Abstract Rendering: Certified Rendering Under 3D Semantic Uncertainty

Chenxi Ji[*], **Yangge Li**[*], **Xiangru Zhong**[*], **Huan Zhang, Sayan Mitra**
University of Illinois Urbana-Champaign
{chenxij2, li213, xiangru4, huanz, mitras}@illinois.edu

Abstract

Rendering produces 2D images from 3D scene representations, yet how continuous variations in camera pose and scenes influence these images—and, consequently, downstream visual models—remains underexplored. We introduce **abstract rendering**, a framework that computes provable bounds on all images rendered under continuously varying camera poses and scenes. The resulting abstract image, expressed as a set of constraints over the image matrix, enables rigorous uncertainty propagation through downstream neural networks and thereby supports certification of model behavior under realistic 3D semantic perturbations, far beyond traditional pixel-level noise models. Our approach propagates camera pose uncertainty through each rendering step using efficient piecewise linear bounds, including custom abstractions for three rendering-specific operations—matrix inversion, sorting-based aggregation, and cumulative product summation—not supported by standard tools. Our implementation, ABSTRACTRENDER, targets two state-of-the-art photorealistic scene representations—3D Gaussian Splats and Neural Radiance Fields (NeRF)—and scales to complex scenes with up to 1M Gaussians. Our computed abstract images achieve up to 3% over-approximation error compared to sampling results (baseline). Through experiments on classification (ResNet), object detection (YOLO), and pose estimation (GATENet) tasks, we demonstrate that abstract rendering enables formal certification of downstream models under realistic 3D variations—an essential step toward safety-critical vision systems.

1 Introduction

Rendering produces 2D images from 3D scenes and underpins visual computing. Two prominent neural scene representations are Gaussian Splats [1] and Neural Radiance Fields (NeRFs) [2]. Gaussian Splats represent a scene as a collection of 3D Gaussians whose colors are blended after projection onto the image plane, whereas NeRFs encode scenes as neural networks mapping 3D positions and viewing directions to color and density, rendered via ray casting and volumetric integration. These methods exemplify rasterization- and ray-casting-based paradigms that achieve photorealistic reconstruction and novel-view synthesis [3]. However, rigorous analysis of how variations in camera pose or scene geometry affect the rendered outputs—and consequently the predictions of downstream models such as classifiers, object detectors, and pose estimators—remains limited, leaving the robustness of vision systems under realistic 3D perturbations largely unexplored.

Although formal verification has been extensively developed for standalone neural networks [4, 5], verifying rendering pipelines poses a fundamentally different challenge. A recent work [6] considers a version of this problem for simpler mesh-based scenes, but a general treatment remains open. Formal verification of rendering seeks to compute the set of *all* possible images generated under continuous variations in camera pose and scene. When coupled with downstream perception models this enables guarantees like "no misclassification occurs during camera panning within a specified range" or

[*]Equal contribution.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

"the relative pose error remains bounded within tolerance across viewpoints." Such guarantees are crucial for safety-critical applications [7] including aircraft auto-landing [8] and formation-flight[9]. Verifying rendering is challenging because ray-casting and rasterization involve neural components, volumetric integration, and nonsmooth operations such as sorting. Consequently, reasoning about rendering requires composing verification techniques across heterogeneous computational layers, a capability absent from existing frameworks.

In this paper, we introduce **Abstract Rendering**, a method for computing the set of all possible images rendered from a continuously varying range of camera poses looking at a scene with a range variations. This set of images, referred to as an **Abstract Image**, is compactly represented using constraints on the image matrix, such as interval or linear bounds. This computed abstract image can then be passed through an existing neural network verification tool [4] to verify models for visual tasks. Our framework is illustrated in Figure 1, with representative results in Figure 4. For example, consider a ResNet classifier downstream of a camera rotating 360° around an airplane in a 3D scene; our method certifies the subrange of viewing angles over which the airplane is classified correctly.

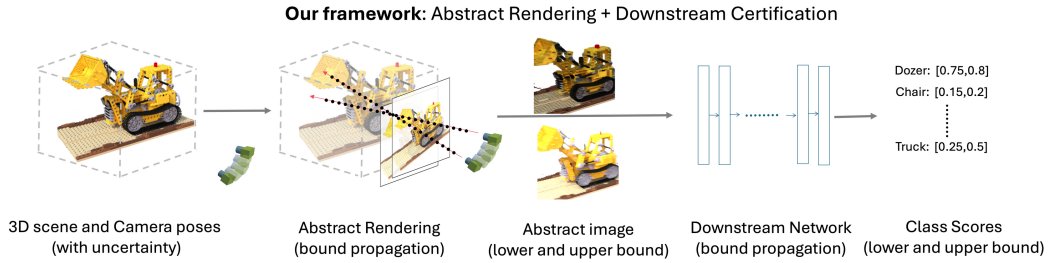

Figure 1: Pipeline of applying abstract rendering to certify downstream network. A range of camera pose uncertainty in a NeRF scene (Left) is propagated step by step through the rendering process (Left Mid) to produce abstract images (Mid). These abstract images are then passed through the downstream network layer by layer (Right Mid), yielding per-class output bounds (Right). For classification, if the lower bound of the Dozer class exceeds the upper bounds of all other classes, the prediction is certified under the given camera uncertainty.

Our abstract rendering algorithm applies **linear sets** to represent and propagate uncertainties from camera pose and scene parameters, through the rendering pipeline. By decomposing the rendering process into composition of basic operations, we develop linear bound propagation rules that maintain tight output bounds when transforming linear input sets. This compositional approach ultimately produces an abstract image that preserves uncertainty information. To optimize the trade-off between bound precision and computational efficiency, we adopt CROWN [10, 11] for handling standard operations like division and matrix multiplication.

However, certain operators in the rendering process lack native support. To address this, we develop novel linear bound propagation techniques for three rendering-specific operations: matrix inversion, sorting-based aggregation, and cumulative product summation. For matrix inversion required in computing projected 2D Gaussian probability densities, our Taylor series-based approximation produces tighter linear bounds than standard adjugate methods. Sorting-based aggregation, critical for occlusion determination between Gaussians, is reformulated as an equivalent indicator-based process to mitigate over-approximation caused by independently bounding sorted outputs while ignoring index mutual exclusivity. Cumulative product summation, used in opacity/occlusion-weighted value aggregation, employs an ordered strategy that significantly improves bound tightness (particularly beyond 1,000 terms) at moderate computational expense.

In summary, our key contributions are as follows. (1) We introduce the first abstract rendering algorithm to compute abstract images for scenes represented by Gaussian Splats and NeRFs under semantic variations in the 3D world. (2) We develop a unified mathematical and software framework for abstracting rendering pipelines, with novel techniques for matrix inversion, sorting-based aggregation, and cumulative product summation, extending beyond what is supported by existing tools like CROWN. (3) We integrate abstract rendering with neural network verification tool — CROWN, to certify visual tasks such as image classification, pose estimation, and object detection, enabling statements like: "Across camera viewpoints or scene variations, the target model consistently classifies, estimates, or detects object within specified error bounds."

2 Related Work

The only closely related work computes abstract images for triangular mesh scenes [6]. To handle camera-pose uncertainty, that method collects each pixel's RGB values into intervals over all possible viewpoints. In practice, this amounts to computing partial interval depth-images for each triangle and merging them using a depth-based union, resulting in an interval image that over-approximates all concrete images for that region of poses. This work considers only camera translations, not rotations. Our method works with Gaussian splatting or NeRF which involve more complex operations and are learnable.

Numerous methods exist in neural network verification [5, 12, 13, 14, 15, 16], including SMT-based approaches [17] and MIP-based techniques [18]. Among these, linear bound propagation has emerged as a prominent strategy, leveraging linear relaxations of nonlinear activations and propagating these bounds layer-wise to efficiently compute output guarantees [19, 11, 20]. Researchers have further advanced the field by introducing complex geometric abstractions such as polyhedra [21], zonotopes [22], and starsets [16, 14]. However, our work diverges conceptually from traditional neural network verification: rather than verifying neural networks themselves, we focus on verifying a broader algorithmic framework that embeds neural networks as components. Crucially, many operations encountered in this context, such as matrix inverse, sorting or cumulative product, are not handled by existing neural network verification tools, necessitating novel theoretical and technical innovations.

Adversarial robustness has been studied most deeply for pixel-space perturbations, where attacks [23, 24, 25, 26, 27] and certificates [11, 20, 14, 16, 28] are constrained to small l_p balls around the original image. Follow-up work broadened the threat model to 2D semantic transforms [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40], including global hue or brightness shifts, in-plane rotations, translations, elastic warps, but still treats the scene as a rigid 2D grid. Our method, instead, handles semantic, 3D perturbations that stem from changing the camera pose itself (e.g., orbiting or translating the camera around a fixed object), which jointly affect all pixels in a viewpoint-consistent manner.

Only a handful of papers explore this regime. Athalye [41] synthesizes 3D-printable adversarial objects that fool classifiers over many viewpoints, yet offer no formal guarantee of success across the entire pose distribution. Camera-Motion Smoothing (CMS) by Hu [42] provides the first certificates for small six-DoF camera perturbations via randomized smoothing, but it relies on dense point-cloud supervision and is restricted to narrow pose radii. In Hu's later work [43], he partitions pose space and smooths in the image domain to improve efficiency, yet still yields conservative bounds over only modest displacements. View-Invariant Adversarial Perturbations (VIAP) [44] craft attacks that remain effective across multiple viewpoints, but—as with Athalye et al.—provide no certification. Our approach is the first to deliver tight, end-to-end formal guarantees for standard vision networks across a full 360° camera sweep with realistic rendering.

3 Preliminaries: Linear Approximations and Rendering Algorithms

Abstract Rendering bounds uncertainty in rendered images by propagating linear bound through each operation within the rendering pipeline. In this section, we delineate the fundamental operations integral to rendering algorithms and demonstrate their amenability to tight linear over-approximations. Subsequently, we provide an overview of two prominent neural rendering techniques: 3D Gaussian Splatting and Neural Radiance Fields (NeRF).

