Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Sparse-Mono: Sparse Gaussian Splatting for Monocular Simultaneous Localization and Mapping

Jianpeng Xu
People's Public Security University of China
Yifan Wang
People's Public Security University of China
Fanliang Bu
bufanliang@sina.com

People's Public Security University of China

Research Article

Keywords: Gaussian splatting, incremental mapping, monocular simultaneous localization and mapping, sparse Gaussian generation

Posted Date: November 12th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11445289/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Sparse-Mono: Sparse Gaussian Splatting for Monocular Simultaneous Localization and Mapping

Jianpeng Xu, Yifan Wang, and Fanliang Bu

School of Information Network Security, People's Public Security University of China, Beijing 100038, China

Corresponding author: Fanliang Bu; email: bufanliang@sina.com

Abstract

Feed-forward Gaussian simultaneous localization and mapping (SLAM) can reduce scene-specific optimization by directly predicting Gaussian primitives, but many existing feed-forward methods still generate a dense set of Gaussians for each mapping keyframe, even when much of the current view is already represented in the persistent map. This causes redundant prediction and unnecessary map growth during continuous mapping. We propose Sparse-Mono, a persistent-map-conditioned neural framework for sparse incremental Gaussian generation in monocular SLAM. Instead of reconstructing each keyframe independently, Sparse-Mono jointly encodes the current observation, the rendered persistent map, and their photometric and geometric discrepancies. A multi-scale network with spatial Transformer adapters and a causal event-state Transformer produces map-conditioned features from the current observation, the persistent map, and preceding mapping context. Separate location and count heads then determine where new Gaussians are needed and how many should be generated, after which complete geometry and appearance are decoded only at the selected locations. Valid frontend depths are locally refined, while missing or invalid depths are predicted relative to the current event scale. Experiments on Replica, BundleFusion, and KITTI Odometry show that Sparse-Mono achieves higher average peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) with substantially more compact maps. On Replica, it improves PSNR by 1.05 dB over MonoGS and uses 89.3% fewer Gaussians than VINGS-Mono, while maintaining comparable online processing latency. These results demonstrate that conditioning Gaussian generation on the existing map provides an effective way to control map growth while maintaining strong rendering quality.

Keywords: Gaussian splatting; incremental mapping; monocular simultaneous localization and mapping; sparse Gaussian generation

1 Introduction

Gaussian splatting provides an explicit scene representation and efficient differentiable rendering [1,2]. Recent visual SLAM systems have adopted this representation for joint tracking and mapping [3–5]. Optimization-based Gaussian SLAM systems repeatedly refine Gaussian parameters as observations arrive, and their mapping cost generally increases with map size. Feed-forward methods such as Flash-Mono [6] reduce this cost by directly predicting camera states and Gaussian attributes. Dense frame-level prediction, however, can still generate Gaussians in regions already represented by earlier observations. In continuous mapping, the need for additional Gaussians depends on both the current image and the accumulated map: a region may require substantial support when first observed but few additions on a later visit. Predicting complete attributes across every keyframe can therefore repeat computation and expand the map unnecessarily. An incremental mapper should instead determine where additional support is still needed and how much should be added before decoding complete Gaussian attributes. Figure 1 illustrates the overlap between frame-level additions and the existing map in representative Gaussian SLAM mapping events.

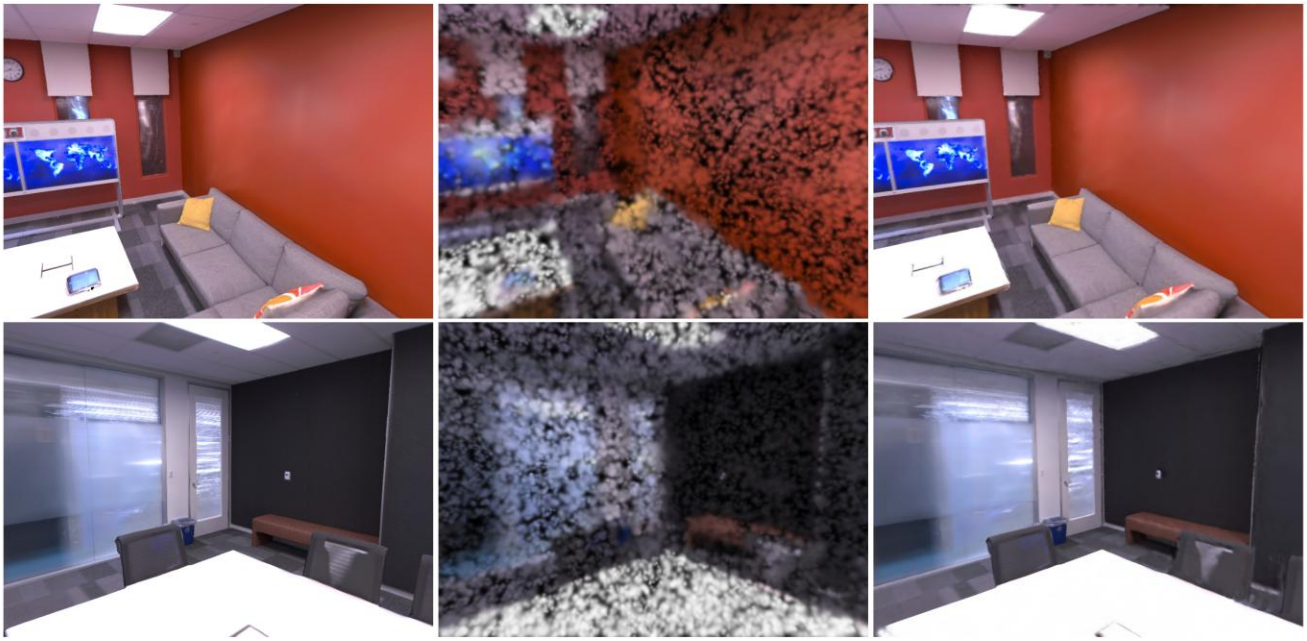


Fig. 1 Overlap between frame-level additions and the persistent map. Each row shows one mapping event: the current RGB observation (left), Gaussians created by MonoGS for the current keyframe before integration into the global map (middle), and the pre-update persistent map rendered from the same viewpoint (right). The visual overlap illustrates how new additions can cover regions already represented in the map

We propose Sparse-Mono: Sparse Gaussian Splatting for Monocular SLAM, a persistent-map-conditioned framework for sparse incremental Gaussian generation. At each mapping event, three input streams encode the current RGB observation and frontend geometry, the persistent map rendered from the current viewpoint, and their photometric and geometric discrepancies. Spatial Transformer adapters capture within-event context, while multi-scale fusion and a causal event-state Transformer combine the current map-conditioned evidence with latent context retained from preceding mapping events. Sparse allocation is decomposed into two separately supervised decisions: a location head ranks where new Gaussians should be added, and an independent count head estimates how many are required. Complete 2D Gaussian attributes are then decoded only for the selected candidates. Valid frontend depths receive bounded local corrections, whereas missing or invalid depths are predicted relative to the current event scale. Training combines masked feature pretraining, offline allocation supervision, and causal event- and window-level rendering supervision. During online operation, the network weights remain fixed, while the persistent map and event state are updated sequentially to condition subsequent mapping events.

The main contributions are as follows:

First, we formulate monocular Gaussian mapping as sparse incremental generation conditioned on the persistent map. Rather than treating each mapping keyframe independently, the current observation is interpreted relative to what the existing map already represents, allowing the mapper to distinguish regions that require additional support from those that are already sufficiently covered.

Second, we develop a three-stream multi-scale network with spatial Transformer adapters and causal event-state encoding. Separate location and count heads determine where new Gaussians should be added and how many are required before complete Gaussian decoding, restricting candidate-specific geometry and appearance prediction to selected locations.

Third, we combine masked feature pretraining, offline allocation supervision, and causal event- and window-level rendering supervision, while using reference geometry only during offline training. Experiments evaluate rendering quality, map compactness, and online processing cost; ablations isolate the effects of the input streams, spatial and cross-event processing, and allocation policies.

Section 2 reviews related work, Section 3 formulates the incremental mapping problem, and Section 4 presents the method. Section 5 reports the experiments, and Section 6 discusses the findings and limitations.

2 Related Work

2.1 Feed-Forward Gaussian Splatting SLAM and Incremental Mapping

Gaussian Splatting has been increasingly adopted as an explicit scene representation for SLAM, where the map is progressively updated as new observations arrive. MonoGS integrates Gaussian representations with monocular tracking and mapping [3]. SplaTAM uses rendered-map information for tracking and map expansion in RGB-D scenes [7], while GS-SLAM adapts Gaussian insertion and optimization during mapping [8]. RTG-SLAM distinguishes stable and unstable Gaussians to improve the efficiency of continuous reconstruction [9]. In monocular settings, VINGS-Mono incrementally maintains a persistent 2D Gaussian map from estimated geometry [4], whereas Splat-SLAM jointly optimizes camera poses, depth, and Gaussian representations [5]. These methods demonstrate the importance of persistent map maintenance, but map information is mainly used for tracking, insertion, or refinement.

