grained adjustments, allowing users to steer the learned color transformation towards a desired style without global interference or model retraining.

Given the current GLUT output at $\mathbf{c}_{\text{in}}$, we compute the correction residual:

$$\boldsymbol{\delta} = \mathbf{c}_{\text{out}} - f(\mathbf{c}_{\text{in}}) \in \mathbb{R}^3. \tag{9}$$

Rather than modifying all N Gaussians, we select the K most influential ones at $\mathbf{c}_{\text{in}}$ based on the normalized influence weights from Eq. (2):

$$\mathcal{K} = \text{argtop}_K \{w_i(\mathbf{c}_{\text{in}})\}_{i=1}^N. \tag{10}$$

Although $\mathcal{K}$ can be chosen freely by the user, selecting the K most influential Gaussians at $\mathbf{c}_{\text{in}}$ provides a natural and automatic proxy. The parameter K controls the spatial extent of the edit: a smaller K confines the modification to the immediate neighbourhood of $\mathbf{c}_{\text{in}}$, whereas a larger K distributes $\boldsymbol{\delta}$ across more Gaussians, producing a broader effect in the color space.

We distribute the residual $\boldsymbol{\delta}$ among the selected Gaussians proportionally to their normalised influence:

$$\alpha_k = \frac{w_k(\mathbf{c}_{\text{in}})}{\sum_{j \in \mathcal{K}} w_j(\mathbf{c}_{\text{in}})}, \quad k \in \mathcal{K}. \tag{11}$$

The bias term of each selected Gaussian is then updated as:

$$\mathbf{b}_k \leftarrow \mathbf{b}_k + s \cdot \alpha_k \cdot \boldsymbol{\delta}, \quad k \in \mathcal{K}, \tag{12}$$

where $s \in [0, 1]$ is a user-controlled *strength* parameter. We update only the bias terms $\mathbf{b}_k$ and keep the local color transformation matrices $\mathbf{M}_k$ fixed. This better preserves the local color relationships (*i.e*. contrast, saturation, and structure) encoded by each Gaussian, restricting the edit to a global shift of the output color within that region.

The edit is performed in real time and is inherently localized, without requiring an explicit locality constraint. The densities p_k of the modified Gaussians decay exponentially with the Mahalanobis distance from their centers. Consequently, at colors $\mathbf{c}$ where other primitives dominate, the normalized weights of the modified Gaussians tend to be small, making the effect of the bias update on $f(\mathbf{c})$ negligible. This locality naturally arises from the explicit Gaussian representation in color space. An interactive GLUT editing tool is provided in the Appendix, with further details in Section D.

4 Experiments

4.1 Experimental Setup

Implementation Details. The model is implemented in PyTorch and optimized using the Adam optimizer on a single NVIDIA GeForce RTX 4090 GPU. We employ a cosine annealing learning rate schedule, starting from 10^{-3} over the entire training duration. GLUT is trained for 20 epochs with a batch size of 1024, while CGLUT is trained for 40 epochs with a batch size of 8192. The loss weights are empirically set to $\lambda_{\text{hc}} = 10$ and $\lambda_{\text{sparse}} = 0.001$. Additional implementation details are provided in Section B.

Dataset. Our dataset comprises 300 LUTs collected from publicly available online sources, split into 225 LUTs with resolutions of 33^3 or 32^3 and 75 LUTs at 64^3. The 7 LUTs used for multi-LUT modeling are taken from [8], each defined on a standard 33^3 grid, and are also included in our 225-LUT subset. For training, we extract dense color mappings from the corresponding `.cube` files. We uniformly sample the full 8-bit RGB space to construct a 128^3 training sample, reserving the remaining colors for evaluation to verify that the model learns a continuous representation. The resulting mappings are reshaped into Hald images [1] of resolutions $1024\times2048\times3$ and $3584\times4096\times3$ for the training and test colors, respectively. All LUTs are open-source under Creative Commons licensing.

Evaluation. Color fidelity is measured via PSNR, CIEDE2000 (ΔE_{00}), and CIEDE1976 (ΔE_{76}), covering both RGB and perceptual CIELAB color spaces. We evaluate GLUT on two settings: direct Hald image retouching and applying learned neural LUTs to 100 natural images from MIT-Adobe FiveK [4], where we additionally report SSIM [39] and LPIPS [49] for structural similarity and perceptual quality. Model efficiency is reported in GFLOPs (measured on $512\times512\times3$ image) and parameter count. **Bold** and *italic* indicate the best and second-best results in each table, respectively.

Comparison Methods. We evaluate GLUT against NILUT [8] on single-LUT modeling, and CGLUT against CNILUT [8] and ENNELUT [46] on multi-LUT representation. NILUT and CNILUT follow official configurations with extended training iterations (10K and 60K, respectively) to ensure adequate convergence. ENNELUT is re-implemented because its code is unavailable. NILUT (64×2) uses two hidden layers of 64 neurons each, while GLUT-16 uses 16 Gaussian primitives. ENNELUT is trained with L_2 loss or ΔE loss, respectively. INRetouch [10] addresses local retouching rather than global color transformations, so it is compared separately in Section C.1.2 in the Appendix.

4.2 Experiment Results

Comparison with classical approaches. We compare GLUT with classical approaches from three categories: *root-polynomial regression* [11], *Gaussian-based local models*: RBF network [27] and Gaussian Locally Linear Mapping (GLLiM) [9], and *LUT compression and interpolation*: Keypoint diffusion [38], Tetrahedral interpolation [17], and Tricubic interpolation [12]. For all baselines except root-polynomial regression [11], we match the parameter count to that of GLUT-32 for a controlled comparison. The implementation details are provided in Section B.4 in the Appendix. As shown in Table 1, GLUT substantially outperforms the RBF network, demonstrating the benefit of modeling each Gaussian primitive with a local affine transformation rather than a constant output.

Table 1: Quantitative comparison between classic baselines and GLUT.

Method	75 LUTs			225 LUTs			#Params
	PSNR↑	ΔE_{00}↓	ΔE_{76}↓	PSNR↑	ΔE_{00}↓	ΔE_{76}↓	
Root-polynomial regression [11]	23.55	5.10	9.96	25.15	4.66	9.23	18
RBF network (Our GD) [27]	42.90	0.52	0.96	47.82	0.38	0.73	717
GLLiM (EM) [9]	38.03	1.01	1.93	41.88	0.78	1.50	726
GLLiM (Our GD) [9]	*44.72*	*0.44*	*0.82*	*49.13*	*0.34*	*0.66*	726
Keypoint diffusion [38]	31.41	1.75	3.64	33.81	1.82	3.96	720
6^3 LUT, tetrahedral [17]	33.55	1.47	3.20	36.20	1.31	2.87	648
6^3 LUT, tricubic [12]	34.81	1.22	2.49	38.51	1.04	2.12	648
GLUT-32 (Ours)	**45.47**	**0.41**	**0.77**	**50.42**	**0.30**	**0.58**	716

Table 2: Quantitative comparison of single-LUT modeling on Hald images and natural images.

Method	Hald images			Natural images					#Params	GFLOPs
	PSNR↑	ΔE_{00} ↓	ΔE_{76} ↓	PSNR↑	SSIM↑	LPIPS↓	ΔE_{00} ↓	ΔE_{76} ↓		
NILUT (64×2) [8]	36.37	1.18	2.26	38.17	0.979	0.017	2.95	3.82	8,771	4.60
NILUT (128×2) [8]	38.84	0.86	1.62	40.84	0.990	0.009	2.07	2.54	33,923	17.78
GLUT-16	41.50	0.64	1.22	43.01	0.994	0.005	1.31	1.75	364	0.25
GLUT-32	*45.47*	*0.41*	*0.77*	*45.92*	*0.997*	*0.003*	*0.98*	*1.25*	716	0.49
GLUT-64	**48.42**	**0.31**	**0.56**	**47.90**	**0.998**	**0.002**	**0.76**	**0.91**	1,420	0.98

The performance gap between the two GLLiM variants further suggests that our gradient-descent (GD) optimization framework provides a more effective fit than classical expectation-maximization (EM). The relative ranking of the methods remains consistent across the 75-LUT and 225-LUT datasets, with GLUT-32 achieving the best PSNR and ΔE_{00} under a matched parameter budget in both settings.

Single GLUT. Table 2 reports the quantitative results of GLUT and NILUT on 75-LUT Hald images and 100 natural images. GLUT achieves higher PSNR and lower ΔE with substantially fewer parameters and less computation. For a GLUT model with N Gaussian primitives, the total parameter count is $22N+12$. In a typical configuration with $N = 32$, the model consists of only 716 parameters, resulting in a significantly smaller memory footprint than INR-based models. Results on 225-LUT Hald images and additional qualitative comparisons are shown in Table 5 and Figure 7 of Section C.1 in the Appendix.

Multiple LUTs. Table 3 and Table 8 report the performance of CGLUT in representing multiple color styles. We evaluate CGLUT on 7, 75, and 225 LUTs, comparing it against CNILUT across two model scales. We consider two CGLUT variants, Small and Large, with increasing modeling capacity by using 64 and 128 neurons, respectively, in each layer of the parameter generator.

During inference, CGLUT maintains a constant input dimensionality regardless of the library size, *i.e.*, the number of learned LUTs. The parameter generation process is executed only once per style, incurring negligible computational cost (about 4 MFLOPs for CGLUT-Large). The generated parameters are then used for pixel-wise color mapping. Consequently, the computational cost of CGLUT lies in the color transformation stage rather than the conditioning mechanism. This decoupled architecture keeps the inference cost on a natural image largely independent of the number of LUTs in the library. In contrast, CNILUT's computational cost scales with the library size because the style condition must be processed at each pixel, resulting in substantial overhead for large LUT collections.

In terms of reconstruction quality, CGLUT outperforms both CNILUT and ENNELUT on the 75- and 225-LUT datasets. In the 7-LUT setting, only ENNELUT (L_2) achieves better results than CGLUT-32 (Small), further demonstrating the competitive performance of our explicit framework.

Efficiency and Model Size. Table 4 summarizes the runtime performance on an AMD EPYC 9634 CPU (84-core) and an NVIDIA RTX 4090 GPU (24 GB), along with the compression ratios of different neural LUTs. Runtime is measured as the average processing time over 100 natural images from MIT5K [4] at three resolutions: Small (512×512), HD (1280×720), and 4K UHD (3840×2160). The compression ratio is defined as the size of the neural LUT (`.pth` file) relative to that of the standard grid-based LUT (`.cube` file).

Grid-based LUTs remain the fastest option for applying a fixed color transformation, as GPU texture units perform lookup and interpolation in hardware. Among compact representations, GLUT achieves

Table 3: Quantitative comparison of multiple-LUT modeling on Hald image dataset.

Method	7 LUTs PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs	75 LUTs PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs	225 LUTs PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs
CNILUT (128×3) [8]	44.90	0.43	0.85	51K	26.92	38.60	0.99	1.88	60K	31.48	38.52	1.16	2.21	79K	41.54
CNILUT (256×3) [8]	48.38	0.32	0.61	201K	105.36	41.20	0.72	1.33	218K	114.48	41.46	0.80	1.52	257K	134.62
ENNELUT (ΔE) [46]	49.99	0.27	0.55	14K	3.55	41.82	0.62	1.24	20K	5.26	39.43	0.95	1.95	34K	9.03
ENNELUT (L_2) [46]	53.07	0.20	0.40	14K	3.55	44.29	0.55	1.00	20K	5.26	40.68	0.96	1.87	34K	9.03
CGLUT-32 (Small)	50.76	0.21	0.41	84K	0.49	45.66	0.39	0.73	89K	0.49	46.68	0.41	0.82	98K	0.49
CGLUT-32 (Large)	53.55	0.17	0.34	233K	0.49	46.69	0.36	0.66	238K	0.49	*49.84*	*0.30*	*0.58*	247K	0.49
CGLUT-64 (Small)	*54.06*	*0.16*	*0.32*	130K	0.98	*48.43*	*0.30*	*0.55*	135K	0.98	49.39	*0.30*	0.59	144K	0.98
CGLUT-64 (Large)	**55.10**	**0.14**	**0.29**	324K	0.98	**49.37**	**0.28**	**0.51**	328K	0.98	**52.41**	**0.22**	**0.43**	338K	0.98

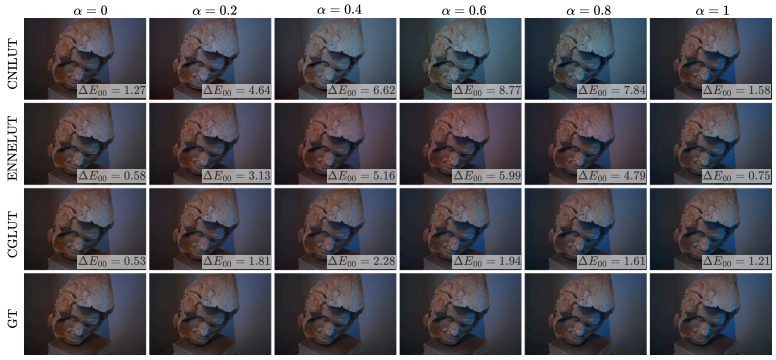


Figure 3: Qualitative comparison of style blending with different α. Rows 1–4 illustrate the results of CNILUT (256×3), ENNELUT (L_2), CGLUT-32L (Shared Geo.), and image-space blending (GT), respectively.

