DiskChunGS: Large-Scale 3D Gaussian SLAM Through Chunk-Based Memory Management

Casimir Feldmann[1], Maximum Wilder-Smith[1], Vaishakh Patil[1],
Michael Oechsle[2], Michael Niemeyer[2], Keisuke Tateno[2], and Marco Hutter[1]

***Abstract*—Recent advances in 3D Gaussian Splatting (3DGS) have demonstrated impressive results for novel view synthesis with real-time rendering capabilities. However, integrating 3DGS with SLAM systems faces a fundamental scalability limitation: methods are constrained by GPU memory capacity, restricting reconstruction to small-scale environments. We present DiskChunGS, a scalable 3DGS SLAM system that overcomes this bottleneck through an out-of-core approach that partitions scenes into spatial chunks and maintains only active regions in GPU memory while storing inactive areas on disk. Our architecture integrates seamlessly with existing SLAM frameworks for pose estimation and loop closure, enabling globally consistent reconstruction at scale. We validate DiskChunGS on indoor scenes (Replica, TUM-RGBD), urban driving scenarios (KITTI), and resource-constrained Nvidia Jetson platforms. Our method uniquely completes all 11 KITTI sequences without memory failures while achieving superior visual quality, demonstrating that algorithmic innovation can overcome the memory constraints that have limited previous 3DGS SLAM methods.**

***Index Terms*—Mapping, SLAM, 3D Gaussian Splatting, Large-Scale Reconstruction**

I. Introduction

RECENT advances in neural representations for 3D scene reconstruction have revolutionized novel view synthesis, with 3D Gaussian Splatting (3DGS) [1] emerging as an exceptionally efficient and high-quality approach. Unlike volume-based methods [2]–[4] that struggle with rendering speed due to expensive ray marching, 3DGS provides real-time rendering capabilities while maintaining impressive visual fidelity. However, extending 3DGS to large-scale environments for Simultaneous Localization and Mapping (SLAM) applications introduces a fundamental bottleneck: existing methods require the entire scene representation to fit within GPU Video Random Access Memory (VRAM), severely limiting the size of environments that can be reconstructed.

Our work presents a novel 3DGS-based SLAM system that overcomes this memory limitation through an out-of-core architecture inspired by virtual memory systems. The key insight is to divide the scene into spatial regions, like tiles on a map, and maintain only the currently relevant areas of GPU memory while storing the rest on disk. We call these spatial regions "chunks" and dynamically load them based on which parts of the scene are visible from the current camera position. This approach allows our system to reconstruct substantially larger environments without compromising visual quality.

Beyond memory constraints, large-scale SLAM faces additional challenges, including pose drift accumulation and loop closure detection in expansive environments. To ensure robust localization and global consistency, our system integrates with the proven ORB-SLAM3 [5] framework and introduces chunk-aware loop closure mechanisms that operate seamlessly within our out-of-core architecture.

Our approach enables several practical advantages for real-world deployment. The spatial partitioning allows efficient serialization and persistent storage of individual chunks, supporting incremental map updates. The system scales across hardware platforms, from memory-constrained mobile robots to high-end desktop systems. This capability opens applications in autonomous navigation, AR/VR, digital twins, and cultural heritage preservation, where photorealistic reconstruction at scale is essential.

We validate our approach through extensive evaluations on diverse datasets spanning different scales and environments, including Replica [7] (indoor scenes), TUM [8] (office environments), and KITTI [6] (urban driving scenarios). Our evaluation demonstrates robust performance across settings, from highly detailed indoor spaces to challenging outdoor environments. We also validate real-world deployment by demonstrating efficient online processing on indoor datasets and successful large-scale reconstruction on the compute-constrained Nvidia Jetson Orin platform.

Our contributions include:

- A novel out-of-core chunk-based architecture that enables large-scale 3DGS SLAM by partitioning scenes into spatial regions and dynamically managing them between disk and VRAM, overcoming the fundamental memory limitations of previous methods.
- Comprehensive evaluation demonstrating state-of-the-art performance across indoor and outdoor datasets, with our method uniquely completing all 11 KITTI [6] sequences without memory failures while achieving superior visual quality.
- Production-ready deployment validated on edge hardware (Jetson Orin) and integrated with ROS for robotic platforms, bridging the gap between research prototypes and real-world applications.

This work was funded by NCCR Automation, Swiss Federal Railways (SBB) via ETH Mobility Initiative, and ETHAR.

[1]Casimir Feldmann, Maximum Wilder-Smith, Vaishakh Patil, and Marco Hutter are with Robotic Systems Lab, ETH Zurich, Switzerland casimir.feldmann@gmail.com, and {mwilder, patilv, mahutter}@ethz.ch

[2]Michael Oechsle, Michael Niemeyer, and Keisuke Tateno are with Google, Switzerland {michaeloechsle, mniemeyer, ktateno}@google.com