Feed-forward Gaussian reconstruction reduces reliance on repeated scene-specific optimization through neural prediction. pixelSplat predicts Gaussian primitives from image pairs [10], MVSplat introduces multi-view matching and plane-sweep cost volumes for geometry estimation [11], and DepthSplat couples depth prediction with Gaussian reconstruction [12]. More recently, feed-forward prediction has also been incorporated into Gaussian SLAM. Flash-Mono aggregates temporal visual features and jointly predicts camera poses and dense per-pixel Gaussian attributes, which are subsequently integrated and refined by a 2D Gaussian backend [6]. DepthGS combines monocular depth estimation with feed-forward pose prediction and Gaussian mapping [13], while S3PO-GS employs scale-consistent Gaussian pointmaps for outdoor monocular SLAM [14].

Despite these advances, existing feed-forward Gaussian SLAM methods do not explicitly condition Gaussian allocation on the current persistent map before complete attribute decoding. In particular, they do not separately determine where additional Gaussians are needed and how many should be generated. This may lead to unnecessary prediction in regions that are already sufficiently represented. To address this issue, we propose Sparse-Mono, which conditions sparse incremental Gaussian generation on the existing persistent map.

2.2 Sparse and Adaptive Gaussian Generation

Many feed-forward Gaussian methods associate primitives with regular pixel samples, causing the number of predicted Gaussians to grow with image resolution or the number of input views. Recent works relax this dense correspondence by learning more flexible primitive placement and density allocation. SparseSplat adapts Gaussian density to local scene structure and information content [15], while Off The Grid formulates primitive placement as local detection and predicts subpixel locations with content-dependent density [16]. These methods demonstrate that adaptive Gaussian placement can substantially reduce redundant primitives.

Other approaches control Gaussian generation through geometric priors or explicit budgets. AnchorSplat uses 3D geometric priors to construct an anchor-aligned representation, reducing its dependence on image resolution and view count [17]. EcoSplat instead ranks primitives by importance and adjusts the representation according to a user-specified target budget [18].

Overall, existing sparse generation methods determine Gaussian placement or density mainly from image content, scene geometry, or predefined budgets. They do not explicitly consider what has already been represented in a persistent map during continuous SLAM. Sparse-Mono addresses this gap by conditioning sparse Gaussian allocation on the current state of the persistent map.

3 Problem Formulation

3.1 Online Incremental Gaussian Mapping

We model monocular Gaussian SLAM as an online incremental mapping process. Let t index mapping events after initialization, where each event is an accepted observation submitted to the mapper; therefore, not every incoming camera frame triggers a map update. At event t, the retained monocular frontend provides the RGB image $\boldsymbol{I}_t$, estimated depth $\boldsymbol{D}_t$, camera intrinsics $\boldsymbol{K}_t$, estimated camera-to-map transform $\boldsymbol{T}_t$, and uncertainty-related metadata $\boldsymbol{U}_t$, summarized as $\mathcal{O}_t$. Before the event, the system maintains a persistent 2D Gaussian map $\mathcal{G}_{t-1}$. Rendering this map from the current viewpoint gives view-aligned evidence:

$$\mathcal{R}_t = \mathcal{R}\big(\mathcal{G}_{t-1}; \boldsymbol{K}_t, \boldsymbol{T}_t\big) = \{\hat{\boldsymbol{I}}_t, \hat{\boldsymbol{D}}_t, \hat{\boldsymbol{N}}_t, \hat{\boldsymbol{A}}_t, \hat{\boldsymbol{V}}_t\}. \tag{1}$$

Here, $\hat{\boldsymbol{I}}_t$, $\hat{\boldsymbol{D}}_t$, $\hat{\boldsymbol{N}}_t$, $\hat{\boldsymbol{A}}_t$, and $\hat{\boldsymbol{V}}_t$ denote rendered color, depth, normals, accumulated opacity, and validity, respectively, describing what the existing map already represents in the current view. The online setting is strictly causal: the mapper may use the current observation, the pre-update map, and preceding accepted events, but not future observations or reference geometry. Ground-truth depth, poses, normals, and reference geometry are used only for offline training or evaluation.

At each event, the mapper may introduce an incremental Gaussian set $\Delta\mathcal{G}_t$. The persistent map is updated as:

$$\tilde{\mathcal{G}}_t = \mathcal{G}_{t-1} \cup \Delta\mathcal{G}_t. \tag{2}$$

The formulation here concerns the construction of the incremental Gaussian set; subsequent optimization and maintenance of the updated map are outside the problem definition.

3.2 Map-Conditioned Sparse Incremental Generation

In continuous mapping, much of the current view may already be represented by the persistent map. Dense frame-level generation in such regions introduces redundant prediction and unnecessary map growth. The incremental mapping problem therefore involves two coupled decisions: how many new Gaussians should be introduced at the current mapping event, and where they should be placed.

Let $\mathcal{C}_t = \{\boldsymbol{u}_{t,i}\}_{i=1}^{N_t}$ denote the set of valid candidate locations at mapping event t, where N_t is the number of valid candidates. The mapper should determine an addition count N_t^{add} and a selected subset $\mathcal{S}_t \subseteq \mathcal{C}_t$, satisfying

$$\text{card}(\mathcal{S}_t) = N_t^{\text{add}}. \tag{3}$$

Both decisions should depend on the current observation $\mathcal{O}_t$ and the pre-update persistent map $\mathcal{G}_{t-1}$, while using only causally available information from preceding mapping events. Regions that are already sufficiently represented should receive few or no additions, whereas newly observed or inadequately represented regions should receive additional Gaussian support.

For each selected location $\boldsymbol{u}_{t,i} \in \mathcal{S}_t$, let $g_{t,i}$ denote the Gaussian introduced at that location. The incremental Gaussian set is

$$\Delta\mathcal{G}_t = \{g_{t,i} \mid \boldsymbol{u}_{t,i} \in \mathcal{S}_t\}. \tag{4}$$

The objective is to preserve rendering quality while avoiding unnecessary map growth and candidate-specific decoding. Section 4 describes how Sparse-Mono learns the addition count, location selection, and Gaussian attributes under this formulation.

4 Method

Figure 2 summarizes one Sparse-Mono mapping event. The retained VINGS monocular frontend [4] provides the current RGB observation, frontend geometry, uncertainty cues, and camera state, while the pre-update persistent 2D Gaussian map is rendered from the same viewpoint. Sparse-Mono first compares the current observation with what the map already represents through three view-aligned input streams. Multi-scale encoding and causal event-state reasoning produce shared map-conditioned features (Sec. 4.1). Sparse allocation is then separated into two decisions: the count head estimates how many new Gaussians are needed, and the location head ranks where they should be placed (Sec. 4.2). Complete geometry and appearance are decoded only for the selected candidates (Sec. 4.3). Section 4.4 describes how these components are trained with offline supervision and causal rendering losses.