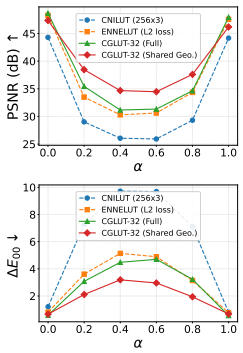


Figure 4: Quantitative comparison of blending with different α.

the highest efficiency. For single-LUT models, GLUT-16 runs at approximately 1 FPS on a 4K image on the CPU and at 241 FPS with our fused CUDA kernel, compared with 47 FPS for NILUT (64×2) accelerated with tiny-cuda-nn. For multi-LUT models, CGLUT is the second fastest on the CPU, after ENNELUT, and it reaches 194 FPS at 4K with CUDA, compared with 10 FPS for ENNELUT. At high resolutions, the cost of CGLUT is dominated by applying the generated GLUT, since the generator runs once per image whereas the color transform runs once per pixel. Accordingly, the Small and Large variants achieve nearly identical speeds of 194 and 193 FPS on 4K images.

Furthermore, neural LUTs offer a compact, continuous representation that significantly reduces memory overhead. For instance, standard 64^3 and 32^3 LUTs require 7.1 MB and 800 kB, while a GLUT-64 model occupies only 9.7 kB, resulting in compression rates of 0.14% and 1.05% for 75-LUT and 225-LUT settings. Such compactness makes GLUT ideal for memory-constrained applications like mobile caching or network transmission. Although ENNELUT provides the highest compression ratio, our method achieves a superior balance of fidelity and processing speed.

Table 4: Comparison of runtime and compression ratio. "-" denotes FPS<0.01. "†" denotes time on GPU texture unit. "⋆" denotes implementation with tiny-cuda-nn [28].

Method	FPS (CPU)↑ 512^2	HD	4K	FPS (GPU)↑ 512^2	HD	4K	Comp. Ratio (%)↓ 75-LUT	225-LUT
Grid LUT	3489	759	148	26673†	5577†	1510†	100	100
NILUT (64×2) [8]	11.42	1.82	0.47	538⋆	239⋆	47⋆	0.55	4.32
NILUT (128×2) [8]	5.87	0.83	0.20	822⋆	111⋆	28⋆	1.96	15.70
GLUT-16	30.12	3.69	0.97	6496	952	241	0.08	0.60
GLUT-32	14.05	1.96	0.49	6170	891	226	0.10	0.75
GLUT-64	6.80	1.02	0.25	4917	693	176	0.14	1.05
CNILUT (128×3) [8]	0.34	0.01	-	1.86	0.03	-	0.05	0.16
CNILUT (256×3) [8]	0.30	-	-	1.83	0.03	-	0.17	0.52
ENNELUT (L_2) [46]	4.92	0.66	0.18	406	39	10	0.02	0.07
CGLUT-32 (Small)	1.01	0.14	0.04	1147	545	194	0.07	0.20
CGLUT-32 (Large)	0.87	0.13	0.03	1146	545	193	0.18	0.50
CGLUT-64 (Small)	0.52	0.07	0.02	1100	473	160	0.10	0.30
CGLUT-64 (Large)	0.48	0.07	0.02	1100	465	156	0.25	0.68

Blending LUTs. Figure 3 illustrates a qualitative comparison of style blending across different models. By interpolating the learned style embeddings, CGLUT achieves high-quality transitions that are consistent with direct linear blending in image space. In contrast, although CNILUT and ENNELUT perform comparably on the two endpoint styles, their intermediate results differ: CNILUT exhibits significant color shifts,

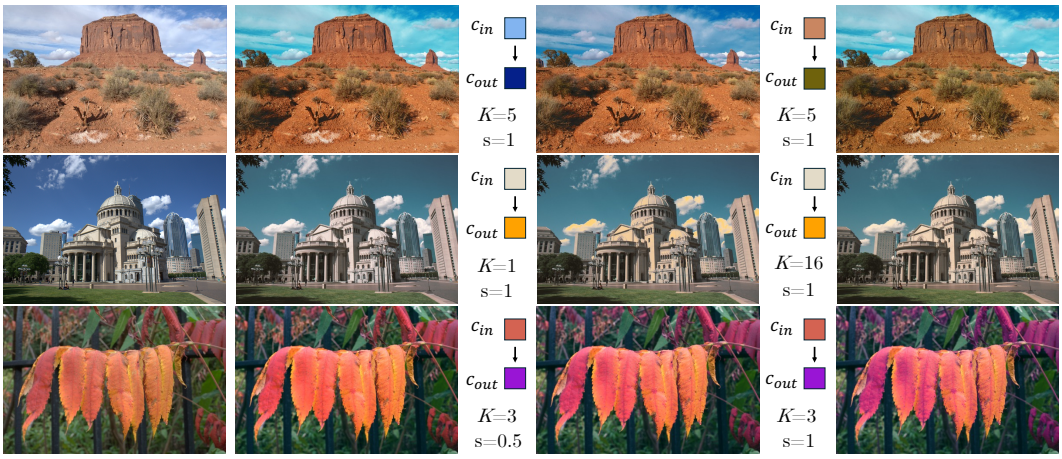


Figure 5: Qualitative results of editing GLUT representations. Each row shows an input image, a retouched image by GLUT, and two edited versions obtained with different color constraints $\mathbf{c} = (\mathbf{c}_{\text{in}}, \mathbf{c}_{\text{out}})$, number of Gaussians to edit K, and strength s.

whereas ENNELUT produces smaller shifts. This observation is further validated in Figure 4, which evaluates the blended LUTs on 100 images from the MIT5K dataset [4]. Both CGLUT variants, using fully generated parameters and shared geometry, outperform ENNELUT and CNILUT in terms of PSNR and ΔE, with the shared-geometry variant achieving the best overall performance. Additional blending results are presented in Section C.2.2 of the Appendix.

GLUT Editing. Figure 5 shows six examples of editing trained GLUT representations using our proposed method. In the first row, adjusting $\mathbf{c}$ shifts the sky toward a darker blue or the ground toward a darker orange, while leaving the rest of the color transformation largely unchanged. In the second row, varying K controls the locality of the edit within the RGB cube. When $K = 1$, the correction is concentrated near $\mathbf{c}_{\text{in}}$, as only the most influential Gaussian primitive is updated. When $K = 16$, the edit spreads more globally as a larger set of Gaussians contributes to the correction. In the third row, with $\mathbf{c}$ and K fixed, increasing s progressively shifts the orangish leaves toward purple, demonstrating that s provides intuitive, continuous control over the edit magnitude.

We emphasize that GLUT editing is not designed for semantically or spatially aware manipulation. Rather, it provides a real-time mechanism to adjust the *mood* or *style* of an existing color transformation without retraining. We analyze the locality of GLUT editing in Section C.3.2 and describe an interactive editing tool in Section D of the Appendix.

Ablation Studies. We conduct ablation studies with the GLUT-32 model on the 75-LUT Hald dataset. We evaluate the number of Gaussian primitives, different local transformation formulations, key architectural components, loss terms, and training strategies, including hard sample mining and initialization. The results in Section C.4 of the Appendix assess the contribution of each design choice and demonstrate the effectiveness of our proposed framework.

5 Concluding Remarks

We propose GLUT, a continuous color look-up table representation based on explicit Gaussian primitives. Unlike previous discrete or implicit methods, GLUT provides a compact, interpretable, and fully differentiable framework for global color grading and fine-grained manipulation. Our results show that GLUT achieves high color fidelity while offering remarkable computational efficiency. We further introduce CGLUT, a conditional variant that represents multiple LUTs within a unified model and supports smooth blending between color styles. By replacing rigid grids with optimizable, spatially localized primitives, GLUT offers a flexible approach to color manipulation. We envision this representation serving as a building block for a broader range of applications, including real-time ISP tuning and interactive artistic grading.

Acknowledgments and Disclosure of Funding

This work was supported by Grant PID2024-162555OB-I00 funded by MCIN/AEI/10.13039/501100011033 and by ERDF, EU, AIA2025-163919-C52 funded by MCIN/AEI/10.13039/501100011033, the grant Catedra ENIA UAB-Cruïlla (TSI-100929-2023-2) from the Ministry of Economic Affairs and Digital Transformation of Spain, and by the Generalitat de Catalunya CERCA Program. Danna Xue and Javier Vazquez-Corral also acknowledge the 2025 Leonardo Grant for Scientific Research and Cultural Creation from the BBVA Foundation. The BBVA Foundation accepts no responsibility for the opinions, statements and contents included in the project and/or the results thereof, which are entirely the responsibility of the authors. David Serrano-Lozano also acknowledges the FPI grant from the Spanish Ministry of Science and Innovation (PRE2022-101525). Shaolin Su was supported by the HORIZON MSCA Postdoctoral Fellowship funded by the European Union (project number 101152858). The authors declare no competing interests.

References

[1] 3d lut creator - hald images description. `https://3dlutcreator.com/3d-lut-creator---materials-and-luts.html`. Accessed: 2026-03-01.

[2] Aravindh Balaji, Gaurav Sharma, Mark Q. Shaw, and Randall Guay. Hierarchical compression of color look-up tables. In *Color and Imaging Conference*, 2007.

[3] Amir Barda, Matheus Gadelha, Vladimir G Kim, Noam Aigerman, Amit H Bermano, and Thibault Groueix. Instant3dit: Multiview inpainting for fast editing of 3d objects. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2025.

[4] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input/output image pairs. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2011.

[5] Du Chen, Liyi Chen, Zhengqiang Zhang, and Lei Zhang. Generalized and efficient 2d gaussian splatting for arbitrary-scale super-resolution. In *Int. Conf. Comput. Vis.*, 2025.

[6] Wenbo Chen and Ligang Liu. Deblur-gs: 3d gaussian splatting from camera motion blurred images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–15, 2024.

[7] Yaosen Chen, Han Yang, Yuexin Yang, Yuegen Liu, Wei Wang, Xuming Wen, and Chaoping Xie. Nlut: Neural-based 3d lookup tables for video photorealistic style transfer. *arXiv preprint arXiv:2303.09170*, 2023.

[8] Marcos V Conde, Javier Vazquez-Corral, Michael S Brown, and Radu Timofte. Nilut: Conditional neural implicit 3d lookup tables for image enhancement. In *AAAI*, 2024.

[9] Antoine Deleforge, Florence Forbes, and Radu Horaud. High-dimensional regression with gaussian mixtures and partially-latent response variables. *Statistics and Computing*, 25(5):893–911, 2015.

[10] Omar Elezabi, Marcos V Conde, Zongwei Wu, and Radu Timofte. Inretouch: Context aware implicit neural representation for photography retouching. In *Winter Conference on Applications of Computer Vision*, 2026.

[11] Graham D Finlayson, Michal Mackiewicz, and Anya Hurlbert. Color correction using root-polynomial regression. *IEEE Trans. Image Process.*, 24(5):1460–1470, 2015.

[12] Teruo Fumoto, Osamu Yamada, and Hideto Motomura. Extraction of non-linear components from 3d-lut in icc profiles and re-composition of new profile for better color reproduction. In *International Conference on Digital Printing Technologies*, pages 475–478, 1997.

[13] Eric Garcia and Maya Gupta. Lattice regression. *Adv. Neural Inform. Process. Syst.*, 22, 2009.

[14] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH*, pages 1–11, 2024.

[15] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[16] Hakki Can Karaimer and Michael S Brown. Improving color reproduction accuracy on cameras. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[17] James M Kasson, Sigfredo I Nin, Wil Plouffe, and James Lee Hafner. Performing color space conversions with three-dimensional linear interpolation. *Journal of Electronic Imaging*, 4(3):226–250, 1995.

[18] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4), 2023.

[19] Heewon Kim and Kyoung Mu Lee. Learning controllable isp for image enhancement. *IEEE Trans. Image Process.*, 33:867–880, 2023.

