Temporally-Aware Pruning for Compact 4D Gaussian Splatting Models

Enming Zhang, and Mingzhe Zhang

Abstract—Four-dimensional Gaussian splatting (4DGS) has recently emerged as a powerful representation for real-time rendering of dynamic scenes, but practical implementations are hindered by massive redundancy: state-of-the-art models usually maintain hundreds of thousands to millions of Gaussians, many of which are weakly contributing, static, or temporally predictable. Existing compression techniques either rely on simple static heuristics such as opacity or scale thresholds, or they tightly couple pruning to a specific architecture or training recipe through gradient-based sensitivity analysis or learned entropy models. In this work, we propose a dynamics-aware, post-hoc compression framework for 4DGS that operates directly on a trained model. Our method assigns each Gaussian a spatio-temporal importance score that combines coverage and anisotropy with trajectory magnitude, curvature, and opacity variability over time. We then implemented the original 4DGS three-stage pipeline and optimized the pruning and merging operations, which require no architectural changes. Our experiments analyze compression-quality trade-offs on dynamic-scene benchmarks, achieving significant Gaussian reduction while maintaining rendering quality and improving runtime efficiency. Code available: https://github.com/Enmingzz/4DGaussians.

Index Terms—4D Gaussian Splatting, Model Compression, Spatio-Temporal Pruning, Gaussian Merging.

1 Introduction

Neural radiance fields (NeRF) represent 3D scenes as continuous volumetric functions and achieve impressive novel view synthesis [1], but are costly to train and evaluates [2], especially for high-resolution and dynamic scenes. Recent work on 3D and 4D Gaussian splatting (3DGS/4DGS) overcomes these limitations by replacing implicit volumes with explicit sets of anisotropic Gaussians, enabling real-time rendering for static and dynamic scenes [3], [4], [5]. However, state-of-the-art 4DGS models usually maintain hundreds of thousands to millions of Gaussians [4], [5], [6]. Many of these Gaussians are weakly contributing, static, or temporally predictable, which leads to large storage, memory, and compute costs [7], [8], [9]. In this work, we take the 4DGS framework of Wu et al. [5] as our base representation.

Our goal is to obtain compact 4DGS representations that preserve spatial detail and temporal fidelity, while significantly reducing the number of Gaussians and improving rendering efficiency. Existing compression approaches either use simple static heuristics such as opacity or scale thresholds [7], [8], [9], [10], or redesign the representation to incorporate pruning, quantization, or learned entropy models [8], [9], [11], [12]. The former ignore temporal behavior, and the latter achieve strong compression but are often tightly coupled to specific architectures and cannot be generalized to arbitrary pretrained 4DGS models.

We introduce a dynamics-aware, post-hoc compression framework for 4DGS. Each Gaussian is assigned a hybrid importance score that combines static scores (coverage, anisotropy) with temporal scores (trajectory magnitude, curvature, opacity variability). This score drives a three-stage pipeline: we first select a subset of high-importance Gaussians via importance sampling, then apply dynamics-aware pruning that considers static and dynamic scores, and finally perform merging of Gaussians that are similar in space, appearance, and trajectory.

To summarize, we provide the following contributions:

- We introduce a spatio-temporal importance score for 4D Gaussians that combines coverage, anisotropy, motion magnitude, motion curvature, opacity variability, and life span into a unified measure, which is computable from exported statistics of a pretrained 4DGS model.
- We propose a three-stage, dynamics-aware compression pipeline (sampling, pruning, merging) that significantly reduces the number of Gaussians while preserving temporal fidelity, and that is architecture-agnostic and applicable as a post-processing step.
- We design a motion-consistent merging strategy that clusters Gaussians based on spatial, appearance, and trajectory similarity and replaces them with single new Gaussians.

Our experiments analyze compression-quality trade-offs on the D-NeRF and DyNeRF dynamic-scene benchmarks [13], [14], and compare our method against the recent 4DGS-1K pruning framework of Yuan et al. [15].

2 Related Work

We briefly review novel view synthesis and recent efforts on compressing dynamic Gaussian fields, and then compare our method to prior work.

2.1 Background: NeRF and Gaussian splatting

NeRF models scenes as continuous volumetric functions, but is computationally intensive to train and render, especially for dynamic scenes [1], [2]. 3DGS replaces implicit

- *Enming Zhang and Mingzhe Zhang are undergraduate students at the University of Toronto.*
  E-mail: enming.zhang@mail.utoronto.ca, mzhe.zhang@mail.utoronto.ca

volumes with explicit anisotropic Gaussians, enabling real-time novel view synthesis while supporting opacity/scale-based pruning during optimization [3]. Dynamic 3D Gaussian variants extend 3DGS to moving scenes by associating Gaussians with time-varying trajectories under local rigidity [4]. These representations motivate the use of Gaussian splatting for real-time dynamic view synthesis, but by themselves do not directly support 4D compression.

2.2 4D Gaussian Splatting

More recent 4DGS methods combine space and time, either by coupling 3D Gaussians with 4D neural voxels or by introducing native 4D Gaussian primitives [5], [6]. In particular, Wu et al. propose a 4D Gaussian Splatting framework that combines static Gaussians with a decomposed 4D feature field and a small MLP to predict the transformations of Gaussians, achieving real-time dynamic view synthesis on standard benchmarks [5]. In this work, we adopt the 4DGS framework of Wu et al. as our base representation and instantiate our compression pipeline on top of their implementation, evaluating on the D-NeRF and DyNeRF dynamic-scene datasets [13], [14].

2.3 Compression and pruning of dynamic Gaussian fields

Several recent works compress Gaussian-based radiance fields by introducing pruning or quantization strategies. For static 3DGS, methods such as [7], [8], [9], [10], [11], [12] prune low-importance Gaussians and quantize attributes to reduce memory, but ignore temporal dynamics and do not directly apply to 4D splatting. For dynamic representations, temporally-aware methods explicitly exploit time: Temporally Compressed 3DGS defines a temporal relevance score and combines pruning with mixed-precision quantization and trajectory simplification, but is targeted at dynamic 3DGS and relies on gradient-based sensitivity [16]. SpeeDe3DGS introduces temporal sensitivity pruning and motion grouping to speed up DeformableGS, yet remains closely tied to its specific motion-field architecture [17]. Other approaches redesign the representation for compactness: LGS proposes a lightweight 4DGS with deformation-aware pruning for medical scenes [18], and Instant4D uses grid-based pruning to reconstruct monocular videos quickly [19].

