LoBE-GS: Load-Balanced and Efficient 3D Gaussian Splatting for Large-Scale Scene Reconstruction

Anonymous authors
Paper under double-blind review

Abstract

3D Gaussian Splatting (3DGS) has established itself as an efficient representation for real-time, high-fidelity 3D scene reconstruction. However, scaling 3DGS to large and unbounded scenes such as city blocks remains difficult. Existing divide-and-conquer methods alleviate memory pressure by partitioning the scene into blocks, but introduce new bottlenecks: (i) partitions suffer from severe load imbalance since uniform or heuristic splits do not reflect actual computational demands, and (ii) coarse-to-fine pipelines fail to exploit the coarse stage efficiently, often reloading the entire model and incurring high overhead. In this work, we introduce LoBE-GS, a novel ***Lo**ad-**B**alanced and **E**fficient* 3D Gaussian Splatting framework, that re-engineers the large-scale 3DGS pipeline. LoBE-GS introduces a depth-aware partitioning method that reduces preprocessing from hours to minutes, an optimization-based strategy that balances visible Gaussians—a strong proxy for computational load—across blocks, and two lightweight techniques, visibility cropping and selective densification, to further reduce training cost. Evaluations on large-scale urban and outdoor datasets show that LoBE-GS consistently achieves up to $2\times$ faster end-to-end training time than state-of-the-art baselines, while maintaining reconstruction quality and enabling scalability to scenes infeasible with vanilla 3DGS.

1 Introduction

Recent advances in 3D scene reconstruction and novel view synthesis have shifted from classical photogrammetry and Neural Radiance Fields (NeRFs) toward explicit, real-time representations. While photogrammetry offers geometric precision but poor rendering efficiency, NeRFs achieve photorealism but remain computationally expensive. 3D Gaussian Splatting (3DGS) addresses these limitations by representing scenes with millions of anisotropic Gaussian primitives optimized through a GPU-friendly rasterization pipeline, delivering both high fidelity and real-time performance. Its efficiency has quickly established 3DGS as a leading representation for scalable 3D content creation.

Despite its success in bounded scenes, scaling 3DGS to large and unbounded environments, such as city-scale reconstructions, remains an open challenge. The memory and computational costs scale with the number of Gaussian primitives, leading to optimization times and GPU usage that quickly become prohibitive. To mitigate this, recent works such as CityGaussian (CityGS) (Liu et al., 2025), VastGaussian (VastGS) (Lin et al., 2024), and DOGS (Chen & Lee, 2024) adopt a divide-and-conquer strategy, partitioning large scenes into spatial blocks that can be processed in parallel. While effective in reducing raw memory pressure, this paradigm introduces new bottlenecks as follows.

- **Lack of load balancing:** Current partitioning strategies do not explicitly account for computational load balance. Heuristics such as uniform grid splits or block size normalization often yield sub-regions with highly uneven optimization demands. As a result, the slowest block dominates the total training time, creating a long-tail bottleneck.
- **Inefficient coarse-to-fine pipelines:** Methods employing a coarse-to-fine pipeline, such as CityGS (Liu et al., 2025), fail to fully exploit the coarse stage for accelerating fine-level

optimization. The coarse model is typically reloaded in full, incurring heavy computational overhead.

To overcome these limitations, we introduce **LoBE-GS**, a novel framework that fundamentally re-engineers the large-scale 3DGS pipeline for load-balanced and efficient parallel training. LoBE-GS addresses the inefficiency of heuristic partitioning, improves the utilization of coarse models, and establishes a standardized evaluation protocol. We first introduce a novel partitioning approach that radically reduces the data partitioning time. Existing methods can result in a complex $O(M \times N)$ projection problem, where M is the number of blocks and N is the number of camera views, requiring up to several hours. Our method leverages depth information from a coarse model to assign each camera to its corresponding block with a single, highly efficient projection per camera. This reduces the projection complexity to a linear $O(N)$ time and shortens the preprocessing time from hours to minutes.

To avoid unbalanced loading in each block for the parallel training, we employee an optimization to scene partitioning that directly addresses the load-balancing problem. Our experiments revealed a strong correlation between the initial number of visible Gaussians in the blocks and the subsequent optimization time. We therefore adopt the number of visible Gaussians as a reliable *proxy for computational load*. By explicitly balancing this metric across blocks, our framework eliminates long-tailed training bottlenecks and ensures more uniform computational demands. Moreover, we propose two complementary techniques to reduce the computational load of each block. First, we introduce *visibility cropping*, a technique applied after scene partitioning to prune irrelevant Gaussians from each block, which reduces the training time without sacrificing the quality of the final reconstruction. Second, we propose *selective densification* to further reduce the computational load of each block by strategically adding or cloning Gaussians only when needed.

We evaluate LoBE-GS on diverse large-scale datasets, including urban and outdoor scenes spanning hundreds of meters. Experimental results show that our method consistently delivers faster training and more balanced computation than prior approaches, while maintaining or improving reconstruction quality. In particular, LoBE-GS reduces end-to-end training time by up to $2\times$ over baselines that use coarse models and achieves stable scalability on scenes that are otherwise infeasible for vanilla 3DGS. The main contributions of this work are summarized as follows:

- We identify load-balancing limitations in prior approaches and introduce a proxy that more closely correlates with fine-training runtime, enabling improved load balancing.
- We present LoBE-GS, featuring (i) load balance-aware scene partitioning for evenly distributed computational workloads, (ii) fast camera selection to minimize partition overhead, and (iii) visibility cropping and selective densification for accelerated fine-training.
- Extensive experiments show that LoBE-GS achieves a $2\times$ training speedup over existing methods while preserving rendering quality.

2 Related Work

2.1 Novel View Synthesis

Given a set of captured images, novel view synthesis seeks to render photorealistic 3D scenes from previously unseen viewpoints. Neural Radiance Fields (NeRF) (Mildenhall et al., 2020) model radiance fields with an MLP and use volumetric ray marching to integrate color along camera rays. NeRF delivers high fidelity but incurs substantial training time and inference latency due to dense sampling and repeated neural evaluations. In contrast, 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) adopts Gaussian primitives, enabling differentiable rasterization for real-time rendering and training that often converges within minutes. While 3DGS yields strong quality, open issues include aliasing under wide baselines, semi-transparent geometry leakage, memory growth from millions of primitives, and robustness under sparse views or imperfect calibration. These advances and limitations motivate our design choices and evaluation, specially for large scale reconstruction.

2.2 Large-Scale Scene Reconstruction

For decades, reconstructing large-scale 3D scenes has been a central goal for researchers and engineers (Snavely et al., 2006; Agarwal et al., 2011). At city and regional scales, such reconstruction, especially for aerial views (Jiang et al., 2025; Tang et al., 2025), faces prohibitive memory demands and computational performance, motivating scalable training and rendering strategies.