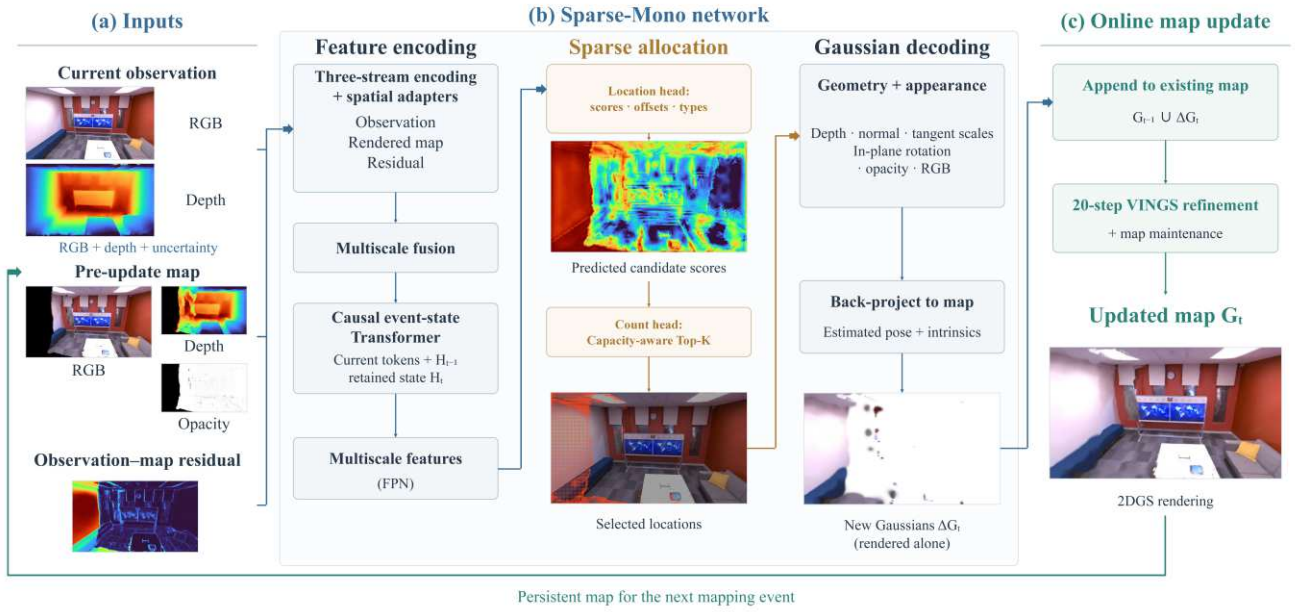


Fig. 2 Sparse-Mono at one mapping event. (a) Online inputs include the current RGB observation and frontend geometry, the pre-update persistent map rendered in the same view, and their photometric and geometric discrepancies. (b) Three-stream encoding, multi-scale fusion, and causal event-state reasoning produce shared features. The count head estimates how many new Gaussians are needed, while the location head ranks lightweight candidate slots. Complete geometry and appearance are decoded only at the capacity-aware Top-K locations. (c) The selected Gaussians are appended to the persistent map, followed by 20 VINGS-backend refinement iterations and map maintenance. The updated map rendering and retained event state condition the next mapping event. Network weights remain fixed online, and ground-truth depth, normals, and poses are never used as online inputs

4.1 Map-Conditioned Feature Encoding

Input construction. At mapping event t, the current observation provides the RGB image, frontend depth, and auxiliary uncertainty-related geometry metadata, while the pre-update persistent map provides a rendering aligned to the same viewpoint. To make depth cues comparable across events, frontend and rendered depths are normalized by an event scale defined from the median valid frontend depth; if no valid frontend depth exists, the scale is set to one. The normalized depths are defined in Eq. (5).

$$d_t(p) = \log\left(\frac{\boldsymbol{D}_t(p)}{\ell_t}\right), \qquad \hat{d}_t(p) = \log\left(\frac{\hat{\boldsymbol{D}}_t(p)}{\ell_t}\right). \tag{5}$$

Equation (5) is evaluated only where the corresponding depth is valid, with invalid entries set to zero. Rendered depth additionally requires accumulated opacity above 0.05. In the discrepancy stream, the signed normalized log-depth residual is clipped to $[-10,10]$ as shown in Eq. (6). The uncertainty channel is sanitized before use and serves only as an auxiliary cue. These quantities are organized into the three online input streams in Eq. (6).

$$\begin{aligned} \boldsymbol{X}_t^{\text{obs}} &= \left[\boldsymbol{I}_t, \boldsymbol{d}_t, \overline{\boldsymbol{U}}_t, \boldsymbol{V}_t\right], \\ \boldsymbol{X}_t^{\text{map}} &= [\hat{\boldsymbol{I}}_t, \hat{\boldsymbol{d}}_t, \hat{\boldsymbol{N}}_t, \hat{\boldsymbol{A}}_t, \hat{\boldsymbol{V}}_t], \\ \boldsymbol{X}_t^{\text{diff}} &= [|\boldsymbol{I}_t - \hat{\boldsymbol{I}}_t|, \text{clip}(\boldsymbol{d}_t - \hat{\boldsymbol{d}}_t, -10, 10), \boldsymbol{V}_t\hat{\boldsymbol{V}}_t]. \end{aligned} \tag{6}$$

The observation, map, and discrepancy streams contain 6, 9, and 5 channels, respectively. They represent three complementary views of the same mapping event: what is currently observed, what the persistent map already explains, and where the two disagree. The discrepancy stream contains absolute RGB residuals, signed normalized log-depth residuals, and a joint depth-validity mask. Rendered normals are masked where invalid, whereas RGB evidence is retained independently of depth validity.

Multi-scale encoding. Each stream is encoded at $1/4$, $1/8$, and $1/16$ resolution with 48, 96, and 160 channels. Fine-scale features preserve spatial detail for later localization, while the $1/16$ features provide a larger receptive field for contextual reasoning at lower spatial cost. A residual Transformer adapter is applied to the $1/16$ feature of each stream using fixed two-dimensional sinusoidal positional encoding, four layers, width 384, and six attention heads. The encoded streams are then fused across scales as defined in Eq. (7).

$$\boldsymbol{F}_l = \mathcal{N}\left(\Phi_l\left(\mathcal{N}([\boldsymbol{E}_l^{\text{obs}}, \boldsymbol{E}_l^{\text{map}}, \boldsymbol{E}_l^{\text{diff}}, \boldsymbol{E}_l^{\text{obs}} - \boldsymbol{E}_l^{\text{map}}, \boldsymbol{E}_l^{\text{obs}} \odot \boldsymbol{E}_l^{\text{map}}])\right)\right). \tag{7}$$

Both occurrences of $\mathcal{N}$ in Eq. (7) denote the same parameter-free, per-pixel channel normalization: it is applied first to the concatenated inputs before Φ_l and then to the fusion output. The operator subtracts the channel mean and divides by the square root of the mean squared channel deviation plus 10^{-5}, without learned scale or bias. Φ_l denotes the trainable fusion module at scale l. The output widths are 96, 128, and 192 channels at 1/4, 1/8, and 1/16 resolution, respectively. Dilated convolutions further process the coarsest feature before temporal reasoning.

Causal event-state encoding. Let $\boldsymbol{F}_t^c$ denote the fused 192-channel feature at $1/16$ resolution. Its spatial grid is flattened, projected to 896-dimensional tokens, and augmented with fixed two-dimensional positional encoding to form $\boldsymbol{Z}_t$. When a preceding event state $\boldsymbol{H}_{t-1}$ is available, a learned elementwise gate injects historical information through a residual branch scaled by $\tanh(\gamma)$, with γ initialized to zero. The temporal path therefore starts from the current event alone, and the contribution of history is learned during training.

A 28-layer Transformer with 14 attention heads produces $\boldsymbol{H}_t$, which is retained as the latent state for the next accepted mapping event. $\boldsymbol{H}_t$ is projected back to the coarse feature grid and combined with $\boldsymbol{F}_t^c$; a top-down feature pyramid network (FPN) then merges this event-aware feature with the $1/8$- and $1/4$-resolution features to obtain the shared quarter-resolution map $\boldsymbol{Q}_t$. Importantly, the retained state acts as recurrent feature memory rather than an explicit pixelwise correspondence across camera views; viewpoint-aligned geometric evidence is supplied by the rendered persistent-map stream. During causal training windows the state remains attached to the computation graph, while during validation and online inference it is detached after each event.

Table 1 Sparse-Mono modules and execution scope.

Module	Role and output	Execution scope
Stream encoders + spatial adapters	Multi-scale observation, rendered-map, and discrepancy features	Once per mapping event
Fusion + event-state Transformer + FPN	Shared quarter-resolution feature $\boldsymbol{Q}_t$ and retained state $\boldsymbol{H}_t$	Once per mapping event
Location head	Eight slots per cell: addition score, offset, and insertion type	All lightweight candidates
Count head + capacity-aware Top-K	Independent event count and selected candidate indices	Once per mapping event
Geometry + appearance decoders	Center, normal, scale, orientation, opacity, and RGB	Selected candidates only

4.2 Sparse Gaussian Allocation

Location prediction. The location head operates on $\boldsymbol{Q}_t$ and asks where new support should be added. Each quarter-resolution cell corresponds to a 4×4 region in the input image and emits eight lightweight candidate slots. A slot is only a proposal, not a fully decoded Gaussian. For each slot, a pointwise head predicts an addition score, a two-dimensional subcell offset, and two insertion-type logits; Eq. (8) converts the offset into an image-space candidate coordinate. Multiple slots allow the model to retain alternative hypotheses near boundaries, thin structures, or locally ambiguous regions without paying the cost of full Gaussian decoding.

$$\begin{aligned} \boldsymbol{q}_i &= 4\big(\boldsymbol{c}_i^{\text{cell}} + \sigma(\boldsymbol{o}_i)\big), \\ \boldsymbol{c}_i^{\text{cell}} &\in \mathbb{Z}^2, \quad \boldsymbol{o}_i \in \mathbb{R}^2. \end{aligned} \tag{8}$$

For brevity, $\boldsymbol{q}_i$ denotes the candidate location $\boldsymbol{u}_{t,i}$ at the current event. The sigmoid in Eq. (8) confines each proposal to its parent cell, and candidates outside the image are discarded. Missing frontend depth does not invalidate a candidate because Sec. 4.3 provides a depth-prediction fallback. The insertion type distinguishes two common cases: uncovered-surface insertion, where the pre-update map does not sufficiently cover the location, and residual-driven densification, where the map is valid but disagrees with the new observation. This type is embedded as an additional cue for Gaussian decoding.

