LEANSPLAT: LOW-OVERHEAD LANGUAGE GAUSSIAN SPLATTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modeling 3D language fields with Gaussian Splatting suffers from high storage and computational costs due to the large number of Gaussian primitives, high-dimensional language features, and complicated retrieval procedures. Primitives from a pretrained 3DGS scene are typically carried into the language branch, so even those that contribute little to semantic prediction require high-dimensional language features and participate in semantic rendering. Query pipelines that reconstruct high-dimensional pixel-level language features before text matching repeat this high-dimensional computation for each query. In this paper, we propose LeanSplat, a low-overhead framework for Language Gaussian Splatting. LeanSplat alternates semantic-opacity optimization with multi-view language-feature aggregation to progressively refine primitive-level semantic visibility. The resulting cross-view semantic rendering contributions guide pruning of low-contribution primitives, while semantic opacity weights the construction of shared prototypes that compress the high-dimensional language features of retained primitives. At query time, text scores are computed directly in prototype space, mapped to primitives via prototype indices, and rasterized without reconstructing the full high-dimensional language feature field. Experiments on LERF show that LeanSplat removes 83.25% of language primitives while achieving an average mIoU of 65.5%. With a prototype budget of $K = 256$, the average language storage is reduced to 2.94 MiB/scene, while query throughput exceeds 500 queries/s on an A100 GPU.

1 INTRODUCTION

3D Gaussian Splatting (3DGS) represents 3D scenes with explicit Gaussian primitives and achieves high-quality real-time novel-view rendering through differentiable rasterization and alpha compositing (Kerbl et al., 2023). These primitives can be accessed directly and can therefore carry vision-language features alongside appearance and geometric attributes. Language information from multiple views is associated with shared 3D primitives to form a scene-level language representation. This representation can be rendered from different viewpoints and used to localize objects or segment regions in response to natural-language queries, linking text descriptions to specific targets in the 3D scene.

Previous Language Gaussian Splatting methods typically build language representations upon visual primitives. These primitives are densely distributed to preserve fine-grained appearance and geometry, yet some contribute little to semantic prediction and are still carried into the language branch. This issue is further exacerbated by the high dimensionality of language embeddings. Storing such features for a dense set of primitives incurs substantial storage overhead. In addition, query pipelines that rely on high-dimensional pixel-level feature reconstruction must recover language features before matching them with text features. This reconstruction step incurs significant computational cost. Taken together, large primitive sets, high-dimensional per-primitive features, and complex semantic querying pipelines increase both storage and computational costs for Language Gaussian representations. Previous methods have explored autoencoders, quantization, codebooks, and compact indices to reduce parts of these costs (Li et al., 2025; Kim et al., 2025; Tian et al., 2025; Bang et al., 2026; Jia et al., 2026). However, compressing features alone does not determine which primitives should remain in the language branch. RGB opacity in a pretrained 3DGS scene serves appearance reconstruction and cannot directly measure a primitive's semantic contribution. If

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

low-contribution primitives are retained, they continue to occupy language-branch storage and add semantic rendering cost.

In this paper, we propose LeanSplat, a low-overhead framework for Language Gaussian Splatting. To reduce the primitive count, LeanSplat introduces semantic opacity independent of RGB opacity and alternates its optimization with multi-view language-feature aggregation. In each round, the current aggregated features are fixed while semantic opacity is optimized by aligning rendered features with 2D language features. The updated semantic opacity changes the semantic compositing weights used in the next aggregation, progressively refining primitive-level semantic visibility. Primitives with low cross-view maximum semantic rendering contribution are then pruned. To compress the high-dimensional features of the retained primitives, the features are normalized to the unit sphere, and weighted spherical K-means constructs shared semantic prototypes using semantic opacity as sample weights. At query time, linear text scores are first computed in prototype space. Primitive scores are then obtained through prototype-index lookup and rasterized. Since text projection commutes with linear alpha compositing, querying does not require reconstruction of the high-dimensional language feature field.