[20] Wontae Kim and Nam Ik Cho. Image-adaptive 3d lookup tables for real-time image enhancement with bilateral grids. In *Eur. Conf. Comput. Vis.*, pages 91–108. Springer, 2024.

[21] Wontae Kim, Keuntek Lee, and Nam Ik Cho. Lightweight and fast real-time image enhancement via decomposition of the spatial-aware lookup tables. In *Int. Conf. Comput. Vis.*, 2025.

[22] Hyeongmin Lee, Kyoungkook Kang, Jungseul Ok, and Sunghyun Cho. Cliptone: Unsupervised learning for text-based image tone adjustment. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2024.

[23] Chengxu Liu, Huan Yang, Jianlong Fu, and Xueming Qian. 4d lut: learnable context-aware 4d lookup table for image enhancement. *IEEE Trans. Image Process.*, 32:4742–4756, 2023.

[24] Tianren Ma, Mingxiang Liao, Xijin Zhang, and Qixiang Ye. Acetone: Bridging words and colors for conditional image grading. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2026.

[25] 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. *Communications of the ACM*, 65(1):99–106, 2021.

[26] Vishal Monga and Zhigang Fan. Imaging device color characterization including color look-up table construction via tensor decomposition, December 11 2012. US Patent 8,331,662.

[27] John Moody and Christian J Darken. Fast learning in networks of locally-tuned processing units. *Neural computation*, 1(2):281–294, 1989.

[28] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):1–15, 2022.

[29] Ante Poljicak, Jurica Dolic, and Jesenka Pibernik. An optimized radial basis function model for color characterization of a mobile device display. *Displays*, 41:61–68, 2016.

[30] Peter Postma and Bob Chorley. Color grading with color management. *SMPTE Motion Imaging Journal*, 125(8):69–74, 2016.

[31] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *Int. Conf. Mach. Learn.*, 2019.

[32] Mark Shaw, Randall G Guay, Gaurav Sharma, and Aravindh BS Rajagopalan. Lossless compression of color look-up table via hierarchical differential encoding or cellular interpolative prediction, October 23 2012. US Patent 8,294,953.

[33] Seunghyun Shin, Dongmin Shin, Jisu Shin, Hae-Gon Jeon, and Joon-Young Lee. Video color grading via look-up table generation. In *Int. Conf. Comput. Vis.*, 2025.

[34] Zijie Tan, Yuxin Yue, and Bahador Rashidi. Chameleontuner: Automatic isp color tuning in subjective scenarios. In *Winter Conference on Applications of Computer Vision*, 2026.

[35] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Adv. Neural Inform. Process. Syst.*, 2020.

[36] Chuohao Tang, Weibao Wang, Sean Collison, Mark Shaw, Jay Gondek, Amy Reibman, and Jan Allebach. Icc profile color table compression. In *Color and Imaging Conference*, 2016.

[37] David Tschumperlé, Christine Porquet, and Amal Mahboubi. 3d color clut compression by multi-scale anisotropic diffusion. In *International Conference on Computer Analysis of Images and Patterns*, 2019.

[38] David Tschumperlé, Christine Porquet, and Amal Mahboubi. Reconstruction of smooth 3d color functions from keypoints: Application to lossy compression and exemplar-based generation of color luts. *SIAM Journal on Imaging Sciences*, 13(3):1511–1535, 2020.

[39] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004.

[40] Jing Wu, Jia-Wang Bian, Xinghui Li, Guangrun Wang, Ian Reid, Philip Torr, and Victor Adrian Prisacariu. Gaussctrl: Multi-view consistent text-driven 3d gaussian splatting editing. In *Eur. Conf. Comput. Vis.*, 2024.

[41] Canqian Yang, Meiguang Jin, Xu Jia, Yi Xu, and Ying Chen. Adaint: Learning adaptive intervals for 3d lookup tables on real-time image enhancement. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[42] Canqian Yang, Meiguang Jin, Yi Xu, Rui Zhang, Ying Chen, and Huaida Liu. Seplut: Separable image-adaptive lookup tables for real-time image enhancement. In *Eur. Conf. Comput. Vis.*, 2022.

[43] Sidi Yang, Binxiao Huang, Mingdeng Cao, Yatai Ji, Hanzhong Guo, Ngai Wong, and Yujiu Yang. Taming lookup tables for efficient image retouching. In *Eur. Conf. Comput. Vis.*, 2024.

[44] Sidi Yang, Binxiao Huang, Yulun Zhang, Dahai Yu, Yujiu Yang, and Ngai Wong. Dnlut: Ultra-efficient color image denoising via channel-aware lookup tables. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2025.

[45] Sipeng Yang, Qingchuan Zhu, Junhao Zhuge, Qiang Qiu, Chen Li, Yuzhong Yan, Huihui Xu, Ling-Qi Yan, and Xiaogang Jin. Mob-fgsr: Frame generation and super resolution for mobile real-time rendering. In *ACM SIGGRAPH*, 2024.

[46] Vahid Zehtab, David B Lindell, Marcus A Brubaker, and Michael S Brown. Efficient neural network encoding for 3d color lookup tables. In *AAAI*, 2025.

[47] Hui Zeng, Jianrui Cai, Lida Li, Zisheng Cao, and Lei Zhang. Learning image-adaptive 3d lookup tables for high performance photo enhancement in real-time. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(4):2058–2073, 2020.

[48] Fengyi Zhang, Hui Zeng, Tianjun Zhang, and Lin Zhang. Clut-net: Learning adaptively compressed representations of 3dluts for lightweight image enhancement. In *ACM Int. Conf. Multimedia*, 2022.

[49] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[50] Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. In *Eur. Conf. Comput. Vis.*, 2024.

[51] Yunxiang Zhang, Bingxuan Li, Alexandr Kuznetsov, Akshay Jindal, Stavros Diolatzis, Kenneth Chen, Anton Sochenov, Anton Kaplanyan, and Qi Sun. Image-gs: Content-adaptive image representation via 2d gaussians. In *ACM SIGGRAPH*, 2025.

Appendix

In this appendix, we provide the statement about limitations and broader impacts, *etc*., in Section A, additional implementation details in Section B, followed by additional experimental results and ablation studies in Section C. Furthermore, we offer a step-by-step user instruction for our interactive GLUT-based LUT editing tool in Section D.

A Statements

Limitations and Discussion. While GLUT demonstrates strong performance in color transformation tasks, several limitations should be acknowledged. First, GLUT is designed to improve the efficiency and interpretability of neural color LUTs, which are inherently limited to global color transformations, where the output depends solely on the input RGB values. Although GLUT currently focuses on global color transformations, it could potentially be extended to context-dependent tasks, such as spatially varying image retouching, with further research, as would be required for conventional LUTs. Second, for both GLUT and CGLUT, the number of Gaussian primitives N is set as a fixed hyperparameter prior to training, yet the optimal N may vary across different color styles and datasets. While we provide empirical guidelines, automatically determining the optimal N during training remains an open problem. Techniques analogous to adaptive density control in 3D Gaussian Splatting could potentially be adapted for this purpose. Third, although GLUT supports localized LUT editing, its locality depends on the spatial extent of the neighboring Gaussian primitives. Extending the editing framework to support finer-grained control, for example by adding Gaussian primitives in a specific region or fine-tuning additional Gaussian parameters, is a potential direction for future work.

Broader Impacts. This work advances the efficiency and interpretability of color transformation technology, with potential positive impacts across several domains. In professional photography and cinematography, GLUT offers a compact and editable representation of color grading styles, enabling efficient storage, transmission, and application of color presets in large-scale media production pipelines. The interactive GLUT Editor facilitates intuitive color correction workflows, potentially making professional-grade color grading tools more accessible for independent creators. In computational photography, the learned continuous LUT representation could improve camera ISP calibration and cross-device color matching, benefiting consumer imaging applications. The compact model size, which can be substantially smaller than that of traditional LUTs, makes color transformation more efficient on resource-constrained devices such as smartphones and embedded systems, broadening access to professional color science tools.

We do not foresee significant negative societal impacts specific to this work. Color transformation is a well-established component of standard media production workflows, and GLUT improves its efficiency rather than introducing qualitatively new capabilities. However, as with any image processing technology, efficient color transformations could potentially be used in image manipulation or deepfake production pipelines. We emphasize that GLUT is designed for color transformation rather than content generation, and its application to deceptive media manipulation would require additional components beyond the scope of this work.

Ethical Statement. This work addresses the technical problem of efficient color transformation representation and does not involve human subjects, personally identifiable information, or sensitive data collection. All experiments were conducted on publicly available image datasets. The proposed GLUT Editor is designed to assist professional colorists and photographers in their creative workflows, with all designs prioritizing transparency and user control over the color transformation process. The explicit Gaussian representation provides interpretable parameters, including the location and spatial extent of each primitive in RGB space, allowing users to inspect and directly edit the color transformation. We have released all code, pretrained models, and training configurations to support open and reproducible research in color science and image processing.

Reproducibility Statement. To facilitate reproducibility, we have released the complete source code, including model implementations, training and evaluation scripts, pretrained model checkpoints, and the GLUT Editor interface at https://github.com/CVC-Color/glut. Hyperparameter configurations for all reported experiments are provided in the appendices. All experiments reported in this paper were conducted with fixed random seeds. The datasets used in this work are either publicly available or

accompanied by detailed instructions for data preparation. The GLUT Editor is implemented as a self-contained browser application (HTML/JavaScript, CPU-only) that runs without installation.

Future Work. A possible future direction involves integrating spatial or semantic features into the Gaussian parameter generator, extending CGLUT into a spatially varying representation that enables content-aware color manipulation while preserving the efficiency of explicit primitives. Moreover, the compact style embeddings of CGLUT provide a natural interface for multimodal control. Another possible direction involves leveraging vision-language models to integrate text prompts into GLUT parameter generation. This will enable intuitive, language-driven color grading, thereby further expanding the applications of image color editing.

B Implementation Details

B.1 Training and Evaluation

Initialization. For GLUT initialization, Gaussian means are initialized by distributing them uniformly on a regular grid covering the RGB cube $[0, 1]^3$. Covariances are initialized isotropically with a scale of $\sigma = 0.15$ through logarithmic Cholesky parameters. Opacities are initially set to 1.0, and the affine color transforms are initialized as identity matrices with zero bias.

Training of CGLUT. Since CGLUT comprises multiple components with different roles, we apply a lower learning rate ($0.1\times$ the base rate) to the style embeddings and shared geometry parameters, while the generator (shared feature encoder and parameter heads) uses the base learning rate of 10^{-3}. We set $\epsilon = 10^{-6}$ in Equation (2) and Equation (8) for numerical stability.

Hard Sample Mining. We employ a curriculum hard-mining strategy to guide the model toward learning complex and high-error color transformations. Specifically, from epoch 5 to 20, the proportion of samples with the highest L_1 errors selected for mining increases linearly from 10% to 40%. This strategy encourages the model to focus progressively on challenging mappings and reduces its tendency to prioritize easy-to-fit samples.

Datasets. As is common practice, Hald images [1] are widely used for representing LUTs and as training samples for neural LUT methods [8, 46]. We uniformly sample the full 8-bit RGB space to construct a 128^3 training set, reserving the remaining colors for evaluation to verify that the model learns a continuous representation. The resulting mappings are reshaped into Hald images of resolutions $1024\times2048\times3$ and $3584\times4096\times3$, respectively. By applying different LUTs to the original Hald image, we obtain output Hald images of the same resolution but with different color styles. These Hald image pairs are used for training and testing. For evaluation on Hald images, we report PSNR, ΔE_{00}, and ΔE_{76}.

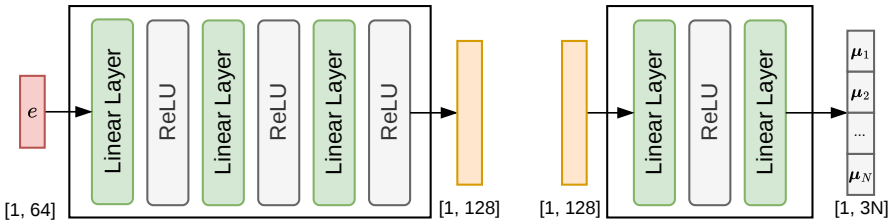
To assess color transformation quality in more general scenarios, we also test on 100 natural images from the MIT-Adobe FiveK (MIT5K) dataset [4] (images #4501 to #4600). We use the images processed by traditional LUTs (`.cube` files) as ground truth, and compare them with images processed by applying the learned NILUT, CNILUT, ENNELUT, GLUT and CGLUT models. These natural images are not used for training. The color fidelity is evaluated using PSNR, SSIM, LPIPS, ΔE_{00}, and ΔE_{76}.