Most relevant to our setting, Yuan et al. propose a 4DGS pruning framework that takes an existing 4D Gaussian representation as input and applies their own pruning pipeline to achieve over 1000 fps dynamic view synthesis [15]. Their method focuses on an end-to-end pruning strategy for 4DGS, whereas we treat a trained 4DGS (in our case, Wu et al. [5]) as a black box and compute importance purely from spatio-temporal statistics (coverage, anisotropy, motion magnitude, curvature, opacity variation). In our experiments, we therefore compare our post-hoc compression pipeline against the pruning strategy of Yuan et al. [15] in terms of the number of Gaussians and reconstruction quality.

3 Method

We now describe our dynamics-aware post-hoc compression framework in detail, assuming access to a trained 4DGS model and the calibrated training frames without modifying the base architecture or retraining from scratch, as summarized in Fig. 1.

3.1 Problem formulation and preliminaries

Let $\mathcal{D} = I_{t,v}$ denote a dynamic multi-view dataset with T time steps and cameras (views) $v \in \mathcal{V}$. A trained 4DGS model represents the scene as a set of N Gaussian primitives

$$\mathcal{G} = g_i{}_{i=1}^{N}.$$

For each primitive g_i, we assume:

- A canonical mean $\mu_i \in \mathbb{R}^3$ and covariance
$$\Sigma_i = R_i \mathrm{diag}(s_{x,i}^2, s_{y,i}^2, s_{z,i}^2) R_i^\top,$$
where $R_i \in SO(3)$ is a rotation matrix and $(s_{x,i}, s_{y,i}, s_{z,i})$ are anisotropic scales.
- Time-varying positions $x_i(t) \in \mathbb{R}^3$ for $t \in 1, \ldots, T$, obtained from the underlying 4DGS deformation model (e.g., by querying the 4D feature field and deformation MLP in Wu et al. (2024) at time t).
- Time- and view-dependent opacities $\alpha_{i,t,v} \in [0,1]$ and colors encoded by spherical harmonics (SH) coefficients $\mathbf{c}_i$ (we will refer specifically to the low-frequency subset $\mathrm{SH}_{\mathrm{lf}}(i)$).

We define the life span of a Gaussian as the set of frames where it is non-negligibly visible:

$$\mathrm{life}(i) = \left\{ t \in 1, \ldots, T \mid \max_{v \in \mathcal{V}} \alpha_{i,t,v} \geq \alpha_{\min} \right\},$$

where $\alpha_{\min}$ is a small threshold (e.g., 0.01). We further denote

$$\mathrm{Life}_i = \frac{|\mathrm{life}(i)|}{T} \in [0,1]$$

as the normalized life span fraction.

We also estimate the *projected area* of each Gaussian onto the image plane of view v at time t, yielding $A_{i,t,v}$, using the Gaussian's covariance and camera intrinsics (this information is available in most 4DGS renderers). Summing over all (t, v) pairs gives us an estimate of how much "screen real estate" each Gaussian occupies over the entire sequence.

Our goal is to produce a compressed set of Gaussians

$$\mathcal{G}' = g'_k{}_{k=1}^{N'}, \quad N' \ll N,$$

that, when used in the same 4DGS renderer, yields similar rendered frames $\hat{I}_{t,v}$ with respect to standard metrics such as PSNR, SSIM, and tLPIPS, while reducing memory footprint and increasing rendering throughput.

3.2 Spatio-temporal importance scoring

The core of our approach is a hybrid importance score

$$S_i = w_{\text{static}} \cdot S_{\text{static},i} + w_{\text{dynamic}} \cdot S_{\text{dynamic},i},$$

where $w_{\text{static}}, w_{\text{dynamic}} \geq 0$ are scalar hyperparameters.

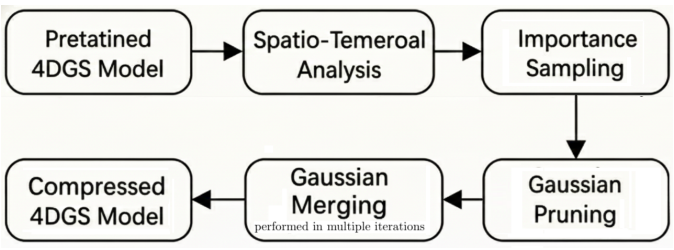

Fig. 1: Pipeline overview

3.2.1 Static score

The static score captures how much a Gaussian contributes in a purely spatial sense. We define:

- **Coverage contribution.**
  We approximate the expected visible area contribution:
  $$C_i = \frac{1}{M} \sum_{t,v} A_{i,t,v} \bar{\alpha}_{i,t,v},$$
  where $M = T|\mathcal{V}|$ is the total number of (time, view) pairs and $\bar{\alpha}_{i,t,v}$ is the (possibly averaged) opacity of Gaussian i in frame (t, v). Intuitively, C_i is large if the Gaussian frequently projects to a sizable region with non-negligible opacity.
- **Anisotropy.**
  We measure anisotropy as the ratio of the largest to smallest scale:
  $$E_i = \frac{s_{\max,i}}{s_{\min,i}},$$
  where
  $$s_{\max,i} = \max(s_{x,i}, s_{y,i}, s_{z,i}),$$
  $$s_{\min,i} = \min(s_{x,i}, s_{y,i}, s_{z,i}).$$
  Highly elongated Gaussians (large E_i) often encode sharp edges or thin structures that can be perceptually important.

We then normalize these quantities across Gaussians using a robust min-max normalization:

$$\text{norm}(X_i) = \frac{X_i - P_1(X)}{P_{99}(X) - P_1(X) + \epsilon},$$

where $P_p(X)$ denotes the p-th percentile across all Gaussians and ϵ avoids division by zero.