Count prediction. The count head answers the complementary question of how many candidates should be admitted. Global average pooling of the event-state-enhanced 192-channel coarse feature $\boldsymbol{F}_t^{\text{evt}}$ is concatenated with a five-dimensional event descriptor $\boldsymbol{b}_t$ summarizing the visible map, the valid candidate population, mean accumulated opacity, remaining map capacity, and the fraction of pixels with a valid map rendering. The resulting 197-dimensional descriptor $\boldsymbol{a}_t$ is passed through a $197 \to 128 \to 64 \to 1$ MLP to produce the normalized allocation output ρ_t in Eq. (9).

$$\begin{aligned} \boldsymbol{a}_t &= [\mathrm{GAP}(\boldsymbol{F}_t^{\text{evt}}); \boldsymbol{b}_t], \\ \rho_t &= \sigma\big(\mathrm{MLP}_{\text{count}}(\boldsymbol{a}_t)\big). \end{aligned} \tag{9}$$

The continuous predicted count is $n_t = \rho_t N_t$, where N_t is the number of valid candidates defined in Sec. 3.2. At inference, it is rounded to the nearest integer (ties to even), then clipped by the valid candidate population, the per-event limit of 16,384 additions, the rounded 15% candidate-fraction limit, and the remaining capacity of the 600,000-Gaussian map. Thus, a continuous prediction below 0.5 yields zero additions; the Sigmoid output itself need not be exactly zero. No valid candidates or zero remaining capacity also forces the admitted count to zero. If the admitted count is zero, no Gaussian is added; otherwise, Top-K selection retains that many highest-scoring valid proposals. The count head predicts the addition count, which is then adjusted by the stated constraints, whereas the location head determines which candidates are admitted.

Only the selected candidates enter the complete geometry and appearance decoders, so dense shared feature extraction is separated from candidate-specific Gaussian construction.

4.3 Selected Gaussian Decoding

For each selected location, Sparse-Mono bilinearly samples $\boldsymbol{Q}_t$ and the corresponding multi-scale features. These features are combined with local observation and rendered-map cues, the normalized camera ray, the predicted count context, and the embedded insertion type to form the geometry descriptor in Eq. (10).

$$\boldsymbol{h}_i = \left[\boldsymbol{z}_i, \boldsymbol{f}_i, \boldsymbol{I}_t(\boldsymbol{q}_i), d_t(\boldsymbol{q}_i), \overline{\boldsymbol{U}}_t(\boldsymbol{q}_i), \hat{d}_t(\boldsymbol{q}_i), \hat{A}_t(\boldsymbol{q}_i), \overline{\boldsymbol{r}}_i, \boldsymbol{e}_i^{\text{count}}, \boldsymbol{e}_i^{\text{type}}\right]. \tag{10}$$

Here, $\boldsymbol{z}_i$ and $\boldsymbol{f}_i$ denote the sampled shared and multi-scale features, respectively, while $\boldsymbol{e}_i^{\text{count}}$ and $\boldsymbol{e}_i^{\text{type}}$ encode the allocation-count context and insertion type. The geometry head predicts depth-related parameters and a camera-frame surface normal only for the selected candidates. This design concentrates candidate-specific computation on locations that have already been judged useful for the map update.

Depth prediction and correction. When the frontend depth at a selected location is finite and positive, the geometry head predicts a gated, bounded log-depth residual around that estimate. The resulting correction is multiplicative in metric depth, as specified below.

If the frontend depth is missing or invalid, the residual branch is disabled and the decoder instead predicts log depth relative to the event scale defined in Sec. 4.1. The binary depth-validity variable selects between these two cases, giving the final depth in Eq. (11). This fallback allows useful candidates to be decoded even when monocular frontend depth is unavailable.

$$D_i^{\text{new}} = \begin{cases} D_i^{\text{local}}, & \text{if} v_i = 1, \\ \ell_t \exp(\zeta_i), & \text{if} v_i = 0. \end{cases} \tag{11}$$

Here, $D_i = D_t(q_i)$ is the sampled frontend depth and v_i marks finite, positive values. The local correction is $D_i^{\text{local}} = D_i \exp(g_i \delta_i)$, with $\delta_i = b\tanh(r_i)$, $g_i = \sigma(a_i)$, and $b = \ln 2$. The geometry head predicts the unconstrained scalars r_i and a_i. Thus $0 < g_i < 1$, $|\delta_i| \le b$, and the corrected-to-input depth ratio is bounded between 1/2 and 2. For an invalid frontend depth, the local branch is bypassed and $\zeta_i = \text{clip}(\zeta_i^{\text{raw}}, -5, 5)$ is used in Eq. (11), where ζ_i^{raw} is the unconstrained log-depth prediction relative to the event scale. The decoded optical-axis depth is back-projected with the estimated intrinsics and transformed into map coordinates as in Eq. (12).

$$\begin{aligned} \boldsymbol{\mu}_i^{\text{new}} &= \boldsymbol{R}_t\left(D_i^{\text{new}} \boldsymbol{K}_t^{-1} \tilde{\boldsymbol{q}}_i\right) + \boldsymbol{t}_t, \\ \boldsymbol{n}_i^{\text{map}} &= \boldsymbol{R}_t \boldsymbol{n}_i^{\text{cam}}. \end{aligned} \tag{12}$$

The camera-frame normal is predicted as a residual around the fixed canonical offset $\boldsymbol{n}_0 = (0, 0, -1)^{\mathsf{T}}$ and then normalized. Under our camera convention, this non-trainable offset points toward the camera and provides a stable reference for front-facing surfaces. It is neither an online input nor a reference normal.

The appearance decoder uses the selected features together with the decoded depth and normal to predict tangent scales, in-plane orientation, opacity, and RGB appearance. For the 2D Gaussian representation [2], the tangent scales are parameterized by predicted depth as in Eq. (13).

$$s_{ij} = \exp\left(\text{clip}\left(\text{clip}\left(\eta_{ij}, -4, 4\right) + \log\left(\frac{2D_i^{\text{new}}}{f_{x,t}}\right), -10, -0.7\right)\right), \quad j \in \{1, 2\}. \tag{13}$$

The horizontal focal length provides the depth-dependent image-to-surface scale, while the predicted in-plane angle rotates a deterministic tangent basis around the decoded normal before transformation into map coordinates. Opacity and color are defined in Eq. (14).

$$\begin{aligned} \alpha_i &= \sigma(\beta_i), \\ \boldsymbol{c}_i &= \text{clip}\left(\boldsymbol{I}_t(q_i) + 0.25\tanh(\xi_i), 0, 1\right). \end{aligned} \tag{14}$$

Each selected location therefore produces one 2D Gaussian with a 3D center, tangent-frame orientation, two tangent scales, opacity, and RGB appearance. The selected Gaussians together form the incremental set appended to the persistent map.

Online map update. After insertion, the retained VINGS backend performs 20 refinement iterations per mapping event, reduced from the 50-iteration setting used by the original VINGS-Mono configuration, together with the existing map-maintenance operations. The updated map is rendered to provide view-aligned evidence for the next mapping event, while $\boldsymbol{H}_t$ is carried forward as the causal event state. Sparse-Mono network weights remain fixed during online operation.

4.4 Training and Incremental Supervision

Training strategy. Training proceeds in three stages. First, the convolutional feature pathway and Gaussian prediction heads are initialized with single-event supervision while the Transformer branches are inactive. Second, the spatial adapters and event-state Transformer are pretrained by reconstructing 40% randomly masked coarse features with a smooth-L_1 objective. Third, the complete sparse incremental mapper is trained on causal windows. Geometry and appearance prediction, count estimation, and location ranking are stabilized before the final joint stage, where selection is driven by the model's own predicted Top-K candidates.

Allocation supervision. The allocation targets are generated offline from the current observation, the pre-update map rendering, and aligned reference geometry. They identify regions that require additional support because the map is missing, weakly covered, or inconsistent with the observation, while well-explained regions are excluded. Within each 4×4 proposal cell, at most one target is matched to a candidate slot. The stored targets specify whether an addition is required, its subcell location and insertion type, and a capacity-consistent event-level count. Reference geometry is used only to construct these targets and is never supplied as an online model input.

Count supervision. Before inference rounding and clipping, the continuous prediction is supervised by $L_{count} = \text{smooth} - \text{L1}(\ln(1 + n_t), \ln(1 + N_t^*))$, where N_t^* is the capacity-consistent target count. For a nonempty valid candidate set, zero-target events remain supervised: $\ln(1 + 0) = 0$, so the loss is finite and penalizes unnecessary predicted additions. When no valid candidate exists, the count loss is set to a graph-connected zero, avoiding division by zero and contributing no count gradient.