B.2 CGLUT Architecture

Figure 6 illustrates the architecture of the Gaussian parameter generator in our CGLUT. Given a 64-dimensional style embedding, a shared encoder, consisting of three linear layers with 128 (64 for "small" setup) hidden units each, extracts a latent feature representation. This feature is then fed into multiple task-specific heads to predict the Gaussian parameters. As shown in Figure 6 (b), the head responsible for mean values $\boldsymbol{\mu}$ comprises two linear layers: an initial layer with 128 (64 for "small" setup) units, followed by an output layer whose dimension is determined by the number of Gaussians N. For instance, to predict 3D mean values, the output dimension is $3N$. Other parameter heads follow a similar structure with adjusted output dimensions tailored to their respective parameters. For the specific case of the global affine transform, since each GLUT incorporates a single global affine transformation, the corresponding head outputs 12 parameters (9 for the matrix and 3 for the bias). All the parameter heads have the same structure as the mean head with two linear layers, except for the local color head, which has three linear layers to learn more complicated local color

transformation parameters. All intermediate layers are interleaved with ReLU activations to ensure sufficient representational capacity.

In the *Shared Geometry* configuration of CGLUT, the mean and covariance values are shared across all styles. Consequently, the mean and covariance heads are replaced by a set of globally learnable parameters, similar to the single-GLUT setting, while only the opacities and color transformations are dynamically generated.



(a) Shared encoder (b) Mean head

Figure 6: Architecture of the Gaussian Parameter Generator. A style embedding is first projected through a shared MLP encoder to extract common latent features. Subsequently, the representation is branched into task-specific heads to predict different parameters of the Gaussian primitives.

Scalability of CGLUT. The output dimensionality of the CGLUT generator does not grow with the number of styles. In each forward pass, the generator maps a single style embedding $\mathbf{e}_l$ to the parameters of one GLUT instance, which has $22N + 12$ values, or $13N + 12$ values per style under shared geometry. This dimensionality is the same whether CGLUT is trained on 75 or 225 styles. Adding styles increases the number of training samples and style embeddings, but each prediction remains a fixed-size regression. Consequently, the model size also grows slowly. Our largest configuration, CGLUT-64-Large trained on 225 styles, has 338K parameters in total, of which 323.6K belong to the shared generator and the remaining 14.4K to the 64-dimensional style embeddings. Each additional style therefore adds only one 64-dimensional embedding, increasing the total parameter count by just 64.

B.3 CUDA Acceleration of GLUT

To accelerate inference, we implement a custom CUDA kernel for the GLUT forward pass. The kernel fuses the Gaussian weight computation, local color transformation, and weighted color blending into a single GPU kernel, eliminating intermediate memory reads and writes that would otherwise dominate the runtime. The thread organization assigns one pixel per thread along the first dimension and distributes the N Gaussian primitives across threads along the second dimension within each thread block. The block dimensions are set to $(32, 32)$, aligning the first dimension with the warp size of modern NVIDIA GPUs.

Gaussian parameters shared across all pixels are precomputed once per forward pass and cached in registers and shared memory, avoiding redundant recomputation for each pixel. For training, the standard PyTorch autograd path is retained to preserve gradient flow. The custom CUDA kernel is activated exclusively during inference.

B.4 Classical Baselines

All baselines in Table 1 are fitted on the training colors of the 75-LUT and 225-LUT Hald datasets and evaluated on the held-out test colors, following the protocol in Section 4.1. Except for root-polynomial regression [11], all baselines are configured to match the 716-parameter budget of GLUT-32. The baselines are divided into two groups. The learned baselines, namely the RBF network [27] and GLLiM-GD [9], are trained end-to-end with GLUT's pipeline. They share the objective $\mathcal{L}_{\text{rec}} + \lambda_{hc}\mathcal{L}_{hc}$, without the sparsity regularizer, as these models do not involve opacity parameters. The remaining baselines are fitted in closed form or using their official implementations. The grid-based [12, 17] and keypoint-based baselines [38] require a reference LUT defined on a complete regular grid, which we obtain from the training Hald image covering the full 128^3 RGB grid.

Root-polynomial regression [11]. This model maps each color through a fixed global feature expansion followed by a linear transformation. We use the second-order expansion $(r, g, b, \sqrt{rg}, \sqrt{gb}, \sqrt{rb})$ without an intercept, following the original formulation, together with a 3×6 matrix, giving 18 parameters. The matrix is fitted by least squares over all training colors. We do not match the budget for this baseline. It is a single global mapping originally designed for device characterization, so it has no mechanism to allocate capacity locally.

RBF network [27]. We use a normalized RBF network with isotropic Gaussian kernels and constant outputs:

$$f(\mathbf{x}) = \sum_i \frac{\varphi_i(\mathbf{x})}{\sum_j \varphi_j(\mathbf{x})}\, \mathbf{c}_i + \mathbf{c}_0, \qquad \varphi_i(\mathbf{x}) = \exp\Big(- \frac{\|\mathbf{x} - \boldsymbol{\mu}_i\|^2}{2\sigma_i^2} \Big). \tag{13}$$

Each unit stores a center $\boldsymbol{\mu}_i \in \mathbb{R}^3$, a bandwidth $\sigma_i \in [0, 1]$, and an output color $\mathbf{c}_i \in \mathbb{R}^3$, which amounts to 7 parameters, and a shared bias $\mathbf{c}_0 \in \mathbb{R}^3$ adds 3 parameters. We use 102 units, yielding 717 parameters to approximately match the budget of GLUT-32. Compared with GLUT, this model uses constant rather than affine local transformations, isotropic rather than anisotropic Gaussian kernels, and a shared bias rather than a global affine transformation. It has no opacity parameters. The centers are initialized on the same regular grid as GLUT, and each output color is initialized to its corresponding center, making the initial mapping close to the identity.

GLLiM (GD/EM) [9]. GLLiM models the mapping as a Gaussian-gated mixture of local affine experts:

$$f(\mathbf{x}) = \sum_k r_k(\mathbf{x})\, (\mathbf{A}_k \mathbf{x} + \mathbf{b}_k), \qquad r_k(\mathbf{x}) = \frac{\pi_k\, \mathcal{N}(\mathbf{x}; \mathbf{c}_k, \boldsymbol{\Gamma}_k)}{\sum_j \pi_j\, \mathcal{N}(\mathbf{x}; \mathbf{c}_j, \boldsymbol{\Gamma}_j)}. \tag{14}$$

Each component has 22 parameters: a mean, a full covariance, a mixing weight, and an affine expert, which equals the per-primitive count of GLUT. GLLiM differs from GLUT in three aspects: its mixing weights lie on the probability simplex, it has no global affine term, and it does not use the sparsity regularizer. We use 33 components to match the budget, giving 726 parameters. Both variants use the same initialization as GLUT, with means on a regular grid, isotropic covariances with scale $\sigma = 0.15$, uniform mixing weights, and identity affine experts. **GLLiM (GD)** trains this predictor end-to-end with our training pipeline. It parameterizes each covariance by its Cholesky factor and the mixing weights by a softmax over learnable logits. **GLLiM (EM)** fits the generative model of [9] using expectation-maximization with closed-form updates. We run 500 EM iterations on all training color pairs.

Keypoint diffusion [38]. We use the official G'MIC implementation of Tschumperlé *et al*. [38]. Compression iteratively inserts keypoints where the reconstruction error is largest, and decompression reconstructs a dense LUT from the keypoints by multi-scale anisotropic diffusion. Each keypoint stores an input position and an output color, which amounts to 6 parameters. We cap the number of keypoints at 120, giving 720 parameters. We reconstruct the LUT at the training resolution of 128^3 in sRGB space and query it with trilinear interpolation.

Sparse grid LUTs [12, 17]. The largest regular grid within the budget of GLUT-32 has 6^3 nodes, giving $6^3 \times 3 = 648$ parameters. We obtain the node values by downsampling the reference LUT to this grid and query the grid with two interpolation schemes. Tetrahedral interpolation [17] splits each cell into six tetrahedra that share its main diagonal and interpolates linearly within the tetrahedron containing the query, which yields a continuous, piecewise-linear mapping. Tricubic interpolation [12] fits a tensor-product cubic B-spline that passes through the node values, which yields a twice continuously differentiable mapping. We do not further optimize the node values in this comparison. Fitting them to the training colors, as in lattice regression [13], improves the PSNR on the 75-LUT dataset to 37.79 dB for tetrahedral and 37.40 dB for tricubic interpolation, which remains below that of GLUT-32.

C Additional Experiment Results

C.1 Single-LUT Results

C.1.1 Single LUT Fitting

Table 5 compares the performance of our proposed GLUT against NILUT on 225-LUT Hald images and 100 natural images from the MIT5K dataset [4]. GLUT achieves superior PSNR and ΔE while maintaining a significantly smaller parameter footprint and lower computational cost. Notably, the color fidelity on this set is higher than that observed on the 75-LUT dataset (Table 2 in the main paper). This is because the LUTs in the 75-LUT set feature a higher resolution (64^3) than those in the 225-LUT set (32^3 or 33^3). Higher-resolution LUTs can represent finer color variations, potentially making accurate reconstruction more challenging.

Table 5: Quantitative comparison of single-LUT modeling on Hald images and natural images (225 LUTs).

Method	Hald images			Natural images					#Params	GFLOPs
	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	PSNR↑	SSIM↑	LPIPS↓	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$		
NILUT (64×2) [8]	42.49	0.76	1.45	41.98	0.987	0.0109	2.14	2.81	8,771	4.60
NILUT (128×2) [8]	44.77	0.57	1.08	44.21	0.992	0.0066	1.61	2.04	33,923	17.78
GLUT-16	46.69	0.45	0.88	46.23	0.994	0.0037	1.06	1.44	364	0.25
GLUT-32	*50.42*	*0.30*	*0.58*	*48.74*	*0.997*	*0.0020*	*0.75*	*0.99*	716	0.49
GLUT-64	**52.73**	**0.22**	**0.43**	**50.56**	**0.998**	**0.0011**	**0.58**	**0.76**	1,420	0.98

Figure 7 and Figure 16 show the qualitative comparison between images retouched with the same color styles with NILUT and GLUT. As the number of Gaussian primitives N increases, the model's ability to capture complex color mappings improves consistently, leading to lower reconstruction errors. For both NILUT and GLUT, the residual errors are mainly concentrated in darker intensity ranges, such as shadowed regions. This suggests that representing high-contrast transitions in low-luminance areas remains a common challenge for both implicit and explicit neural color representations. The last columns of Figure 7 and Figure 16 also visualize the GLUT-64 representation alongside the ground-truth LUT. Notably, the Gaussian distribution exhibits a structural correspondence with the target LUT, with more Gaussians located in regions associated with larger color shifts.

C.1.2 Comparison with INRetouch

Since INRetouch [10] primarily targets preset-based color style transfer, its retouched outputs involve spatially varying transformations, where identical input RGB values can be mapped to different output values depending on their spatial coordinates. In contrast, our GLUT focuses on globally consistent color transformations. Similar to our single-LUT model GLUT, each INRetouch model is trained specifically for a single preset. Given that INRetouch's architecture is designed to capture contextual information from natural images, training it on Hald images, which are merely tabular representations of the RGB color space, would be unsuitable. To ensure an appropriate comparison, we train both models using a single image pair processed by a specific LUT and subsequently evaluate them on 100 test images from MIT5K [4].

As shown in Table 6, our GLUT-32 consistently outperforms INRetouch across all metrics, including PSNR, SSIM, and ΔE_{76}, on seven different LUTs. Both models achieve lower performance when trained on a single image pair than when trained using a Hald image. This performance gap stems from the fact that natural images typically do not span the entire RGB color space. Consequently, the model is only supervised on a sparse subset of colors present in the scene, leading to less robust generalization in unobserved color regions compared to the exhaustive supervision provided by a Hald image.

C.1.3 Runtime and Implementation Efficiency

We compare the runtime of GLUT with traditional grid-based LUTs and neural LUT representations under different implementations.