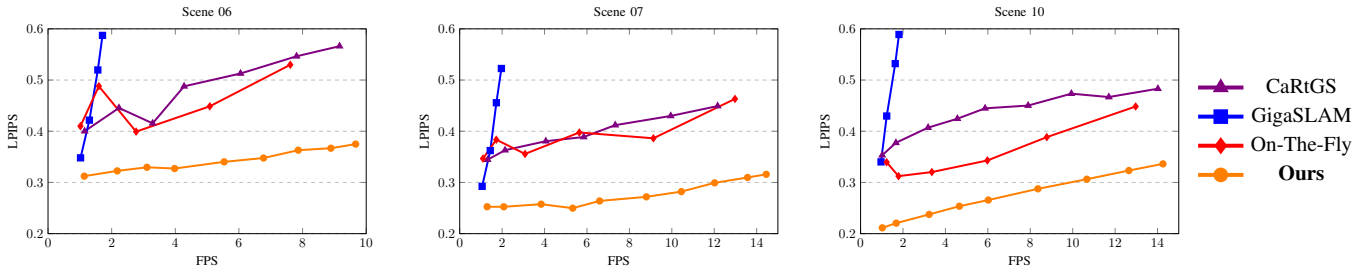

Fig. 1. **Pareto curves for KITTI [6] scenes.** With more iterations and, as a consequence, more processing time, 3DGS SLAM methods can optimize for longer, achieving higher reconstruction quality. Our method achieves superior visual quality in less time than competing methods across all three scenes.

II. Related Work

Neural Large-Scale Reconstructions

Recent neural approaches have pursued large-scale scene representation through spatial partitioning. Block-NeRF [9] partitions city-scale scenes into individually trained NeRF models with blending, while Mega-NeRF [10] uses geometry-aware pixel-data partitioning for parallel training of spatial cell submodules. For 3DGS, CityGaussian [11] enables real-time rendering through divide-and-conquer training and block-wise Level-of-Detail strategies that dynamically adjust detail based on viewing distance. However, these approaches sacrifice real-time performance or require substantial computational resources for training and inference.

Gaussian Splatting SLAM

The integration of 3DGS into SLAM represents a significant advancement in real-time photorealistic reconstruction. Early methods like GS-SLAM [12] first demonstrated this potential. Photo-SLAM [13] advanced the field by combining ORB-SLAM3 localization [5] with hyper primitives that merge explicit geometry with implicit photometric representations.

CaRtGS [14] improves rendering efficiency through splat-wise backpropagation, adaptive keyframe optimization, and opacity regularization for model size control. GigaSLAM [15] targets kilometer-scale environments using hierarchical sparse voxel maps with level-of-detail rendering. While they demonstrate results on urban driving scenarios spanning multiple kilometers, achieving high visual quality requires substantial offline post-processing.

Scaling these methods to large environments on standard hardware remains challenging. GigaSLAM [15] utilizes systems with up to 48GB VRAM for long outdoor sequences. On-The-Fly [16] addresses GPU memory constraints through incremental clustering and anchoring, storing distant content in CPU RAM rather than disk. While this shifts the memory bottleneck from VRAM to system RAM, it does not fundamentally eliminate scalability limits and lacks loop closure detection for maintaining global consistency. Gaussian-SLAM [17] employs a sub-map strategy to address this issue, but has not been validated on large-scale outdoor datasets.

In contrast, our approach scales through algorithmic innovation rather than hardware requirements. By treating reconstruction as an out-of-core problem with disk-based chunk management, we enable large-scale mapping on standard hardware while maintaining full representation fidelity (3rd-degree spherical harmonics) and incorporating robust loop closure for global consistency.

III. Method

Our approach combines robust visual SLAM with chunk-based 3D Gaussian Splatting to enable scalable dense mapping of large-scale environments. As shown in Figure 2, the system operates through parallel tracking and mapping threads, where accurate pose estimation guides efficient Gaussian placement and optimization within our hierarchical scene representation.

Tracking

Our system supports two tracking modes for different deployments. In standalone mode, we employ ORB-SLAM3 [5] for feature-based visual odometry, sharing co-visibility optimized keyframes between tracking and mapping threads. For robotic integration, our ROS wrapper enables external pose input, allowing integration with existing SLAM systems that may achieve superior localization through multi-sensor fusion. The system accepts monocular, stereo, and RGB-D camera inputs. All experiments use the standalone ORB-SLAM3 mode.

Sampling

The sparse point cloud $P \in \mathbb{R}^3$ from ORB-SLAM3 [5] provides one source of Gaussians, with keypoints from each keyframe contributing Gaussians to the scene representation. To efficiently add complementary Gaussians, we simultaneously adopt the direct primitive sampling method from Meuleman et al. [16] for each keyframe. This method estimates the probability that each pixel should generate a Gaussian primitive based on the norm of the Laplacian of Gaussian (LoG) operator, which identifies high-frequency details and edges in the image. To avoid placing redundant Gaussians, the current scene representation is rendered, and a penalty map is computed from the rendered image's LoG response. The final sampling probability is then given by:

$$P_s(x, y) = \max\left(P_{\mathrm{L}}(x, y) - \tilde{P}(x, y), 0\right) \tag{1}$$

where P_{L} is the LoG norm of the input image and $\tilde{P}$ is the LoG norm of the currently rendered scene. This ensures