3.1 Linear Over-approximability

Notations. For vectors or matrices x, y of the same shape, $x < y$ denotes element-wise comparison. The Frobenius norm of a vector or matrix x is denoted by $\|x\|$. Boldface (e.g., $\mathbf{pc}$) indicates set-valued variables, while non-bold (e.g., pc) denotes fixed values. A *linear set* (or polytope) is defined as $\{x \in \mathbb{R}^n \mid Ax \leq b\}$, where $Ax \leq b$ specifies linear constraints. A *piecewise linear relation* $R \subseteq \mathbb{R}^n \times \mathbb{R}^m$ has the form $\left\{(x, y) \mid \underline{A}x + \underline{b} \leq y \leq \overline{A}x + \overline{b},\ Ax \leq b\right\}$, where $\underline{A}, \overline{A}, \underline{b}, \overline{b}$ define linear bounds on y given x. A *constant relation* is a special case where $\underline{A} = \overline{A} = 0$.

We consider a class of functions that can be tightly over-approximated by piecewise linear lower and upper bounds over any compact input domain.

Definition 1. *A function $f : S \to \mathbb{R}^m$ is called* linearly over-approximable *if, for any compact $X \subseteq S$ and $\varepsilon > 0$, there exist piecewise linear maps $\ell_X, u_X : X \to \mathbb{R}^m$ such that for all $x \in X$, $\ell_X(x) \leq f(x) \leq u_X(x)$ and $|u_X(x) - \ell_X(x)| < \varepsilon$.*

Proposition 1. *Any continuous function is linearly over-approximable.*

Although Proposition 1 implies piecewise linear bounds on continuous functions can be adjusted sufficiently tight by shrinking the neighborhood, finding accurate bounds over larger neighborhoods remains a significant challenge. Our implementation of abstract rendering employs CROWN [10] for computing such bounds. Except for Ind and Sort, all the basic operations listed in Table 1 are continuous and therefore linearly over-approximable. Moreover, since Ind is a piecewise constant function, it can be tightly bounded on each subdomain by partitioning at $x = 0$, allowing it to be treated as a linearly over-approximable function. This is formalized in Corollary 1.

Corollary 1. *All operations in Table 1, except for* Sort*, are linearly over-approximable.*

Table 1: Basic Operation Table. Conditional $A?B : C$ returns B if A is true and otherwise C. Permuting x based on y means reordering the elements of x according to the indices that sort y in ascending order. E.g., permuting $(9, 3, 7)$ based on $(5, 13, 8)$ results in $(9, 7, 3)$.

Operation	Inputs	Output	Math Representation
Element-wise add (Add)	$x, y \in \mathbb{R}^{n\times m}$	$z \in \mathbb{R}^{n\times m}$	$z_{ij} = x_{ij} + y_{ij}$
Element-wise multiply (Mul)	$x, y \in \mathbb{R}^{n\times m}$	$z \in \mathbb{R}^{n\times m}$	$z_{ij} = x_{ij} \cdot y_{ij}$
Division (Div)	$x \in \mathbb{R}^{n\times m}_{>0}$	$z \in \mathbb{R}^{n\times m}_{>0}$	$z_{ij} = 1/x_{ij}$
Matrix multiplication (Mmul)	$x \in \mathbb{R}^{n\times m}, y \in \mathbb{R}^{m\times k}$	$z \in \mathbb{R}^{n\times k}$	$z = x \times y$
Matrix inverse (Inv)	$x \in \mathbb{R}^{n\times n}, \det(x) \neq 0$	$z \in \mathbb{R}^{n\times n}$	$z = x^{-1}$
Matrix power (Pow)	$x \in \mathbb{R}^{n\times n}, k \in \mathbb{N}$	$z \in \mathbb{R}^{n\times n}$	$z = x^k$
Summation (Sum)	$x \in \mathbb{R}^n$	$z \in \mathbb{R}$	$z = \sum_{i=1}^{k} x_i$
Product (Prod)	$x \in \mathbb{R}^n$	$z \in \mathbb{R}$	$z = \prod_{i=1}^{k} x_i$
Matrix transpose ($\top$)	$x \in \mathbb{R}^{n\times m}$	$z \in \mathbb{R}^{m\times n}$	$z_{ij} = x_{ji}$
Element-wise exponential (Exp)	$x \in \mathbb{R}^{n\times m}$	$z \in \mathbb{R}^{n\times m}$	$z_{ij} = e^{x_{ij}}$
Frobenius norm (Norm)	$x \in \mathbb{R}^{n\times m}$	$z \in \mathbb{R}_{\geq 0}$	$z = \|x\|$
Element-wise indicator (Ind)	$x \in \mathbb{R}^{n\times m}$	$z \in \mathbb{R}^{n\times m}$	$z_{ij} = (x_{ij} > 0)?1 : 0$
Sorting (Sort)	$z \in \mathbb{R}^n$	$x \in \mathbb{R}^n, y \in \mathbb{R}^n$	permute x based on y

3.2 Rendering Algorithms: Gaussian Splat and NeRF

Scenes and Camera Model A *3D Gaussian scene* ScG is defined as a finite collection of 3D Gaussians in world coordinate frame w indexed by I. Each 3D Gaussian is specified by its mean μ_{w}, covariance Σ_{w}, opacity o and RGB color c. For simplicity, we use RGB instead of spherical harmonics to represent 3D Gaussians' colors. A *neural radiance field* ScN consists of an opacity network $\mathsf{F_o}$ mapping 3D points to opacity, a color network $\mathsf{F_c}$ mapping 3D points and view directions to color, maximum sampling distance L, and sampling count N. A *camera* C is characterized by translation $\mathsf{T} \in \mathbb{R}^3$, rotation $\mathsf{R} \in \mathbb{R}^{3\times 3}$, focal lengths $(\mathsf{f_x}, \mathsf{f_y}) \in \mathbb{R}^2$, and principal point offsets $(\mathsf{c_x}, \mathsf{c_y}) \in \mathbb{R}^2$.

Gaussian Splat rendering Algorithm 1 takes a Gaussian scene ScG, a camera C and a pixel coordinate u, and outputs the rendered RGB value pc at pixel u. Rendering proceeds in four steps: (1) it transforms 3D Gaussians from world to camera coordinate $(\mu_{\mathsf{c}}, \Sigma_{\mathsf{c}})$ (Line 2-3); (2) it projects 3D Gaussians onto image plane $(\mu_{\mathsf{p}}, \Sigma_{\mathsf{p}})$ and computes its weighed probability density a at pixel u (Line 4-10); (3) it sorts the Gaussians according to the distance of their means to the image plane (Line 11-14); (4) it computes pc by aggregating the colors of all Gaussians weighted according to their relative positions (Line 16-18). The full image is rendered by applying GAUSSIANSPLAT at every pixel location.

NeRF rendering Algorithm 2 takes as input a neural radiance field ScN, a camera C, and a pixel coordinate u, and outputs the rendered RGB value pc at that pixel. The rendering proceeds in three steps: (1) it computes the normalized direction of the camera ray $\mathsf{dir_w}$ corresponding to u and samples N points x along the ray within the maximum range L (Line 1–4); (2) it evaluates the opacity and

Algorithm 1 GAUSSIANSPLAT($\mathsf{ScG}, \mathsf{C}, \mathsf{u}$)

1: **for all** $\mathsf{i} \in \mathsf{I}$ **do**
2: $\quad \mu_\mathsf{c}[\mathsf{i}] \leftarrow \mathsf{R} \times (\mu_\mathsf{w}[\mathsf{i}] - \mathsf{T})$
3: $\quad \Sigma_\mathsf{c}[\mathsf{i}] \leftarrow \mathsf{R} \times \Sigma_\mathsf{w}[\mathsf{i}]$
4: $\quad \mathsf{K}[\mathsf{i}] \leftarrow \begin{bmatrix} \frac{\mathsf{f_x}}{\mu_\mathsf{c}[\mathsf{i},2]} & 0 & \frac{\mathsf{c_x}}{\mu_\mathsf{c}[\mathsf{i},2]} \\ 0 & \frac{\mathsf{f_y}}{\mu_\mathsf{c}[\mathsf{i},2]} & \frac{\mathsf{c_y}}{\mu_\mathsf{c}[\mathsf{i},2]} \end{bmatrix}$
5: $\quad \mathsf{J}[\mathsf{i}] \leftarrow \begin{bmatrix} \frac{\mathsf{f_x}}{\mu_\mathsf{c}[\mathsf{i},2]} & 0 & -\frac{\mathsf{f_x} \cdot \mu_\mathsf{c}[\mathsf{i},0]}{\mu_\mathsf{c}{}^2[\mathsf{i},2]} \\ 0 & \frac{\mathsf{f_y}}{\mu_\mathsf{c}[\mathsf{i},2]} & -\frac{\mathsf{f_y} \cdot \mu_\mathsf{c}[\mathsf{i},1]}{\mu_\mathsf{c}{}^2[\mathsf{i},2]} \end{bmatrix}$
6: $\quad \mu_\mathsf{p}[\mathsf{i}] \leftarrow \mathsf{K} \times \mu_\mathsf{c}[\mathsf{i}]$
7: $\quad \Sigma_\mathsf{p}[\mathsf{i}] \leftarrow \mathsf{J}[\mathsf{i}] \times \Sigma_\mathsf{c}[\mathsf{i}] \times \mathsf{J}[\mathsf{i}]^\top$
8: $\quad \mathsf{Conic}[\mathsf{i}] \leftarrow \mathrm{Inv}(\Sigma_\mathsf{p}[\mathsf{i}])$
9: $\quad \mathsf{q}[\mathsf{i}] \leftarrow (\mathsf{u} - \mu_\mathsf{p}[\mathsf{i}])^\top \times \mathsf{Conic}[\mathsf{i}] \times (\mathsf{u} - \mu_\mathsf{p}[\mathsf{i}])$
10: $\quad \mathsf{a}[\mathsf{i}] \leftarrow \mathsf{o}[\mathsf{i}] \cdot \mathrm{Exp}(-\frac{1}{2} \cdot \mathsf{q}[\mathsf{i}])$
11: $\quad \mathsf{d}[\mathsf{i}] \leftarrow \mu_\mathsf{c}[\mathsf{i}, 2]$
12: **end for**
13: $\mathsf{as} \leftarrow \mathrm{Sort}(\mathsf{a}, \mathsf{d})$
14: $\mathsf{cs} \leftarrow \mathrm{Sort}(\mathsf{c}, \mathsf{d})$
15: **for all** $\mathsf{i} \in \mathsf{I}$ **do**
16: $\quad \mathsf{oc}[\mathsf{i}] \leftarrow \prod_{\mathsf{j}=1}^{\mathsf{i}-1}(1 - \mathsf{as}[\mathsf{j}])$
17: **end for**
18: $\mathsf{pc} \leftarrow \sum_{\mathsf{i}=1}^{\mathsf{N}}(\mathsf{oc}[\mathsf{i}] \cdot \mathsf{as}[\mathsf{i}] \cdot \mathsf{cs}[\mathsf{i}])$
19: **return** pc