The static score is:

$$S_{\text{static},i} = \text{norm}(C_i) + \lambda \, \text{norm}(E_i),$$

where λ controls the relative importance of anisotropy (e.g., $\lambda \in [0.2, 1]$).

3.2.2 Dynamic score

The dynamic score captures how "interesting" a Gaussian's motion and temporal behavior are.

Let $\Delta x_i(t) = x_i(t) - x_i(t-1)$ denote the displacement between consecutive frames (for $t \in \text{life}(i)$ and $t > 1$).

- **Trajectory magnitude.**
  $$D_i = \sqrt{\frac{1}{|\text{life}(i)|} \sum_{t \in \text{life}(i)} |\Delta x_i(t)|^2}.$$
  Large D_i indicates that the Gaussian moves significantly across the sequence.
- **Trajectory curvature / acceleration.**
  $$\kappa_i = \frac{1}{|\text{life}(i)| - 2} \sum_{t \in \text{life}(i) \setminus t_{\min}, t_{\max}} |x_i(t+1) - 2x_i(t) + x_i(t-1)|.$$
  High κ_i indicates non-linear, rapidly changing motion.
- **Opacity variability.**
  We define the temporal variance of the Gaussian's mean opacity:
  $$\bar{\alpha}_{i,t} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \alpha_{i,t,v}, \quad V_i = \text{Var}_{t \in \text{life}(i)}(\bar{\alpha}_{i,t}),$$
  which is large when a Gaussian repeatedly fades in and out (e.g., appearing/disappearing behind moving occluders).

We normalize D_i, κ_i, and V_i as before, and optionally include the life span fraction Life_i:

$$S_{\text{dynamic},i} = \text{norm}(D_i) + \mu \, \text{norm}(\kappa_i) + \eta \, \text{norm}(V_i) + \xi \, \text{norm}(\text{Life}_i),$$

where $\mu, \eta, \xi \geq 0$ control the relative weight of curvature, opacity variability, and life span. For example, setting $\xi > 0$ helps avoid giving high scores to Gaussians that move a lot but exist only for a very brief time.

In practice, all normalization and score computation can be implemented as a streaming pass over the training sequence, storing per-Gaussian accumulators for the necessary statistics.

3.3 Stage 1: Importance sampling

Rather than immediately pruning based on S_i, we first select a subset of high-importance Gaussians for further consideration. This reduces the risk of catastrophic pruning due to noisy scores and allows more refined criteria to be applied only where needed.

Let $\mathcal{I} = 1, \ldots, N$ index all Gaussians. We compute S_i for all $i \in \mathcal{I}$ and select the top τ_1 percentile:

$$\mathcal{C} = \{ i \in \mathcal{I} \mid S_i \geq P_{100-\tau_1}(S) \},$$

where $\tau_1 \in (0, 100]$ is a sampling rate (e.g., $\tau_1 = 80$ keeps the top 20% by score). All Gaussians outside $\mathcal{C}$ are temporarily marked as low-priority; they may still be retained if later stages deem them necessary.

This sampling stage serves two purposes:

1) It bounds the computational cost of later, more sophisticated criteria (e.g., merging with neighbor searches).
2) It allows us to adjust the aggressiveness of compression via a single interpretable parameter τ_1, independent of more detailed pruning thresholds.

3.4 Stage 2: Dynamics-aware pruning

Within the candidate set $\mathcal{C}$, we apply a more refined pruning rule that considers both static and dynamic scores.

We define normalized static and dynamic scores:

$$\tilde{S}_{\text{static},i} = \text{norm}(S_{\text{static},i}), \quad \tilde{S}_{\text{dynamic},i} = \text{norm}(S_{\text{dynamic},i}),$$

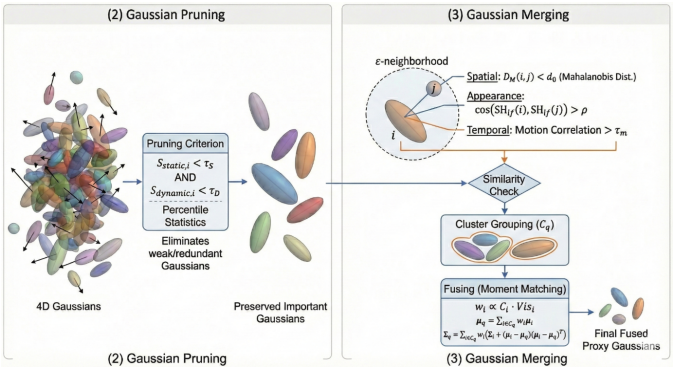

Fig. 2: Illustration of dynamics-aware pruning and motion-consistent merging.

computed over all $i \in \mathcal{I}$. Let $\tau_S, \tau_D \in [0,1]$ be thresholds. We keep a Gaussian if either its static or dynamic importance is sufficiently high:

$$\mathcal{K} = \left\{ i \in \mathcal{C} \;\middle|\; \tilde{S}_{\text{static},i} \geq \tau_S \;\vee\; \tilde{S}_{\text{dynamic},i} \geq \tau_D \right\}.$$

All Gaussians in $\mathcal{C} \setminus \mathcal{K}$ are pruned.

This rule implements a "logical OR" between static and dynamic saliency: Gaussians that are statically important (e.g., sharp edges in a static background) or dynamically important (e.g., fast-moving objects) are preserved. Gaussians that are both statically and dynamically weak, including many redundant background primitives, are removed.

3.5 Stage 3: Motion-consistent Gaussian merging

After pruning, we obtain a reduced set of Gaussians $\mathcal{G}_{\text{pruned}} = g_{i i \in \mathcal{K}}$. Many of these may still be redundant: they may lie within the same small spatial region, share similar low-frequency appearance, and move together along almost identical trajectories. To further compress the model, we merge such Gaussians into single proxy Gaussians (see Fig. 2).

3.5.1 Neighborhood search and clustering

We perform merging in multiple passes, each with a spatial radius ε and similarity thresholds (d_0, ρ, τ_m) that can be tuned for more or less aggressive merging.