Supervision and rendering losses. The location head is supervised for addition ranking, subcell offset, and insertion type, while the count head uses the independent log(1 + count) objective above. Selected candidates receive direct supervision on Gaussian geometry and appearance. To couple these local predictions with map quality, rendering supervision is applied at two causal levels: an event-level term renders the current increment together with its pre-update map, and a window-level term accumulates predicted increments in temporal order and renders the resulting map in each reached view. Both terms use observed RGB and offline reference depth, with the per-rendering loss defined as

$$L_{\text{render}} = 10L_{\text{RGB-MSE}} + 2L_{\text{SSIM}} + 0.1L_{\text{log-depth}}. \tag{15}$$

In Eq. (15), the RGB and SSIM terms measure photometric and structural consistency. The log-depth term is a smooth-L1 loss between rendered and reference log depths, restricted to valid reference pixels with finite, positive depths in both images and rendered opacity above 0.05. For a window of T supervised events, L_{event} and L_{window} are the arithmetic means of the per-rendering loss in Eq. (15) over the event-level and window-level renderings, respectively. Equation (16) combines these means with weights of 0.5 each; the rendering contribution is therefore their average, not their sum. The remaining terms retain direct Gaussian, location, count, and feature-drift supervision, with λ denoting the feature-drift weight.

$$L = L_{\text{dir}} + L_{\text{loc}} + L_{\text{count}} + 0.5L_{\text{event}} + 0.5L_{\text{window}} + \lambda L_{\text{feat}}. \tag{16}$$

Training windows contain temporally contiguous mapping events from the same scene, with the window length increased from 4 to 6 and then 8 events. The event state is reset at window boundaries but remains connected within each window, allowing later losses to update earlier state transitions through one window-level backward pass. Teacher-based selection is used only during early stabilization and is removed before the final joint stage; validation and online inference are fully autonomous. Ground-truth depth, normals, poses, and other reference geometry are used only for offline target construction or loss computation.

5 Experimental Evaluation

We evaluate rendering quality, map compactness, online processing cost, tracking accuracy, and final-map depth accuracy. Tables 2–8 compare complete systems, while Tables 9–11 examine input streams, spatial and cross-event processing, and allocation policies. Network weights remain fixed throughout evaluation; the map and event state are updated sequentially.

5.1 Experimental Setup

Datasets and training. Indoor evaluation uses eight Replica scenes [19] and a common set of five BundleFusion scenes [20]. Outdoor evaluation uses KITTI Odometry sequences 00, 05, 06, and 07 [21]. Training uses 856 ScanNet++ scenes [22], with 100 mapping events sampled per scene, for a total of 85,600 events. ScanNet++ supplies the reference geometry for offline supervision. The three-stage training schedule follows Sec. 4.4: single-event initialization of the convolutional and Gaussian-prediction pathways, masked feature pretraining of the Transformer branches, and causal window-based incremental training with allocation and rendering supervision. Reference geometry is used only for offline supervision and evaluation; it is never provided to the model during online tracking or mapping.

Baselines and reproduction. Indoor baselines are MonoGS [3], DepthGS [13], and VINGS-Mono [4]; outdoor baselines are S3PO-GS [14] and DepthGS. All reproduced systems are evaluated on an NVIDIA RTX 4070 SUPER GPU under the same reporting protocol. Flash-Mono [6] is the closest feed-forward monocular Gaussian SLAM system. However, its official implementation was not publicly available at the time of our experiments, preventing controlled reproduction under the same hardware and evaluation protocol. We therefore do not report direct numerical comparisons with Flash-Mono. Sparse-Mono retains the VINGS monocular frontend, so the reported cross-method comparisons concern complete system configurations, including their respective tracking and mapping components.

Metrics and implementation. We measure rendering quality with PSNR, SSIM [23], and LPIPS [24]; map compactness with final Gaussian count and exported PLY size; online processing cost with CUDA-synchronized full-frame latency and full-sequence time; tracking accuracy with absolute trajectory error (ATE) RMSE after per-scene $\mathrm{Sim}(3)$ alignment, reported in centimeters; and final-map depth accuracy with Depth L_1 error in meters. Depth L_1 is evaluated on the common evaluation frames using each method's native estimated poses, with the scale component of the per-scene $\mathrm{Sim}(3)$ alignment used to recover metric depth. All valid ground-truth depth pixels are included, and unrendered pixels are counted as zero depth. All methods use the same evaluation frames and a rendering resolution of 320×240. Rendering quality and map size are evaluated from the final online map, without post-sequence offline refinement. Processing rate is derived from mean frame latency. Sparse-Mono uses Python 3.9.19, PyTorch 2.0.1+cu118, and CUDA 11.8. The network has approximately 300.3 million parameters, separate from the scene-specific Gaussian population. Its weights are fixed during evaluation, and the event state is reset at sequence boundaries.

5.2 Rendering Quality

Indoor benchmarks. Sparse-Mono has the highest average PSNR and SSIM on Replica and BundleFusion (Table 2). It also uses fewer Gaussians than DepthGS and VINGS-Mono (Table 4). The gains are metric-dependent: MonoGS has the lowest LPIPS on Replica, and DepthGS has the lowest LPIPS on BundleFusion. Figure 3 compares RGB and depth renderings on Replica.

Table 2 Indoor rendering quality on Replica and BundleFusion.

Method	Replica			BundleFusion		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
MonoGS [3]	26.40	0.830	0.177	13.34	0.56	0.71
DepthGS [13]	20.69	0.663	0.412	15.39	0.53	0.61
VINGS-Mono [4]	15.67	0.494	0.558	13.47	0.47	0.64
Sparse-Mono	27.45	0.866	0.182	16.37	0.63	0.66

PSNR is measured in dB. Higher PSNR and SSIM and lower LPIPS indicate better rendering quality. Results are averaged with equal weight across the eight Replica scenes and the same five BundleFusion scenes for all methods.

On Replica, Sparse-Mono exceeds MonoGS, the strongest PSNR baseline, by 1.05 dB. In Fig. 3, its RGB renderings retain door and cabinet boundaries in room0 and the main wall and sofa structure in office3. Its rendered depths also show coherent regions where some baseline outputs are fragmented. These examples illustrate how the sparse map retains visible scene structure.

On BundleFusion, Sparse-Mono improves average PSNR by 0.98 dB over DepthGS and average SSIM by 0.07 over MonoGS, the strongest baselines for these metrics. Together with Table 4, these results show that the PSNR and SSIM gains are compatible with a smaller map than those of DepthGS and VINGS-Mono.