Distributed training approaches train a unified model jointly across multiple GPUs. NeRF-XL (Li et al., 2024) shares NeRF parameters and activations across devices to maintain a single global model, executing multi-GPU volume rendering and loss computation with low inter-GPU communication, while DOGS (Chen & Lee, 2024) and Grendel-GS (Zhao et al., 2024) distribute Gaussian primitives via consensus or sparse all-to-all routing. However, such systems typically require customized multi-GPU infrastructure to support frequent synchronization and communication, which limits their practicality on standard hardware setups.

Divide-and-conquer approaches partition a large scene into subregions, train submodels in parallel with multiple GPUs, and compose their outputs. Block-NeRF (Tancik et al., 2022) partitions a city into spatial blocks and assigns training views by camera position; Mega-NeRF (Turki et al., 2022a) decomposes space into grids and routes each pixel to the grids intersected by its ray; Switch-NeRF (Mi & Xu, 2023) learns the decomposition and routing end-to-end via a mixture-of-NeRF-experts. Within 3DGS representations, VastGS (Lin et al., 2024) introduces a progressive spatial partitioning strategy that divides a large scene into blocks and assigns training cameras and point clouds using an *airspace*-aware visibility criterion. Each block is optimized in parallel and subsequently fused to a seamless global 3DGS reconstruction. CityGS (Liu et al., 2025) leverages a coarse 3DGS prior to guide scene partitioning and parallel 3DGS submodel training, improving coherence and reconstruction quality across spatial partitions. They map unbounded scenes into a normalized unit cube and then partition the contracted scenes with a uniform grid for parallel training. However, most of the aforementioned works underemphasize load balancing of the submodels during partitioning, which limits parallel scalability. Moreover, CityGS loads the entire coarse model during the parallel stage, which is inefficient. To address these, LoBE-GS balances the 3DGS prior across submodels within each subregion and trains them efficiently in parallel.

2.3 Efficient Gaussian Splatting Reconstruction

As new 3DGS methods emerge, many research efforts target efficient 3D Gaussian Splatting reconstruction and rendering. With limited resources, 3DGS compression (Navaneet et al., 2024; Papantonakis et al., 2024) reduces on-disk storage, while Taming 3DGS (Mallick et al., 2024) addresses budget-constrained training and rendering via guided, purely constructive densification that steers growth toward high-contribution Gaussians. For large-scale scenes, level-of-detail (LoD) 3DGS representations enable efficient rendering (Ren et al., 2024; Kerbl et al., 2024). CityGaussianV2 (Liu et al., 2024) builds on CityGS (Liu et al., 2025) with an optimized parallel training pipeline that incorporates 2DGS for accurate geometric modeling. Momentum-GS (Fan et al., 2024) extends Scaffold-GS (Lu et al., 2024) to large-scale scenes by introducing scene momentum self-distillation and reconstruction-guided block weighting, allowing scalable parallel training with improved reconstruction quality. CityGS-X (Gao et al., 2025) proposes a scalable hybrid hierarchical representation with multitask batch rendering and training, eliminating merge–partition overhead while achieving efficient and geometrically accurate large-scale reconstruction. In this work, we focus on an efficient 3DGS reconstruction for large-scale scenes with coarse 3DGS prior and load-balanced parallel training.

3 Analysis of Scene Partitioning and Load balancing

In this section, we first show that existing scene partitioning strategies fail to achieve satisfactory load balancing during the fine-training stage. We then provide a principled analysis to identify a reliable proxy for estimating the per-block fine-training runtime.

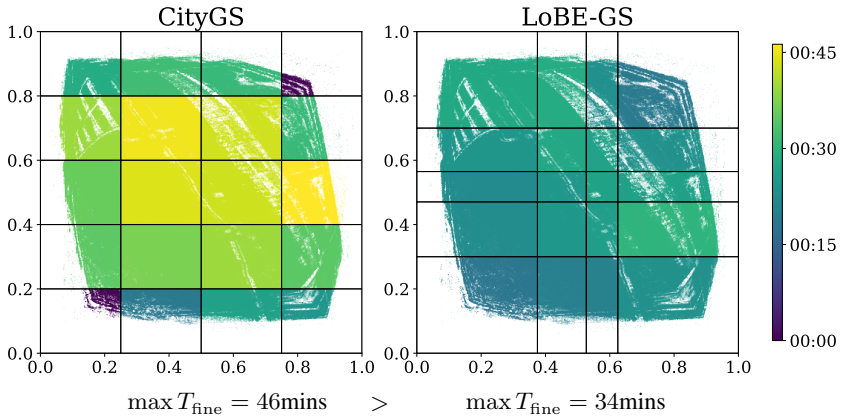

Figure 1: **Illustration of per-block training time under different partitioning strategies.** (Left) The uniform area partitioning in CityGS. (Right) The load-balanced partitioning in LoBE-GS.

3.1 Impact of Scene Partition on Load Balancing

Large-scale 3DGS pipelines typically adopt a *partition–and–merge* paradigm: the scene is divided into B spatial blocks, each optimized independently in parallel, and then merged into a complete model. Some methods further employ a coarse-to-fine strategy, where a coarse model is trained first, followed by scene partitioning and parallel fine training before the final merging stage. Let T_{coarse} denote the coarse-stage optimization time, $T_{\text{partition}}$ the partitioning time, and $T_{\text{fine}}^{(b)}$ the fine-stage runtime of block $b \in \{1, \ldots, B\}$. Assuming sufficient computational resources to run all fine-stage processes in parallel, the end-to-end runtime is defined as:

$$T_{\text{E2E}} = T_{\text{coarse}} + T_{\text{partition}} + \max_{b \in \{1,\ldots,B\}} T_{\text{fine}}^{(b)}. \tag{1}$$

Thus, an effective partitioning strategy must balance the workloads across blocks to minimize the runtime of the slowest block while maintaining reconstruction quality. Prior work has relied on heuristics such as equalizing *area*, *camera counts*, or *point counts*, yet their ability to predict fine-stage runtime were underexplored.

As a motivational example, consider the CityGS pipeline, which partitions the scene by equalizing block *areas* in contracted space. Figure 1 illustrates the fine-stage runtime per block on the *Building* dataset. Figure 1(a) shows that the strategy adopted by CityGS leads to significant load imbalance in fine-stage training. In contrast, Figure 1(b) shows that LoBE-GS achieves a more balanced runtime distribution by employing a different proxy. Similar patterns are observed across other datasets (see Appendix A.2), suggesting that existing heuristics are often suboptimal for actual fine-stage runtimes. As a result, they lead to skewed per-block runtimes and longer end-to-end runtime T_{E2E}.