Algorithm 2 NERF($\mathsf{ScN}, \mathsf{C}, \mathsf{u}$)

1: $\mathsf{dir_c} \leftarrow \left[\frac{\mathsf{u_x} - \mathsf{c_x}}{\mathsf{f_x}}, \frac{\mathsf{u_y} - \mathsf{c_y}}{\mathsf{f_y}}, 1\right]$
2: $\mathsf{dir_w} \leftarrow \frac{\mathsf{dir_c} \times \mathsf{R}^\top}{||\mathsf{dir_c}||}$
3: **for all** $\mathsf{i} \in \mathsf{range}(\mathsf{N})$ **do**
4: $\quad \mathsf{x}[\mathsf{i}] \leftarrow \mathsf{T} + \frac{(2\mathsf{i}-1) \cdot \mathsf{L} \cdot \mathsf{dir_w}}{2\mathsf{N}}$
5: $\quad \mathsf{c}[\mathsf{i}] \leftarrow \mathsf{F_c}(\mathsf{x}[\mathsf{i}], \mathsf{dir_w})$
6: $\quad \mathsf{o}[\mathsf{i}] \leftarrow \mathsf{F_o}(\mathsf{x}[\mathsf{i}])$
7: $\quad \mathsf{a}[\mathsf{i}] \leftarrow 1 - \mathrm{Exp}\left(-\frac{\mathsf{o}[\mathsf{i}] \cdot \mathsf{L}}{\mathsf{N}}\right)$
8: $\quad \mathsf{oc}[\mathsf{i}] \leftarrow \prod_{\mathsf{j}=1}^{\mathsf{i}-1}(1 - \mathsf{a}[\mathsf{j}])$
9: **end for**
10: $\mathsf{pc} \leftarrow \sum_{\mathsf{i}=1}^{\mathsf{N}}(\mathsf{oc}[\mathsf{i}] \cdot \mathsf{a}[\mathsf{i}] \cdot \mathsf{c}[\mathsf{i}])$
11: **return** pc

Algorithm 3 MATRIXINV($\mathsf{X}, \mathsf{X_{ref}}, \mathsf{k}$)

1: $\Delta\mathsf{X} \leftarrow -(\mathsf{X} - \mathsf{X_{ref}}) \times \mathsf{X_{ref}^{-1}}$
2: **assert** $||\Delta\mathsf{X}|| < 1$
3: $\mathsf{Xp} \leftarrow \sum_{\mathsf{i}=0}^{\mathsf{k}} \mathsf{X_{ref}^{-1}} \times \mathrm{Pow}(\Delta\mathsf{X}, \mathsf{i})$
4: $\mathsf{X_R} \leftarrow ||\mathsf{X_{ref}^{-1}}|| \cdot \frac{||\Delta\mathsf{X}||^{\mathsf{k}+1}}{1 - ||\Delta\mathsf{X}||}$
5: $\mathsf{lXinv} \leftarrow \mathsf{Xp} - \mathsf{X_R}$
6: $\mathsf{uXinv} \leftarrow \mathsf{Xp} + \mathsf{X_R}$
7: **return** $\langle \mathsf{lXinv}, \mathsf{uXinv} \rangle$

color of each sampled point using the networks $\mathsf{F_o}$ and $\mathsf{F_c}$ (Line 5–6); (3) it aggregates the colors along the ray by computing a weighted sum based on the opacity and depth ordering of the samples (Line 7–10).

Having introduced two representative rendering algorithms, we now present a formal definition of abstract rendering.

Abstract rendering problem. Our goal is to design an algorithm that for all pixel $\mathsf{u} \in \mathsf{WH}$, it takes as input a linear set of scenes $\mathbf{Sc}$ and cameras $\mathbf{C}$, and outputs a linear set of pixel color $\mathbf{pc}$ such that GAUSSIANSPLAT$(\mathsf{Sc}, \mathsf{C}, \mathsf{u}) \in \mathbf{pc}$ or NERF$(\mathsf{Sc}, \mathsf{C}, \mathsf{u}) \in \mathbf{pc}$) for each $\mathsf{Sc} \in \mathbf{Sc}$ and $\mathsf{C} \in \mathbf{C}$.

4 Methodology: Abstract Rendering

Our abstract rendering algorithm processes scene and camera inputs by incrementally building piecewise linear relations between intermediate variables and inputs through standard rendering steps, proceeding until pixel-input relationships are established. These relations enable linear bound computation for each pixel, yielding linear bound for image. Since most GAUSSIANSPLAT and NERF operations are linear, their relations compose directly. In this section, we highlight three rendering-specific nonlinear operations (matrix inverse, sorting-based aggregation and cumulative product summation) and introduce tailored techniques to tightly bound them with linear over-approximations.

4.1 Abstracting Matrix Inverse by Taylor Series

Bounding matrix inversion (Line 8 of GAUSSIANSPLAT) via the adjugate method[2] often introduces significant over-approximation due to uncertainty in the denominator. To address this, we propose a Taylor series-based algorithm, MATRIXINV (Algorithm 3), which reformulates matrix inversion using only addition, multiplication, and the inverse of a fixed reference matrix. Although the remainder term still involves division, its influence diminishes as the Taylor order increases.

[2]The adjugate method computes the inverse as the adjugate matrix divided by the determinant.

Algorithm 4 VR-IND($\mathsf{a}, \mathsf{c}, \mathsf{d}$)

1: **for all** $\mathsf{i} \in \mathsf{I}$ **do**
2: $\quad \mathsf{oc}[\mathsf{i}] \leftarrow \prod_{\mathsf{j}=1}^{\mathsf{N}} \left(1 - \mathsf{a}[\mathsf{j}] \cdot \mathrm{Ind}(\mathsf{d}[\mathsf{i}] - \mathsf{d}[\mathsf{j}])\right)$
3: **end for**
4: $\mathsf{pc} \leftarrow \sum_{\mathsf{i}=1}^{\mathsf{N}} (\mathsf{oc}[\mathsf{i}] \cdot \mathsf{a}[\mathsf{i}] \cdot \mathsf{c}[\mathsf{i}])$
5: **return** pc

Algorithm 5 SUMCUMPROD(a, c)

1: $\mathsf{pc} \leftarrow [0, 0, 0]$
2: **for all** $\mathsf{i} \in \mathrm{reverse}(\mathsf{I})$ **do**
3: $\quad \mathsf{pc} \leftarrow \mathsf{a}[\mathsf{i}] \cdot \mathsf{c}[\mathsf{i}] + (1 - \mathsf{a}[\mathsf{i}]) \cdot \mathsf{pc}$
4: **end for**
5: **return** pc

MATRIXINV takes a non-singular matrix X, a non-singular reference matrix $\mathsf{X}_{\mathsf{ref}}$, and a Taylor order k as input, and returns lower and upper bounds lXinv and uXinv for X^{-1}. The algorithm first checks the convergence condition (Line 2), then computes the k-th order Taylor approximation and its remainder (Lines 3–4), and finally derives bounds via basic addition and subtraction (Lines 5–6).

Lemma 1. *Given any non-singular matrix* X*, the output of Algorithm* MATRIXINV $\langle \mathsf{lXinv}, \mathsf{uXinv} \rangle$ *satisfies that:* $\mathsf{lXinv} \leq \mathrm{Inv}(\mathsf{X}) \leq \mathsf{uXinv}$.

Lemma 1 guarantees that MATRIXINV yields valid bounds for the Inv operation with fixed input. When extended to a linear set of input matrices, the linear bounds of lXinv and uXinv can be obtained by propagating linear relations line by line. Their union defines the overall linear bounds for the inversion. Example 1 illustrates the advantage of using MATRIXINV over the adjugate method in terms of bound tightness.

Example 1. *Given the constant lower and upper bounds of input matrices,*

$$\underline{X} = \begin{bmatrix} 0.60 & -0.2 \\ -0.02 & 0.90 \end{bmatrix}, \quad \overline{X} = \begin{bmatrix} 0.90 & 0.02 \\ 0.02 & 1.30 \end{bmatrix},$$

we combine lower bound of lXinv *and upper bound of* uXinv *as overall bounds for* Inv *operation, and take* $||\mathsf{uXinv} - \mathsf{lXinv}||$ *as measurement for bound tightness. The adjugate method yields a bound width of* 1.22*, while* MATRIXINV *reduces it to* 0.70*, closely matching the empirical value of* 0.66*.*

4.2 Abstracting Sorting-based Summation by Indicator

The sorting step in Lines 13–14 of GAUSSIANSPLAT models occlusion: a front Gaussian occludes those behind, but not vice versa. Bounding sorted elements independently introduces large over-approximations in the final pixel color (Line 18), as it neglects the inherent ordering constraint. To mitigate this, we introduce VR-IND (Algorithm 4), which replaces sorting with pairwise indicator-based occlusion modeling. This approach preserves the output while significantly improving bound tightness in linear relaxation.

Lemma 2. *For any given inputs — effective opacities* a*, colors* c *and depth* d*,* VR-IND *outputs the same value as computation from Line 13 to Line 18 in* GAUSSIANSPLAT.

Lemma 2 confirms that VR-IND produces the same output as Lines 13–18 in GAUSSIANSPLAT, capturing both occlusion (Line 2) and color aggregation (Line 4) without explicit sorting. Example 2 further demonstrates its advantage in reducing over-approximation.

Example 2. *Consider a scene with three Gaussians (red, green, blue) and a camera at the origin facing upward, with each coordinate perturbed by* ± 0.3*. As shown in Figure 2, the upper bound obtained via linear relaxation through* VR-IND *is significantly tighter than that derived from direct bound propagation through* GAUSSIANSPLAT*. An analytical example is given in Appendix A.*

4.3 Abstract Summation of Cumulative Product

The summation of cumulative products arises in Lines 8–10 of NERF, where occlusion effects are computed and colors are aggregated to produce the final pixel color. Naively bounding each element of oc independently and then aggregating can lead to significant over-approximation, since the intermediate terms a are shared across multiple entries. To mitigate this, we propose a factorization-based formulation, SUMCUMPROD (Algorithm 5), which computes each element of a only once in the final expression for pc. This approach preserves exactness for fixed inputs while enabling substantially tighter bounds during linear relaxation.