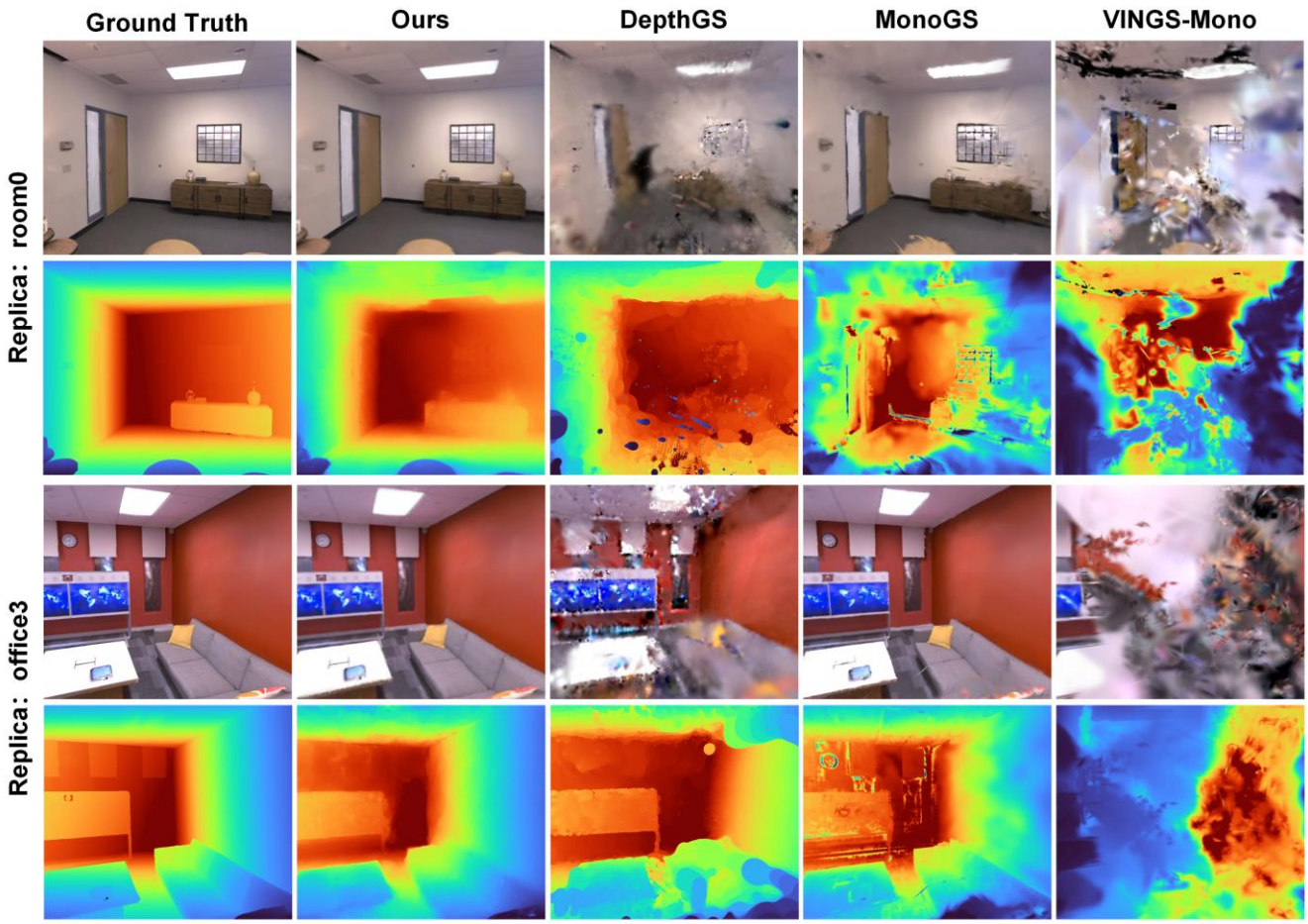


Fig. 3 RGB and depth renderings on Replica. For room0 and office3, an RGB row is followed by the corresponding depth row. Columns show ground truth, Sparse-Mono, DepthGS, MonoGS, and VINGS-Mono. Method outputs are rendered from their final Gaussian maps

Table 3 Outdoor rendering quality on KITTI Odometry sequences 00, 05, 06, and 07.

Method	Metric	00	05	06	07	Avg
S3PO-GS [14]	PSNR ↑	16.55	17.02	15.23	16.33	16.28
	SSIM ↑	0.50	0.52	0.53	0.57	0.53
	LPIPS ↓	0.53	0.54	0.54	0.54	0.54
DepthGS [13]	PSNR ↑	14.854	15.274	15.403	16.156	15.422
	SSIM ↑	0.456	0.477	0.450	0.533	0.479
	LPIPS ↓	0.508	0.493	0.556	0.447	0.501
Sparse-Mono	PSNR ↑	16.23	17.44	16.13	16.87	16.67
	SSIM ↑	0.59	0.58	0.49	0.62	0.57
	LPIPS ↓	0.50	0.48	0.61	0.47	0.52

Outdoor sequences. Sparse-Mono has the highest average PSNR and SSIM on the four KITTI Odometry sequences and the highest PSNR on three of them (Table 3). Performance varies by sequence and metric: S3PO-GS has higher PSNR on sequence 00. On sequence 06, Sparse-Mono has lower SSIM than S3PO-GS and higher LPIPS than both baselines. DepthGS has the lowest average LPIPS. The outdoor results therefore show average PSNR and SSIM gains without a consistent perceptual-quality advantage.

5.3 Map Compactness

Table 4 reports the final Gaussian population and actual exported PLY size on the indoor benchmarks. These values describe scene-map storage; they do not include neural-network parameters or total system memory. Figure 4 shows the spatial distribution of new additions relative to the observation and pre-update map.

Table 4 Final Gaussian count and exported PLY size on Replica and BundleFusion.

Method	Replica		BundleFusion	
	Gaussians ↓	PLY size (MiB) ↓	Gaussians ↓	PLY size (MiB) ↓
MonoGS [3]	73,076	17.0	103,061	24.0
DepthGS [13]	180,205	41.9	661,022	153.8
VINGS-Mono [4]	825,906	192.2	3,569,350	830.6
Sparse-Mono	88,450	20.6	400,241	93.1

Results are averaged with equal weight across eight Replica scenes and the same five BundleFusion scenes for all methods. PLY size is measured from the exported final map. Percentage reductions are calculated from the table entries.

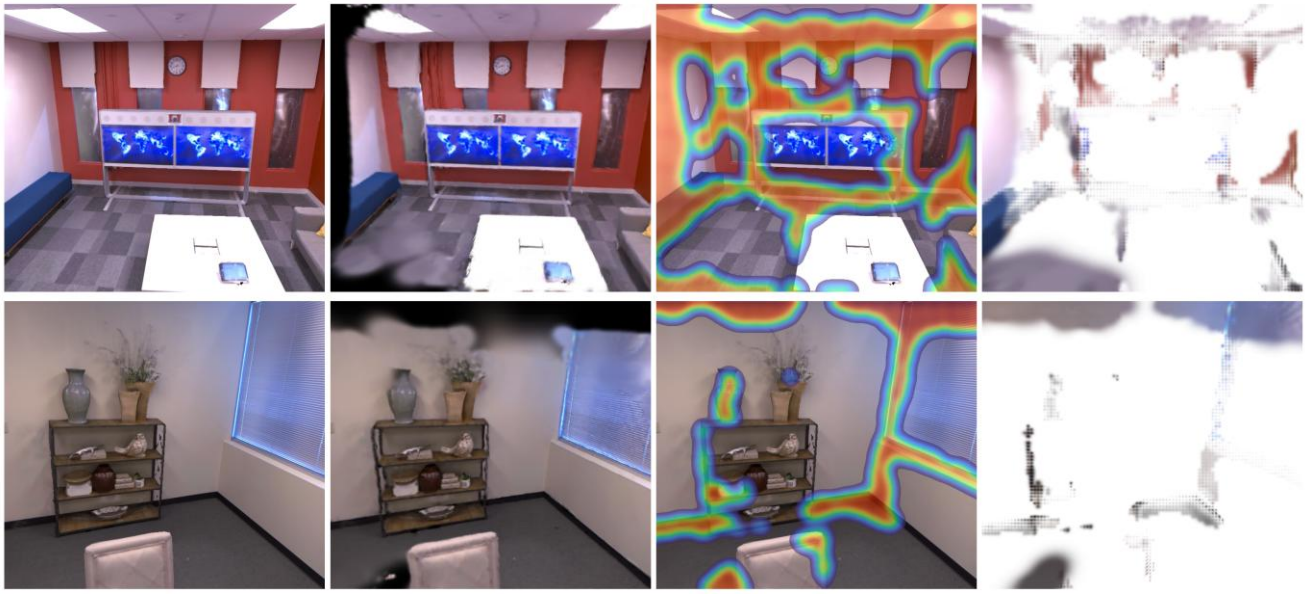


Fig. 4 Map-conditioned additions at two Replica mapping events. From left to right: current observation, pre-update persistent map rendered from the current viewpoint, observation–map discrepancy, and a rendering of only the new 2D Gaussians generated at that event

Compared with VINGS-Mono, Sparse-Mono uses 89.3% fewer Gaussians on Replica and 88.8% fewer on BundleFusion, with higher PSNR on both datasets (Table 2). These reductions describe final map populations; they do not quantify the fraction of redundant primitives in the baseline.

The comparison evaluates the complete mapping configuration, including learned allocation, explicit count and map-capacity constraints, and backend maintenance; it does not isolate the contribution of the count head.

In Fig. 4, the pre-update map already reproduces much of the visible room, and the rendering of new Gaussians is concentrated in selected regions. This pattern is consistent with adding support where the current map differs from the observation.

MonoGS produces the smallest maps, but Sparse-Mono has higher PSNR and SSIM on both indoor datasets. Relative to DepthGS and VINGS-Mono, Sparse-Mono improves these metrics while using fewer Gaussians. The compactness results should therefore be read alongside rendering quality: a smaller map is useful only if it still represents the observed scene adequately.

5.4 Online Processing Efficiency

Timing protocol. All methods use the RTX 4070 SUPER and CUDA-synchronized wall-clock measurements. Full-frame latency includes input preparation, depth estimation or preprocessing, tracking, Gaussian generation or insertion, and backend optimization and maintenance. Sparse-Mono measurements also include Transformer inference and event-state updates. Sparse-Mono uses 20 VINGS-backend refinement iterations per mapping event, reduced from the 50-iteration setting of the original VINGS-Mono configuration. Other mapping and maintenance settings follow each method's native configuration. The results compare complete configurations with different refinement budgets and do not isolate the runtime benefit of sparse allocation. Post-sequence offline refinement is excluded.

Table 5 Online frame-processing time on Replica (RTX 4070 SUPER).