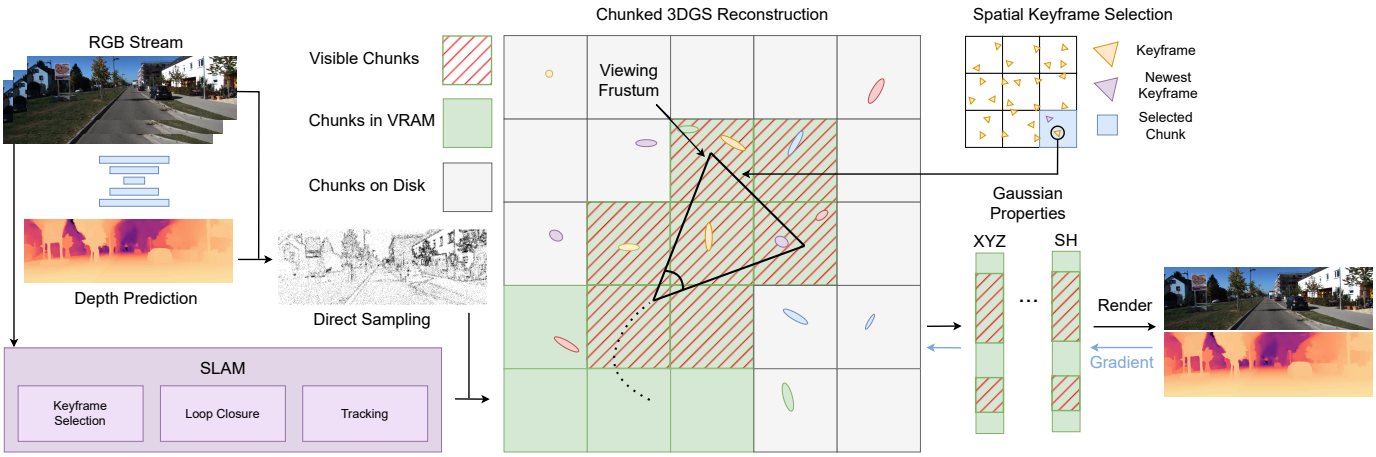

Fig. 2. **Overview of DiskChunGS.** For each SLAM keyframe, we estimate depth and perform direct primitive placement based on image content analysis instead of iterative densification. For optimization of a keyframe, frustum culling is performed to identify visible chunks, which are loaded from disk into VRAM. On the other hand, old chunks are evicted from VRAM to disk to free up memory. The visible subset of Gaussians in VRAM is then rasterized, and image/depth losses are calculated.

new Gaussians are only placed where additional detail is needed rather than in already well-reconstructed areas. This direct sampling approach replaces the iterative densification process commonly used in 3D Gaussian Splatting with a more immediate placement strategy that positions Gaussians based on image content analysis.

To determine the depth at which sampled Gaussians should be placed and to provide depth supervision during optimization, we use Depth-Anything-2 [18] for monocular and Fast-ACVNet [19] for stereo depth estimation. Next, we follow the approach of Meuleman et al. [16], where the depth is then aligned to triangulated matches and refined through guided stereo matching to correct for monocular depth errors.

Losses

We optimize Gaussian parameters through differentiable rendering using a multi-component loss:

$$\mathcal{L} = \mathcal{L}_{image} + \lambda_{depth}\mathcal{L}_{depth} \tag{2}$$

For the image loss, the L1 term measures pixel-wise differences, and the Structural Similarity Index Measure (SSIM) term captures structural similarities between rendered and ground truth images, balanced by λ_s.

$$\mathcal{L}_{image} = (1-\lambda_s)\,|I - I_{gt}|_1 + \lambda_s(1 - \text{SSIM}(I, I_{gt})) \tag{3}$$

To enforce geometric accuracy, we implement a depth loss function that measures the absolute difference between the rendered and ground truth depth maps:

$$\mathcal{L}_{depth} = |D_{\text{r}} - D_{\text{gt}}|_1 \tag{4}$$

Chunk 3DGS Mapping

Chunking: Our system employs a chunk-based scene management approach for efficient rendering and optimization of large-scale Gaussian splatting scenes. By partitioning the 3D space into discrete chunks, each containing a subset of Gaussians, we achieve selective loading, processing, and memory management that scales to massive environments while maintaining interactive performance.

The 3D world space is divided into regular cubic chunks of size s. We can determine into which chunk each Gaussian with position p falls using centered chunks:

$$\text{ChunkCoord}(p) = \left(\left\lfloor \frac{p_i + s/2}{s} \right\rfloor \right)_{i \in \{x,y,z\}} \tag{5}$$

To enable $O(1)$ chunk location queries, chunk coordinates are encoded into a single 64-bit integer using a shifted base conversion. Each coordinate is allocated 21 bits after adding an offset of 2^{20} to handle negative values, providing a range of $\pm$1M chunks per axis.

$$\text{EncodedID} = (c_x + 2^{20}) \cdot 2^{42} + (c_y + 2^{20}) \cdot 2^{21} + (c_z + 2^{20}) \tag{6}$$

Frustum Culling: Our system uses hierarchical frustum culling with spatial subdivision to determine visible chunks for each keyframe efficiently. We extract frustum planes from the view-projection matrix and recursively test chunk regions at multiple levels, rejecting entire regions outside the frustum early and subdividing intersecting regions until reaching individual chunks. The process is parallelized using OpenMP and includes distance-based filtering and pose-aware caching to minimize redundant computations.