Lemma 3. *For any given inputs — effective opacities* a*, colors* c*,* SUMCUMPROD *outputs the same value as computation from Line 8 to Line 10 in* NERF.

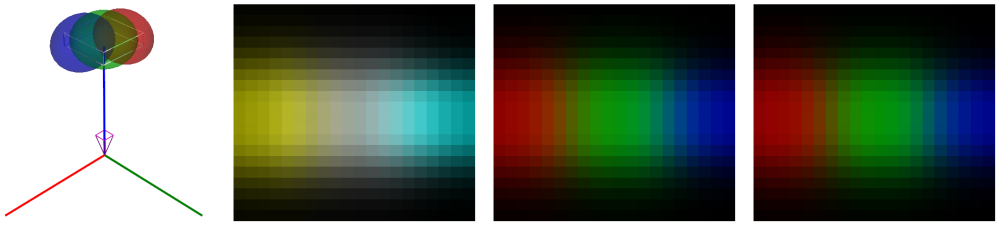

Figure 2: Example 2 (left) shows the three-Gaussian scene. The corresponding upper bounds are visualized as follows: via GAUSSIANSPLAT (mid-left), via VR-IND (mid-right), and via sampling (right). Notably, the bounds produced by VR-IND are tighter (darker) and closely align with the sampled results.

Lemma 3 confirms that, for fixed input, SUMCUMPROD exactly reproduces the original NERF output without explicitly computing occlusion for each sampled point. Example 3 shows that it produces much tighter bounds than the standard method. However, SUMCUMPROD has two limitations: (1) its recursive structure necessitates iterative bound propagation, thereby increasing computational cost; (2) it assumes a fixed order of a, restricting its applicability to Gaussian-represented scenes where the Gaussian depth order remains unchanged, such as in scenarios involving camera translation uncertainty.

Example 3. *Consider a single-channel color* $\mathsf{c} = [0.8, 1.0, 0.9, 0.8, 0.5]$ *and effective opacity* $\mathsf{a} = [0.2, 0.5, 0.6, 0.8, 0.1]$ *with* ± 0.1 *perturbations on each* $\mathsf{a}[\mathsf{i}]$. SUMCUMPROD *yields a tight bound on output pixel color,* $[0.826, 0.926]$*, versus a looser* $[0.780, 1.029]$ *from Lines 8–10 of* NERF. *Sampling 10,000 random perturbations gives* $[0.831, 0.922]$*, closely matching the result from* SUMCUMPROD.

4.4 Linear Approximability of ABSTRACTRENDER

The operations in MATRIXINV, VR-IND, and SUMCUMPROD fall within those listed in Table 1. Since sorting can be eliminated using VR-IND, and all remaining operations are linear over-approximable, we arrive at Theorem 1. It states that both GAUSSIANSPLAT and NERF can be rewritten using only linearly approximable operations, enabling computation of tight bounds via input domain partitioning.

Theorem 1. *By replacing components of* GAUSSIANSPLAT *and* NERF *with* MATRIXINV, VR-IND, *and* SUMCUMPROD, *both rendering algorithms are fully linear over-approximable.*

5 Experiments with Abstract Rendering

We implemented the abstract rendering algorithm ABSTRACTRENDER described in Section 4 with both GAUSSIANSPLAT and NERF. As mentioned earlier, the linear approximation of the continuous operations in Table 1 is implemented using CROWN [10].

Scene Description. We evaluate ABSTRACTRENDER on scenes of varying scales and complexities, represented using GAUSSIANSPLAT and NERF. The scenes include: `Lego`, `Chair`, and `Drums` [2], which are single-object scenes on empty backgrounds; `PineTree` [6], a synthetic boulevard scene with trees; `Airport` [8], a large-scale photorealistic airport environment; `Garden` [1], a real-world scene; and `Airplane`, `Truck`, and `Car`, containing objects corresponding to CIFAR-10 classes. GAUSSIANSPLAT reconstructions are generated using Splatfacto [45], while those of NERF are trained with the vanilla NERF algorithm [2], both implemented with standard Nerfstudio settings. Further details of these scenes are provided in the appendix.

Experimental Setup. For each scene, we evaluate ABSTRACTRENDER under varying camera poses and input perturbations. As a baseline, we construct **empirical bound images** by (1) sampling 50 images per input partition (the full perturbation range is divided into hundreds or thousands of partitions) and (2) computing pixel-wise lower and upper bounds across the samples. Since exhaustive enumeration is infeasible, these empirical bound images are **under-approximations** of the exact set of renderable images, whereas our framework provides sound **over-approximations**. The proximity of ABSTRACTRENDER results to the empirical bound images indicates the tightness of our computed bounds.

Metrics and Evaluation. To measure the tightness between the pixel-wise lower and upper bound images, we report **Mean Pixel Gap** (MPG) and **Max Pixel Gap** (XPG), defined as:

$$\mathsf{MPG} = \frac{1}{|\mathsf{WH}|} \sum_{\mathsf{j} \in \mathsf{WH}} \|\overline{\mathsf{pc_j}} - \underline{\mathsf{pc_j}}\|, \quad \mathsf{XPG} = \max_{\mathsf{j} \in \mathsf{WH}} \|\overline{\mathsf{pc_j}} - \underline{\mathsf{pc_j}}\|$$

where $\overline{\mathsf{pc_j}}$ and $\underline{\mathsf{pc_j}}$ denote the pixel-wise upper and lower bounds, and WH represents the set of all pixel coordinates on the image plane. Smaller values of MPG or XPG indicate a smaller difference between the lower and upper bound images. Table 2 summarizes the results across different scene representations and perturbations, while Figure 3 visualizes the comparison between AbstractRender results and empirical bounds for two scenes: `Airplane` and `Lego`.

Table 2: AbstractRender results for scenes represented by GaussianSplat (GS) and NeRF, along with empirical bounds. CPR: Camera Perturbation Range; Dim: Perturbation Dimension; Res: Rendered Image Resolution; Rt: Runtime (min); MPG: mean pixel gap; XPG: maximum pixel gap.

Scene	CPR	Dim	Res	**GS**			**NeRF**			**Empirical**	
				Rt	MPG	XPG	Rt	MPG	XPG	MPG	XPG
Lego	0.1 (rad)	yaw	80×80	22	0.51	1.73	25	0.40	1.73	0.22	1.57
Chair	0.1 (rad)	yaw	50×50	17	0.46	1.73	20	0.40	1.73	0.40	1.72
Drums	0.1 (m)	x	50×50	5.3	0.19	1.62	5.8	0.16	1.55	0.13	1.54
PineTree	2 (m)	x	72×72	4.1	0.27	1.73	4.9	0.23	1.73	0.06	1.37
PineTree	10 (m)	x	72×72	24	0.47	1.73	28	0.36	1.73	0.18	1.37
Airport	0.027 (rad)	roll	160×160	27	0.56	1.69	35	0.42	1.63	0.21	1.38
Airport	0.03 (rad)	roll	160×160	30	0.59	1.70	43	0.51	1.68	0.22	1.41
Garden	0.5 (m)	x	100×100	20	0.53	1.59	11	0.37	1.53	0.34	1.21
Airplane	2 (m)	x	80×80	16	0.41	1.68	19	0.33	1.61	0.15	1.39

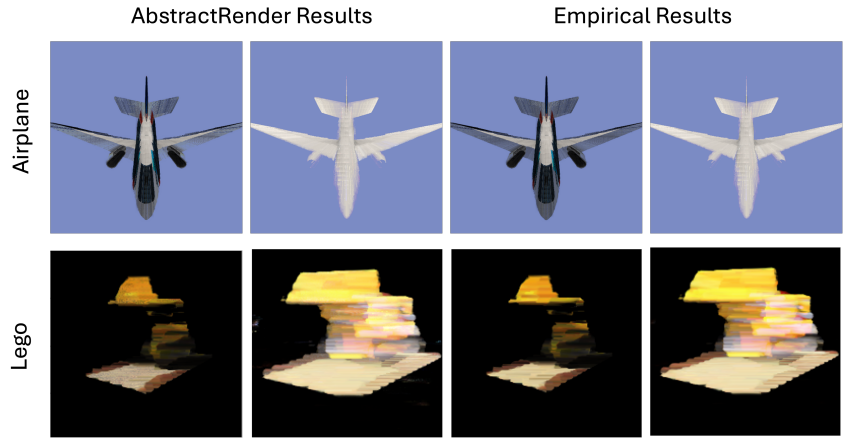

Figure 3: Comparison of lower and upper images produced by AbstractRender (Left) and empirical approach (Right). The `Airplane` scene (Top) is represented by GaussianSplat, while the `Lego` scene (Bottom) is represented by NeRF. The experimental settings for both cases follow those reported in Table 2.

From Table 2, we see that AbstractRender is applicable to diverse scenes, including single-object (`Lego`), synthetic (`PineTree`), and large-scale realistic (`Airport`, `Garden`) scenes, with reasonable runtime at 200×200 resolution. The framework supports camera perturbations in both position (x, y, z) and orientation (roll, pitch, yaw). The metrics (MPG and XPG) computed by AbstractRender (under GS and NeRF columns in Table 2) are larger than the corresponding empirical bounds, as expected, since our framework produces over-approximations, which inherently encompass a larger image space than the under-approximated empirical bounds. We further observe that bounds computed for NeRF exhibit tighter margins than those for GaussianSplat, as indicated by smaller MPG/XPG gaps relative to empirical samples. This is because NeRF can leverage our linear bound

relaxation for cumulative product summation (Section 4.3), whereas GAUSSIANSPLAT cannot, due to variations in the depth of 3D Gaussians on the image plane under camera rotation uncertainty.

Overall, these experimental results confirm that our method produces sound over-approximations that are meaningfully tight for rendered images from both GAUSSIANSPLAT and NERF under camera pose uncertainty.

ABSTRACTRENDER for Certified Classification This experiment evaluates ABSTRACTRENDER for certifying neural classifier robustness against camera pose perturbations. We test whether a pretrained CIFAR-10 ResNet [46] maintains correct predictions as the camera orbits the target object azimuthally over 360° at fixed distance and elevation. The 360° range is partitioned into angular intervals, and ABSTRACTRENDER computes abstract images for each, which are propagated through the classifier via CROWN [10] for set-based certification. An interval is robust if the lower bound of the target label exceeds the upper bounds of all other classes. Verification coverage is the percentage of certifiably robust intervals. Table 3 shows results for `Airplane`, `Car`, and `Truck` in GAUSSIANSPLAT and NERF scenes. Figure 4 visualizes camera regions where the classifier is certifiably correct (green) and regions without such guarantees (red). Regions capturing more of the object or its distinctive features are more likely to be verified as robust, e.g., the lateral views of the airplane and car, and the front-left view of the truck.