3.2 Runtime Correlation with Per-Block Predictors

To address this, we analyze the correlation between candidate proxy variables and observed fine-stage runtimes to determine which predictors most accurately reflect the computational cost of each block. For each block b, we computed the Pearson correlation between its fine-stage runtime $T_{\text{fine}}^{(b)}$ (in minutes) and the following quantities, all available prior to fine-stage optimization:

- $A^{(b)}$: area of block b in contracted space.
- $C^{(b)}$: number of cameras assigned to block b.
- $G_{\text{blk}}^{(b)}$: initial number of Gaussians inside block b at the start of fine-stage optimization.
- $G_{\text{vis}}^{(b)}$: initial number of Gaussians visible across all cameras assigned to block b.
- $G_{\text{avg\_vis}}^{(b)} = G_{\text{vis}}^{(b)} / C^{(b)}$: initial average number of visible Gaussians per assigned camera.

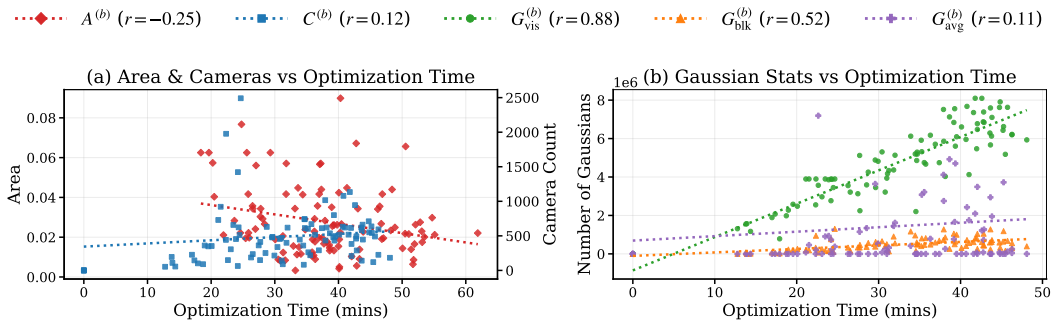

Figure 2: **Correlation between per-block training time and block-level statistics under CityGS's partitioning.** (a) Plots of block area $A^{(b)}$ and camera count $C^{(b)}$. (b) Plots of Gaussian-based measures ($G^{(b)}_{\text{blk}}$, $G^{(b)}_{\text{vis}}$, $G^{(b)}_{\text{avg\_vis}}$). $G^{(b)}_{\text{vis}}$ yields the strongest and most consistent correlation across datasets.

Figure 2 presents scatter plots across five representative datasets, *Building*, *Rubble*, *Residence*, *SciArt*, and *MatrixCity*, evaluated under fixed hardware and hyperparameters. Each point is color-coded by a candidate proxy variable, with fine-stage runtime on the x-axis and the corresponding proxy value on the y-axis. For each proxy, a dashed line of the same color hue is fit using linear regression. The legend also reports the Pearson correlation coefficients (r) between $T^{(b)}_{\text{fine}}$ and the respective block-level quantities.

The results indicate that the *area* proxy $A^{(b)}$, commonly adopted in prior works (Liu et al., 2025; 2024; Fan et al., 2024), exhibits relatively weak correlation with fine-stage runtime. Similarly, the per-block Gaussian count $G^{(b)}_{\text{blk}}$ shows minimal correlation, implying that considering only Gaussians physically contained within a block underestimates the effective optimization load. In contrast, the visibility-augmented measure $G^{(b)}_{\text{vis}}$ achieves the strongest and most consistent correlation across datasets, confirming its suitability as a reliable predictor of per-block training cost. Normalizing this quantity by camera count, resulting in $G^{(b)}_{\text{avg\_vis}}$, weakens the correlation, while the camera count alone, $C^{(b)}$, also used in previous studies (Chen & Lee, 2024; Yuan et al., 2025), exhibits only weak correlation. Overall, these findings suggest that balancing partitions by the number of initial visible Gaussians $G^{(b)}_{\text{vis}}$, as implemented in the proposed LoBE-GS, provides a more principled strategy than traditional equal-area or equal-camera approaches.

4 Methodology

Prior large-scale 3DGS systems have demonstrated strong results but continue to face challenges with load imbalance and training efficiency. To address these limitations, we propose LoBE-GS, a coarse-to-fine training framework where each block is fine-trained independently, following prior works (Liu et al., 2025; 2024). The overall pipeline is illustrated in Figure 3. Section 4.1 introduces *load balance-aware scene partition* that iteratively refines initial uniform cuts to minimize a proxy for fine-stage runtime. In Section 4.2, *fast camera selection* is proposed to improve efficiency over existing camera selection strategies. Finally, Section 4.3 describes *visibility cropping* and *selective densification*, two techniques that further reduce memory and computation costs during fine-training.

4.1 Load balance-aware scene partition

To mitigate load imbalance, we propose *load balance-aware scene partition* that minimizes maximum fine-training time $\max_b T^{(b)}_{\text{fine}}$ by leveraging proxy metrics $\max_b G^{(b)}_{\text{vis}}$, which exhibit strong correlation with fine-stage runtimes as analyzed in Section 3.2. For a grid partition with $B = m \times n$ blocks, given a coarse model $\mathcal{G}_{\text{coarse}}$ and a set of c camera views, the objective is to optimize vertical

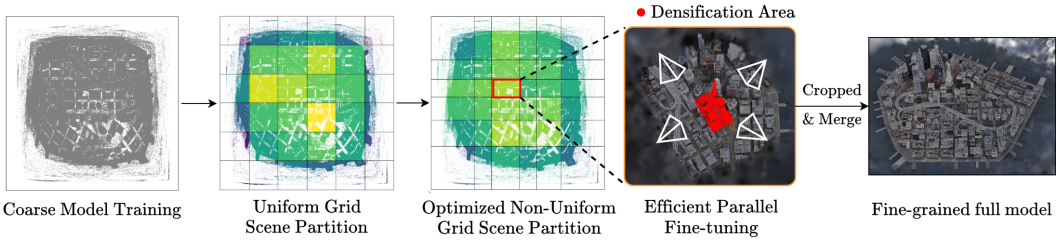

Figure 3: **Overview of our framework.** Our approach begins with training a coarse 3DGS model. Using our load balance–aware data partition, we optimize the grid cuts to achieve a more balanced division of the scene. We then apply visibility cropping and selective densification before and during the parallel fine-training stage, enabling faster and more efficient training. Finally, we prune regions outside each block and merge the results into a unified, high-quality model.

and horizontal cut positions $(\boldsymbol{v}, \boldsymbol{h})$ such that:

$$(\boldsymbol{v}^*, \boldsymbol{h}^*) = \arg\min_{\boldsymbol{v},\boldsymbol{h}} \max_b G_{\text{vis}}^{(b)}(\boldsymbol{v}, \boldsymbol{h}), \tag{2}$$

where $\boldsymbol{v} = (v_1, \ldots, v_{m-1}) \in (0,1)^{m-1}$ and $\boldsymbol{h} = (h_1, \ldots, h_{n-1}) \in (0,1)^{n-1}$ denotes monotonically increasing cut positions in contracted space. The proxy $G_{\text{vis}}^{(b)}(\boldsymbol{v}, \boldsymbol{h})$ denotes the number of visible Gaussians in block $b = (i-1)(n+1)+j$ for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, as defined by the corresponding cut boundaries $\mathcal{B}^{(b)}$ for the i-th row, j-th column block.