Traditional LUTs. Table 7 reports FPS of a grid-based 33^3 3D LUT using trilinear interpolation under three implementations. GPU texture units achieve the highest throughput across all image

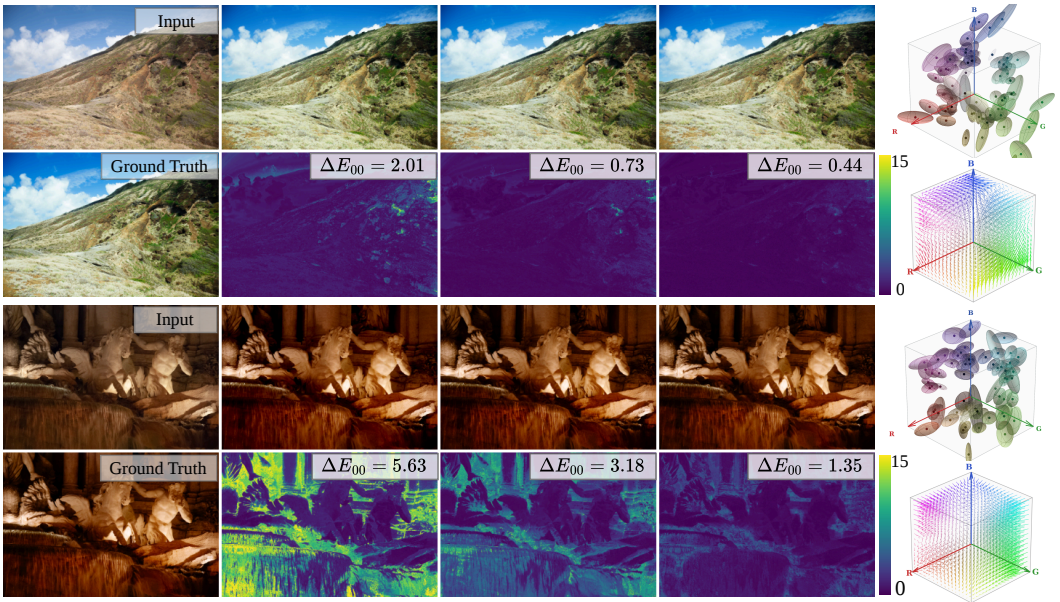


Figure 7: Qualitative results on images from the MIT5K [4]. The first column presents the input and the retouched image with a LUT. The top row shows the retouched version of the input image of NILUT (128×2) [8], GLUT-16, and GLUT-64. Under each image, we present the ΔE_{00} error map. The last column shows the visualization of our GLUT-64 representation and its corresponding LUT. Each Gaussian primitive is visualized using its learned mean values.

Table 6: Quantitative comparison of single-LUT modeling training with a single natural image pair.

Metric	Method	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	LUT7	Average
PSNR↑	INRetouch	34.91	33.32	34.15	40.16	33.00	35.81	30.15	34.50
	GLUT-32	**39.57**	**36.55**	**38.63**	**41.18**	**39.47**	**44.53**	**35.78**	**39.39**
SSIM↑	INRetouch	0.967	0.967	0.960	0.977	0.954	0.977	0.916	0.960
	GLUT-32	**0.994**	**0.991**	**0.991**	**0.989**	**0.992**	**0.997**	**0.980**	**0.991**
ΔE_{76} ↓	INRetouch	4.120	5.096	5.464	2.643	5.948	3.625	7.016	4.845
	GLUT-32	**2.190**	**2.360**	**2.640**	**1.890**	**3.570**	**1.430**	**2.530**	**2.370**

resolutions, making fixed-function texture sampling highly efficient for a precomputed static LUT. However, texture units are specialized hardware in the GPU graphics pipeline and are generally unavailable on neural processing units (NPUs), which primarily target tensor and matrix operations.

Neural LUTs. Table 7 further compares NILUT and GLUT using both standard PyTorch and optimized implementations. For NILUT, we use the fully fused MLP implementation from tiny-cuda-nn [28], which is specifically optimized for small coordinate-based MLPs such as NILUT (128×2). For GLUT, we implement a custom fused CUDA kernel that evaluates all Gaussian primitives in a single kernel launch. With standard PyTorch implementations, GLUT is slower than NILUT despite its compact representation. This is largely due to implementation overhead: NILUT primarily relies on large matrix multiplications, which are efficiently handled by optimized CUDA libraries, whereas GLUT performs multiple small operations for each Gaussian primitive, resulting in frequent kernel launches and intermediate memory accesses. Fusing these operations into a single CUDA kernel yields an approximately 80× speedup over the PyTorch implementation. With optimized implementations for both methods, GLUT achieves approximately 8× higher throughput than NILUT across all tested resolutions. These results indicate that the runtime of the standard GLUT implementation is dominated by execution overhead from unfused operations, and that GLUT's structured computation is well suited to kernel fusion.

In Table 4 in the main paper, we only report CNILUT and ENNELUT with their PyTorch implementations, since tiny-cuda-nn gives CNILUT no measurable speedup in our tests. ENNELUT injects its

Table 7: Runtime comparison of traditional and neural LUT representations under different implementations. FPS is reported at different image resolutions.

Method	Backend	512^2	HD	4K
3D LUT	CPU Numba	3489	759	148
3D LUT	CUDA PyTorch	5936	1025	265
3D LUT	GPU texture unit	26673	5577	1510
NILUT(128×2) [8]	CUDA PyTorch	295	39	8
NILUT(128×2) [8]	tiny-cuda-nn	822	111	28
GLUT-32	CUDA PyTorch	79	10	3
GLUT-32	Fused CUDA kernel	6170	891	226

condition inside each block, uses the LipSwish activation, and has hidden layers of varying width, none of which the fully fused MLP of tiny-cuda-nn supports.

C.2 Multiple-LUT Results

C.2.1 CGLUT Results on Natural Images

Table 8 presents the performance evaluation on 100 test images from the MIT5K dataset [4]. Our CGLUT consistently outperforms CNILUT [8] and ENNELUT [46] across both 7-LUT, 75-LUT, and 225-LUT configurations. The testing results on natural images align with the results on Hald images shown in Table 3 in the main paper.

Table 8: Quantitative comparison of multiple-LUT modeling on the natural images from MIT5K [4].

Method	7 LUTs					75 LUTs					225 LUTs				
	PSNR↑	SSIM↑	LPIPS↓	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	PSNR↑	SSIM↑	LPIPS↓	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	PSNR↑	SSIM↑	LPIPS↓	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$
CNILUT (128×3) [8]	40.12	0.995	0.019	1.74	2.21	39.78	0.987	0.027	2.45	2.95	38.93	0.984	0.028	2.77	3.58
CNILUT (256×3) [8]	44.24	*0.998*	0.011	1.18	1.48	42.75	0.992	0.020	1.84	2.16	41.98	0.990	0.019	2.07	2.65
ENNELUT (ΔE) [46]	45.83	*0.998*	0.008	0.79	1.03	43.23	0.993	0.016	1.26	1.52	40.50	0.989	0.020	2.02	2.61
ENNELUT (L_2) [46]	47.93	*0.998*	0.008	0.79	0.96	43.43	*0.994*	0.018	1.52	1.76	40.47	0.989	0.024	2.41	3.04
CGLUT-32 (Small)	46.42	*0.998*	0.007	0.70	0.92	46.61	**0.998**	*0.011*	0.88	1.07	46.52	0.995	0.009	1.03	1.39
CGLUT-32 (Large)	48.31	*0.998*	0.005	0.58	0.76	47.18	**0.998**	*0.011*	0.82	0.99	*48.74*	*0.997*	*0.006*	*0.76*	*1.01*
CGLUT-64 (Small)	*48.84*	*0.998*	*0.004*	*0.55*	*0.72*	*48.38*	**0.998**	**0.010**	*0.73*	*0.87*	48.53	*0.997*	*0.006*	0.78	1.03
CGLUT-64 (Large)	**49.84**	**0.999**	**0.003**	**0.45**	**0.59**	**48.69**	**0.998**	**0.010**	**0.70**	**0.84**	**50.61**	**0.998**	**0.004**	**0.57**	**0.75**

C.2.2 Blending LUTs

Table 9 presents the numerical blending results for all possible pairwise LUT combinations under the 7-LUT. Specifically, we evaluate $C_7^2 = 21$ unique style pairs, with metrics averaged over 100 test images from the MIT-Adobe FiveK dataset [4].

Table 9 illustrates a common trend across all methods, where blending qualities are better when alpha is near 0 or 1. Among the models, CGLUT with fully generated parameters (Full) achieves better performance than CNILUT and ENNELUT. Notably, using CGLUT with shared geometry (Shared Geo.) further enhances the results. This demonstrates that our Gaussian representation forms a more linear and robust latent manifold for color styles, enabling smooth manipulation between disparate color profiles. It is worth noting that no additional constraints were applied to optimize blending during the training of all models. Thus, these results reflect the inherent blending capabilities of different LUT representations.

Figure 15 compares the LUT blending results between CNILUT [8], ENNELUT [46], and two setups of our CGLUT (*Full Generation* and *Shared Geometry*). The visualization reveals that CNILUT tends to introduce undesired color shifts that deviate from the two anchor styles. In contrast, ENNELUT and CGLUT produce smoother transitions while better preserving the characteristics of the anchor styles.

Regarding implicit models, we hypothesize that ENNELUT achieves better blending results compared to CNILUT because it injects the LUT index into every residual block of the network. In contrast, CNILUT only provides the index to the initial layer, which limits the network's ability to leverage index information. Unlike these implicit approaches, CGLUT explicitly generates Gaussian

Table 9: Quantitative comparison of blending with different α on the 100 natural images from MIT5K [4].

Metric	Method	0	0.2	0.4	0.6	0.8	1
PSNR↑	CNILUT (128×3)	39.97	26.80	23.37	23.99	27.71	40.27
	CNILUT (256×3)	44.30	29.02	26.06	25.92	29.31	44.17
	ENNELUT (L_2)	*48.29*	33.50	30.29	30.62	34.35	*47.56*
	CGLUT-32L (Full)	**48.67**	*35.44*	*31.16*	*31.33*	*34.64*	**47.95**
	CGLUT-32L (Shared Geo.)	47.36	**38.46**	**34.67**	**34.47**	**37.60**	46.18
$\Delta E_{00}\downarrow$	CNILUT (128×3)	1.87	7.88	10.70	10.13	7.13	1.15
	CNILUT (256×3)	1.21	7.27	9.72	9.69	7.11	0.80
	ENNELUT (L_2)	0.81	3.61	5.14	4.89	3.14	0.77
	CGLUT-32L (Full)	**0.59**	*3.08*	*4.48*	*4.70*	*3.23*	**0.56**
	CGLUT-32L (Shared Geo.)	*0.66*	**2.11**	**3.19**	**2.96**	**1.94**	*0.68*
$\Delta E_{76}\downarrow$	CNILUT (128×3)	2.40	10.87	14.91	13.72	9.17	2.02
	CNILUT (256×3)	1.50	9.70	12.86	12.78	9.16	1.45
	ENNELUT (L_2)	0.99	4.38	6.48	6.22	3.93	0.93
	CGLUT-32L (Full)	**0.79**	*4.03*	*5.86*	*6.03*	*4.06*	**0.73**
	CGLUT-32L (Shared Geo.)	*0.86*	**2.62**	**3.93**	**3.69**	**2.46**	*0.87*
LPIPS↓	CNILUT (128×3)	0.015	0.104	0.166	0.154	0.098	0.010
	CNILUT (256×3)	0.008	0.086	0.130	0.131	0.086	0.007
	ENNELUT (L_2)	*0.004*	0.036	0.066	0.063	*0.035*	0.012
	CGLUT-32L (Full)	**0.003**	*0.026*	*0.049*	*0.054*	*0.035*	*0.008*
	CGLUT-32L (Shared Geo.)	*0.004*	**0.011**	**0.025**	**0.024**	**0.014**	**0.007**

parameters to construct each LUT. Interpolating the style embeddings therefore provides a direct mechanism for blending the corresponding LUT representations.

Table 10: Quantitative comparison of CGLUT under *Shared Geometry* (†) and *Full Generation*.