Optimization

Our system follows a keyframe-driven architecture where each selected keyframe determines which chunks are loaded into VRAM based on its visible set. The optimization pass follows a structured workflow (visualized in Fig. 2). The process begins by selecting a keyframe and performing frustum culling to identify visible chunks. These visible chunks are then loaded into VRAM if not already present.

To maintain memory efficiency, our system operates under a configurable VRAM budget, which we set to 1.5 million

Fig. 3. **Qualitative results on the KITTI [6] dataset.** Reconstruction results on three scenes by all methods. On-The-Fly suffers from tracking drift and lacks loop closure. GigaSLAM's neural approach fails without expensive post-processing. CaRtGS shows floating artifacts from missing depth supervision. Our method achieves superior quality through robust tracking, depth-supervised Gaussian placement, and efficient chunk-based optimization.

Gaussians for our experiments. When loading new chunks would cause the system to exceed this limit, we evict and save existing chunks from VRAM to disk using a Least Recently Used (LRU) principle. After ensuring all necessary chunks are in memory, we use the chunk visibility mask to index only the visible Gaussians for rendering. These selected Gaussians are passed to the rasterizer, which renders RGB and depth images from the selected keyframe. The rendered images are then used to compute gradients and optimize the Gaussian parameters according to the loss function defined in Equation 2.

LRU Eviction Strategy: The LRU eviction operates as a greedy selection process. Given evictable chunks sorted by access time (oldest first), we iteratively select the minimum number of chunks needed to accommodate incoming data. This ensures minimal disruption by preserving recently accessed regions and maintaining spatial coherence.

Beyond chunk eviction, we employ an independent LRU queue to manage keyframe memory with a budget of 400 keyframes. Selected keyframes are loaded from disk on demand and added to the front of the queue; when the budget is exceeded, the least recently used keyframes are saved to disk and evicted.

Keyframe Selection

Intelligent keyframe selection is paramount for our chunk-based 3DGS architecture. The fundamental challenge lies in balancing two competing objectives: maintaining spatial locality for efficient I/O operations while ensuring sufficient gradient diversity for stable optimization. Poor keyframe selection can result in either computational thrashing due to excessive chunk swapping or catastrophic forgetting resulting from insufficient viewing constraints.

Let $\mathcal{K}_t$ be the set of available keyframes at time t, $\mathcal{C}_{active}$ represent the set of chunks currently loaded in VRAM, and $\mathcal{C}_v(k)$ represent the set of visible chunks for keyframe k. When operating near our VRAM budget limit of 1.5 million Gaussians, the chunk overlap between a candidate keyframe's visible chunks and the currently active chunk set becomes critical for I/O efficiency:

$$\text{Overlap}(k) = \frac{|\mathcal{C}_v(k) \cap \mathcal{C}_{active}|}{|\mathcal{C}_v(k)|} \tag{7}$$

High overlap values (approaching 1.0) indicate that most chunks required by keyframe k are already in VRAM, minimizing loading operations. Conversely, low overlap values require loading many new chunks. In large-scale scenes where the system operates near its VRAM budget, low overlap values trigger our LRU eviction mechanism, causing excessive chunk swapping that would bring training to a near halt due to I/O bottlenecks.

To address this fundamental tension between spatial locality and gradient diversity, we organize keyframes into a spatial grid with resolution $g = 200$m. Note that this keyframe selection grid operates independently from and at a much coarser resolution than the chunk grid used for Gaussian partitioning. For any keyframe position p_{latest} of the most recently added keyframe, we define the spatial candidate set as:

$$\mathcal{K}(p) = \{k \in \mathcal{K}_t : \lfloor p_k/g \rfloor = \lfloor p/g \rfloor\} \tag{8}$$

where $\lfloor p/g \rfloor$ applies element-wise division and floor to discretize 3D positions to grid coordinates at resolution g.

At each optimization step, we identify the spatial candidate set $\mathcal{K}(p_{latest})$ containing all keyframes within the same spatial

grid cell as the most recent keyframe. Within this spatially constrained set, we apply the same usage-based and loss-weighted selection strategy as CaRtGS [14], which prioritizes keyframes with remaining usage allocations and provides additional allocations to high-loss keyframes for adaptive focus on challenging views.

This spatial grid approach provides several key advantages: when revisiting previously mapped regions, all historically relevant keyframes within the spatial locality contribute their viewing constraints regardless of temporal distance, preventing catastrophic forgetting. By constraining selection to keyframes within spatial grid cell g, we minimize I/O operations through high chunk overlap.

Loop Closure

While traditional voxel-based systems often require expensive global optimization or complete rebuilding of affected regions during loop closure, our chunked 3DGS system enables more targeted corrections. Our approach identifies affected keyframes and selectively transforms only the visible Gaussian points in relevant chunks.

Cross-Chunk Transformation: When loop closure updates keyframe poses, we apply the same transformation to the affected Gaussians to maintain consistency between the scene representation and corrected camera trajectory. For each affected keyframe, we perform frustum culling to identify visible chunks and apply the pose correction transformation to both the positions and rotations of their Gaussians.