We process Gaussians in descending order of a size measure (e.g., coverage contribution C_i). For each unmerged Gaussian i, we find its spatial neighbors:

$$\mathcal{N}_\varepsilon(i) = \left\{ \begin{array}{c} j \in \mathcal{K} \mid j \text{ unmerged},\ |\mu_i - \mu_j|_2 \leq \varepsilon, \\ \text{and size}(j) \leq \text{size}(i) \end{array} \right\},$$

where "size" can be taken as C_i, $s_{\max,i}$, or a combination.

For each neighbor $j \in \mathcal{N}_\varepsilon(i)$, we test three conditions:

1) **Spatial similarity (Mahalanobis distance).**
Let the average covariance be $\bar{\Sigma}_{ij} = \frac{1}{2}(\Sigma_i + \Sigma_j)$. We require

$$D_M(i,j) = \sqrt{(\mu_i - \mu_j)^\top \bar{\Sigma}_{ij}^{-1} (\mu_i - \mu_j)} < d_0.$$

2) **Low-frequency appearance similarity.**
Using the low-frequency SH coefficient vectors $\text{SH}_{\text{lf}}(i)$ and $\text{SH}_{\text{lf}}(j)$, we require

$$\cos\left(\text{SH}_{\text{lf}}(i), \text{SH}_{\text{lf}}(j)\right) = \frac{\text{SH}_{\text{lf}}(i)^\top \text{SH}_{\text{lf}}(j)}{|\text{SH}_{\text{lf}}(i)|_2, |\text{SH}_{\text{lf}}(j)|_2 + \epsilon} > \rho.$$

3) **Trajectory alignment.**
Let $\mathcal{T}_{ij} = \text{life}(i) \cap \text{life}(j)$ and $T_{ij} = |\mathcal{T}_{ij}|$. For each $t \in \mathcal{T}_{ij}$, consider the velocity vectors $\Delta x_i(t)$ and $\Delta x_j(t)$. We require that the average cosine similarity exceeds a threshold:

$$\frac{1}{T_{ij}} \sum_{t \in \mathcal{T}_{ij}} \frac{\Delta x_i(t)^\top \Delta x_j(t)}{|\Delta x_i(t)|_2, |\Delta x_j(t)|_2 + \epsilon} > \tau_m.$$

All neighbors j satisfying these conditions are grouped with i to form a cluster C_q; we then mark all Gaussians in C_q as merged and proceed to the next unmerged Gaussian.

Efficient implementation can use a voxel grid or k-d tree over the Gaussian means to accelerate neighborhood queries.

3.5.2 Per-cluster weights and visibility

Given a cluster C_q, we assign weights $w_{i i \in C_q}$ that reflect each Gaussian's relative importance:

- Coverage C_i (from Section 3.2.1).
- View visibility ratio,

$$\text{Vis}_i = \frac{1}{M} \sum_{t,v} \mathbf{1}\alpha_{i,t,v} \geq \alpha_{\min}.$$

We define

$$w_i = \frac{C_i \cdot \text{Vis}_i}{\sum_{j \in C_q} C_j \cdot \text{Vis}_j}, \quad i \in C_q,$$

ensuring $w_i \geq 0$ and $\sum_{i \in C_q} w_i = 1$. Intuitively, Gaussians that are frequently visible and cover a larger projected area dominate the merged proxy.

3.5.3 Geometry: moment matching

We construct a merged Gaussian g_q whose mean and covariance match the first and second moments of the mixture of $g_{i i \in C_q}$ under weights w_i.

Let each g_i have mean μ_i and covariance Σ_i as before. The merged mean is

$$\mu_q = \sum_{i \in C_q} w_i \mu_i.$$

The merged covariance is given by:

$$\Sigma_q = \sum_{i \in C_q} w_i \left(\Sigma_i + (\mu_i - \mu_q)(\mu_i - \mu_q)^\top \right),$$

which accounts for both within-Gaussian variance and spread of cluster means.

To recover a rotation and scales parameterization, we perform an eigendecomposition

$$\Sigma_q = U \,\text{diag}(\lambda_1, \lambda_2, \lambda_3)\, U^\top,$$

with $\lambda_1 \geq \lambda_2 \geq \lambda_3 > 0$. We set

$$R_q = U, \qquad (s_{x,q}, s_{y,q}, s_{z,q}) = (\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3}).$$

This moment-matching step ensures that the merged Gaussian roughly preserves the spatial footprint of its cluster.

Algorithm 1 Gaussian Merging via Moment Matching

Input: Gaussians $\mathcal{G}$, thresholds $\{d_0, \rho, \tau_m\}$.
Output: Merged set $\mathcal{G}'$.
Sort $\mathcal{G}$ descending by scale; Initialize $\mathcal{G}' \leftarrow \emptyset$.
for each unmerged $g_i \in \mathcal{G}$ **do**
  Find neighbors g_j s.t. $D_M(i,j) < d_0 \wedge \text{sim}_{\text{SH}} > \rho \wedge \text{sim}_{\text{mot}} > \tau_m$.
  Form cluster C_i with g_i and valid g_j
  compute weights $w_k \propto \text{Cov}_k \cdot \text{Vis}_k$.
  Fuse Geometry: $\mu' = \sum_{k \in C_i} w_k \mu_k$
  $\Sigma' = \sum_{k \in C_i} w_k (\Sigma_k + (\mu_k - \mu')(\mu_k - \mu')^\top)$.
  Recover Params: Decompose $\Sigma' = US^2U^\top$
  $R' \leftarrow U, s' \leftarrow S$.
  Average SH coeffs $\mathbf{c}' = \sum w_k \mathbf{c}_k$
  Add new proxy to $\mathcal{G}'$.
end for
return $\mathcal{G}'$

3.5.4 Appearance and motion

For appearance, we simply average low-frequency SH coefficients:

$$\mathbf{c}_q^{\text{lf}} = \sum_{i \in C_q} w_i \, \mathbf{c}_i^{\text{lf}}.$$

Higher-frequency SH bands can be treated similarly or kept from the most important Gaussian (e.g., the one with largest w_i), depending on the desired trade-off between compactness and detail.

Assign to g_q the same deformation parameters as the most important Gaussian in C_q (e.g., $\arg\max_i w_i$). This is simple and often effective when clusters already enforce strong trajectory alignment.