Method	7 LUTs					75 LUTs					225 LUTs				
	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs
CGLUT-32† (Small)	49.35	0.25	0.49	58K	0.49	43.55	0.49	0.95	62K	0.49	41.97	0.69	1.38	72K	0.49
CGLUT-64† (Small)	52.02	0.19	0.36	85K	0.98	46.89	0.35	0.65	89K	0.98	44.81	0.49	0.99	99K	0.98
CGLUT-32† (Large)	49.58	0.24	0.48	163K	0.49	43.79	0.48	0.93	168K	0.49	44.02	0.56	1.12	177K	0.49
CGLUT-64† (Large)	52.62	0.18	0.35	217K	0.98	47.13	0.34	0.64	222K	0.98	47.54	0.37	0.74	231K	0.98
CGLUT-32 (Small)	50.76	0.21	0.41	84K	0.49	45.66	0.39	0.73	89K	0.49	46.68	0.41	0.82	98K	0.49
CGLUT-64 (Small)	54.06	0.16	0.32	130K	0.98	48.43	0.30	0.55	135K	0.98	49.39	0.30	0.59	144K	0.98
CGLUT-32 (Large)	53.55	0.17	0.34	233K	0.49	46.69	0.36	0.66	238K	0.49	49.84	0.30	0.58	247K	0.49
CGLUT-64 (Large)	55.10	0.14	0.29	324K	0.98	49.37	0.28	0.51	328K	0.98	52.41	0.22	0.43	338K	0.98

Table 10 compares two CGLUT configurations for multi-LUT fitting: *Shared Geometry* and *Full Generation*. While shared geometry achieves better blending quality with fewer parameters, its multi-LUT fitting accuracy is lower than that of the *full generation* model. This is because sharing spatial attributes across all Gaussian primitives limits the model's expressiveness. Notably, this decline in LUT reconstruction quality becomes more pronounced as the number of LUTs increases. Therefore, for applications where blending is not required, we recommend the *full generation* setting to leverage its more flexible Gaussian representations.

C.3 LUT Editing Results

C.3.1 Additional GLUT Editing Results

Figure 8 presents additional editing results under various configurations. Specifically, we evaluate the impact of different color constraints (Rows 1–2), the number of edited primitives K (Rows 3–4), and the manipulation strength s (Rows 5–6) on a single input image. The results demonstrate that our method consistently produces natural-looking color transitions. Since K determines the number of primitives involved in the update, a larger K spreads the edit across a broader region of the color manifold, thereby affecting a wider range of colors while minimizing abrupt shifts in specific hues. Similarly, the strength s controls the degree to which the source color $\mathbf{c}_{\text{in}}$ is displaced toward the target $\mathbf{c}_{\text{out}}$. A higher s yields a result closer to the target color. Furthermore, applying edits iteratively on the output image allows for a progressively more pronounced and customized effect.

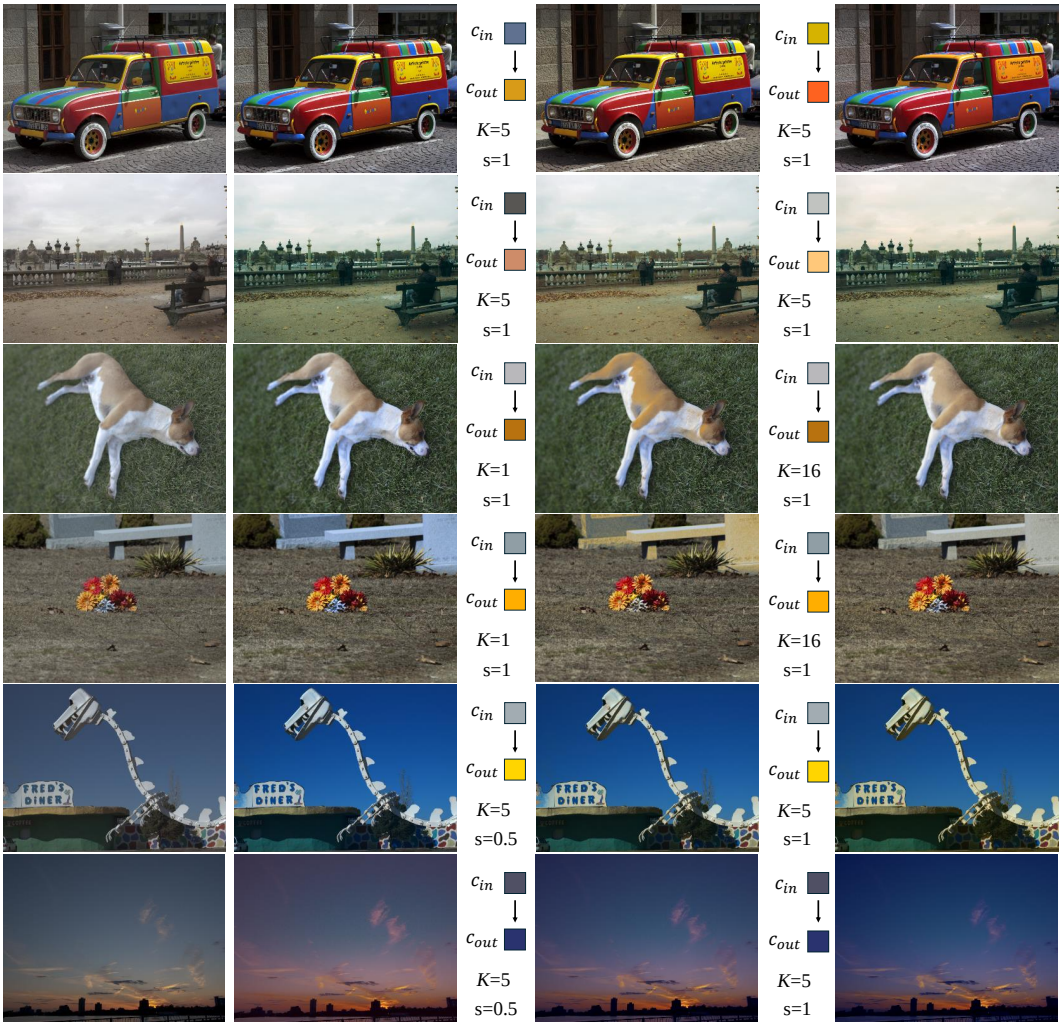


Figure 8: Qualitative results of editing GLUT representations. Each row shows an input image, a retouched image by GLUT, and two edited versions obtained with different color constraints $\mathbf{c} = (\mathbf{c}_{\text{in}}, \mathbf{c}_{\text{out}})$, number of Gaussians to edit K, and strength s.

C.3.2 Locality evaluation

To evaluate locality, we compute the color change before and after each edit at every node of a regular 33^3 RGB grid, measured as $\Delta E_{00}\big(f_{\text{before}}(\mathbf{x}), f_{\text{after}}(\mathbf{x})\big)$. We pool the nodes across all edits and divide them into 20 bins according to their Euclidean distance $\|\mathbf{x} - \mathbf{c}_{\text{in}}\|_2$, which ranges from 0 to $\sqrt{3} \approx 1.73$. For each bin, we report the mean color change. We define the leakage radius as the center of the first bin at which the mean color change falls below one JND.

Comparison with fine-tuning an implicit LUT. We compare GLUT editing with fine-tuning NILUT [8] under identical editing constraints. For each of the seven LUTs, we use a GLUT-32 and a NILUT (128×2) trained on the same Hald image pair. For each LUT, we sample 50 source colors $\mathbf{c}_{\text{in}}$ uniformly from the RGB cube. We obtain each target color $\mathbf{c}_{\text{out}}$ by perturbing the reference LUT output at $\mathbf{c}_{\text{in}}$ by 10–20 ΔE_{00} in a random direction in CIELAB, simulating a local color adjustment. Both models are thus evaluated on the same set of 350 edits.

For GLUT, we apply each edit with $K = 1$ and $s = 1$. A single application may move $f(\mathbf{c}_{\text{in}})$ only partway toward $\mathbf{c}_{\text{out}}$, so we repeat the edit until the ΔE_{00} between $f(\mathbf{c}_{\text{in}})$ and $\mathbf{c}_{\text{out}}$ falls below one JND. This requires an average of about 10 repetitions and at most 30.

For NILUT, we fine-tune all network weights m using Adam with a learning rate of 5×10^{-4} and the following loss:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}\sim\mathcal{U}([0,1]^3)} \|m(\mathbf{x}) - m_0(\mathbf{x})\|_1 + \lambda \|m(\mathbf{c}_{\text{in}}) - \mathbf{c}_{\text{out}}\|_1, \qquad \lambda = 10, \tag{15}$$

where m_0 is a frozen copy of the pretrained network. The first term encourages the original mapping to be preserved and is estimated using 4096 newly sampled uniform RGB values at each step. The second term encourages the output at $\mathbf{c}_{\text{in}}$ to match the target. We stop fine-tuning when the ΔE_{00} at $\mathbf{c}_{\text{in}}$ falls below 1.0 or after 8000 steps. Within this budget, NILUT fails to reach the target for 35 of the 350 edits, whereas GLUT reaches it for all edits. We exclude these 35 edits from both methods so that the locality analysis compares only edits successfully completed by both.

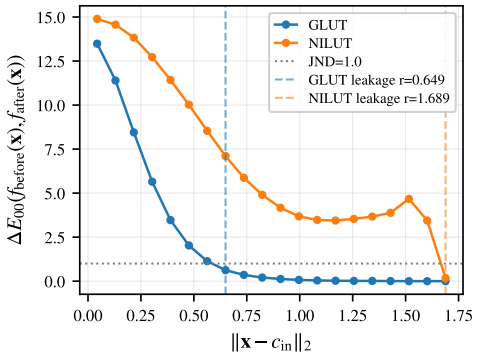


Figure 9: Edit locality of GLUT and NILUT. Mean ΔE_{00} change as a function of the RGB distance to the source color $\mathbf{c}_{\text{in}}$, pooled over 315 edits on 7 LUTs. The dashed line marks one JND.

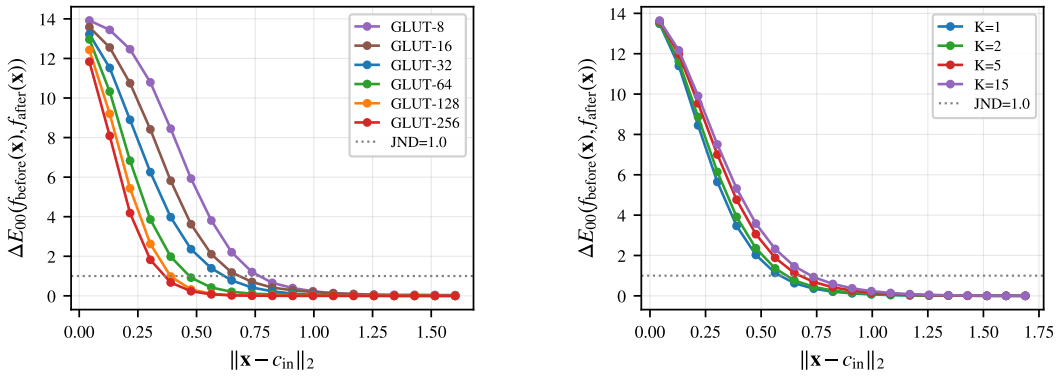
Figure 9 shows the resulting curves for GLUT and NILUT. For both methods, the mean color change decreases with distance from $\mathbf{c}_{\text{in}}$, but it decays much faster for GLUT. The leakage radius is 0.65 for GLUT and 1.69 for NILUT. For NILUT, the mean change remains above 10 ΔE_{00} across most of the RGB cube, whereas for GLUT it approaches zero at a distance of approximately 1.0. This difference is consistent with NILUT updating over 80% of its network weights, indicating that the edit is not localized in parameter space and offers no explicit control over its spatial extent in RGB space.

This difference arises even though NILUT is fine-tuned with an explicit regularization term to preserve the original mapping, whereas GLUT editing uses no such term. The difference stems from the two representations. A GLUT edit modifies only the biases of the K primitives with the largest weights at $\mathbf{c}_{\text{in}}$, so its effect is governed by their influence over RGB space. NILUT lacks this explicit local structure, and fine-tuning its fully connected weights can alter the mapping far from $\mathbf{c}_{\text{in}}$.

Influence of N and K on Editing Locality. Figure 10 shows how the locality of GLUT editing depends on the number of Gaussians N and the number of edited primitives K. In Figure 10a, we fit GLUTs with different values of N to the same LUT and edit them with $K = 1$. As N increases, the curves decay faster, and the leakage radius decreases. This is because, with more Gaussians, each primitive generally covers a smaller region of the RGB cube, causing its influence to decay over a shorter distance and keeping the edit more localized. In Figure 10b, we edit a single GLUT with different values of K. Smaller values of K produce steeper decay, although the curves remain close across different values of K. This is because the additional primitives selected at larger K have relatively small weights at $\mathbf{c}_{\text{in}}$ and therefore receive only a small share of the edit, resulting in only a modest change in its spatial extent.

Figure 11 and Figure 12 illustrate these trends for a single edit. We apply the edit with $s = 1$ and repeat it until ΔE_{00} between the current output at $f(\mathbf{c}_{\text{in}})$ and the target $\mathbf{c}_{\text{out}}$ falls below one JND, ensuring that all settings reach the same target. This experiment illustrates the effect of N and K under a common stopping criterion. In typical editing scenarios, users may stop after fewer iterations once the desired result is achieved. The top row shows the edited outputs, and the bottom row shows the corresponding ΔE_{00} between the edited and unedited outputs.