Method	Full-frame time (ms) ↓	Full-sequence online time (s) ↓	Effective processing rate (frames/s) ↑
MonoGS [3]	699.0	1398.0	1.4
DepthGS [13]	582.9	1165.9	1.7
VINGS-Mono [4]	213.9	427.9	4.7
Sparse-Mono	211.5	423.0	4.7

Results are averaged with equal weight across eight Replica scenes. Full-frame time includes the online stages listed above. Full-sequence time is the sum of frame-processing times and excludes loading, final evaluation, export, and checkpoint serialization. Processing rate is calculated as 1,000 divided by mean frame time in milliseconds and rounded to one decimal place.

Replica. Sparse-Mono and VINGS-Mono have similar mean full-frame latencies, at 211.5 and 213.9 ms, respectively. Under the reported configurations, Sparse-Mono achieves higher PSNR and a smaller map at comparable total processing cost. Map-size reductions do not translate directly into proportional latency reductions because frontend processing and shared neural inference still contribute to the total cost.

Table 6 Online frame-processing time on BundleFusion (RTX 4070 SUPER).

Method	Full-frame time (ms) ↓	Full-sequence online time (s) ↓	Effective processing rate (frames/s) ↑
MonoGS [3]	616.3	3304.5	1.6
DepthGS [13]	871.1	4724.9	1.1
VINGS-Mono [4]	310.1	1871.6	3.2
Sparse-Mono	279.1	1569.3	3.6

Results are averaged with equal weight across the same five BundleFusion scenes for all methods. Timing boundaries and processing-rate calculation follow Table 5. Mean frame time and mean full-sequence time are averaged separately across scenes.

BundleFusion. Sparse-Mono has the lowest mean latency among the evaluated systems, 10.0% below VINGS-Mono. A smaller map can reduce rendering, optimization, and maintenance work, although the measurements also include neural inference and the different refinement budgets. The complete Sparse-Mono configuration therefore combines a smaller map with competitive online processing cost.

5.5 Tracking and Final-Map Depth Accuracy

Tables 7 and 8 report Replica tracking and final-map depth accuracy.

Table 7 Tracking accuracy on Replica measured by ATE RMSE (cm) ↓.

Method	room0	room1	room2	office0	office1	office2	office3	office4	Avg
MonoGS [3]	37.324	64.473	24.457	32.224	39.702	47.680	18.721	96.692	45.159
DepthGS [13]	61.718	14.907	86.621	38.926	12.373	43.817	45.290	16.235	39.986
Sparse-Mono	2.655	7.313	4.695	17.647	16.961	5.185	4.639	5.800	8.112

Replica ATE. VINGS-Mono is omitted from Table 7 and Fig. 5 because Sparse-Mono retains the same VINGS/DROID tracking frontend and map updates do not feed back to camera tracking; the two systems therefore produce identical trajectories and ATE values in this configuration. The ATE results show that Sparse-Mono preserves the trajectory accuracy of the retained frontend while achieving lower mean trajectory error than MonoGS and DepthGS on Replica.

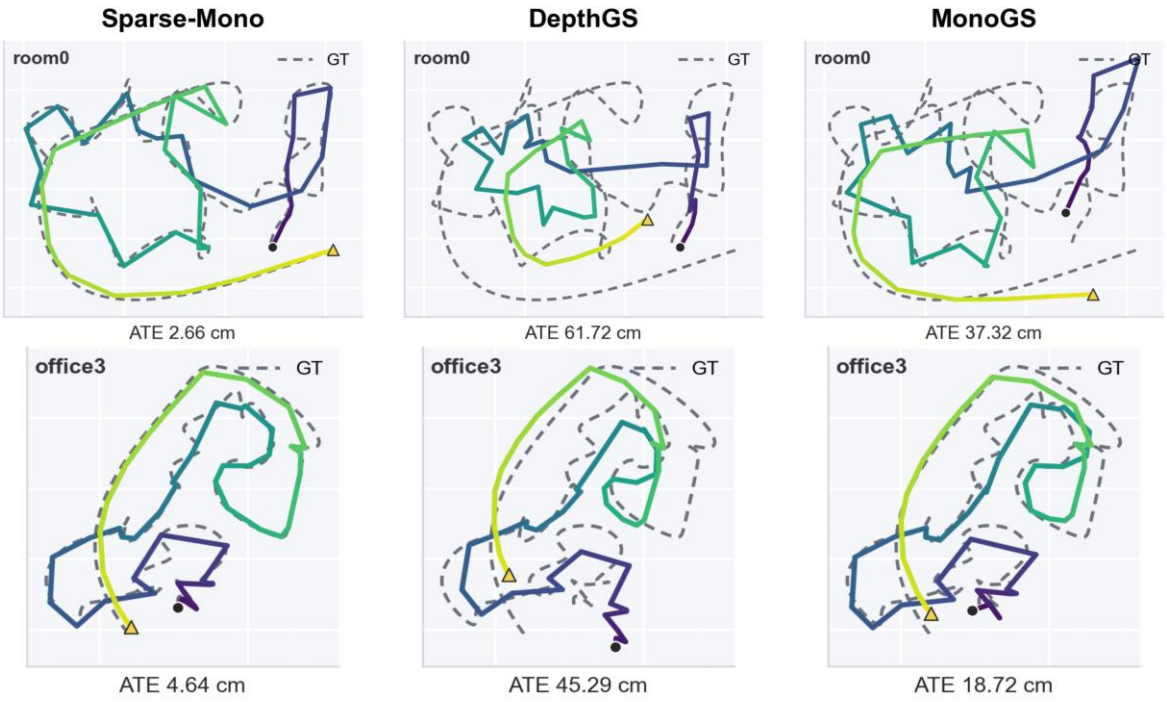


Fig. 5 Camera trajectory comparison on the Replica dataset. Each row represents a scene, and each column represents a method. Gray dashed lines indicate the ground-truth camera trajectories, while colored solid lines show the estimated trajectories, progressing from purple to yellow over time

Table 8 Final-map depth accuracy on Replica measured by Depth L_1 error (m) ↓.

Method	room0	room1	room2	office0	office1	office2	office3	office4	Avg
MonoGS [3]	0.759	0.918	0.478	0.489	0.376	0.684	0.358	1.560	0.703
DepthGS [13]	1.411	0.205	1.747	0.228	0.579	0.232	1.031	0.195	0.704
VINGS-Mono [4]	1.844	1.181	1.136	1.343	0.765	1.293	2.280	1.313	1.394
Sparse-Mono	0.250	0.274	0.220	0.322	0.421	0.298	0.287	0.427	0.312

Replica depth. Sparse-Mono achieves the lowest mean Depth L_1 error among the evaluated methods, although it does not have the lowest error on every scene.

5.6 Ablation Studies

Map-conditioned inputs. Table 9 compares separately trained variants under otherwise identical training settings. The observation-stream-only variant disables the rendered-map and discrepancy image streams; the two-stream variant adds the rendered-map stream; and the full model also includes the discrepancy stream. Map-derived scalars and decoder-side cues remain enabled in all variants. This ablation therefore tests the image-stream design, not the removal of all map information.

Table 9 Effect of the observation, rendered-map, and discrepancy input streams.

Enabled image streams	PSNR ↑	SSIM ↑	LPIPS ↓
Observation stream only	23.30	0.765	0.201
Observation + rendered-map streams	27.04	0.821	0.191
Full three streams	27.45	0.866	0.182

Adding the rendered-map stream increases PSNR by 3.74 dB, the largest gain in Table 9. This stream provides a view-aligned description of the existing map, giving the encoder information about what earlier observations have already represented.

The discrepancy stream adds a further 0.41 dB and improves SSIM and LPIPS. Explicit residuals make local observation–map disagreement available to the encoder, complementing the observation and map streams.

Spatial and cross-event processing. Table 10 compares four configurations on the same eight Replica scenes. A uses convolutional encoders only; B adds spatial Transformer adapters; C adds the fused-feature Transformer but disables the preceding event state during training and inference; and D enables cross-event state propagation. C and D use the same Transformer architecture. All variants retain the persistent Gaussian map and use the same training data, observation sequences, initial maps, and backend settings. The C–D comparison isolates the contribution of recurrent feature context beyond the history already stored in the map.

Table 10 Ablation of spatial Transformer processing and cross-event state.

Configuration	PSNR ↑	SSIM ↑	LPIPS ↓	Gaussians ↓	Frame time (ms) ↓
A Convolution only	25.14	0.811	0.226	122,630	187.8
B + spatial adapters	26.26	0.827	0.202	103,212	194.6
C + current-event Transformer	27.01	0.838	0.197	97,631	209.0
D + cross-event state (full model)	27.45	0.866	0.182	88,450	211.5

Results are averaged with equal weight across the same eight Replica scenes. PSNR is measured in dB, Gaussian count is measured from the final map, and full-frame time is measured in milliseconds.