To minimize I/O overhead, we collect all unique chunks across affected keyframes and estimate the total number of Gaussians requiring transformation. If this remains within our memory budget, we batch load all chunks in a single operation and use a global transformation mask to prevent redundant processing of Gaussians visible to multiple keyframes. For extensive loop closures that exceed memory constraints, we process keyframes sequentially using per-keyframe chunk loading.

Chunk Redistribution: After transformation, Gaussians may cross chunk boundaries. We handle this by recomputing chunk assignments based on updated positions and redistributing moved Gaussians to their correct chunks, ensuring the spatial hierarchy remains consistent.

Post-Correction Refinement: After geometric transformation and redistribution, we perform targeted optimization to refine the affected regions. We pause new frame ingestion, and for keyframes where independent reconstructions merge (loop closure detection points), we reset Gaussian opacity and optimizer states to allow fresh optimization of the junction region. This extended refinement runs for 1k iterations (roughly 10s), ensuring better blending between previously disconnected map sections.

IV. Experiments

Datasets and Metrics

While most current 3DGS SLAM methods are limited to small-scale indoor environments due to memory constraints, our approach enables evaluation on long-sequence outdoor scenarios. To demonstrate the versatility of our method across different settings, we evaluate on both established indoor datasets and outdoor sequences. More specifically, we evaluate our method on datasets Replica [7], TUM-RGBD [8], and KITTI [6]. Replica [7] is a high-quality indoor dataset featuring photorealistic 3D reconstructions of various room environments. TUM-RGBD [8] is a dataset containing RGB-D sequences captured in different indoor environments (offices, hallways, households) with handheld cameras. KITTI [6] is a comprehensive multi-kilometer outdoor dataset collected from a moving vehicle that includes stereo images, LiDAR scans, and GPS measurements.

We evaluate localization accuracy using Absolute Tracking Error (ATE), and assess reconstruction quality through Peak Signal-to-Noise Ratio (PSNR), SSIM, and Learned Perceptual Image Patch Similarity (LPIPS). Among these metrics, LPIPS best correlates with perceptual reconstruction quality [20]. For computational efficiency, we measure the total wall-clock time required to process each complete sequence, which provides the most transparent measure of real-world deployment performance. We report this as processing FPS (total frames ÷ total time) to enable fair comparison across methods processing identical frame sequences, independent of their internal optimization strategies. Note that datasets with lower capture rates (e.g., KITTI at 10 FPS) naturally yield lower processing FPS values due to fewer total frames, regardless of per-frame processing efficiency.

TABLE I
SLAM Tracking Accuracy on KITTI [6] dataset (ATE in meters) ↓. **Best** and second best results are highlighted. × = OOM crash, – = tracking loss. S = Stereo, M = Mono. OTF = On-The-Fly [16], Giga = GigaSLAM [15]

Seq.	CaRtGS (S)	OTF (M)	Giga (M)	Ours-5 (S)	Ours-20 (S)
00	×	20.90	×	**0.82**	0.89
01	26.14	–	74.48	16.41	**9.99**
02	×	–	×	4.50	**4.05**
03	0.37	–	1.49	**0.32**	0.33
04	**0.17**	–	1.78	**0.17**	**0.17**
05	0.43	2.38	×	0.41	**0.37**
06	**0.42**	9.69	1.20	0.58	0.72
07	0.44	21.21	3.41	**0.35**	0.40
08	×	–	×	**2.99**	3.18
09	**0.99**	–	3.86	**0.99**	1.02
10	1.34	19.75	2.45	1.34	**1.18**

TABLE II
Processing Speed on KITTI [6] dataset (Processing FPS) ↑. **Best** and second best results are highlighted. × = OOM crash, – = tracking loss. S = Stereo, M = Mono. OTF = On-The-Fly [16], Giga = GigaSLAM [15]

Seq.	CaRtGS (S)	OTF (M)	Giga (M)	Ours-5 (S)	Ours-20 (S)
00	×	**11.39**	×	1.51	4.94
01	0.59	–	**3.30**	0.59	2.26
02	×	–	×	1.15	**3.69**
03	1.80	–	2.55	1.81	**6.54**
04	0.88	–	2.25	0.89	**3.35**
05	1.59	**15.85**	×	1.53	4.69
06	1.15	**11.02**	1.88	1.13	3.95
07	2.11	**11.83**	2.26	2.05	6.59
08	×	–	×	1.61	**5.82**
09	1.20	–	1.85	1.18	**4.16**
10	1.65	**12.17**	2.09	1.66	5.90

TABLE III

RENDERING QUALITY ON KITTI [6]. NOVEL VIEW SYNTHESIS COMPARISON. – INDICATES TRACKING LOSS AND × INDICATES OUT-OF-MEMORY. **BEST** AND SECOND BEST RESULTS ARE HIGHLIGHTED.