Since the computation of $G_{\text{vis}}^{(b)}(\boldsymbol{v}, \boldsymbol{h})$ is non-differentiable, we adopt Bayesian Optimization (BO) with a Gaussian Process (GP) surrogate for iterative cut refinement. The process begins with an initial uniform partition $(\boldsymbol{v}^{[0]}, \boldsymbol{h}^{[0]})$, where $(v_i^{[0]}, h_j^{[0]}) = (\frac{i}{m}, \frac{j}{n})$. To preserve ordering, each cut is constrained to move at most halfway toward its neighbors, i.e., $v_i \in [\frac{1}{2}(v_{i-1}^{[0]} + v_i^{[0]}), \frac{1}{2}(v_i^{[0]} + v_{i+1}^{[0]})]$ with $v_0 = 0$ and $v_m = 1$ (defined analogously for h_j). At each iteration l, BO proposes candidate cuts $(\boldsymbol{v}^{[l]}, \boldsymbol{h}^{[l]})$. The corresponding block regions are set to slightly enlarged grid cell, $\mathcal{B}^{(b)} = [v_{i-1}^{[l]} - \delta_v, v_i^{[l]} + \delta_v] \times [h_{j-1}^{[l]} - \delta_h, h_j^{[l]} + \delta_h]$, following prior works. Each block is then assigned a camera set $\mathcal{C}^{(b)}$ using standard view assignment strategies, and the number of visible Gaussians $G_{\text{vis}}^{(b)}$ is calculated. The GP surrogate is updated to fit the observed $\max_b G_{\text{vis}}^{(b)}(\boldsymbol{v}^{[l]}, \boldsymbol{h}^{[l]})$, and the best solution is tracked. After L iterations, the best solution is returned. In practice, $L = 100$ and $(\delta_v, \delta_h) = (\frac{0.1}{m}, \frac{0.1}{n})$ yield satisfactory results, eliminating the need for reinitialization or nested search-space refinements.

4.2 Fast Camera Selection

Camera selection is performed to assign a subset of views $\mathcal{C}^{(b)}$ to each block for fine-training. The goal is to reduce per-block fine-training cost by discarding views with negligible coverage of the corresponding block region. This ensures that each block is optimized with only the most relevant views, improving efficiency without compromising reconstruction quality.

Despite its importance, prior studies often overlook the computational burden of this process, which can account for nearly half of the overall end-to-end runtime (see Section 5.3). For instance, given M partitioned blocks and N camera views, CityGS assigns cameras by computing the SSIM between the full coarse render and each per-block render, where the latter is obtained by filtering out Gaussians outside the block boundaries. This requires rendering every view for every block, resulting in at least $(M+1) \times N$ projections, which constitutes the main computational bottleneck.

To eliminate this overhead, we introduce *fast camera selection*, which reduces the computation to only N projections. First, for each camera view, we compute the per-pixel depth D using the α-blending equation: $D = \sum_{i \in \mathcal{N}} d_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$, where $\mathcal{N}$ is the ordered set of points along the ray, d_i the depth of point i, and α_i its opacity determined by covariance and opacity. The resulting depth map is then back-projected into 3D space, forming a dense point cloud $\mathcal{P}^{(c)} = \{\boldsymbol{p}_{c,k} \mid k =$

$1, \ldots, K\}$, with $\boldsymbol{p}_{c,k} \in \mathbb{R}^3$ and K denotes the total number of points for camera c. Next, for each camera c and block b, we compute the visibility ratio of points inside the block:

$$V_{c,b} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}[\boldsymbol{p}_{c,k} \in \mathcal{B}^{(b)}], \tag{3}$$

where $\mathbb{1}$ denotes the indicator function, and $\mathcal{B}^{(b)}$ is the spatial region of block b. Finally, the assigned camera set for block b is defined as $\mathcal{C}^{(b)} = \{c \mid V_{c,b} \geq \tau\}$, where τ is a predefined threshold (with $\tau = 0.15$) to prune views with negligible block coverage. This makes the procedure substantially faster, even enabling its use as a subroutine in BO, where the back-projection is computed once and reused throughout all iterations.

4.3 Visibility Cropping and Selective Densification

Prior coarse-to-fine 3DGS pipelines load the entire coarse model during per-block fine-training. This introduces both memory and runtime overhead. The memory overhead arises from storing the entire coarse model in GPU memory, while the runtime overhead arises from the Adam optimizer rather than rendering, as frustum culling already excludes non-visible points. Since Adam maintains momentum terms, it still updates parameters of all Gaussians, including those not observed by any camera in $\mathcal{C}^{(b)}$. Similar effects have also been observed in Mallick et al. (2024).

As fine-trained models are cropped before being merged into the final model, one naive solution is to retain only Gaussians strictly within each block $\mathcal{G}^{(b)}_{\text{blk}} = \{\boldsymbol{g} \in \mathcal{G}_{\text{coarse}} \mid \boldsymbol{g} \in \mathcal{B}^{(b)}\}$. However, this leads to degraded results due to over-pruning of Gaussians that lie outside block boundaries that remain visible in some views. To address this, we introduce *visibility cropping* that retain the visible Gaussians $\mathcal{G}^{(b)}_{\text{vis}} = \{\boldsymbol{g} \in \mathcal{G}_{\text{coarse}} \mid \boldsymbol{g}$ visible from some $c \in \mathcal{C}^{(b)}\}$ for each block prior to fine-training. This visibility-based filtering substantially reduces the number of Gaussians involved in optimization. In addition, since $G^{(b)}_{\text{vis}} = |\mathcal{G}^{(b)}_{\text{vis}}|$ must be recomputed at every BO iteration, we implement its evaluation entirely in NVIDIA Warp, achieving near-native CUDA performance and significantly reducing partition time. More implementation details are presented in Appendix A.1.

While *visibility cropping* preserves all visible Gaussians necessary for fine-training, it also includes those outside the block, i.e., $\mathcal{G}^{(b)}_{\text{vis}} \setminus \mathcal{G}^{(b)}_{\text{blk}}$, which are ultimately discarded prior to merging. Although retaining these Gaussians is essential to prevent quality degradation, they need not participate in densification. Motivated by this observation, we introduce *selective densification*, which restricts densification to Gaussians strictly within the block. This approach reduces the number of new Gaussians created during training, thereby lowering memory consumption and improving optimization efficiency, while maintaining per-block fidelity.