Figure 11 varies N with $K = 1$. We select a red color from the hat and turn it orange. We fit GLUT with $N = 16$, 32, and 64 to the same style and apply each model to the input image. As N increases, each primitive generally covers a smaller region of RGB space, making the edit more localized. With



(a) Number of Gaussians N

(b) Number of edited primitives K

Figure 10: Edit locality of GLUT. (a) GLUTs with different numbers of Gaussians fitted to the same LUT, edited with $K = 1$. (b) A single GLUT edited with different K.

GLUT-16, the edit also changes the sky, the white shirt, and the grass, whereas with GLUT-64, it is largely confined to the hats. However, this increased locality comes at a cost: the shaded parts of the hats retain their original red, while other parts turn yellow. Since GLUT operates solely on RGB values without accounting for semantic regions, it cannot ensure consistent color changes across the same object. Incorporating semantic constraints into the editing process is a promising direction for future work.

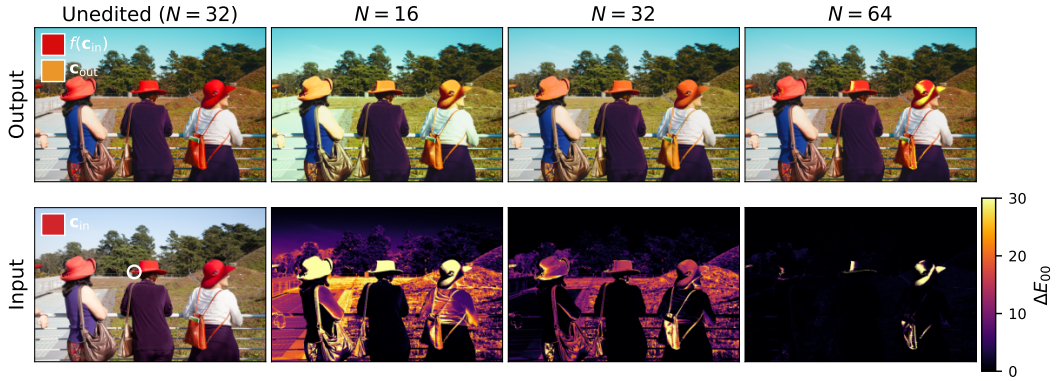


Figure 11: An editing example with different numbers of Gaussians N.

Figure 12 varies K with GLUT-32. We select a green-blue color from the sea and shift it toward ocean blue. As K increases, the edit affects more regions, including the sky. The results for $K = 5$ and $K = 10$ are almost identical because the additional primitives have small weights at $\mathbf{c}_{\text{in}}$ and therefore receive only a small share of the edit. Both observations are consistent with the locality curves in Figure 10b.

Locality on Overlapping Gaussians. We investigate whether editing becomes less effective in regions where multiple Gaussians have comparable influence. We define the ambiguity of a color $\mathbf{x}$ as $a(\mathbf{x}) = w_{(1)}(\mathbf{x})/w_{(2)}(\mathbf{x})$, the ratio of its largest to second-largest weight. Values close to one indicate substantial overlap, with multiple primitives contributing similarly, whereas large values indicate that a single primitive dominates. Using GLUT-32 trained on the 75-LUT dataset, we compute $a(\mathbf{x})$ on a 64^3 grid and sample 50 overlap points with $a(\mathbf{x}) \leq 1.5$ and 50 dominated points with $a(\mathbf{x}) \geq 10$. For each point, we specify a target $\mathbf{c}_{\text{out}}$ whose ΔE_{00} distance from $f(\mathbf{c}_{\text{in}})$ is between 10 and 20. We then apply edits with $K = 1$ and $s = 1$, repeating each edit until the error falls below one JND or 30 iterations are reached.

All 100 edits reach the error threshold within 30 iterations, indicating that overlap does not prevent successful editing under our experimental settings. However, overlap slows convergence: overlap

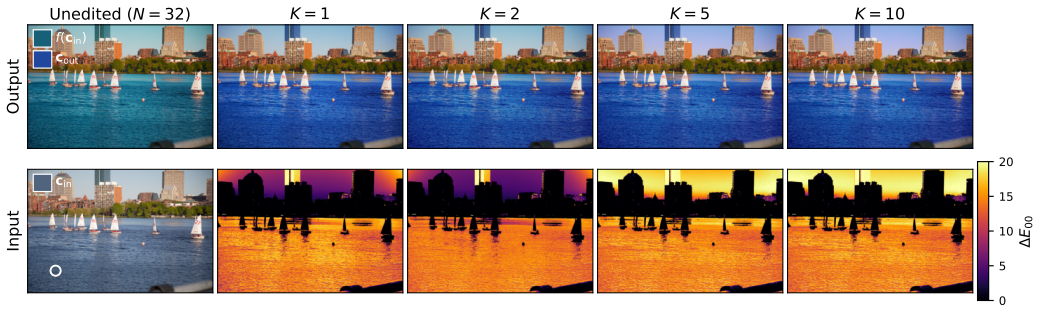


Figure 12: An editing example with different numbers of edited primitives K.

points require an average of 13.3 iterations, compared with 3.4 for dominated points. When the weights at $\mathbf{c}_{\text{in}}$ are more evenly distributed, each edit corrects a smaller fraction of the residual. Overlap also reduces locality. The mean leakage radius is 0.776 for overlap points and 0.584 for dominated points. This is consistent with the edit affecting multiple primitives, potentially extending its influence across the regions they cover.

C.4 Ablation Studies

We conduct several ablation studies to verify the proposed Gaussian-based LUT representation. We train the GLUT-32 model under different settings and test it on the 75-LUT Hald image dataset.

C.4.1 Number of Gaussian Primitives

Table 11 and Figure 13 present the performance of our GLUT across varying numbers of Gaussian primitives. As shown, the parameter size is linearly proportional to the number of Gaussians. While increasing the primitive count consistently improves performance, the growth rate of PSNR and the reduction in ΔE_{00} exhibit diminishing returns, suggesting an optimal balance between model complexity and reconstruction accuracy.

Table 11: Ablations on the number of Gaussian primitives.

#Gaussian	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params
8	37.01	1.068	2.119	188
16	41.50	0.636	1.223	364
32	45.47	0.414	0.770	716
64	48.42	0.310	0.560	1,420
128	50.31	0.263	0.464	2,828

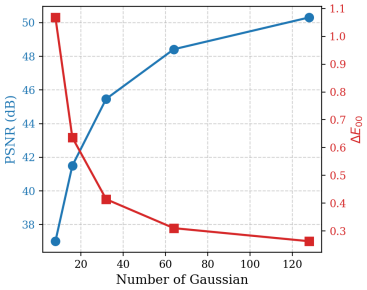


Figure 13: Quantitative results of GLUT with different numbers of Gaussians.

C.4.2 Local Color Transformation

Table 12a shows different formulations for the local color mapping function, including a learnable RGB value, an affine transformation (our default), and second- and third-order Spherical Harmonics (SH) (with $9 \times 3 = 27$ and $16 \times 3 = 48$ parameters). These variants introduce differences in model capacity and color accuracy. Generally, we observe that performance is positively correlated with parameter size. Despite the higher representational power of SH bases, we select the affine transformation as our final design, as it shows a better tradeoff between color fidelity and efficiency, and also provides a more robust and intuitive representation for our GLUT editing in Section 3.3.

Table 12: Ablation on Gaussian parameters.

Function	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params	GFLOPs
RGB	42.58	0.565	1.059	428	0.30
SH-2	45.67	0.405	0.740	1,196	0.73
SH-3	**46.78**	**0.360**	**0.651**	1,868	1.12
Affine	45.47	0.414	0.770	716	0.49

(a) Local color transformation functions.

Configuration	PSNR ↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$	#Params
w/o weight normalization	45.17	0.419	0.776	716
w/o Opacity (o_i)	45.43	0.415	0.774	684
w/o Global ($\mathbf{G}$, $\mathbf{g}$)	45.28	0.419	0.776	704
w/o Global w/o Residual	40.54	0.661	1.273	704
Full Model	**45.47**	**0.414**	**0.770**	716

(b) Opacity and Global transformation.

Table 13: Ablation on training settings.

$\mathcal{L}_{\text{rec}}$	$\mathcal{L}_{\text{hc}}$	$\mathcal{R}_{\text{sparse}}$	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$
✓			45.36	0.425	0.785
✓	✓		45.38	0.419	0.776
✓	✓	✓	**45.47**	**0.414**	**0.770**

(a) Loss functions.

Mining	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$
	45.13	0.422	0.793
✓	**45.47**	**0.414**	**0.770**

(b) Hard sample mining strategy.

Mean init	PSNR↑	$\Delta E_{00}\downarrow$	$\Delta E_{76}\downarrow$
Random	45.45	0.420	0.782
Uniform	**45.47**	**0.414**	**0.770**

(c) Mean value initialization.

C.4.3 Other Components

Table 12b summarizes ablations on other key parameters. To verify the impact of weight normalization, we replace the weights in Equation (2) by $w_i(\mathbf{x}) = p_i(\mathbf{x})\, o_i$. The results in the first row show a decrease in PSNR and larger color differences when normalization is removed. To investigate the impact of opacity, we consider a configuration excluding o_i, where the weighted activation in Equation (2) is replaced by a standard softmax-like normalization: $w_i(\mathbf{x}) = p_i(\mathbf{x})/(\sum_{j=1}^{N} p_j(\mathbf{x}) + \epsilon)$. Moreover, we evaluate the role of the global transformation branch by comparing two variants: a residual connection without global transform, defined as $f(\mathbf{x}) = \sum_{i=1}^{N} w_i(\mathbf{x}) f_i(\mathbf{x}) + \mathbf{x}$, and a pure local mapping that relies solely on weighted local transformations, $f(\mathbf{x}) = \sum_{i=1}^{N} w_i(\mathbf{x}) f_i(\mathbf{x})$, compared to Equation (5). The results underscore the effectiveness of residual learning. Specifically, predicting the color displacement relative to the input is significantly more efficient than directly regressing the absolute target color values. This allows the Gaussian primitives to focus on modeling the non-linear residuals. Furthermore, the learned opacity and global transformation also enhance the representation, resulting in higher color mapping fidelity.

C.4.4 Loss Terms

Table 13a shows the contribution of each loss component. Starting with the basic reconstruction loss $\mathcal{L}_{\text{rec}}$, we observe that the introduction of the hue-chroma loss $\mathcal{L}_{\text{hc}}$ effectively reduces color distortion, leading to lower ΔE_{00} and ΔE_{76} metrics. Furthermore, the inclusion of the sparsity regularization $\mathcal{R}_{\text{sparse}}$ yields the best overall performance on both PSNR and ΔE.

C.4.5 Hard Sample Mining

To investigate the effect of hard sample mining, we compare the baseline model with and without this strategy. As shown in Table 13b, applying hard sample mining brings consistent improvements across all metrics. These results demonstrate that focusing on challenging samples during training effectively guides the model to achieve better color fidelity.

C.4.6 Initialization

Table 13c compares the impact of random and uniform initialization on the Mean parameter. Since uniform initialization performs marginally better, we adopt it as our default configuration. Notably, these results also demonstrate that our GLUT can consistently converge to an optimal distribution of Gaussians regardless of the specific initialization strategy, highlighting the robustness of our optimization process.

D Interactive GLUT-based LUT editing tool

In the supplementary material, we provide an interactive demo for the LUT editing tool by implementing the GLUT editing method described in Section 3.3 of the main paper. The demo runs entirely in the browser (HTML/JavaScript, CPU-only, no external dependencies) in real time. Figure 14 shows a snapshot of the interactive LUT editing tool. A step-by-step guide is provided below, with each step referenced by the corresponding red number annotated in Figure 14.