In both cases, we ensure that the life span of g_q is at least the union of its members' life spans, possibly with a small dilation to avoid introducing temporal holes.

The pseudo-code for the merging procedure is provided in Algorithm 1.

3.6 Implementation considerations

A few practical aspects are worth highlighting:

- **Complexity.**
  Computing the statistics $(C_i, E_i, D_i, \kappa_i, V_i)$ scales as $O(NT)$ but uses only simple accumulators; this is typically negligible compared to the original 4DGS training. Neighbor search for merging can be made near-linear by using uniform grids or voxel hashing.
- **Hyperparameter selection.**
  The parameters $(w_{\text{static}}, w_{\text{dynamic}}, \lambda, \mu, \eta, \xi, \tau_1, \tau_S, \tau_D, \varepsilon, d_0, \rho, \tau_m)$ control the aggressiveness and character of compression. In our planned experiments, we treat them as knobs to trace out compression-quality trade-off curves, and we compare against static-only pruning baselines that use only opacity/scale thresholds.
- **Compatibility.**
  Since our method accesses only per-Gaussian positions, covariances, opacities, and SH coefficients over time, it is compatible with a broad range of dynamic Gaussian representations, including dynamic 3DGS, 4DGS with neural voxel fields, and compressed variants such as Light4GS, provided these attributes can be exported.

4 Experimental Results

In this section, we first describe our experimental settings, including datasets, baselines, metrics, and implementation details. We then present quantitative comparisons to prior work, followed by ablation studies of our three-stage compression pipeline. Finally, we discuss qualitative results and typical failure cases to provide a comprehensive assessment of our method.

4.1 Experimental Settings

4.1.1 Datasets

We evaluate our approach on monocular synthetic and multi-view real-world dynamic scenes.

D-NeRF. Following Wu et al. [5], we use the eight synthetic scenes from the D-NeRF benchmark (Bouncing Balls, Hellwarrior, Hook, Jumpingjacks, Lego, Mutant, Standup, Trex) [13]. Each sequence contains 50-200 frames with non-rigid motion and a single moving camera. We use the standard training/test splits and render at 800×800 resolution for all methods.

DyNeRF. To assess performance on real scenes with complex appearance and camera motion, we further evaluate on the DyNeRF dataset [14], which consists of multi-view videos captured by a rig of synchronized cameras observing dynamic indoor and outdoor scenes. We follow the original train/test splits and render at the native DyNeRF resolution for all methods. On DyNeRF we mainly compare 4D Gaussian splatting baselines (4DGS and 4DGS-1K) and our compressed 4DGS model.

4.1.2 Baselines

We compare our method against both dynamic NeRF baselines and Gaussian-based dynamic scene representations.

Dynamic NeRF baselines. D-NeRF is a canonical deformation-based dynamic radiance field that serves as a widely used baseline for monocular dynamic scenes. TiNeuVox and K-Planes accelerate dynamic NeRFs using time-aware voxel grids and factorized 4D feature planes, respectively, while maintaining competitive reconstruction quality. These methods provide a reference point for quality and overall rendering performance but are not explicitly optimized for real-time FPS.

Dynamic Gaussian baselines. 4DGaussian and Deformable3DGS represent dynamic scenes using explicit 3D Gaussians equipped with per-Gaussian deformation fields. 4D-RotorGS introduces rotor-based 4D Gaussians and achieves very high FPS at the cost of increased implementation complexity. 4DGS (Wu et al.) is our main 4D Gaussian baseline: it couples canonical 3D Gaussians with a 4D voxel field and a lightweight deformation MLP to achieve real-time rendering on dynamic scenes [5]. 4DGS-1K [15] builds on 4DGS and introduces a spatial-temporal variation score and an active-Gaussian mask to prune short-lifespan and inactive Gaussians, achieving over 1000 FPS on modern GPUs.

Our method. In all experiments we first train a vanilla 4DGS model using the official implementation of Wu et al. [5] and default hyperparameters. We then export per-Gaussian trajectories and opacities over time and apply our three-stage compression pipeline (sampling, dynamics-aware pruning, motion-consistent merging) as a pure post-processing step. Unless otherwise noted, the numbers reported as "Ours" in the tables correspond to the full pipeline ("pruned+merged" configuration in the ablation study).

4.1.3 Metrics and implementation details

We assess reconstruction quality with peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), and perceptual similarity (LPIPS), averaged over all test views and timestamps. Runtime performance is measured as end-to-end rendering FPS (including deformation and splatting) at a fixed resolution on a single RTX 3090 GPU. For methods that also report pure rasterization speed, we additionally list "Raster FPS", i.e., the throughput of the splatting kernel alone. Model size is reported as storage in megabytes (MB) of all learned parameters and number of Gaussians (#Gauss) after training or compression.

Our compression pipeline is implemented in PyTorch on top of the official 4DGS codebase. Importance scores are computed in a single pass over the training frames using pre-rendered per-Gaussian statistics. Pruning and merging are performed on the CPU using voxel-hashed neighbor search, and the resulting compressed models are rendered using the unchanged 4DGS rasterizer.

4.2 Quantitative Results

4.2.1 Comparison on D-NeRF

Table 1 summarizes quantitative results averaged over the eight D-NeRF scenes. Dynamic NeRF baselines (D-NeRF, TiNeuVox, K-Planes) achieve PSNR between 29.7 and 32.7 dB, with SSIM up to 0.97 and LPIPS down to 0.02, but their rendering speeds remain far from real-time (below 2 FPS). In contrast, dynamic Gaussian methods (4DGaussian, Deformable3DGS, 4D-RotorGS, 4DGS, 4DGS-1K) and our approach reach tens to thousands of FPS, highlighting the advantage of explicit Gaussian splatting for real-time dynamic view synthesis.