5 Experiments

5.1 Experimental Setup

Datasets. We conducted experiments on five large-scale scenes, including four real-world datasets and one synthetic dataset. For the real-world datasets, we used *Building* and *Rubble* from Mill19 (Turki et al., 2022b), and *Residence* and *Sci-Art* from UrbanScene3D (Lin et al., 2022). For the synthetic dataset, we adopted *Aerial*, which represents a small city region from MatrixCity (Li et al., 2023). Following prior work (Liu et al., 2025), all images in MatrixCity were resized to a width of 1600 pixels. For a fair comparison on real-world datasets, we downsampled all images by a factor of four, consistent with previous methods.

Baselines. We compare our framework against state-of-the-art large-scale 3DGS methods, including CityGS (Liu et al., 2025), VastGS (Lin et al., 2024), and DOGS (Chen & Lee, 2024). We also include 3DGS†, which follows the original 3DGS pipeline but extends training to 60k iterations, sets the densification interval to 200 iterations, and applies densification until 30k iterations. For VastGS and DOGS, we directly adopt the metrics reported in DOGS paper, where VastGS was evaluated without appearance modeling. For runtime analysis, we use an unofficial implementation of VastGS

Methods	MatrixCity-Aerial			Mill19			UrbanScene3D		
	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)
3DGS†	23.67	0.735	0.384	22.97	0.749	0.291	21.25	0.814	0.239
CityGS	27.46	0.865	0.204	23.66	0.796	**0.237**	**21.70**	0.825	0.221
Ours	**27.74**	**0.875**	**0.186**	**23.87**	**0.797**	0.240	21.33	**0.826**	**0.213**

Table 1: **Quantitative comparison.** Results on MatrixCity-Aerial, Mill19 (average of Rubble and Building), and UrbanScene3D (average of Residence and Sci-Art).

Methods	MatrixCity-Aerial			Mill19			UrbanScene3D		
	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)
VastGS†	28.33	0.835	0.220	23.50	0.735	0.245	21.83	0.730	0.261
DOGS	28.58	0.847	0.219	24.26	0.762	**0.231**	23.18	0.772	0.232
Ours	**28.91**	**0.879**	**0.187**	**24.68**	**0.795**	0.241	**23.55**	**0.832**	**0.213**

Table 2: **Quantitative comparison with color-corrected metrics (denoted by the "C-" prefix).** Results on MatrixCity-Aerial, Mill19 (average of Rubble and Building), and UrbanScene3D (average of Residence and Sci-Art).

(also without appearance modeling) to enable a fairer comparison of training efficiency. For consistency, we denote both variants as VastGS† throughout our experiments. We do not report runtime results for DOGS, as its distributed training setup involves interconnect communication overhead, which is not directly comparable to our parallel but independent runtime setting.

Metrics. We evaluate reconstruction quality using PSNR, SSIM, and LPIPS. Since some prior works, such as DOGS and VastGS, apply color correction before computing these metrics, we also adopt the color-corrected versions to ensure fair comparison. In contrast, when comparing against 3DGS and CityGS, which do not apply color correction, we report the standard PSNR, SSIM, and LPIPS values.

Efficiency metrics & runtime protocol. We use T_{coarse}, $T_{\text{partition}}$, $\max T_{\text{fine}}$, and T_{E2E} (as defined in Equation 1) as our efficiency metrics. For all runtime analysis presented in this paper, we adopt the same block configurations as CityGS: 36 blocks for *MatrixCity-Aerial*, 20 for *Building*, 20 for *Residence*, 9 for *Rubble*, and 9 for *Sci-Art*. All runtimes are measured on identical compute hardware, with detailed specifications provided in Appendix A.1.

5.2 Quantitative Results

From Table 1 and Table 2, our method achieves competitive or superior reconstruction quality across datasets. Compared to CityGS, performance is largely on par, with modest gains ($\approx 1.0-1.02\times$) in PSNR/SSIM where applicable and consistently better LPIPS, at the cost of a slight PSNR drop on one dataset in exchange for improved perceptual quality. Compared to 3DGS†, we observe consistent improvements, typically $\approx 1.05-1.2\times$ higher PSNR/SSIM and up to $\sim 2\times$ lower LPIPS. With color-corrected metrics, our method also surpasses VastGS† and DOGS on most datasets, leading in C-PSNR and C-SSIM. Overall, these results demonstrate parity with CityGS while clearly outperforming VastGS†, DOGS, and 3DGS†. Additional quantitative results are provided in Appendix A.4.

5.3 Load Balance and Runtime Analysis

As shown in Table 3, our method consistently achieves the lowest coarse-stage runtime and slowest-block fine-stage runtime across all datasets, yields the best partition time on two of three datasets, and achieves the best end-to-end runtime on MatrixCity-Aerial and UrbanScene3D; on Mill19, Notably, although our T_{E2E} on Mill19 is slightly longer than the reported VastGS† runtime (which omits T_{coarse}), our method delivers *higher reconstruction quality*—surpassing VastGS† on PSNR, SSIM, and LPIPS (see Table 2)—highlighting a favorable quality–latency trade-off.

Methods	MatrixCity-Aerial				Mill19				UrbanScene3D			
	T_{coarse}	$T_{\text{partition}}$	$\max T_{\text{fine}}$	T_{E2E}	T_{coarse}	$T_{\text{partition}}$	$\max T_{\text{fine}}$	T_{E2E}	T_{coarse}	$T_{\text{partition}}$	$\max T_{\text{fine}}$	T_{E2E}
3DGS†	01:50	–	–	01:50	01:20	–	–	01:20	01:01	–	–	01:01
VastGS†	–	00:48	01:13	02:01	–	**00:05**	00:42	**00:47**	–	00:17	00:40	00:57
CityGS	00:52	01:39	01:00	03:31	01:03	00:15	01:10	02:28	00:43	00:20	01:04	02:07
Ours	**00:38**	**00:16**	**00:30**	**01:24**	**00:24**	00:07	**00:36**	01:07	**00:21**	**00:07**	**00:28**	**00:55**

Table 3: **End-to-end runtime comparison.** A value of "–" indicates that the method does not include the corresponding stage.

FCS	LB-SP	VC	SD	MatrixCity-Aerial		Residence		Building	
				Max T_{fine}	$T_{\text{partition}}$	Max T_{fine}	$T_{\text{partition}}$	Max T_{fine}	$T_{\text{partition}}$
✓				01:00	00:14	01:01	00:04	01:06	00:03
✓		✓		00:52	00:14	00:47	00:04	00:45	00:03
✓	✓	✓		00:47	00:16	00:36	00:07	00:34	00:08
✓		✓	✓	00:32	00:14	00:33	00:04	00:39	00:03
✓	✓	✓	✓	**00:30**	00:16	**00:30**	00:07	**00:30**	00:08

Table 4: **Ablation on model components.** Evaluate the effectiveness of individual components: Fast Camera Selection (FCS), Load Balance-aware Scene Partition (LB-SP), Visibility Cropping (VC), and Selective Densification (SD).