Table 3: Classifier certification results. Obj: Target Object; SR: Scene Representation, either GAUSSIANSPLAT or NERF; d: Distance to Object Centroid (m); h: Camera Height to Object's Horizon Plane (m); Npart: Number of Partitions; PCP-S: Percentage of Correct Partitions via Sampling; PCP-V: Percentage of Certified Partitions via Formal Verification; Rt: Runtime (min).

Obj	SR	d	h	Npart	PCP-S	PCP-V	Rt
`Airplane`	GS	40	10	62832	96.0%	74.5%	1140.5
`Car`	GS	8	2.5	62832	96.3%	76.7%	879.6
`Truck`	GS	4	1.2	62832	71.8%	31.9%	963.4
`Airplane`	NERF	40	36	3142	25.6%	22.8%	94.3
`Car`	NERF	4	0.5	3142	75.9%	42.1%	118.6

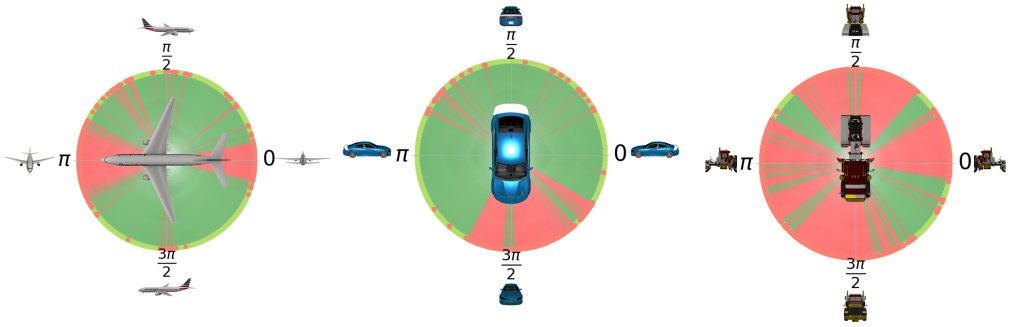

Figure 4: Classifier certification results for `Airplane`, `Car`, and `Truck` in GAUSSIANSPLAT scenes across 0–360° camera rotations. Green: certified regions—images captured within these camera positions are always predicted correctly. Red: uncertified regions—images captured within these camera positions may be predicted incorrectly.

ABSTRACTRENDER for certified pose estimation This experiment evaluates ABSTRACTRENDER for certifying the robustness of neural pose estimators against camera pose perturbations. We assess a target pose estimator built upon GateNet [47] (fine-tuned on our airplane and truck datasets) by partitioning the camera's translational perturbation range into intervals. ABSTRACTRENDER generates abstract images capturing all photometric variations within each interval, which are propagated through the estimator via CROWN to verify whether positional errors between pose estimation and ground truth holds below a given error tolerance (20% divation from ground truth). Verification coverage is measured as the proportion of robust intervals across the workspace. Table 4 reports results

for `Airplane` and `Truck` in GAUSSIANSPLAT and NERF scenes, while Figure 5 visualizes camera regions where the estimator meets the error tolerance (green) versus regions without guarantees (red). Regions where the target object occupies a substantial portion of the rendered image are easier to certify—for example, distant regions in GAUSSIANSPLAT scenes and closer regions in NERF scenes, reflecting the different default camera-object distances in the two settings.

Table 4: Pose estimator and object detecter certification results. Obj: Target Object; SR: Scene Representation, either GAUSSIANSPLAT or NERF; d: Distance to Object Centroid; CPR: Camera Perturbation Range (m); PCP-S: Percentage of Correct Partitions via Sampling; PCP-V: Percentage of Certified Partitions via Formal Verification; Thr: Threshold; Rt: Runtime(min).

Obj	SR	**Certified Pose Estimation**					**Certified Object Detection**				
		d	CPR	PCP-S	PCP-V	Rt	Thr	CPR	PCP-S	PCP-V	Rt
`Airplane`	GS	174.4	6	52.4%	23.3%	362.2	0.45	6	81.5%	51.4%	422.2
`Truck`	GS	7.8	1	37.6%	22.2%	311.3	0.46	1	47.2%	42.4%	367.3
`Airplane`	NERF	40	20	46.7%	37.2%	82.4	0.12	3.5	92.2%	80.4%	121.3
`Truck`	NERF	8	2	75.1%	70.0%	53.4	0.12	1.5	100.0%	71.0%	66.5

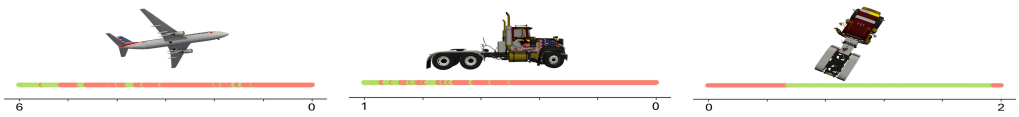

Figure 5: Pose estimator certification results for `Airplane` and `Truck` in GAUSSIANSPLAT scenes (Left, Middle) and `Truck` in a NERF scene (Right) across a linear camera translation. Green: certified regions—camera positions where estimated poses always remain within given error tolerance. Red: uncertified regions—camera positions where estimated poses may exceed given error tolerance.

ABSTRACTRENDER for certified object detection This experiment evaluates ABSTRACTRENDER for certifying neural object detectors under camera pose perturbations. We test a pretrained detector based on the YOLO2 architecture [48], with a backbone from the VNN-COMP 2023 benchmark[3]. The goal is to assess whether the detector can consistently produce bounding boxes that exceeds a predefined confidence threshold under camera translations. The camera's translational range is partitioned into small intervals. For each interval, ABSTRACTRENDER computes abstract images. These images are propagated through the YOLO2 detector. An interval is considered robust if the detector maintains a high-confidence bounding box across all images rendered from that interval. Table 4 reports results for `Airplane` and `Truck` in GAUSSIANSPLAT and NERF scenes, demonstrating ABSTRACTRENDER's capacity to certify object detection models.

6 Conclusions

Limitations. ABSTRACTRENDER currently has several limitations: First, like other verification approaches, it takes more time than sampling-based approaches. Second, the correctness of the analysis assumes the reconstructed scene accurately represents reality. Third, our framework computes bounds for each pixel's color independently, ignoring correlations between neighboring pixels.

Despite these limitations, ABSTRACTRENDER is the first framework for computing abstract images of scenes represented by Gaussian Splats and NeRF under camera pose or scene uncertainty. It is built using novel linear relational approximations of three rendering-specific operations. By integrating ABSTRACTRENDER with CROWN, we have enabled certification of visual tasks with respect to semantic variations in 3D environments. A promising direction for future work is to accommodate scenes inaccuracies and to certify visual control functions.

[3] `https://github.com/VNN-COMP/vnncomp2023_benchmarks`

Acknowledgments and Disclosure of Funding

Chenxi Ji, Yangge Li, and Sayan Mitra are supported by a research grant from The Boeing Company and NSF (FMITF-2525287). Huan Zhang and Xiangru Zhong are supported in part by the AI2050 program at Schmidt Sciences (AI2050 Early Career Fellowship) and NSF (IIS SLES-2331967, CCF FMITF-2525287).

References

[1] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

[2] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[3] Siting Zhu, Guangming Wang, Xin Kong, Dezhi Kong, and Hesheng Wang. 3d gaussian splatting in robotics: A survey. *arXiv preprint arXiv:2410.12262*, 2024.

[4] Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results, 2024.

[5] Matthias Koenig, Annelot W Bosman, Holger H Hoos, and Jan N Van Rijn. Critically assessing the state of the art in neural network verification. *Journal of Machine Learning Research*, 25(12):1–53, 2024.

[6] P Habeeb, Deepak D'Souza, Kamal Lodaya, and Pavithra Prabhakar. Interval image abstraction for verification of camera-based autonomous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):4310–4321, 2024.

[7] Sayan Mitra, Corina Păsăreanu, Pavithra Prabhakar, Sanjit A. Seshia, Ravi Mangal, Yangge Li, Christopher Watson, Divya Gopinath, and Huafeng Yu. *Formal Verification Techniques for Vision-Based Autonomous Systems – A Survey*, pages 89–108. Springer Nature Switzerland, Cham, 2025.

[8] Yangge Li, Chenxi Ji, Jai Anchalia, Yixuan Jia, Benjamin C Yang, Daniel Zhuang, and Sayan Mitra. Lyapunov perception contracts for operating design domains. In *7th Annual Learning for Dynamics\& Control Conference*, pages 1053–1065. PMLR, 2025.

[9] Chiao Hsieh, Yubin Koh, Yangge Li, and Sayan Mitra. Assuring safety of vision-based swarm formation control, 2023.

[10] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

[11] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[12] Hai Duong, ThanhVu Nguyen, and Matthew B Dwyer. Neuralsat: A high-performance verification tool for deep neural networks. In *International Conference on Computer Aided Verification*, pages 409–423. Springer, 2025.

[13] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International conference on computer aided verification*, pages 443–452. Springer, 2019.

[14] Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T Johnson. Nnv 2.0: The neural network verification tool. In *International Conference on Computer Aided Verification*, pages 397–412. Springer, 2023.

[15] Augustin Lemesle, Julien Lehmann, and Tristan Le Gall. Neural network verification with pyrat. *arXiv preprint arXiv:2410.23903*, 2024.

[16] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T Johnson. Improved geometric path enumeration for verifying relu neural networks. In *International Conference on Computer Aided Verification*, pages 66–96. Springer, 2020.

[17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[18] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[19] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*, pages 5286–5295. PMLR, 2018.

[20] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[21] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in neural information processing systems*, 31, 2018.

[22] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

[23] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[24] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[25] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[26] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[27] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. *IEEE access*, 9:155161–155196, 2021.

[28] Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[29] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In *International conference on machine learning*, pages 1802–1811. PMLR, 2019.

[30] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[31] Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1614–1619, 2018.