Method	Metric	00	01	02	03	04	05	06	07	08	09	10
Frames	-	4541	1101	4661	801	271	2761	1101	1101	4071	1591	1201
Length (km)	-	3.72	2.45	5.07	0.56	0.39	2.21	1.23	0.65	3.22	1.71	0.92
Loop Closure	-	✓	×	✓	×	×	✓	✓	✓	✓	✓	×
CaRtGS [14] (Stereo)	PSNR ↑	×	**23.14**	×	19.81	21.97	18.16	17.92	17.43	×	18.29	19.97
	SSIM ↑	×	0.74	×	0.56	0.72	0.58	0.56	0.58	×	0.56	0.63
	LPIPS ↓	×	0.31	×	0.43	0.29	0.39	0.39	0.37	×	0.43	0.37
On-the-fly [16] (Mono)	PSNR ↑	17.01	–	–	–	–	17.10	18.44	18.56	–	–	18.22
	SSIM ↑	0.59	–	–	–	–	0.58	0.61	0.64	–	–	0.58
	LPIPS ↓	0.46	–	–	–	–	0.44	0.40	0.38	–	–	0.47
GigaSLAM [15] (Mono)	PSNR ↑	×	15.86	×	16.09	14.74	×	14.72	14.53	×	15.56	15.75
	SSIM ↑	×	0.50	×	0.39	0.33	×	0.44	0.42	×	0.41	0.44
	LPIPS ↓	×	0.62	×	0.64	0.63	×	0.70	0.68	×	0.66	0.68
Ours 5km/h (Stereo)	PSNR ↑	**20.33**	22.07	**20.35**	**20.84**	**21.98**	**20.08**	19.67	**20.09**	**20.33**	**21.15**	**22.89**
	SSIM ↑	**0.72**	**0.77**	**0.69**	**0.68**	**0.78**	**0.70**	**0.68**	**0.74**	**0.72**	**0.73**	**0.78**
	LPIPS ↓	**0.26**	**0.26**	**0.28**	**0.31**	**0.23**	**0.28**	**0.31**	**0.25**	**0.27**	**0.25**	**0.22**
Ours 20km/h (Stereo)	PSNR ↑	19.65	21.54	19.67	19.15	20.76	19.01	**19.75**	19.83	19.30	20.41	21.63
	SSIM ↑	0.70	**0.77**	0.67	0.64	0.75	0.67	**0.68**	0.73	0.69	0.72	0.75
	LPIPS ↓	0.31	**0.26**	0.31	0.35	0.26	0.31	0.32	0.26	0.31	0.28	0.26

Implementation Details

We conduct all experiments on an AMD Ryzen 5950X CPU, 64GB of RAM, PCIe Gen 4 NVMe SSD, and an Nvidia RTX 3090 GPU (24GB VRAM). Since CaRtGS [14], GigaSLAM [15], and On-The-Fly [16] originally reported results on different hardware, we rerun and evaluate these methods using their publicly available code on our hardware, enabling fair comparison of tracking, visual, and performance metrics. We acknowledge that some methods require GPUs with greater VRAM capacity than our current hardware; however, our disk-based approach doesn't suffer from this memory limitation. To ensure fair comparison of online processing capabilities, we evaluate all methods using their online-only performance. For GigaSLAM [15], this means excluding its offline post-processing stage. We note that GigaSLAM's [15] published results include extensive offline refinement, which can achieve higher quality but requires substantial additional computation time, commonly an hour or more per sequence.

CaRtGS [14] and On-The-Fly [16] have not been previously evaluated on KITTI [6] by their original authors. For On-The-Fly [16], we use the default hyperparameters provided by the authors, which are designed to generalize across datasets. For CaRtGS [14], initially intended for indoor scenes, we fine-tune hyperparameters to optimize performance on outdoor sequences. Our method uses a chunk size of 10 meters for all datasets.

Our method adapts to available processing time, utilizing whatever time exists between incoming frames for map refinement. To demonstrate this adaptability, we evaluate at vehicle speeds of 5 km/h and 20 km/h on the KITTI [6] dataset, corresponding to longer and shorter inter-frame intervals. CaRtGS [14] operates similarly and is evaluated at 5 km/h, while other baseline methods use their default configurations with fixed iteration counts per frame.

Results

Quantitative results for KITTI [6] are shown in Tables I, II and III. Our method is the only approach that successfully reconstructs all 11 sequences without tracking loss or memory crashes at both intake rates. In contrast, CaRtGS [14] and GigaSLAM [15] crash on 3 and 4 sequences, respectively, due to VRAM limitations. In comparison, On-the-fly [16] loses tracking on 6 sequences due to insufficient tracking robustness and a lack of loop closure capabilities. Where direct comparisons are possible, our method consistently achieves superior reconstruction quality, ranking first or second across most metrics, with particular strength in perceptual quality (LPIPS). At the 20 km/h intake rate, our method achieves higher frame rates with graceful quality degradation compared to 5 km/h results, demonstrating effective scalability without algorithmic modifications. While On-the-fly [16] is faster when it succeeds, its frequent tracking failures limit practical applicability. GigaSLAM [15], evaluated in online-only mode as discussed in our implementation details, shows reduced visual quality without its offline refinement stage. These quantitative advantages translate to realistic large-scale reconstructions as visualized in Fig. 3. We note that our comparison involves different input modalities (mono vs. stereo), reflecting each method's design choices for handling challenging outdoor scenarios. However, we believe this difference is secondary to the fundamental scalability challenges we address, which occur regardless of input modality.