5.4 Ablation Studies

To assess the contribution of each component in our framework, we conduct ablation experiments on three representative datasets: MatrixCity-Aerial, Residence, and Building. We evaluate different combinations of four components: (1) **Fast Camera Selection (FCS)**, which accelerates camera-to-block assignment with negligible accuracy loss; (2) **Load Balance-aware Scene Partition (LB-SP)**, which redistributes Gaussians across blocks based on proxy load metrics to mitigate imbalance; (3) **Visibility Cropping (VC)**, which prunes invisible Gaussians to reduce optimization time; and (4) **Selective Densification (SD)**, which restricts densification to block regions. As shown in Table 4, LB-SP consistently reduces the worst-block fine-stage runtime $\max T_{\text{fine}}$: configurations with LB-SP always outperform otherwise identical ones without it. Moreover, enabling all four components halves the worst-block fine-stage runtime compared to the FCS-only baseline ($\sim$01:00 $\rightarrow$ $\sim$00:30), corresponding to a $\sim 2\times$ speedup in $\max T_{\text{fine}}$ and substantially improved end-to-end efficiency. These results highlight that LB-SP's workload rebalancing complements the per-block reductions of VC and SD, yielding the largest cumulative runtime gains when combined.

6 Conclusion

In this paper, we present LoBE-GS, which addresses load balancing and efficiency in the parallel training of 3DGS models. At the core of LoBE-GS is a computational-load proxy that enables an optimization for the scene partition of a coarse 3DGS model. We further introduce fast camera selection to accelerate the scene partitioning, as well as visibility cropping and selective densification to reduce loading in each block. LoBE-GS achieves up to $2\times$ training speedup over existing methods using coarse models for large-scale scene reconstruction while preserving the quality of the 3DGS models. In future work, we plan to experiment with larger and more complex scenes that would benefit from partitioning into a greater number of blocks for fine-training, and to explore the integration of level-of-detail (LoD) and 2DGS representations. We also plan to evaluate the framework on more diverse datasets, including those with sparse camera views in specific regions, and to investigate alternative partitioning strategies beyond the current grid-based approach.

References

Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020. URL http://arxiv.org/abs/1910.06403.

Yu Chen and Gim Hee Lee. DOGS: Distributed-oriented gaussian splatting for large-scale 3d reconstruction via gaussian consensus. *Advances in Neural Information Processing Systems*, 37: 34487–34512, 2024.

Jixuan Fan, Wanhua Li, Yifei Han, and Yansong Tang. Momentum-GS: Momentum gaussian self-distillation for high-quality large scene reconstruction. *arXiv preprint arXiv:2412.04887*, 2024.

Yuanyuan Gao, Hao Li, Jiaqi Chen, Zhengyu Zou, Zhihang Zhong, Dingwen Zhang, Xiao Sun, and Junwei Han. CityGS-X: A scalable architecture for efficient and geometrically accurate large-scale scene reconstruction, 2025. URL https://arxiv.org/abs/2503.23044.

Lihan Jiang, Kerui Ren, Mulin Yu, Linning Xu, Junting Dong, Tao Lu, Feng Zhao, Dahua Lin, and Bo Dai. Horizon-GS: Unified 3d gaussian splatting for large-scale aerial-to-ground scenes. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 26789–26799, 2025.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (SIGGRAPH)*, 42(4): 1–14, 2023.

Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics*, 43(4), July 2024. URL https://repo-sam.inria.fr/fungraph/hierarchical-3d-gaussians/.

Ruilong Li, Sanja Fidler, Angjoo Kanazawa, and Francis Williams. NeRF-XL: Scaling nerfs with multiple GPUs. In *European Conference on Computer Vision (ECCV)*, 2024.

Yixuan Li, Lihan Jiang, Linning Xu, Yuanbo Xiangli, Zhenzhi Wang, Dahua Lin, and Bo Dai. MatrixCity: A large-scale city dataset for city-scale neural rendering and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3205–3215, 2023.

Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, and Wenming Yang. VastGaussian: Vast 3d gaussians for large scene reconstruction. In *CVPR*, 2024.

Liqiang Lin, Yilin Liu, Yue Hu, Xingguang Yan, Ke Xie, and Hui Huang. Capturing, reconstructing, and simulating: the urbanscene3d dataset. In *ECCV*, 2022.

Yang Liu, Chuanchen Luo, Zhongkai Mao, Junran Peng, and Zhaoxiang Zhang. CityGaussianV2: Efficient and geometrically accurate reconstruction for large-scale scenes. *arXiv preprint arXiv:2411.00771*, 2024.

Yang Liu, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. CityGaussian: Real-time high-quality large-scale scene rendering with gaussians. In *European Conference on Computer Vision*, pp. 265–282. Springer, 2025.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-GS: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20654–20664, 2024.

Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3DGS: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400711312. doi: 10.1145/3680528.3687694. URL https://doi.org/10.1145/3680528.3687694.

Zhenxing Mi and Dan Xu. Switch-NeRF: Learning scene decomposition with mixture of experts for large-scale neural radiance fields. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=PQ2zoIZqvm.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, pp. 405–421. Springer, 2020.

KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. CompGS: Smaller and faster gaussian splatting with vector quantization. *ECCV*, 2024.

Miles Olson, Elizabeth Santorella, Louis C. Tiao, Sait Cakmak, David Eriksson, Mia Garrard, Sam Daulton, Maximilian Balandat, Eytan Bakshy, Elena Kashtelyan, Zhiyuan Jerry Lin, Sebastian Ament, Bernard Beckerman, Eric Onofrey, Paschal Igusti, Cristian Lara, Benjamin Letham, Cesar Cardoso, Shiyun Sunny Shen, Andy Chenyuan Lin, and Matthew Grange. Ax: A Platform for Adaptive Experimentation. In *AutoML 2025 ABCD Track*, 2025.

Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1), May 2024. URL https://repo-sam.inria.fr/fungraph/reduced_3dgs/.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-GS: Towards consistent real-time rendering with lod-structured 3d gaussians. *arXiv preprint arXiv:2403.17898*, 2024.

Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*, pp. 835–846. 2006.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8248–8258, 2022.

Jiadong Tang, Yu Gao, Dianyi Yang, Liqi Yan, Yufeng Yue, and Yi Yang. Dronesplat: 3d gaussian splatting for robust 3d reconstruction from in-the-wild drone imagery. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 833–843, 2025.

Haithem Turki, Deva Ramanan, and Mahadev 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)*, pp. 12922–12931, June 2022a.

Haithem Turki, Deva Ramanan, and Mahadev 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)*, pp. 12922–12931, June 2022b.

Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for gaussian splatting. *Journal of Machine Learning Research*, 26(34):1–17, 2025.

Zhensheng Yuan, Haozhi Huang, Zhen Xiong, Di Wang, and Guanghua Yang. Robust and efficient 3d gaussian splatting for urban scene reconstruction. *arXiv preprint arXiv:2507.23006*, 2025.

Hexu Zhao, Haoyang Weng, Daohan Lu, Ang Li, Jinyang Li, Aurojit Panda, and Saining Xie. On scaling up 3d gaussian splatting training, 2024. URL https://arxiv.org/abs/2406.18533.

A APPENDIX

A.1 IMPLEMENTATION DETAILS

Scene Partition. Bayesian Optimization (BO) with Gaussian Process (GP) surrogate modeling (Section 4.1) is implemented using Ax (Olson et al., 2025) with BoTorch (Balandat et al., 2020) backend for GPU-accelerated optimization. Block load computation (Section 4.1), *fast camera selection* (Section 4.2), and *visibility cropping* (Section 4.3), are implemented in NVIDIA Warp (Macklin, 2022), which enables kernel-based programming in Python with performance comparable to native CUDA. In preliminary benchmark on the *MatrixCity-Aerial* scene, the Warp implementation on a single GPU achieves speedups of approximately $450\times$ over sequential CPU code and $5\times$ over Numba-parallelized CPU code executed on 128 logical CPU cores. These performance improvements are enabled by low-level optimizations not exposed in PyTorch, including bitsets, atomics, and Warp tiles, with the latter providing functionality analogous to shared memory and cooperative groups in CUDA C++.

3DGS Training. The coarse-training stage employs the Sparse Adam optimizer to accelerate training, which has minimal impact on final performance. In contrast, the fine-training stage continues to use the standard Adam optimizer, as Sparse Adam was found to degrade performance in this setting. Aside from *selective densification*, fine-training details follows the standard vanilla 3DGS procedure (as in CityGS), with additional code-level optimizations through the *gsplat* library (Ye et al., 2025) and *fused-ssim* (Mallick et al., 2024) for SSIM loss evaluation.

Experimental Setup. For consistency, all CityGS runtimes reported in Section 3 are measured using a modified version of CityGS with *gsplat*, *fused-ssim*, and *visibility cropping* enabled. Moreover, since *selective densification* shortens per-block fine-training time, we disable it in LoBE-GS when reporting results in Section 3. A comparison against the unmodified CityGS with the full LoBE-GS pipeline (including *selective densification*) is provided in Figure A.1. In Section 5, since the official implementation of VastGS† is unavailable, we report performance results based on an unofficial implementation available at `https://github.com/kangpeilun/VastGaussian`.

System Configuration. All experiments are conducted on a cluster consisting of 10 compute nodes, each equipped with 8 NVIDIA L40 GPUs and 128 logical CPU cores (Intel Xeon Platinum 8362), amounting to a total of 80 GPUs across the cluster. The fine-training stage is parallelized across blocks with one GPU per block, whereas all other stages are executed on a single GPU.

Reproducibility. Source code along with a pre-built Docker image will be released upon paper acceptance to ensure reproducibility. All reported runtimes are measured within the Docker environment to eliminate potential discrepancies caused by library mismatches or system-level variations.

Declaration of LLM usage. Large Language Models (LLMs) are only used for editing grammar.

A.2 LOAD BALANCE ACROSS DATASETS

As shown in Figure A.1, our method yields a noticeably more uniform per-block workload distribution across the evaluated datasets. In particular, the load balance-aware partitioning combined with visibility cropping and selective densification systematically reduces the worst-case per-block fine-stage runtime, i.e., the slowest straggler blocks are much faster than under the baselines. This reduction in the tail of the runtime distribution leads to fewer stragglers and improved end-to-end efficiency. These gains are consistent across datasets, demonstrating the robustness of our partitioning strategy in mitigating workload skew.

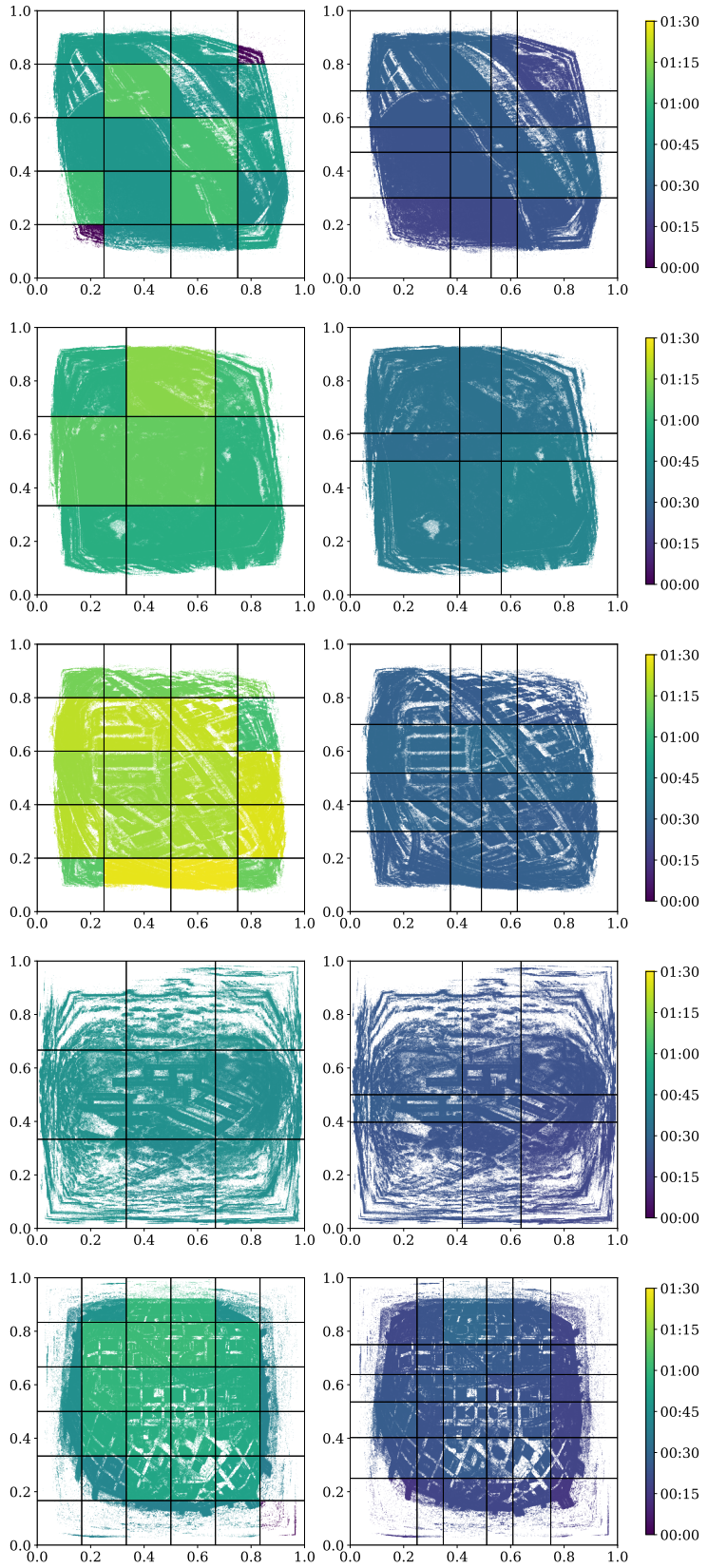

Figure A.1: Comparison of load balance and partitioning between CityGS (Left) and LoBE-GS (Right) across five datasets: *Building*, *Rubble*, *Residence*, *Sci-Art*, and *MatrixCity-Aerial*.