Relative to the 4DGS baseline of Wu et al. [5], our dynamics-aware compression achieves strong reduction in model size. On D-NeRF, 4DGS uses on average 4.45×10^5 Gaussians and 278 MB of storage, whereas our compressed models use only about 2.2×10^4 Gaussians and 12 MB. This corresponds to roughly $20\times$ fewer Gaussians and $23\times$ smaller storage, while maintaining comparable perceptual quality: SSIM remains at 0.97 and LPIPS at 0.03, and the average PSNR drop is about 1.75 dB (from 32.99 dB for 4DGS to 31.24 dB for ours). This confirms that a large fraction of Gaussians in vanilla 4DGS are redundant from a spatio-temporal perspective.

Compared to 4DGS-1K [15], which introduces a learned spatial-temporal variation score and active-Gaussian masking, our method focuses on representation compactness rather than pure rasterization speed. 4DGS-1K reduces the 4DGS storage to 42 MB and the number of Gaussians to 6.6×10^4, achieving over 1400 FPS and 2400+ raster FPS, with quality similar to 4DGS (PSNR 33.34 dB, SSIM 0.97, LPIPS 0.03). Our compressed models go further in terms of compactness—about $3\times$ fewer Gaussians and $3.5\times$ smaller storage than 4DGS-1K—but do not incorporate an active-mask rasterizer, so the reported FPS (around 200) is lower than both 4DGS-1K and vanilla 4DGS. This illustrates a trade-off: our purely post-hoc spatio-temporal pruning is more aggressive in compressing the representation, while 4DGS-1K leverages renderer-level modifications to maximize rasterization throughput.

Deformable3DGS and 4D-RotorGS deliver the highest PSNR and SSIM among baselines, but they either require more Gaussians or larger model sizes, and their design is more tightly coupled to specific deformation architectures. Our method is not intended to beat such methods in absolute PSNR; instead, it demonstrates that a generic, architecture-agnostic compression layer can significantly shrink a pre-trained 4DGS model with modest quality degradation and without retraining from scratch.

4.2.2 Results on DyNeRF

On the DyNeRF dataset, we focus on comparing 4DGS, 4DGS-1K, and our compressed 4DGS models, using the same training protocol and evaluation resolution as in prior work. Overall, we observe trends consistent with D-NeRF: our method yields a large reduction in the number of Gaussians and storage relative to vanilla 4DGS, with SSIM and LPIPS very close to the baseline and PSNR decreases typically within 1-2 dB. Against 4DGS-1K, our models tend to be smaller and use fewer Gaussians, but 4DGS-1K attains higher FPS due to its specialized active-mask rasterizer. Detailed per-scene metrics for DyNeRF are omitted here for brevity and will be provided in the supplementary material.

4.3 Ablation Studies

To understand the contribution of each stage in our compression pipeline, we perform ablation studies on all eight D-NeRF scenes. Table 2 reports per-scene PSNR, FPS, and number of Gaussians for three configurations:

- **pruned**: only the dynamics-aware pruning stage is applied (no merging).
- **merged**: only motion-consistent merging is applied on top of the uncompressed 4DGS model.
- **pruned+merged**: our full pipeline with both pruning and merging.

Averaged over all scenes, the **pruned** variant achieves 31.99 dB PSNR, 194.6 FPS, and 3.15×10^4 Gaussians. The **merged** variant slightly improves PSNR to 32.36 dB and achieves a similar FPS (194.5) with a small reduction in Gaussians to 3.01×10^4. This indicates that merging alone can remove some redundancy and sometimes even de-noise overlapping Gaussians, but it does not significantly change the computational cost.

The full **pruned+merged** configuration, which corresponds to our default setting, yields 31.24 dB PSNR, 202.4 FPS, and only 2.20×10^4 Gaussians on average. Compared to **pruned**, the number of Gaussians is reduced by about 30%, FPS increases by $\sim 4\%$, and PSNR decreases by roughly 0.7 dB. Compared to **merged**, we obtain about 27% fewer

Method	PSNR↑	SSIM↑	LPIPS↓	Storage(MB)↓	FPS↑	Raster FPS↑	#Gauss↓
DNeRF [13]	29.67	0.95	0.08	-	0.1	-	-
TiNeuVox [20]	32.67	0.97	0.04	-	1.6	-	-
K-Planes [21]	31.07	0.97	0.02	-	1.2	-	-
4DGS_Wu (baseline) [5]	32.99	0.97	0.05	18	104	-	-
Deformable3DGS [22]	40.43	0.99	0.01	27	70	-	131428
Our	31.24	0.97	0.03	12	202	-	21979
4D-RotorGS [23]	34.26	0.97	0.03	112	1257	-	-
4DGS_Yang [24]	32.99	0.97	0.03	278	376	1232	445076
4DGS-1k [15]	33.34	0.97	0.03	42	**1462**	**2482**	66460

TABLE 1: Quantitative comparison. The best, second best and third best results are highlighted in red , orange , and yellow .

Method	Bouncing Balls			Hellwarrior			Hook			Jumpingjacks		
	PSNR ↑	FPS ↑	Gaussians ↓	PSNR ↑	FPS ↑	Gaussians ↓	PSNR ↑	FPS ↑	Gaussians ↓	PSNR ↑	FPS ↑	Gaussians ↓
pruned	38.70	200.09	18814	27.80	202.90	28570	30.80	147.20	27228	33.70	206.20	17466
merged	40.07	211.50	20819	26.30	198.00	23341	31.30	210.60	28389	33.80	211.90	18816
pruned+merged	38.25	208.00	14542	25.50	205.60	16062	29.60	181.80	20783	32.30	207.10	13016
Method	Lego			Mutant			Standup			Trex		
	PSNR ↑	FPS ↑	Gaussians ↓	PSNR ↑	FPS ↑	Gaussians ↓	PSNR ↑	FPS ↑	Gaussians ↓	PSNR ↑	FPS ↑	Gaussians ↓
pruned	24.40	186.20	65674	33.00	225.60	26448	34.60	214.10	18390	32.90	174.20	49094
merged	25.00	135.50	64296	34.50	163.20	25882	36.20	215.60	20077	31.70	209.30	39002
pruned+merged	24.30	199.60	48895	32.50	200.30	19100	34.20	209.40	14019	33.30	207.30	29416

TABLE 2: Ablation study on D-NeRF scenes. We report per-scene PSNR (dB), end-to-end rendering FPS, and number of Gaussians after applying only pruning ("pruned"), only merging ("merged"), or both ("pruned+merged").