Experiments show that LEANSPLAT achieves an average mIoU of 65.5% while removing 83.25% of language primitives. With a prototype budget of $K = 256$, the average language storage is reduced to 2.94 MiB per scene. Direct retrieval reaches over 500 queries/s on an A100 GPU.

The main contributions are summarized as follows:

- We propose iterative semantic visibility learning. Semantic-opacity optimization alternates with multi-view language-feature re-aggregation to progressively refine primitive-level semantic visibility.
- We use the learned semantic visibility to build a compact language representation. Cross-view semantic rendering contributions guide pruning of low-contribution primitives, while semantic opacity weights the construction of shared prototypes. Together, these steps reduce the number of language primitives and per-primitive feature storage.
- We develop a semantic rendering scheme that queries in prototype space without reconstructing the full feature field, thus lowering the overhead of semantic querying.

2 RELATED WORK

2.1 3D LANGUAGE FIELDS

Open-vocabulary 3D scene understanding grounds text queries in reconstructed scenes. NeRF provides a radiance-field representation built from posed images (Mildenhall et al., 2021). LERF learns multi-scale CLIP features for localization (Kerr et al., 2023), while OpenNeRF uses pixel-level features and novel-view rendering for open-set segmentation (Engelmann et al., 2024). Both use posed images to build a view-consistent field. A query matches text with the learned features to obtain spatial scores for selection.

3D Gaussian Splatting (3DGS) represents scenes with explicit primitives and efficient rasterization (Kerbl et al., 2023). Language features attached to Gaussians can be queried through rendered pixel features or stored 3D features. The primitive count and feature width directly affect storage and rendering. The two query paths also incur different computation.

2.2 LANGUAGE GAUSSIAN SPLATTING

Language Gaussian methods either optimize primitive features or aggregate multi-view evidence. Feature-3DGS (Zhou et al., 2024) extends Gaussian rendering to semantic features, while LEGaussians (Shi et al., 2024) learns language-aligned Gaussian attributes. 3D VL-GS (Peng et al., 2025) uses cross-modal rasterization to balance visual and semantic representations. LangSplat (Qin et al., 2024) compresses CLIP features into a scene-specific latent space. This avoids storing a full CLIP vector on every primitive. It uses SAM masks to supervise semantics at object and part scales. LaGa (Cen et al., 2025) addresses view-dependent semantics. OpenGaussian (Wu et al., 2024) learns instance-consistent features for point-level tasks, while GOI (Qu et al., 2024) selects relevant Gaussians with a semantic-space hyperplane.

Aggregation-based methods focus on which 2D observations inform each Gaussian. This matters when occlusion or noisy masks assign conflicting evidence to the same primitive. Occam's LGS (Cheng et al., 2025) aggregates features with rendering weights and filters noisy Gaussians. VALA (Wang et al., 2026) measures ray contributions before visibility-aware gating and robust aggregation. THGS (Dai et al., 2025) uses superpoint structure for hierarchical semantics, while ReLaGS (Xie et al., 2026) adds relational reasoning. CCL-LGS (Tian et al., 2025) aligns masks across views and learns a contrastive codebook to reduce semantic inconsistency.

RGB 3DGS work reduces storage through clustering, quantization, and pruning (Niedermayr et al., 2024; Navaneet et al., 2024; Fan et al., 2024; Lee et al., 2024), and accelerates rendering (Hanson et al., 2025; Hou et al., 2025). These methods target appearance and geometry. Appearance-driven primitive selection does not directly measure which Gaussians provide useful language evidence.