We observe that our method produces excellent tracking and visual quality results on the Replica [7] dataset, demonstrating strong generalization capabilities across both small and large-scale scenes. Our approach achieves the highest average processing FPS (32.41) on Replica [7] among compared methods, as shown in Table IV. More importantly, we achieve the best reconstruction quality on Replica [7] across all visual metrics (PSNR: 34.59, SSIM: 0.94, LPIPS: 0.05), with the superior visual quality evident in Fig. 4, where zoomed-in renders reveal reduced artifacts and finer detail preservation compared to baseline methods. On the TUM [8] dataset, our method produces competitive results, ranking second across all metrics while maintaining efficient runtime performance.

The Pareto curves in Figure 1 reveal the complete quality-

time trade-off spectrum that single-point comparisons in traditional result tables cannot capture. We evaluate on KITTI [6] scenes 06, 07, and 10, as these are the only sequences where all competing methods complete processing without crashes or tracking failures. We evaluate performance above 1 FPS to focus on methods suitable for online operation. Our method (orange) consistently dominates this trade-off across all three scenes, achieving superior LPIPS scores in significantly less time than competing approaches.

Table V validates our design choices on the KITTI [6] dataset. Chunking proves essential, as omitting this component causes 6 of 11 sequences to fail due to memory constraints. Removing spatial grid keyframe selection causes 2 failures from excessive chunk thrashing. Depth supervision improves perceptual quality (LPIPS: 0.27 vs. 0.28) and structural similarity (SSIM: 0.73 vs. 0.72) while maintaining full sequence completion. These results confirm that chunking enables scalability, grid-based selection ensures efficient I/O, and depth supervision enhances reconstruction quality.

Our chunk-based architecture incurs modest disk overhead while scaling beyond GPU memory constraints. On KITTI [6] scene 02, the longest sequence at 5.07 km, chunk I/O operations consume 5.5% of total processing time at the 5 km/h intake rate. At 20 km/h, I/O overhead increases to 19.6% since absolute I/O costs remain similar while total processing time decreases with less optimization per frame. Disk storage remains practical across all sequences, ranging from 0.3 GB (scene 04) to 7.2 GB (scene 02), with an average of 2.3 GB per sequence, demonstrating algorithmic chunking with modern NVMe SSDs as a viable alternative to hardware-constrained scaling.

To demonstrate our chunking system's effect on VRAM consumption, we plot the allocated VRAM usage for KITTI [6] scene 02 in Figure 5. The initial rise reflects the growing number of Gaussians and keyframes being added to the active set. Once the system reaches the Gaussian budget limit of 1.5 million (lower dashed line), our LRU-based chunk eviction mechanism begins saving older chunks to disk. Similarly, when the number of keyframes reaches 400 (upper dashed line), our disk-based management system maintains only spatially relevant keyframes in memory. As evident from the plateau in VRAM usage beyond these thresholds, our system successfully maintains constant memory consumption while the total number of Gaussians and keyframes continues to grow throughout the sequence.

We test our method on Nvidia's Jetson AGX Orin, an edge AI computing platform with energy efficiency suitable for robotics, drones, and other mobile computing scenarios. Quantitative results in Table VI show that our method maintains competitive performance on the resource-constrained platform with only modest degradation in rendering quality and increased processing time, demonstrating practical viability for mobile applications.

Fig. 4. **Zoomed qualitative results on the Replica [7] dataset (office0).**

TABLE IV
COMPARISON OF METHODS ON THE TUM [8] AND REPLICA [7] DATASETS. ATE IS IN CM. **BEST** AND SECOND BEST RESULTS ARE HIGHLIGHTED.

Metric	CaRtGS	On-The-Fly	GigaSLAM	Ours
TUM				
ATE ↓	**0.85**	7.12	22.02	1.88
PSNR ↑	20.59	**23.28**	13.40	21.21
SSIM ↑	0.71	**0.84**	0.53	0.75
LPIPS ↓	0.22	**0.14**	0.68	0.22
FPS ↑	24.14	**51.88**	10.99	25.09
Replica				
ATE ↓	**0.30**	31.68	11.54	0.33
PSNR ↑	33.85	26.91	21.11	**34.59**
SSIM ↑	**0.94**	0.89	0.78	**0.94**
LPIPS ↓	0.07	0.16	0.38	**0.05**
FPS ↑	27.65	22.96	5.65	**32.41**

TABLE V
COMPONENT ABLATION: SUCCESS RATE AND QUALITY

Configuration	Success	Failed	PSNR↑	SSIM↑	LPIPS↓
w/o Chunks	5	6	–	–	–
w/o Grid KF	9	2	–	–	–
w/o Depth Sup.	11	0	20.90	0.72	0.28
Full Method	11	0	20.89	0.73	0.27

TABLE VI
RESULTS ON NVIDIA JETSON. ATE IS IN CM.