[32] Cassidy Laidlaw and Soheil Feizi. *Functional adversarial attacks*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[33] Anand Bhattad, Min Jin Chong, Kaizhao Liang, Bo Li, and David A. Forsyth. Unrestricted adversarial examples via semantic manipulation. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[34] Ali Shahin Shamsabadi, Ricardo Sanchez-Matilla, and Andrea Cavallaro. Colorfool: Semantic adversarial colorization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1151–1160, 2020.

[35] Rima Alaifari, Giovanni S. Alberti, and Tandri Gauksson. ADef: an iterative algorithm to construct adversarial deformations. In *International Conference on Learning Representations*, 2019.

[36] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[37] Dan Hendrycks*, Norman Mu*, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple method to improve robustness and uncertainty under data shift. In *International Conference on Learning Representations*, 2020.

[38] Jeet Mohapatra, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Towards verifying robustness of neural networks against a family of semantic perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 244–252, 2020.

[39] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] Marc Fischer, Maximilian Baader, and Martin Vechev. Certified defense to image transformations via randomized smoothing. *Advances in Neural information processing systems*, 33:8404–8417, 2020.

[41] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[42] Hanjiang Hu, Zuxin Liu, Linyi Li, Jiacheng Zhu, and Ding Zhao. Robustness certification of visual perception models via camera motion smoothing. In *Proceedings of the 6th Conference on Robot Learning*, 2022.

[43] Hanjiang Hu, Zuxin Liu, Linyi Li, Jiacheng Zhu, and Ding Zhao. Pixel-wise smoothing for certified robustness against camera motion perturbations. In *International Conference on Artificial Intelligence and Statistics*, pages 217–225. PMLR, 2024.

[44] Christian Green, Mehmet Ergezer, and Abdurrahman Zeybey. Targeted view-invariant adversarial perturbations for 3d object recognition, 2024.

[45] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 conference proceedings*, pages 1–12, 2023.

[46] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in neural information processing systems*, 34:29909–29921, 2021.

[47] Huy Xuan Pham, Ilker Bozcan, Andriy Sarabakha, Sami Haddadin, and Erdal Kayacan. Gatenet: An efficient deep neural network architecture for gate perception using fish-eye camera in autonomous drone racing. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4176–4183. IEEE, 2021.

[48] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

A An analytical example illustrating the advantage of Algorithm VR-IND

Consider two purely white Gaussians (with color values set to 1) whose probability densities and depth to the target pixel are given by $a = \{0.8, 0.5\}$ and $d = \{x, x+1\}$, respectively, where $x \in [-1, 1]$. We aim to compute the worst-case bound of volumetric blending as follows:

$$\begin{aligned} as &= sort(a, d) \\ oc[0] &= 1 \\ oc[1] &= 1 - as[0] \\ pc &= oc[0] \times as[0] + oc[1] \times as[1] \end{aligned}$$

we first determine the output range of pc via Naive interval approach. By plugging in range of x into the expression of d, we have $d[0] = x \in [-1, 1]$, and $d[1] = x+1 \in [0, 2]$. Since the relative ordering between $d[0]$ and $d[1]$ cannot be inferred from their interval bound, we must consider both possible cases: one where $(as[0], as[1]) = (a[0], a[1])$, and the other where the assignment is reversed. Thus, we have

$$\begin{aligned} &as[0], as[1] \in [0.5, 0.8]; \\ &oc[1] \in [0.2, 0.5] \end{aligned}$$

Hence, the resulting interval bound for pc is:

$$pc \in [1 \cdot 0.5 + 0.2 \cdot 0.5, 1 \cdot 0.8 + 0.5 \cdot 0.8] = [0.6, 1.2]$$

Next, we determine the output range of pc via proposed algorithm VR-IND, in which the same computation is encoded symbolically to account for value-dependent assignments. The volumetric blending procedure is reformulated using indicator functions (Ind), making the dependencies on d explicit:

$$\begin{aligned} oc[0] &= (1 - a[0] \times Ind(d[0] - d[0])) \times (1 - a[1] \times Ind(d[0] - d[1])) \\ oc[1] &= (1 - a[0] \times Ind(d[1] - d[0])) \times (1 - a[1] \times Ind(d[1] - d[1])) \\ pc &= oc[0] \times a[0] + oc[1] \times a[1] \end{aligned}$$

Given $d[0]$ and $d[1]$, we can compute the values of indicator functions Ind in the above formulation.

$$\begin{aligned} Ind(d[0] - d[1]) &= Ind(x - (x+1)) = Ind(-1) = 0 \\ Ind(d[1] - d[0]) &= Ind((x+1) - x) = Ind(1) = 1 \\ Ind(d[1] - d[1]) &= Ind(d[0] - d[0]) = Ind(0) = 0 \end{aligned}$$

Substituting these indicator values yields:

$$\begin{aligned} oc[0] &= (1 - 0.8 \times 0) \times (1 - 0.5 \times 0) = 1 \\ oc[1] &= (1 - 0.8 \times 1) \times (1 - 0.5 \times 0) = 0.2 \\ pc &= 1 \times 0.8 + 0.2 \times 0.5 = 0.9 \end{aligned}$$

Hence, using VR-IND, we obtain a precise bound $pc \in [0.9, 0.9]$, which is significantly tighter than the interval-based bound of $[0.6, 1.2]$. Remark: in the real verification pipeline, VR-INDproduces a linear relaxation set instead of a fixed value. Here we use a simplified case for highlighting VR-IND's advantage in terms of bound tightness.

The naive interval method leads to loose bounds due to the loss of dependency information caused by Sort operation. In contrast, the proposed VR-Ind method preserves input-dependent relationships and yields significantly tighter bounds.

B Proofs

Proposition 1. *Any continuous function is linearly over-approximable.*

Proof. We present a simple proof for scalar f and this can be extended to higher dimensions. We fix $x_0 \in \mathbb{R}$ and a finite slope $k \in \mathbb{R}$. Since $f(x) - k(x - x_0)$ is continuous, for any $\varepsilon > 0$, there must exist an open neighborhood $U(x_0)$ around x_0, such that $\forall x \in R$,

$$|f(x) - f(x_0) - k(x - x_0)| < \frac{\varepsilon}{3}. \tag{1}$$

Equivalently,

$$f(x_0) + k(x - x_0) - \frac{\varepsilon}{3} < f(x) < f(x_0) + k(x - x_0) + \frac{\varepsilon}{3}. \tag{2}$$

We construct candidate linear upper and lower bounds over U as:

$$l_U(x) = k(x - x_0) + f(x_0) - \frac{\varepsilon}{3}, \quad u_U(x) = k(x - x_0) + f(x_0) + \frac{\varepsilon}{3}. \tag{3}$$

We can check that these linear functions $l_U(x)$ and $u_U(x)$ are valid lower and upper bounds for $f(x)$ over U, because $\forall x \in U(x_0)$:

$$f(x) - l_U(x) = f(x) - (k(x - x_0) + f(x_0) - \frac{\varepsilon}{3}) > -\frac{\varepsilon}{3} + \frac{\varepsilon}{3} = 0, \tag{4}$$

$$f(x) - u_U(x) = f(x) - (k(x - x_0) + f(x_0) + \frac{\varepsilon}{3}) < \frac{\varepsilon}{3} - \frac{\varepsilon}{3} = 0. \tag{5}$$

Additionally, difference between $l_U(x)$ and $u_U(x)$ is tightly bounded, as $\forall x \in U(x_0)$:

$$|u_U(x) - l_U(x)| = (k(x - x_0) + f(x_0) + \frac{\varepsilon}{3}) - (k(x - x_0) + f(x_0) - \frac{\varepsilon}{3}) = \frac{2\varepsilon}{3} < \varepsilon.$$

Now, for any compact set B, the collection $\mathcal{U} = \{U(x) \mid x \in B\}$ forms an open cover of B. By the definition of compactness, there exists a finite subcover of B, denoted by $\{U_i = U(x_i) \mid i = 1, \ldots, s\}$. We can now define piecewise linear functions on B as follows:

$$\begin{aligned} u_B(x) &= \min_{j \in I} u_{U_j}(x), \\ l_B(x) &= \max_{j \in I} l_{U_j}(x), \quad if\ x \in \{U_j \mid j \in I\} \end{aligned} \tag{6}$$

where I is the index set of U_i that cover x. Consequently, we have

$$u_B(x) = \min_{j \in I} u_{U_j}(x) \geq f(x) \geq \max_{j \in I} l_{U_j}(x) = l_B(x), \tag{7}$$

and

$$|u_B(x) - l_B(x)| \leq u_{U_j}(x) - l_{U_j}(x) \leq \varepsilon, \quad \forall j \in I. \tag{8}$$

Thus, f is linearly over-approximable. □

Lemma 1. *Given any non-singular matrix* X*, the output of Algorithm* MATRIXINV $\langle \mathsf{lXinv}, \mathsf{uXinv} \rangle$ *satisfies that:*

$$\mathsf{lXinv} \leq \mathrm{Inv}(\mathsf{X}) \leq \mathsf{uXinv}$$

Proof. For any non-singular $n \times n$ matrices X and $\mathsf{X_{ref}}$, denote their difference as $\Delta\mathsf{X} = -(\mathsf{X} - \mathsf{X_{ref}})$. The $\mathsf{k^{th}}$ order Taylor Polynomial Xp and remainder $\mathsf{X_R}$ of matrix inverse of X, estimated at $\mathsf{X_{ref}}$, can be written as follows:

$$\mathsf{Xp} = \mathsf{X_{ref}^{-1}} \sum_{\mathsf{i}=0}^{\mathsf{k}} (\Delta\mathsf{X} \cdot \mathsf{X_{ref}^{-1}})^{\mathsf{i}} \tag{9}$$

$$\mathsf{X_R} = \mathsf{X_{ref}^{-1}} \sum_{\mathsf{i}=\mathsf{k}+1}^{\infty} (\Delta\mathsf{X} \cdot \mathsf{X_{ref}^{-1}})^{\mathsf{i}} \tag{10}$$

Assuming that $\|\Delta \mathsf{X} \cdot \mathsf{X}_{\mathsf{ref}}^{-1}\| < 1$, the remainder X_{R} can be bounded by:

$$\|\mathsf{X}_{\mathsf{R}}\| \leq \|\mathsf{X}_{\mathsf{ref}}^{-1}\| \cdot \sum_{\mathsf{i}=\mathsf{k}+1}^{\infty} \|\Delta \mathsf{X} \cdot \mathsf{X}_{\mathsf{ref}}^{-1}\|^{\mathsf{i}} = \|\mathsf{X}_{\mathsf{ref}}^{-1}\| \cdot \frac{\|\Delta \mathsf{X} \cdot \mathsf{X}_{\mathsf{ref}}^{-1}\|^{\mathsf{k}+1}}{1 - \|\Delta \mathsf{X} \cdot \mathsf{X}_{\mathsf{ref}}^{-1}\|} \tag{11}$$

Using the bound in inequality 11, we obtain bounds on X^{-1}:

$$\mathsf{Xp} - \mathsf{X}_{\mathsf{R}} \leq \mathsf{X}^{-1} \leq \mathsf{Xp} + \mathsf{X}_{\mathsf{R}} \tag{12}$$

Given the definition of $\mathsf{lXinv} = \mathsf{Xp} - \mathsf{X}_{\mathsf{R}}$ and $\mathsf{uXinv} = \mathsf{Xp} + \mathsf{X}_{\mathsf{R}}$, we obtain the lower and upper bound for X^{-1} as:

$$\mathsf{lXinv} \leq \mathsf{X}^{-1} \leq \mathsf{uXinv}. \tag{13}$$

□

Lemma 2. *For any given inputs — effective opacities* a*, colors* c *and depth* d*,* VR-IND *outputs the same value as computation from Line 13 to Line 18 in* GAUSSIANSPLAT.