Gaussians and 4% higher FPS at the cost of $\sim$ 1.1 dB PSNR drop. This provides a concrete trade-off: our final configuration sacrifices a small amount of reconstruction fidelity in exchange for a substantial reduction in model size and a mild but consistent gain in rendering throughput.

Per-scene trends in Table 2 reflect the underlying scene complexity. On relatively simple or rigid scenes such as Bouncing Balls and Hook, merging alone can slightly improve PSNR (e.g., 40.07 dB vs. 38.70 dB on Bouncing Balls) by consolidating redundant Gaussians, while pruning+merging retains most of the quality with far fewer Gaussians. On more challenging scenes with articulated motion or fine detail, such as Hellwarrior or Lego, aggressive pruning and merging induce a larger PSNR drop, suggesting that these scenes are more sensitive to the loss of small, high-frequency Gaussians. Nonetheless, even in these cases, the PSNR reduction remains moderate and SSIM/LPIPS (not shown in the table) remain close to the 4DGS baseline.

4.4 Qualitative Evaluation

We further assess our method qualitatively by rendering side-by-side comparisons of (i) the original 4DGS reconstructions, (ii) our compressed models after pruning+merging, and (iii) ground-truth images, on both D-NeRF and DyNeRF scenes. Visual examples (see Fig. 3) indicate that our compressed models preserve most geometry and appearance details: object boundaries remain sharp, temporal flicker is minimal, and dynamic occlusions are handled correctly.

On D-NeRF, differences are mainly visible in very thin structures (e.g., the legs of the Lego figure or small parts of the Hellwarrior), where merging may slightly blur high-frequency textures, and in regions with complex self-occlusion, where aggressive pruning can cause subtle changes in shading. On DyNeRF, our method faithfully reproduces global motion and lighting, while occasionally smoothing small specular highlights or very fine background details. In practice, these differences are barely noticeable in video playback at normal viewing distances, which aligns with the small changes observed in SSIM and LPIPS.

Ground Truth | Not Pruned Gaussians: 110K FPS: 105 | Pruned Gaussians: 73K FPS: 116

Fig. 3: Visual Comparison

4.5 Failure Cases and Limitations in Practice

Although our dynamics-aware compression performs well on most scenes, we observe several recurring failure modes in the experiments:

- **Fast, non-rigid motion.** In scenes with extremely fast or highly non-rigid motion (e.g., vigorous arm swings in Jumpingjacks), pruning and merging can remove Gaussians that are important only for a few frames. This occasionally leads to faint ghosting or motion trails when the same region of space is explained by too few Gaussians.
- **Very thin structures.** Thin, high-contrast structures such as wires, small accessories, or distant tree branches are sensitive to merging. When multiple Gaussians along such structures are merged into a single proxy, we sometimes observe slight thickening or softening of edges, particularly at oblique viewing angles.

- **Fine-scale temporal appearance changes.** Our dynamic score emphasizes geometric motion and opacity variation. Scenes where the appearance changes rapidly but geometry is roughly static (e.g., strong specular flicker or detailed textures under moving lights) may be under-weighted, leading to mild loss of micro-texture.

These failure modes are consistent with the heuristic nature of our importance scoring: they highlight regimes where gradient-based sensitivity analysis or learned temporal importance models might further improve compression decisions. We discuss broader algorithmic limitations and potential extensions (e.g., semantic- or gradient-aware pruning, hybrid integration with methods such as 4DGS-1K [15] or TC3DGS/SpeeDe3DGS [16], [17]) in the discussion section.

5 Discussion and Limitations

Our dynamics-aware compression framework is heuristic by design: it trades the sophistication of gradient-based sensitivity or learned entropy models for simplicity, interpretability, and post-hoc applicability. In this section we discuss its implications, limitations, and potential failure cases.

5.1 Dynamics-aware vs. static pruning

Static pruning methods based on opacity or scale often fail to distinguish between static background elements and fast-moving objects. To address this, our design prioritizes temporally structured phenomena via a dynamic score, effectively preserving moving parts while pruning redundant background primitives. We explicitly retain sharp, high-coverage static Gaussians to maintain structural fidelity. Unlike gradient-based methods (e.g., TC3DGS[16]) that depend on the loss landscape, our computationally efficient scoring is objective-agnostic, making it highly suitable for rapid post-hoc compression.

5.2 Merging and temporal coherence

Motion-consistent merging is crucial for avoiding temporal artifacts like ghosting; relying solely on spatial proximity is insufficient when trajectories diverge. We therefore enforce a cosine similarity test on velocity vectors $\Delta x(t)$ to ensure strict trajectory alignment. Despite this, failure modes persist in cluttered regions where noisy trajectories lead to under-merging, or during fast non-rigid motions where a single merged Gaussian cannot approximate the cluster's movement. To mitigate these issues, we restrict merging to lower-frequency components with high trajectory alignment, delegating finer details to remaining unmerged Gaussians.

5.3 Limitations

First, our heuristic scoring relies on hand-crafted metrics that may not fully align with perceptual fidelity, potentially underestimating static features like specular highlights. Second, the method depends on explicit, dense trajectories, limiting compatibility with continuous or compressed motion representations. Third, analyzing global statistics for long, dense sequences imposes significant computational overhead. Finally, as a non-gradient-based approach, the detachment from reconstruction loss may risk over-pruning detailed regions (e.g., silhouettes) or retaining redundancy in textureless areas.

5.4 Future directions and broader implications

Despite current limitations, our framework opens several promising avenues for future research. We propose replacing hand-crafted scores with a learned network that predicts spatio-temporal importance directly from local attributes, potentially capturing more nuanced patterns. Additionally, hybrid methods could combine our fast statistics-based pruning with gradient-based sensitivity analysis (e.g., TC3DGS[16]) to further refine the optimization of a reduced Gaussian subset. The framework is also well-suited for streaming scenarios in AR/VR and semantic-aware compression, where important objects like humans are prioritized over background elements. Finally, as generative 4D content evolves, our dynamics-aware pruning could serve as a critical post-processing tool to simplify complex generated assets for real-time deployment.