Language-field efficiency also depends on feature storage and querying. Building on feature quantization (Jégou et al., 2011; van den Oord et al., 2017), LangSplatV2 (Li et al., 2025) uses sparse coefficients and a shared codebook. It rasterizes coefficients before reconstructing high-dimensional pixel features. This separates the number of rasterized channels from the width of the queried feature map. Dr. Splat (Kim et al., 2025) assigns product-quantized embeddings to selected Gaussians for direct 3D queries. LightSplat (Bang et al., 2026) stores compact semantic indices with an index-to-feature mapping. CoSAG (Jia et al., 2026) compresses spatial anchor assignments, while CF3 (Lee et al., 2025) combines autoencoding with adaptive Gaussian pruning and merging. Some of these methods already reduce the primitive set. For rendering-based queries, high-dimensional pixel reconstruction can still add computation after feature storage is compressed. We therefore consider primitive count, per-primitive storage, and query-time feature processing together.

3 Preliminaries and Problem Formulation

Consider a trained RGB 3DGS scene containing N Gaussian primitives. Throughout this paper, i, v, and p denote the indices of primitives, views, and pixels, respectively. Let α_i^{rgb} denote the RGB opacity, $G_{iv}(p)$ the projected 2D Gaussian response, and $T_{iv}^{\mathrm{rgb}}(p)$ the accumulated transmittance along the pixel ray before reaching primitive i. The standard RGB compositing weight is defined as $w_{iv}^{\mathrm{rgb}}(p) = T_{iv}^{\mathrm{rgb}}(p)\alpha_i^{\mathrm{rgb}}G_{iv}(p)$.

A conventional Language Gaussian representation typically associates each primitive with a D-dimensional language feature $\mathbf{f}_i \in \mathbb{R}^D$. Using the RGB compositing weights, the pixel-level language feature is given by

$$\mathbf{F}_v(p) = \sum_i w_{iv}^{\mathrm{rgb}}(p)\,\mathbf{f}_i. \tag{1}$$

This representation incurs three main sources of overhead. First, the language branch typically inherits the large primitive set of the RGB 3DGS and must therefore maintain and process N primitive-level semantic representations. Second, storing a D-dimensional language feature for each primitive requires ND feature values. If each value occupies b bytes, this amounts to bD bytes per primitive and bND bytes per scene. With L semantic levels, the corresponding storage rises to bLD and $bNLD$, respectively. Third, open-vocabulary querying typically reconstructs dense language features for L semantic levels at resolution $H \times W$, with D channels per level. This requires processing $LHWD$ feature values before matching pixel features with text embeddings. The query must therefore perform both high-dimensional feature reconstruction and subsequent text matching, with costs growing with the number of semantic levels, image resolution, and feature dimensionality.

Consequently, the overhead of Language Gaussian Splatting stems primarily from large primitive sets, high-dimensional per-primitive language features, and high-dimensional semantic querying. We seek to reduce the representation, storage, and query costs of the language branch while keeping the geometry and appearance representation of the pretrained RGB 3DGS unchanged and preserving open-vocabulary semantic prediction quality.

4 METHOD

4.1 OVERALL FRAMEWORK

Figure 1 shows the LEANSPLAT pipeline. The pretrained RGB 3DGS remains frozen. First, iterative semantic visibility refinement begins with multi-view feature aggregation. The initial aggregation uses RGB compositing weights. Semantic-opacity optimization then alternates with language-feature re-aggregation. Second, semantic visibility guides the construction of a compact representation. Cross-view semantic contributions determine which primitives to prune, while shared semantic prototypes encode the language features of retained primitives. Finally, decode-free prototype-space querying computes text scores in prototype space. Prototype indices map these scores to primitives for rasterization without reconstructing the high-dimensional language feature field.