Metrics	TUM		Replica		KITTI	
	Desktop	Jetson	Desktop	Jetson	Desktop	Jetson
ATE ↓	1.88	4.09	0.33	0.33	262.54	239.23
PSNR ↑	21.21	20.48	34.59	32.73	20.89	19.10
SSIM ↑	0.75	0.74	0.94	0.93	0.73	0.67
LPIPS ↓	0.22	0.24	0.05	0.07	0.27	0.33
FPS ↑	25.09	13.19	32.41	12.29	1.37	1.13

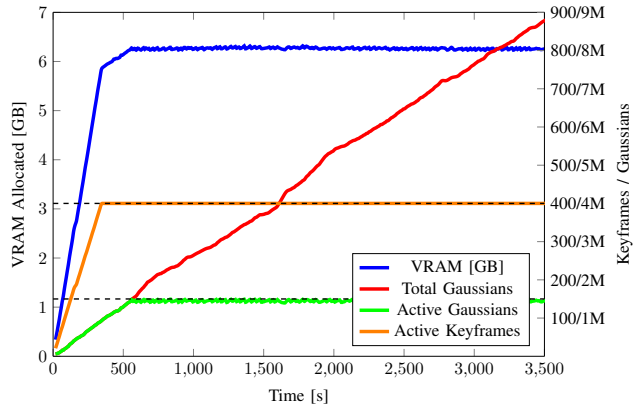

Fig. 5. **Active Gaussians and Keyframes vs VRAM usage.** Our Gaussian and Keyframe disk-based saving and loading system keeps memory usage steady as scene size increases.

V. Limitations & Future Work

Our DiskChunGS method provides an algorithmic solution for scaling 3DGS SLAM through efficient disk-based Gaussian storage. Like other 3DGS methods, it does not explicitly handle capture artifacts such as motion blur, lens flare, or dynamic objects.

Our locality-focused keyframe strategy minimizes I/O by avoiding excessive chunk swapping, enabling efficient scaling. However, regions visible to only a subset of keyframes may receive fewer optimization constraints than methods with global keyframe access, occasionally causing sparse floating artifacts in transitions. Future work could leverage GPU Direct Storage to reduce chunk-swapping costs and enable more flexible optimization strategies.

Additionally, integrating a 3DGS Level of Detail (LoD) system would address current limitations in rendering distant objects, and evaluation on ultra-long sequences (10 km+) would further demonstrate scalability.

VI. Conclusion

We present DiskChunGS, an out-of-core 3DGS SLAM system that overcomes memory limitations through spatial chunking and disk-based management. By treating large-scale reconstruction as an algorithmic rather than hardware challenge, our method scales to multi-kilometer environments on standard GPUs where previous approaches fail. Evaluations demonstrate superior visual quality across diverse scenarios, from indoor spaces to urban driving sequences. Validated on resource-constrained platforms like the Jetson Orin and integrated with ROS, DiskChunGS enables practical photorealistic large-scale 3D reconstruction for real-world robotics.

References

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[2] 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," 2020. [Online]. Available: https://arxiv.org/abs/2003.08934

[3] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2112.05131

[4] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, p. 1–15, July 2022. [Online]. Available: http://dx.doi.org/10.1145/3528223.3530127

[5] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, p. 1874–1890, Dec. 2021. [Online]. Available: http://dx.doi.org/10.1109/TRO.2021.3075644

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, "The replica dataset: A digital replica of indoor spaces," 2019. [Online]. Available: https://arxiv.org/abs/1906.05797

[8] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[9] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-nerf: Scalable large scene neural view synthesis," 2022. [Online]. Available: https://arxiv.org/abs/2202.05263

[10] H. Turki, D. Ramanan, and M. Satyanarayanan, "Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 12 922–12 931.

[11] Y. Liu, H. Guan, C. Luo, L. Fan, N. Wang, J. Peng, and Z. Zhang, "Citygaussian: Real-time high-quality large-scale scene rendering with gaussians," 2024. [Online]. Available: https://arxiv.org/abs/2404.01133

[12] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, and X. Li, "Gs-slam: Dense visual slam with 3d gaussian splatting," 2024. [Online]. Available: https://arxiv.org/abs/2311.11700

[13] H. Huang, L. Li, H. Cheng, and S.-K. Yeung, "Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and rgb-d cameras," 2024. [Online]. Available: https://arxiv.org/abs/2311.16728

[14] D. Feng, Z. Chen, Y. Yin, S. Zhong, Y. Qi, and H. Chen, "Cartgs: Computational alignment for real-time gaussian splatting slam," *IEEE Robotics and Automation Letters*, vol. 10, no. 5, p. 4340–4347, May 2025. [Online]. Available: http://dx.doi.org/10.1109/LRA.2025.3544928

[15] K. Deng, J. Yang, S. Wang, and J. Xie, "Gigaslam: Large-scale monocular slam with hierachical gaussian splats," 2025. [Online]. Available: https://arxiv.org/abs/2503.08071

[16] A. Meuleman, I. Shah, A. Lanvin, B. Kerbl, and G. Drettakis, "On-the-fly reconstruction for large-scale novel view synthesis from unposed images," *ACM Transactions on Graphics*, vol. 44, no. 4, p. 1–14, July 2025. [Online]. Available: http://dx.doi.org/10.1145/3730913

[17] V. Yugay, Y. Li, T. Gevers, and M. R. Oswald, "Gaussian-slam: Photo-realistic dense slam with gaussian splatting," 2024. [Online]. Available: https://arxiv.org/abs/2312.10070

[18] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," 2024. [Online]. Available: https://arxiv.org/abs/2406.09414

[19] G. Xu, Y. Wang, J. Cheng, J. Tang, and X. Yang, "Accurate and efficient stereo matching via attention concatenation volume," 2023. [Online]. Available: https://arxiv.org/abs/2209.12699

[20] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," 2018. [Online]. Available: https://arxiv.org/abs/1801.03924