Proof. We first derive the math expression of pixel color pc from GAUSSIANSPLAT. By denoting $\arg\mathrm{sort}(\mathsf{d})$ as a mapping $\sigma : \mathsf{I} \to \mathsf{I}$, where I refers to the index set of Gaussians, as at Line 13, cs at Line 14 and oc at Line 16 can be written as follows.

$$\mathsf{as}_{\mathsf{i}} = \mathsf{a}_{\sigma(\mathsf{i})} \tag{14}$$

$$\mathsf{cs}_{\mathsf{i}} = \mathsf{c}_{\sigma(\mathsf{i})} \tag{15}$$

$$\mathsf{oc}_{\mathsf{i}} = \prod_{\mathsf{j}=1}^{\mathsf{i}-1} 1 - \mathsf{a}_{\sigma(\mathsf{j})} \tag{16}$$

According to the definition of Ind operation, we have:

$$\mathrm{Ind}(\mathsf{d}_{\mathsf{i}} - \mathsf{d}_{\mathsf{j}}) = \begin{cases} 1 & \text{if } \mathsf{d}_{\mathsf{i}} > \mathsf{d}_{\mathsf{j}} \\ 0 & \text{if } \mathsf{d}_{\mathsf{i}} \leq \mathsf{d}_{\mathsf{j}}. \end{cases} \tag{17}$$

Then by replacing i, j with $\sigma(\mathsf{i}), \sigma(\mathsf{j})$, and multiplying $\mathsf{a}_{\sigma(\mathsf{i})}$, (17) becomes:

$$\mathsf{a}_{\sigma(\mathsf{i})} \cdot \mathrm{Ind}(\mathsf{d}_{\sigma(\mathsf{i})} - \mathsf{d}_{\sigma(\mathsf{j})}) = \begin{cases} \mathsf{a}_{\sigma(\mathsf{i})} & \text{if } \mathsf{d}_{\sigma(\mathsf{i})} > \mathsf{d}_{\sigma(\mathsf{j})} \\ 0 & \text{if } \mathsf{d}_{\sigma(\mathsf{i})} \leq \mathsf{d}_{\sigma(\mathsf{j})} \end{cases} \tag{18}$$

For any fixed index i, by multiplying all the case in the second branch of (18), it follows:

$$\prod_{\mathsf{j}=\mathsf{i}}^{N} (1 - \mathsf{a}_{\sigma(\mathsf{i})} \cdot \mathrm{Ind}(\mathsf{d}_{\sigma(\mathsf{i})} - \mathsf{d}_{\sigma(\mathsf{j})})) = 1. \tag{19}$$

By applying Equation 16, Equation 18 and Equation 19 into Line 16, oc_{i} can be written as follows:

$$\begin{aligned} \mathsf{oc}_{\mathsf{i}} &= \prod_{\mathsf{j}=1}^{\mathsf{i}-1} (1 - \mathsf{a}_{\sigma(\mathsf{i})}) \\ &= \prod_{\mathsf{j}=1}^{i-1} (1 - \mathsf{a}_{\sigma(\mathsf{i})} \cdot \mathrm{Ind}(\mathsf{d}_{\sigma(\mathsf{i})} - \mathsf{d}_{\sigma(\mathsf{j})})) \\ &= \prod_{\mathsf{j}=1}^{N} (1 - \mathsf{a}_{\sigma(\mathsf{i})} \cdot \mathrm{Ind}(\mathsf{d}_{\sigma(\mathsf{i})} - \mathsf{d}_{\sigma(\mathsf{j})})) \end{aligned} \tag{20}$$

Then by applying Equation 20 into Line 18, the pixel color pc computed via Algorithm BLENDSORT can be expressed as:

$$\mathsf{pc} = \sum_{\mathsf{i}=1}^{\mathsf{N}} \left(\prod_{\mathsf{j}=1}^{N} (1 - \mathsf{a}_{\sigma(\mathsf{i})} \cdot \mathrm{Ind}(\mathsf{d}_{\sigma(\mathsf{i})} - \mathsf{d}_{\sigma(\mathsf{j})})) \mathsf{a}_{\sigma(\mathsf{i})} \mathsf{c}_{\sigma(\mathsf{i})} \right) \tag{21}$$

Since the summation and product operations are invariant to the reordering of input elements, and both indices i and j iterate over the entire index set I, the reordering mapping σ in Equation 21 can be eliminated, resulting in:

$$\mathsf{pc} = \sum_{\mathsf{i}=1}^{\mathsf{N}} \left(\prod_{\mathsf{j}=1}^{N} (1 - \mathsf{a_i} \cdot \mathrm{Ind}(\mathsf{d_i} - \mathsf{d_j})) \mathsf{a_i} \mathsf{c_i} \right) \tag{22}$$

The last equation comes from 17.

Next, we derive the math expression for pc in Algorithm VR-IND. $\mathsf{oc_i}$ at Line 2 can be written as:

$$\mathsf{oc_i} = \prod_{\mathsf{j}=1}^{N} (1 - \mathsf{a_i} \cdot \mathrm{Ind}(\mathsf{d_i} - \mathsf{d_j})) \tag{23}$$

Then, by applying Equation 23 into Line 4 of Algorithm VR-IND, the expression for pc becomes:

$$\mathsf{pc} = \sum_{\mathsf{i}=1}^{\mathsf{N}} \left(\prod_{\mathsf{j}=1}^{N} (1 - \mathsf{a_i} \cdot \mathrm{Ind}(\mathsf{d_i} - \mathsf{d_j})) \mathsf{a_i} \mathsf{c_i} \right) \tag{24}$$

Note that the expressions for pc are identical in both Equation 22 and Equation 24. This demonstrates that Algorithms BLENDSORT and VR-IND yield the same input-output relationship while applying different operations, thus completing this proof. □

C Supplementary Experiment Results

C.1 Detailed Scenario Information

Table 5 summarizes scene configurations and rendering quality metrics for the scenarios discussed in Section 5. Figure 6 depicts representative GAUSSIANSPLAT-rendered images for each scene.

Table 5: Reconstruction quality for scenes are evaluated by the following metrics: NGauss: Number of Gaussians used for scene representation, applicable only to Gaussian Splatting; PSNR: Peak Signal-to-Noise Ratio; SSIM: Structural Similarity Index Measure; LPIPS: Learned Perceptual Image Patch Similarity.

Scene	Gaussian Splatting				NeRF		
	NGauss	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
`Lego`	43543	25.60	0.95	0.07	21.38	0.81	0.19
`Chair`	46108	22.98	0.94	0.08	22.56	0.89	0.11
`Drums`	50877	21.19	0.89	0.10	19.46	0.81	0.23
`Pinetree`	113368	31.06	0.97	0.06	22.41	0.79	0.20
`Airport`	617371	18.83	0.84	0.33	20.83	0.72	0.22
`Garden`	524407	18.74	0.37	0.32	22.15	0.80	0.20
`Plane`	51316	28.05	0.95	0.11	22.89	0.70	0.26
`Truck`	47895	24.70	0.94	0.09	23.53	0.75	0.19
`Car`	34699	26.98	0.93	0.10	20.75	0.69	0.27

Figure 6: Example rendered image for scenario (left to right, top to bottom): `Lego`, `Pinetree`, `Airport`, `Garden`, `Plane`, `Truck`, `Car` by GAUSSIANSPLAT.

C.2 Supplementary Results for Certified Classification

Figure 7 compares certified classification regions with those verified by sampling.

C.3 Supplementary Results for Certified Pose Estimation

Figure 8 compares certified estimated pose regions with those verified by sampling.

C.4 Supplementary Results for Certified Object Detection

Figure 9 compares certified detection with those verified by sampling.

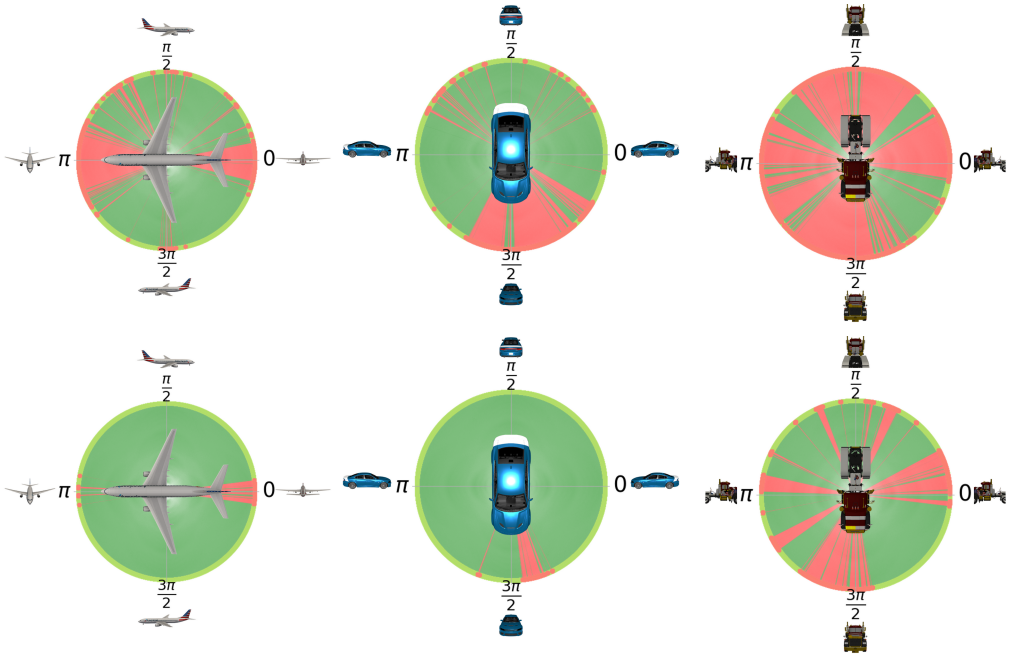

Figure 7: Classifier certification result for AIRPLANE, CAR, and TRUCK by GAUSSIANSPLAT across 0-360° camera rotation. Top row: Certified classification result. Bottom row: Classification result obtained via sampling. Green: Certified/Correct Camera Pose Region. Red: Uncertified/Incorrect Camera Pose Region.