From A to C, spatial adapters and current-event Transformer processing progressively improve rendering metrics and reduce final Gaussian count. Enabling cross-event state in D raises PSNR by 0.44 dB over C and reduces the Gaussian population by 9.4%, while mean full-frame latency increases by 2.5 ms. SSIM and LPIPS also improve. Since C and D have the same Transformer architecture, the gains support retaining event features alongside the explicit map.

Allocation policies. Table 11 separates the effects of location ranking and count prediction on the same eight Replica scenes. The rule-ranked control keeps the predicted count but prioritizes uncovered candidates, then ranks candidates using normalized RGB and depth residuals. The fixed-fraction control keeps learned location scores but replaces the predicted event-dependent count with a fixed 3% of the valid candidate population. The fraction is selected on a held-out ScanNet++ validation set to match the full model's average allocation budget and is then kept fixed for all test sequences. This inference-time control is separate from the fixed offline allocation targets used during training. The full model uses both learned location scores and event-dependent count prediction.

All policies use the same frontend, decoder weights, initial maps, and backend settings. Each policy updates its own map during evaluation, so later scores and predicted counts can differ as the maps diverge. The comparison measures the accumulated effect over a sequence, rather than selection from identical map inputs at a single event.

Table 11 Ablation of sparse Gaussian allocation policies.

Allocation policy	PSNR ↑	SSIM ↑	LPIPS ↓	Gaussians ↓	Frame time (ms) ↓
Rule-ranked locations + predicted count	26.02	0.810	0.202	99,042	214.6
Learned locations + fixed 3% fraction	26.87	0.847	0.188	116,503	216.8
Learned locations + predicted count	27.45	0.866	0.182	88,450	211.5

Results are averaged with equal weight across the same eight Replica scenes. PSNR is measured in dB, Gaussian count is measured from the final map, and full-frame time is measured in milliseconds.

With predicted counts retained, learned location ranking improves PSNR by 1.43 dB and reduces final Gaussian count by 10.7% relative to rule-based ranking. With learned ranking retained, event-dependent count prediction improves PSNR by 0.58 dB and reduces Gaussian count by 24.1% relative to the fixed-fraction control. Both decisions therefore contribute to the rendering and compactness gains. The full policy has the best rendering metrics, smallest map, and lowest measured latency among the three policies.

6 Discussion and Conclusion

Sparse-Mono conditions incremental Gaussian generation on the persistent map and a causal event feature state. Its three-stream network selects addition locations and counts before decoding complete attributes. Offline feature pretraining, allocation supervision, and causal rendering losses train the predictor; online operation updates the map and event state with fixed network weights.

On Replica, BundleFusion, and four KITTI Odometry sequences, Sparse-Mono achieves the highest average PSNR and SSIM among the evaluated methods. Compared with VINGS-Mono, it uses 89.3% fewer Gaussians at comparable latency on Replica and 88.8% fewer with 10.0% lower latency on BundleFusion. On Replica, Sparse-Mono preserves the retained VINGS/DROID frontend trajectory while substantially reducing final-map depth error relative to VINGS-Mono. Ablations support the map-conditioned inputs, spatial and cross-event processing, and learned allocation.

LPIPS gains are less consistent, and the Depth L_1 evaluation does not by itself establish full surface-geometric fidelity. The mapper remains dependent on frontend pose and depth estimates. Smaller maps reduce scene storage, but the 300.3-million-parameter network still incurs storage and inference costs. Different backend refinement budgets also limit attribution of latency gains to sparse allocation.

Future work will test longer sequences and more diverse real-world scenes, extend event-state propagation, and improve robustness to geometric uncertainty. Matched refinement budgets and broader surface-geometry evaluation will further clarify the mapping trade-offs.

Statements and Declarations

Competing interests

The authors declare no conflicts of interest.

Data Availability Statement

The datasets used in this study are available from their respective sources: Replica benchmark sequences (NICE-SLAM repository, https://github.com/cvg/nice-slam), based on the original Replica Dataset (https://github.com/facebookresearch/Replica-Dataset); BundleFusion (https://graphics.stanford.edu/projects/bundlefusion/); KITTI Odometry (https://www.cvlibs.net/datasets/kitti/eval_odometry.php); and ScanNet++ v2 (https://scannetpp.mlsg.cit.tum.de/scannetpp/). Access to ScanNet++ is subject to registration and its terms of use. The quantitative results are reported in this article. Additional processed evaluation data and experimental logs supporting the findings of this study are available from the corresponding author upon reasonable request.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3D Gaussian splatting for real-time radiance field rendering. ACM Trans. Graph. 42(4), 139:1–139:14 (2023) https://doi.org/10.1145/3592433

[2] Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D Gaussian splatting for geometrically accurate radiance fields. In: ACM SIGGRAPH 2024 Conference Papers, Article 32 (2024)

[3] Matsuki, H., Murai, R., Kelly, P.H.J., Davison, A.J.: Gaussian splatting SLAM. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 18039–18048 (2024)

[4] Wu, K., Zhang, Z., Tie, M., Ai, Z., Gan, Z., Ding, W.: VINGS-Mono: Visual-inertial Gaussian splatting monocular SLAM in large scenes. IEEE Trans. Robot. 41, 5912–5931 (2025) https://doi.org/10.1109/TRO.2025.3613536

[5] Sandström, E., Zhang, G., Tateno, K., et al.: Splat-SLAM: Globally optimized RGB-only SLAM with 3D Gaussians. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp. 1686–1697 (2025)

[6] Zhang, Z., Wu, K., Meng, X., Liu, K., Zhao, J., Ding, W.: Flash-Mono: Feed-forward accelerated Gaussian splatting monocular SLAM. In: Proceedings of the International Conference on Learning Representations (ICLR) (2026)

[7] Keetha, N., Karhade, J., Jatavallabhula, K.M., et al.: SplaTAM: Splat, track & map 3D Gaussians for dense RGB-D SLAM. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 21357–21366 (2024)

[8] Yan, C., Qu, D., Xu, D., et al.: GS-SLAM: Dense visual SLAM with 3D Gaussian splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 19595–19604 (2024)

[9] Peng, Z., Shao, T., Liu, Y., et al.: RTG-SLAM: Real-time 3D reconstruction at scale using Gaussian splatting. In: ACM SIGGRAPH 2024 Conference Papers, Article 30 (2024)

[10] Charatan, D., Li, S.L., Tagliasacchi, A., Sitzmann, V.: pixelSplat: 3D Gaussian splats from image pairs for scalable generalizable 3D reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 19457–19467 (2024)

[11] Chen, Y., Xu, H., Zheng, C., et al.: MVSplat: Efficient 3D Gaussian splatting from sparse multi-view images. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 370–386 (2024)

[12] Xu, H., Peng, S., Wang, F., et al.: DepthSplat: Connecting Gaussian splatting and depth. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 16453–16463 (2025)

[13] Zhao, L., Xu, X., Wang, Y., et al.: Pseudo depth meets Gaussian: A feed-forward RGB SLAM baseline. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 8142–8149 (2025)

[14] Cheng, C., Yu, S., Wang, Z., Zhou, Y., Wang, H.: Outdoor monocular SLAM with global scale-consistent 3D Gaussian pointmaps. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 26035–26044 (2025)

[15] Zhang, Z., Meng, X., Wu, K., Ding, W.: SparseSplat: Towards applicable feed-forward 3D Gaussian splatting with pixel-unaligned prediction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5049–5058 (2026)

[16] Moreau, A., Shaw, R., Nazarczuk, M., et al.: Off The Grid: Detection of primitives for feed-forward 3D Gaussian splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 11756–11766 (2026)

[17] Zhang, X., Zheng, X., Yin, Y., et al.: AnchorSplat: Feed-forward 3D Gaussian splatting with 3D geometric priors. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 18924–18933 (2026)

[18] Park, J., Bui, M.Q.V., Gonzalez Bello, J.L., Moon, J., Oh, J., Kim, M.: EcoSplat: Efficiency-controllable feed-forward 3D Gaussian splatting from multi-view images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 26010–26020 (2026)

[19] Straub, J., Whelan, T., Ma, L., et al.: The Replica dataset: A digital replica of indoor spaces. arXiv preprint arXiv:1906.05797 (2019)

[20] Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., Theobalt, C.: BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration. ACM Trans. Graph. 36(3), 24:1–24:18 (2017)

[21] Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The KITTI vision benchmark suite. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3354–3361 (2012)

[22] Yeshwanth, C., Liu, Y.C., Nießner, M., Dai, A.: ScanNet++: A high-fidelity dataset of 3D indoor scenes. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 12–22 (2023)

[23] Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. IEEE Trans. Image Process. 13(4), 600–612 (2004)

[24] Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 586–595 (2018)