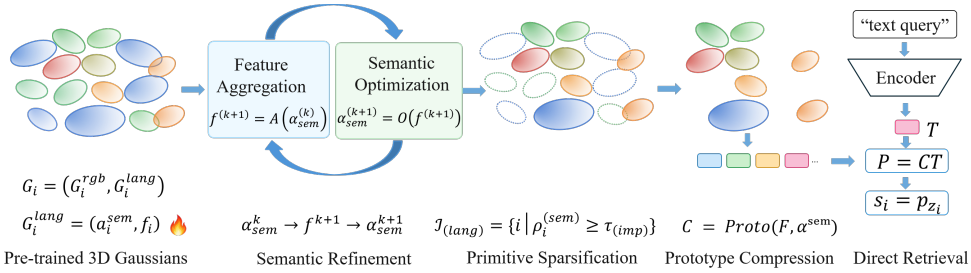

Figure 1: Overview of LEANSPLAT. The RGB branch remains frozen. Semantic-opacity optimization alternates with multi-view feature re-aggregation to progressively refine semantic visibility in the language branch. The learned semantic visibility guides language primitive sparsification and shared semantic prototype construction. The compact representation supports decode-free prototype-space querying.

4.2 ITERATIVE SEMANTIC VISIBILITY REFINEMENT

The language branch introduces semantic opacity independently of RGB opacity α_i^{rgb}. It is defined as $\alpha_i^{\text{sem}} = \sigma(\ell_i^{\text{sem}})$, where ℓ_i^{sem} is a learnable parameter. Semantic rendering uses the same front-to-back alpha compositing as Eq. 1. Replacing α_i^{rgb} with α_i^{sem} yields the semantic compositing weight $\omega_{iv}^{\text{sem}}(p)$. The rendered language feature is

$$\widehat{\mathbf{F}}_v(p) = \sum_i \omega_{iv}^{\text{sem}}(p)\, \mathbf{f}_i. \tag{2}$$

Multi-view feature aggregation. Let $\mathbf{F}_v^{\text{2D}}(p)$ denote the 2D language feature and $\omega_{iv}^{(r)}(p)$ the compositing weight for aggregation round r. The multi-view 2D features are aggregated onto each primitive using these weights:

$$\mathbf{f}_i^{(r)} = \frac{\sum_{v,p} \omega_{iv}^{(r)}(p)\, \mathbf{F}_v^{\text{2D}}(p)}{\sum_{v,p} \omega_{iv}^{(r)}(p) + \epsilon}. \tag{3}$$

Here, ϵ is a numerical stabilizer. The initial aggregation computes the weights from RGB opacity and yields $\mathbf{f}^{(0)}$. Later rounds use the semantic opacity learned in the preceding stage.

Semantic opacity optimization. At the start of each optimization stage, the parameter ℓ_i^{sem} is reset to 0. This gives $\alpha_i^{\text{sem}} = 0.5$. The currently aggregated primitive features $\mathbf{f}^{(r)}$ are then held fixed, and only semantic opacity is optimized. The language features rendered with Eq. 2 are aligned with the corresponding 2D features using the following objective:

$$\mathcal{L}_{\text{vis}}^{(r)} = 1 - \cos\left(\widehat{\mathbf{F}}^{(r)}, \mathbf{F}^{\text{2D}}\right) + \lambda_2 \left\|\widehat{\mathbf{F}}^{(r)} - \mathbf{F}^{\text{2D}}\right\|_2^2 + \lambda_{\text{ent}}(t) H\left(\alpha^{\text{sem}}\right). \tag{4}$$

Here, $H(\cdot)$ denotes Bernoulli entropy and drives semantic opacity toward 0 or 1. Its weight $\lambda_{\text{ent}}(t)$ changes with optimization iteration t.

Iterative semantic update. Semantic opacity is optimized over multiple stages. It changes the compositing weights used for multi-view feature aggregation. The resulting primitive features are then used for semantic-opacity optimization in the next stage.

$$\mathbf{f}^{(0)} = \mathcal{A}\left(\alpha^{\mathrm{rgb}}\right), \qquad \alpha^{\mathrm{sem},(r+1)} = \mathcal{O}\left(\mathbf{f}^{(r)}\right), \qquad \mathbf{f}^{(r+1)} = \mathcal{A}\left(\alpha^{\mathrm{sem},(r+1)}\right). \tag{5}$$