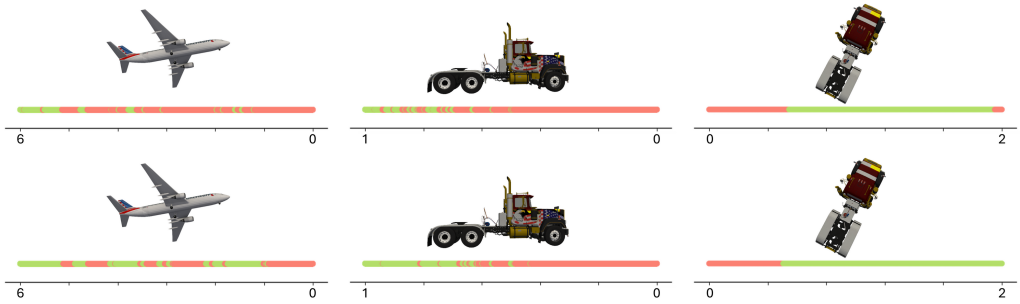

Figure 8: Pose estimation certification results for `Airplane`, `Truck` by GAUSSIANSPLAT scenes and `Truck` by a NERF scene. Top row: Certified pose estimation result. Bottom row: Pose estimation result obtained via sampling. Green: Certified/Correct Camera Regions. Red: Uncertified/Incorrect Camera Regions.

C.5 ABSTRACTRENDER Results for Scene Variations

In addition to handling camera pose variations as discussed in Section 5, ABSTRACTRENDER can also accommodate scene variations. For GAUSSIANSPLAT scenes, we consider variations in meaningful sets of 3D Gaussian parameters—such as color, mean position, and opacity of objects like trees in a street scene. For NERF scenes, we consider variations in hue or saturation across all 3D point colors in the scene. Table 6 presents ABSTRACTRENDER results under these scene variations for both GAUSSIANSPLAT and NERF representations. Figure 10 visualizes the abstract image of the `PineTree` (a GAUSSIANSPLAT scene) scene under uncertainty in color, mean position, and opacity of two tree objects. Figure 11 shows the abstract image of the `Lego` scene (a NERF scene) under uncertainty in hue and saturation across all 3D points.

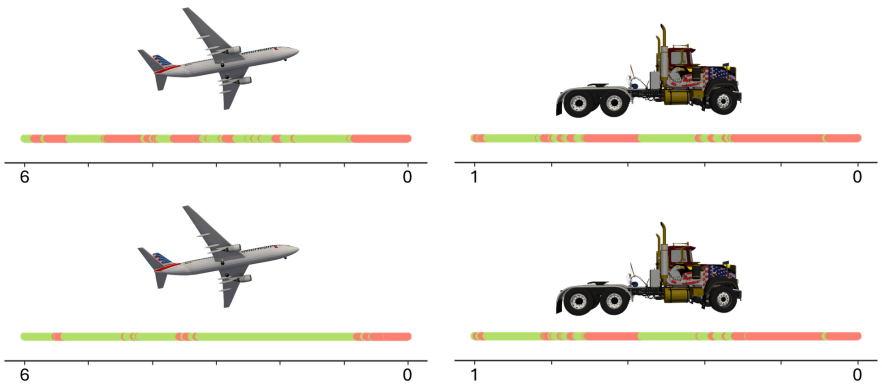

Figure 9: Object detection certification results for `Airplane` and `Truck` in GAUSSIANSPLAT scenes. Top row: Certified object detection result. Bottom row: Object detection result obtained via sampling. Green: Certified/Correct Camera Regions. Red: Uncertified/Incorrect Camera Regions.

Scene	PR	Dim	SR	Res	**Ours**		**Empirical**	
					MPG	XPG	MPG	XPG
Lego	0.15	hue	NeRF	80×80	0.17	1.51	0.14	1.45
Lego	0.30	satur	NeRF	80×80	0.10	1.34	0.09	1.33
Chair	0.20	hue	NeRF	80×80	0.65	1.71	0.37	1.71
Chair	0.50	satur	NeRF	80×80	0.76	1.73	0.52	1.73
Drums	0.20	hue	NeRF	80×80	0.63	1.20	0.62	1.20
Drums	0.50	satur	NeRF	80×80	0.96	1.36	0.88	1.36
PineTree	0.10	mean	GS	96×96	0.27	1.62	0.11	1.27
PineTree	0.20	mean	GS	96×96	0.41	1.73	0.25	1.43
PineTree	0.10	op	GS	96×96	0.29	1.53	0.14	1.25
PineTree	0.20	op	GS	96×96	0.59	1.73	0.27	1.23

Table 6: ABSTRACTRENDER results under scene variations, along with empirical bounds. PR: Perturbation Range; Dim: Perturbation Dimension, satur for saturation, mean for the mean's position of a set of Gaussians, op for opacity of a set of Gaussians; SR: Scene Representation; Res: Rendered Image Resolution; Rt: Runtime (min); MPG: mean pixel gap; XPG: maximum pixel gap.

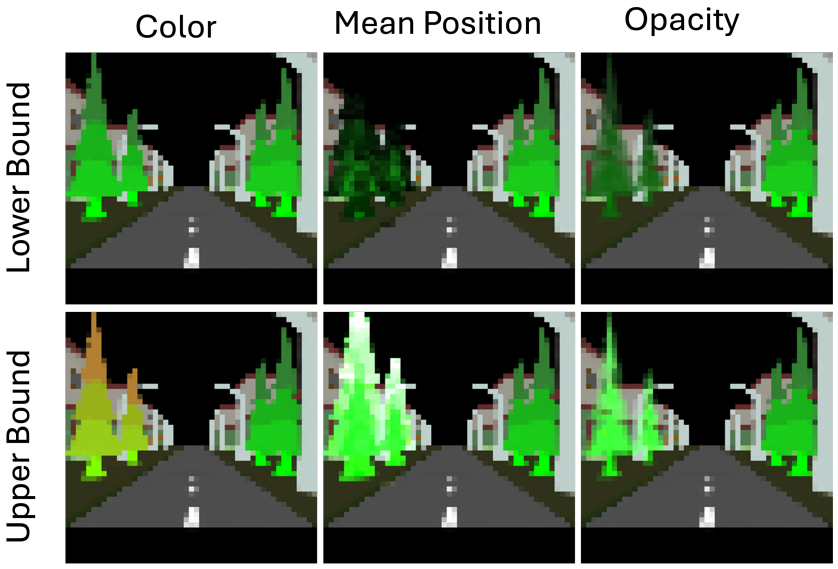

Figure 10: Lower (Top) and upper (Bottom) bound under Gaussian color (Left), mean (Mid) and opacity (Right) perturbation.

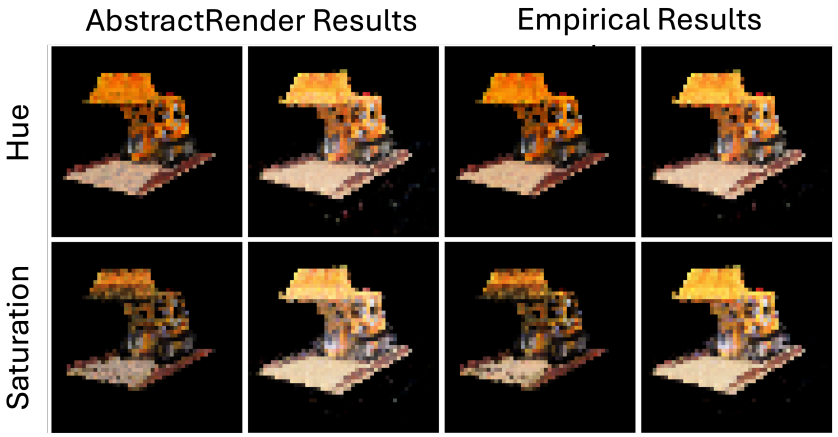

Figure 11: Comparison between ABSTRACTRENDER results and empirical results on `Legot` scene under perturbations in the hue of 3D points (top row) and saturation of 3D points (bottom row).

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: We clearly summarize our work's contribution and main application in the abstract.

   Guidelines:

   - The answer NA 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 NA 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: We discuss limitation of our work in Discussion section.

   Guidelines:

   - The answer NA 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: [Yes]

Justification: The assumptions are clearly stated and proof can be referred in supplementary material.

Guidelines:

- The answer NA 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: We upload our code to github and is reproductive.

   Guidelines:

   - The answer NA 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: Yes, we uploaded our code to github and is reproductive.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and test does not apply for our work, but we provide detail of experiment setting for ours and baseline.

   Guidelines:

   - The answer NA 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: [Yes]

   Justification: Our work computes rigorous bounds of output images/label. Statistical Significance does not apply to our work.

   Guidelines:

   - The answer NA 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: We run our experiment on 32G A100 GPU. Running times are also provide.

   Guidelines:

   - The answer NA 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: No human participants gets involved in our experiment.

   Guidelines:

   - The answer NA 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: Our work aims to verify the safety of downstream neural network, able to mitigating potential negative societal impacts of existing work.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work doesn't have such risks.

    Guidelines:

    - The answer NA 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: We have done this.

    Guidelines:

    - The answer NA 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 provide a readme for our upload code.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: Not relevant to human subjects.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: Not relevant to human subjects.

    Guidelines:

    - The answer NA 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 rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: Not involve LLM.

Guidelines:

- The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.