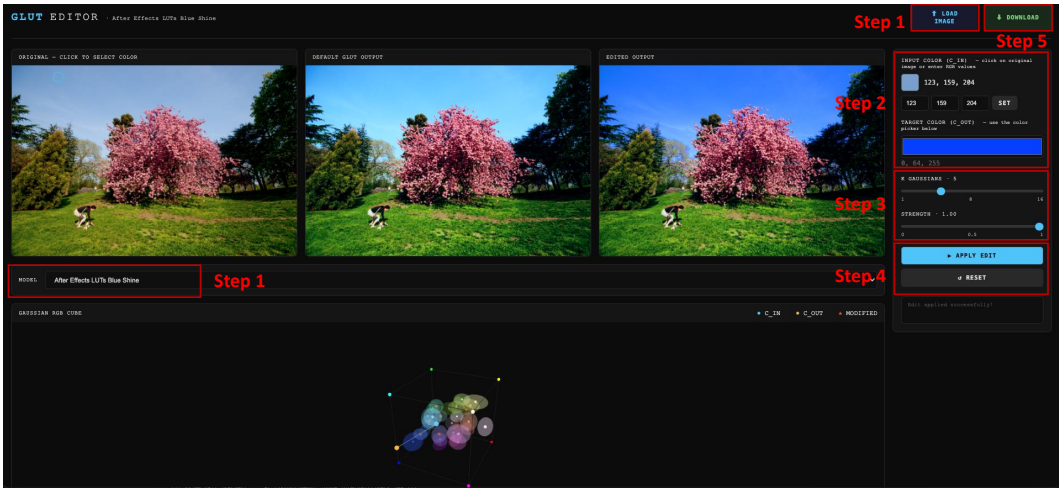


Figure 14: Snapshot of the interactive GLUT editing tool. Red numbers correspond to the operation steps.

The operation instructions of the tool are as follows:

Step 1: Load Image and GLUT Model. Load a source image by clicking the blue "LOAD IMAGE" button in the top-right corner. Select a pretrained GLUT model from the drop-down menu. The interface will display the original input, the retouched output, and the edited output in the "ORIGINAL", "DEFAULT GLUT OUTPUT", and "EDITED OUTPUT" panels, respectively. Note that by default the "EDITED OUTPUT" image is the same as the "DEFAULT GLUT OUTPUT". The selected model name (*e.g.*, "*After Effects LUTs Blue Shine*") will appear at the top. If no model is active, it displays "*No model loaded*". The "GAUSSIAN RGB CUBE" panel visualizes each Gaussian primitive using its learned mean RGB value. Primitives with means outside the standard RGB unit cube are visualized in gray.

Step 2: Define Source ($\mathbf{c}_{\text{in}}$) and Target ($\mathbf{c}_{\text{out}}$) Colors. There are two ways to specify the source color $\mathbf{c}_{\text{in}}$: (i) Directly click on the original image to select the pixel color, or (ii) Manually enter RGB values in the right-hand control panel. To define the target color $\mathbf{c}_{\text{out}}$, click the "TARGET COLOR" box and use the color picker. The selection is visualized in the RGB cube as a blue node ($\mathbf{c}_{\text{in}}$) and an orange node ($\mathbf{c}_{\text{out}}$), connected by a vector illustrating the transformation.

Step 3: Configure Parameters. Adjust the number of affected Gaussians (K) and the manipulation strength using the sliders on the right-hand panel. The parameter K controls the spatial extent of the edit: a larger K involves more Gaussian primitives in the update, thereby affecting a broader region of the color manifold. The Strength slider modulates the intensity of the transformation. Increasing this value pushes the source color $\mathbf{c}_{\text{in}}$ closer to the specified target $\mathbf{c}_{\text{out}}$, allowing for both subtle refinements and bold artistic changes.

Step 4: Apply and Reset Edits. Click "APPLY EDIT" to perform the localized transformation. The result will be displayed in the "EDITED OUTPUT" panel. At the same time, the centers of the modified Gaussian primitives will be highlighted as red stars in the "GAUSSIAN RGB CUBE" panel. Subsequent clicks will apply the edit incrementally. To revert all changes and return to the original GLUT-retouched version, press the "RESET" button.

Step 5: Export Results. Click the "DOWNLOAD" button in the top-right corner to save the edited image to the local device.

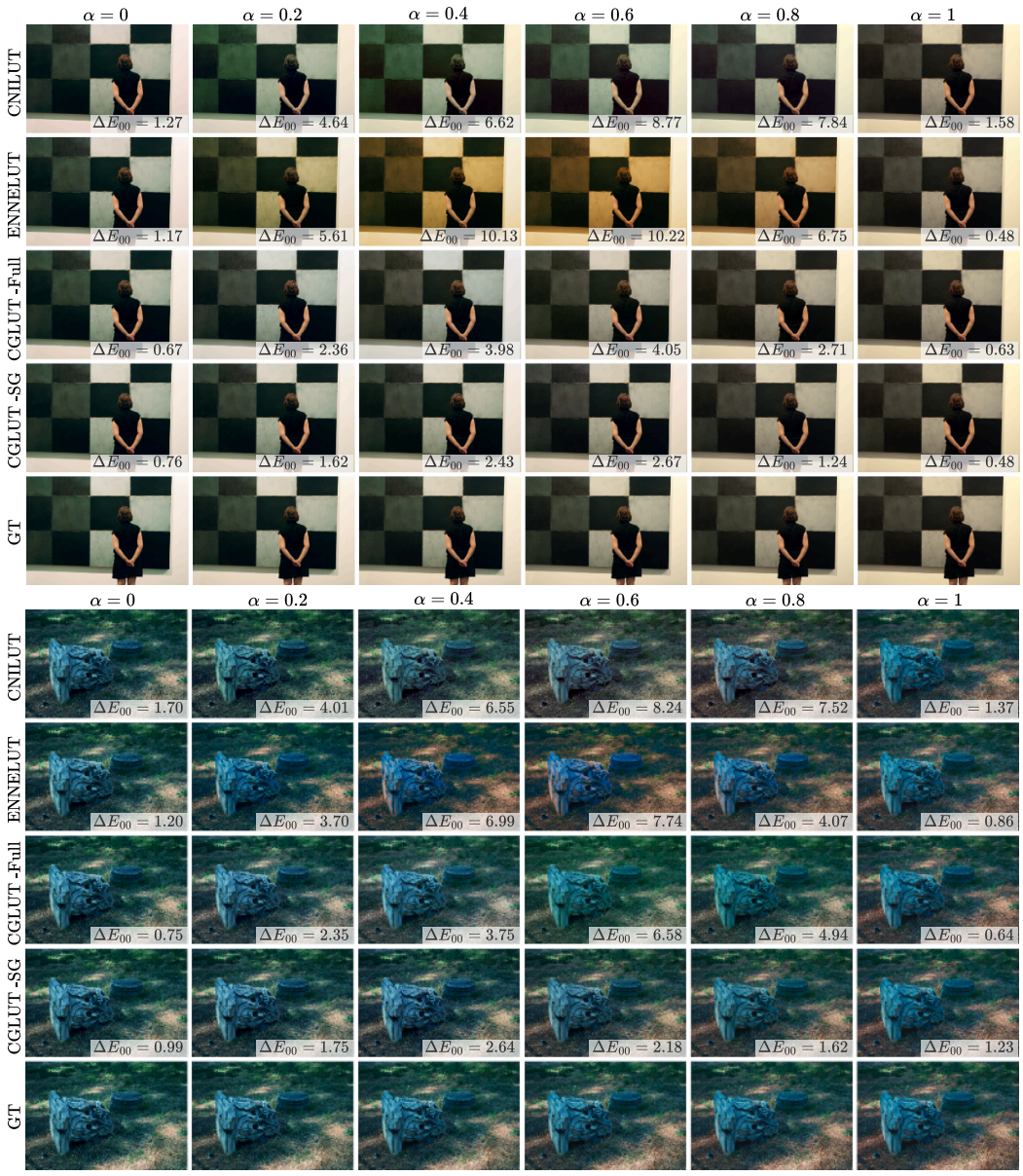


Figure 15: Qualitative comparison of style blending with different α. Rows 1–5 illustrate the results of CNILUT (256×3), ENNELUT (L_2), CGLUT-32 (Full generation), CGLUT-32 (Shared Geometry), and direct image-space blending (ground-truth), respectively.

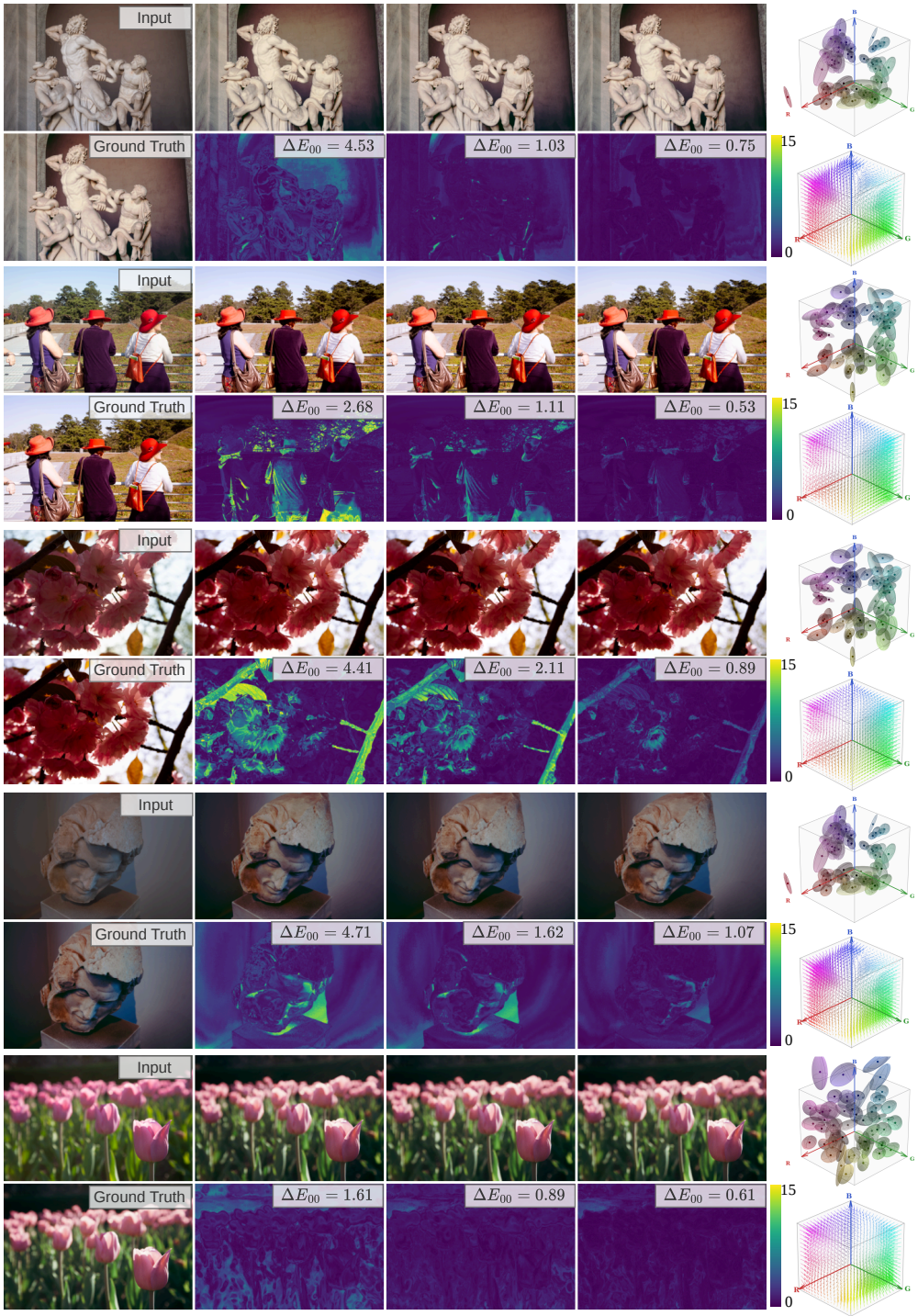


Figure 16: Qualitative results on images from the MIT5K [4]. The first column presents the input and the retouched image with a LUT. The top row shows the retouched version of the input image of NILUT(128×2) [8], GLUT-16, and GLUT-64. Under each image, we present the ΔE_{00} error map. The last column shows the visualization of our GLUT-64 representation and its corresponding LUT. Each Gaussian primitive is visualized using its learned mean value.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims are supported in the rest of the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section A of the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: No theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: All the required details are explained in Section 3, Section 4, and Section B of the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We release the full training and evaluation code, configuration files, and scripts to reproduce the main results, pretrained GLUT and CGLUT weights, and the interactive GLUT Editor at `https://github.com/CVC-Color/glut`. The datasets are publicly available; we provide instructions and preprocessing scripts to obtain and prepare them.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All details are explained in Section 4 of the main text and Section B of the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Each experiment involves optimizing GLUT for a specific set of LUTs, with metrics computed from the image outputs of the LUT reconstructions, giving equal weight to all pixels and all LUTs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All the information is presented in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We followed the code of ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Broader impacts are explained in Section A of the Appendix.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our model is limited to color transformations on existing images and does not perform content generation.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All creators are credited in the manuscript.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have released the code and models upon acceptance, with all the required documentation.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No crowdsourcing or experiments involving human participants were conducted.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No human subjects involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: No LLMs were used as part of the methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.