A.3 Additional Correlation Analysis Across Datasets

In Section 3, we observed a strong correlation with $G_{\text{vis}}^{(b)}$ when using the original CityGS pipeline combined with visibility cropping. In the fine-training stage, however, both visibility cropping and selective densification were enabled to further reduce the per-block load in LoBE-GS. To ensure that the correlation still remains strong under these settings, we additionally conducted experiments with both visibility cropping and selective densification enabled.

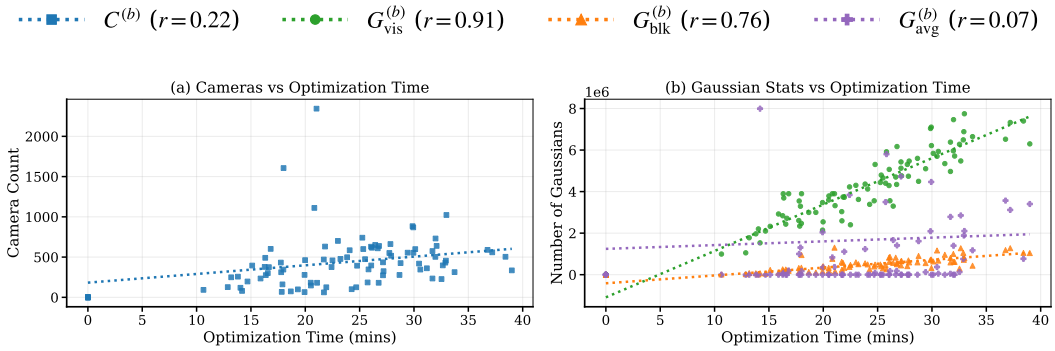

Figure A.2: **Correlation between per-block training time and block-level statistics under CityGS's partitioning with both visibility cropping and selective densification enabled.** (a) Plots camera count $C^{(b)}$. (b) Plots of Gaussian-based measures ($G_{\text{blk}}^{(b)}$, $G_{\text{vis}}^{(b)}$, $G_{\text{avg\_vis}}^{(b)}$). $G_{\text{vis}}^{(b)}$ yields the strongest and most consistent correlation across datasets even when selective densification is enabled.

A.4 Extended Quantitative Comparison

This appendix collects the full numerical results omitted from the main text for space reasons. Table A.1 and Table A.2 report per-dataset qualitative results and their color-corrected counterparts to ensure fair comparison with baselines that apply post-hoc color alignment. Table A.3 summarizes end-to-end timing T_{E2E} (T_{coarse}, $T_{\text{partition}}$, $\max T_{\text{fine}}$) and Table A.4 compares load balance aware data partition times for CPU vs GPU implementations across five datasets.

Methods	Residence			Rubble			Building			Sci-Art		
	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)
3DGS	21.44	0.791	0.236	25.47	0.777	0.277	20.46	0.720	0.305	21.05	0.837	0.242
CityGS	**22.00**	**0.813**	0.211	25.77	**0.813**	**0.228**	21.55	0.778	0.246	**21.39**	0.837	0.230
Ours	21.41	0.808	**0.206**	**25.78**	0.811	0.234	**21.96**	**0.783**	**0.245**	21.24	**0.843**	**0.219**

Table A.1: Quantitative comparison on Mill19 and UrbanScene3D datasets. We report PSNR, SSIM, and LPIPS.

Methods	Residence			Rubble			Building			Sci-Art		
	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)	C-PSNR (↑)	C-SSIM (↑)	C-LPIPS (↓)
VastGS	21.01	0.699	0.261	25.20	0.742	0.264	21.80	0.728	0.225	22.64	0.761	0.261
DoGS	21.94	0.740	0.244	25.78	0.765	0.257	22.73	0.759	**0.204**	24.42	0.804	0.219
Ours	**22.94**	**0.822**	**0.206**	**26.55**	**0.810**	**0.235**	**22.80**	**0.779**	0.247	**24.71**	**0.853**	**0.217**

Table A.2: Quantitative comparison on Mill19 and UrbanScene3D datasets. We report color-corrected (denoted by the "C-" prefix) PSNR, SSIM, and LPIPS.

Methods	Residence				Rubble				Building				Sci-Art			
	T_{coarse}	$T_{\text{partition}}$	Max T_{fine}	T_{E2E}	T_{coarse}	$T_{\text{partition}}$	Max T_{fine}	T_{E2E}	T_{coarse}	$T_{\text{partition}}$	Max T_{fine}	T_{E2E}	T_{coarse}	$T_{\text{partition}}$	Max T_{fine}	T_{E2E}
3DGS	01:22	–	–	01:22	01:10	–	–	01:10	01:30	–	–	01:30	00:40	–	–	**00:40**
VastGS	–	00:08	00:49	**00:57**	–	**00:04**	**00:39**	**00:43**	–	**00:05**	00:44	**00:49**	–	00:25	00:31	00:56
CityGS	00:43	00:31	01:22	02:36	01:06	00:09	01:14	02:29	00:59	00:21	01:06	02:26	00:42	00:08	00:45	01:35
Ours	**00:26**	**00:08**	**00:30**	01:04	**00:23**	00:05	00:41	01:09	**00:25**	00:08	**00:30**	01:03	**00:16**	**00:05**	**00:26**	00:47

Table A.3: End-to-end runtime comparison on Mill19 and UrbanScene3D dataset. For each dataset we report coarse time T_{coarse}, partition time $T_{\text{partition}}$, max fine time (Max T_{fine}), and total T_{E2E}. A value of "–" indicates that the method does not include the corresponding stage.

Methods	MatrixCity-Aerial	Residence	Rubble	Building	SciArt
LB-SP (CPU)	00:47	00:18	00:03	00:15	00:10
LB-SP (GPU)	00:16	00:06	00:05	00:06	00:05

Table A.4: Partition time (`hh:mm`) comparison across CPU and GPU methods for five datasets.