Here, $\mathcal{A}$ aggregates multi-view language features according to Eq. 3, yielding the current primitive features $\mathbf{f}^{(r)}$. With $\mathbf{f}^{(r)}$ fixed, $\mathcal{O}$ optimizes Eq. 4 to obtain $\alpha^{\mathrm{sem},(r+1)}$. Then $\mathcal{A}$ uses $\alpha^{\mathrm{sem},(r+1)}$ to re-aggregate the features, yielding $\mathbf{f}^{(r+1)}$. This yields the iterative loop $\cdots \to \mathbf{f}^{(r)} \to \alpha^{\mathrm{sem},(r+1)} \to \mathbf{f}^{(r+1)} \to \cdots$. The updated semantic opacity is used for the next multi-view feature aggregation. The resulting primitive language features are used for the next round of semantic-opacity optimization. From the second round onward, updates from the previous round affect the current optimization in Eq. 4 through $\mathbf{f}^{(r)}$, even though semantic opacity is re-initialized.

4.3 Semantic-Visibility-Guided Compact Representation

The learned semantic opacity guides both pruning and prototype construction. Pruning uses the cross-view maximum semantic compositing weight computed from it to remove low-contribution primitives. For retained primitives, semantic opacity directly serves as the clustering weight. Shared prototypes then compress their high-dimensional language features.

Semantic sparsification. The entropy regularization encourages semantic opacity toward binary values, reducing ambiguous intermediate opacities and sharpening the semantic contributions used for subsequent sparsification. After semantic-opacity optimization, the maximum semantic rendering contribution of each primitive over all valid views and pixels is taken as its importance:

$$\rho_i^{\mathrm{sem}} = \max_{v,p} \omega_{iv}^{\mathrm{sem}}(p), \qquad \mathcal{I}_{\mathrm{lang}} = \{i \mid \rho_i^{\mathrm{sem}} \geq \tau_{\mathrm{imp}}\}. \tag{6}$$

Here, ρ_i^{sem} is the maximum semantic compositing weight across views and pixels, rather than the semantic opacity α_i^{sem} itself. τ_{imp} is the sparsification threshold, and $\mathcal{I}_{\mathrm{lang}}$ is the retained language primitive set. Low-contribution primitives are removed only from the language branch. The RGB scene remains unchanged.

Visibility-guided prototype construction. On the sparse language primitive set $\mathcal{I}_{\mathrm{lang}}$, the per-primitive high-dimensional language features are further compressed. The retained language features are first normalized onto the unit sphere, and the semantic opacity is used as the sample weight, so that weighted spherical K-means builds the shared semantic prototypes:

$$\min_{\mathbf{C},\{z_i\}} \sum_{i \in \mathcal{I}_{\mathrm{lang}}} \alpha_i^{\mathrm{sem}} \left\|\mathbf{u}_i - \mathbf{c}_{z_i}\right\|_2^2. \tag{7}$$

Here, $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_K]^\top \in \mathbb{R}^{K \times D}$ is the prototype codebook, with K prototypes. $\mathbf{u}_i$ is the normalized language feature of primitive i, z_i is its prototype index, and $\mathbf{c}_{z_i}$ is the shared prototype selected by that index. All prototypes are constrained to the unit sphere. As a clustering weight, semantic opacity gives primitives with higher opacity more influence on the prototype centers. After quantization, each retained primitive uses its prototype index z_i to reference the shared codebook $\mathbf{C}$, avoiding storage of its full D-dimensional language feature. Semantic opacity is retained for subsequent semantic compositing.

Storage Cost. Our method stores L prototype indices and one semantic opacity per retained primitive, together with one $K \times D$ codebook for each language level. Let N_{lang} denote the number of retained primitives and b_{idx} the bytes per index. With semantic opacities and codebook values stored in float32, the language storage is

$$S_{\mathrm{lang}} = N_{\mathrm{lang}}(Lb_{\mathrm{idx}} + 4) + 4LKD \quad \text{bytes}. \tag{8}$$

This replaces dense per-primitive feature storage with indices and shared codebooks.