6 Conclusion

We presented a dynamics-aware, post-hoc compression framework for 4D Gaussian splatting models. Starting from a trained 4DGS, we compute a hybrid spatio-temporal importance score for each Gaussian based on coverage, anisotropy, motion magnitude, curvature, opacity variability, and life span. This score drives a three-stage pipeline, importance sampling, dynamics-aware pruning, and motion-consistent merging, that significantly reduces the number of Gaussians while preserving both spatial detail and temporal fidelity.

Unlike many existing compression approaches, our method treats the base 4DGS as a black box and relies only on exported per-Gaussian statistics, making it easy to integrate with a variety of dynamic Gaussian representations and training pipelines. At the same time, it captures key insights from recent temporally-aware pruning and merging techniques without requiring gradient access or learned entropy models.

Acknowledgments

The authors would like to thank our teaching assistant, Shayan Shekarforoush, for his valuable advice and assistance. We also appreciate Professor David Lindell for designing and teaching this class.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 405–421.

[2] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, Jul. 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530127

[3] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Trans. Graph.*, vol. 42, no. 4, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3592433

[4] Z. Li, Z. Chen, Z. Li, and Y. Xu, "Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis," 2024. [Online]. Available: https://arxiv.org/abs/2312.16812

[5] G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, W. Liu, Q. Tian, and X. Wang, "4D Gaussian Splatting for Real-Time Dynamic Scene Rendering," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 20 310–20 320.

[6] S. Oh, Y. Lee, H. Jeon, and E. Park, "Hybrid 3D-4D Gaussian Splatting for Fast Dynamic Scene Representation," 2025. [Online]. Available: https://arxiv.org/abs/2505.13215

[7] Z. Zheng, D. Zhou, Y. Shao, and X. Yang, "EGU-GS: Efficient Gaussian utilization for real-time 3D Gaussian splatting," *Image and Vision Computing*, vol. 162, p. 105687, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885625002756

[8] M. S. Ali, S.-H. Bae, and E. Tartaglione, "ELMGS: Enhancing Memory and Computation Scalability Through coMpression for 3D Gaussian Splatting," in *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2025, pp. 2591–2600.

[9] S. Qiu, C. Wu, Z. Wan, and S. Tong, "High-Fold 3D Gaussian Splatting Model Pruning Method Assisted by Opacity," *Applied Sciences*, vol. 15, no. 3, 2025. [Online]. Available: https://www.mdpi.com/2076-3417/15/3/1535

[10] A. Hanson, A. Tu, V. Singla, M. Jayawardhana, M. Zwicker, and T. Goldstein, "PUP 3D-GS: Principled Uncertainty Pruning for 3D Gaussian Splatting," in *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025, pp. 5949–5958.

[11] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3D Gaussian Representation for Radiance Field," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 21 719–21 728.

[12] Z. Zhang, T. Song, Y. Lee, L. Yang, C. Peng, R. Chellappa, and D. Fan, "LP-3DGS: Learning to Prune 3D Gaussian Splatting," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 122 434–122 457. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/dd51dbce305433cd60910dc5b0147be4-Paper-Conference.pdf

[13] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural Radiance Fields for Dynamic Scenes," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 10 313–10 322.

[14] T. Li, M. Slavcheva, M. Zollhoefer, S. Green, C. Lassner, C. Kim, T. Schmidt, S. Lovegrove, M. Goesele, R. Newcombe, and Z. Lv, "Neural 3D Video Synthesis from Multi-view Video," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 5511–5521.

[15] Y. Yuan, Q. Shen, X. Yang, and X. Wang, "1000+ FPS 4D Gaussian Splatting for Dynamic Scene Rendering," 2025. [Online]. Available: https://arxiv.org/abs/2503.16422

[16] S. Javed, A. J. Khan, C. Dumery, C. Zhao, and M. Salzmann, "Temporally Compressed 3D Gaussian Splatting for Dynamic Scenes," 2025. [Online]. Available: https://arxiv.org/abs/2412.05700

[17] A. Tu, H. Ying, A. Hanson, Y. Lee, T. Goldstein, and M. Zwicker, "SpeeDe3DGS: Speedy Deformable 3D Gaussian Splatting with Temporal Pruning and Motion Grouping," 2025. [Online]. Available: https://arxiv.org/abs/2506.07917

[18] H. Liu, Y. Liu, C. Li, W. Li, and Y. Yuan, "LGS: A Light-weight 4D Gaussian Splatting for Efficient Surgical Scene Reconstruction," 2024. [Online]. Available: https://arxiv.org/abs/2406.16073

[19] Z. Luo, H. Ran, and L. Lu, "Instant4D: 4D Gaussian Splatting in Minutes," 2025. [Online]. Available: https://arxiv.org/abs/2510.01119

[20] J. Fang, T. Yi, X. Wang, L. Xie, X. Zhang, W. Liu, M. Nießner, and Q. Tian, "Fast Dynamic Radiance Fields with Time-Aware Neural Voxels," in *SIGGRAPH Asia 2022 Conference Papers*, ser. SA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3550469.3555383

[21] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht, and A. Kanazawa, "K-Planes: Explicit Radiance Fields in Space, Time, and Appearance," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 12 479–12 488.

[22] Z. Yang, X. Gao, W. Zhou, S. Jiao, Y. Zhang, and X. Jin, "Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction," 2023. [Online]. Available: https://arxiv.org/abs/2309.13101

[23] Y. Duan, F. Wei, Q. Dai, Y. He, W. Chen, and B. Chen, "4D-Rotor Gaussian Splatting: Towards Efficient Novel View Synthesis for Dynamic Scenes," in *ACM SIGGRAPH 2024 Conference Papers*, ser. SIGGRAPH '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3641519.3657463

[24] Z. Yang, Z. Pan, X. Zhu, L. Zhang, J. Feng, Y.-G. Jiang, and P. H. S. Torr, "4D Gaussian Splatting: Modeling Dynamic Scenes with Native 4D Primitives," 2025. [Online]. Available: https://arxiv.org/abs/2412.20720