4.4 Decode-Free Prototype-Space Querying

Let a query contain J normalized text embeddings, arranged as columns of $T = [\mathbf{t}_1, \ldots, \mathbf{t}_J] \in \mathbb{R}^{D\times J}$. The J texts can be queried together for the same view, sharing geometric projection, depth sorting, and semantic compositing weights. Because the semantic compositing weights do not depend on the text, text projection commutes with linear alpha compositing:

$$\left(\sum_{i\in\mathcal{I}_{\text{lang}}} \omega_{iv}^{\text{sem}}(p)\mathbf{c}_{z_i}\right)^{\top} T = \sum_{i\in\mathcal{I}_{\text{lang}}} \omega_{iv}^{\text{sem}}(p)\mathbf{c}_{z_i}^{\top} T. \tag{9}$$

Prototype and primitive-level text scores. First compute the scores of all prototypes against the query texts as $\mathbf{P} = \mathbf{C}T \in \mathbb{R}^{K\times J}$. Let $\mathbf{p}_k$ denote the k-th row of $\mathbf{P}$. Primitive scores are then retrieved by prototype index:

$$\mathbf{s}_i = \mathbf{p}_{z_i} = [s_{i,1}, \ldots, s_{i,J}]. \tag{10}$$

Pixel-level text scores. Existing Language Gaussian methods typically render or reconstruct D-dimensional pixel language features before text matching. We instead directly composite the J text scores of each primitive using semantic alpha weights:

$$\mathbf{S}_v(p) = \sum_{i\in\mathcal{I}_{\text{lang}}} \omega_{iv}^{\text{sem}}(p)\,\mathbf{s}_i = [a_{1,v}(p), \ldots, a_{J,v}(p)]\,. \tag{11}$$

Thus, the language branch rasterizes only J score channels without constructing a high-dimensional pixel language feature map.

5 Experiments

5.1 Datasets and Details

Datasets. Experiments evaluate open-vocabulary segmentation on LERF (Kerr et al., 2023) and 3D semantic segmentation on ScanNet (Dai et al., 2017), using mIoU on both datasets. Following OpenGaussian (Wu et al., 2024), mAcc is also reported for ScanNet settings with 19, 15, and 10 classes. Waldo refers to the Waldo-kitchen scene in LERF.

Implementation Details. All experiments start from an RGB 3DGS pretrained for 30K iterations, with geometry and appearance frozen. For both LERF and ScanNet, 2D language features are extracted following the CLIP-SAM protocol of LangSplat (Qin et al., 2024). LERF uses three language-feature levels, whereas ScanNet uses one. In the multi-level setting, semantic opacity and the retained primitive set are shared across levels. Semantic opacity is optimized for 9K iterations. All optimization-strategy ablations use this budget. The stage-wise schedule consists of three 3000-iteration stages. Primitive importance is computed using Eq. 6, followed by pruning at $\tau_{\text{imp}} = 0.1$. Each language level builds a separate codebook with $K = 256$ prototypes. LERF queries follow the relevancy protocol of LERF (Kerr et al., 2023) and LangSplat (Qin et al., 2024). Query efficiency is evaluated on LERF-OVS. Further implementation and evaluation details are given in Appendix A.1, and timing details in Appendix A.7.

5.2 Semantic Segmentation Results

Tables 1 and 2 report semantic segmentation on LERF and ScanNet. Figure 2 shows the qualitative comparison on LERF. Baseline values are transcribed from the original papers.

On LERF, LeanSplat achieves a mean mIoU close to the best result. On ScanNet, LeanSplat achieves the highest mIoU across all class settings and the highest mAcc in most settings. These results show that LeanSplat maintains competitive semantic prediction quality while reducing language-branch overhead.

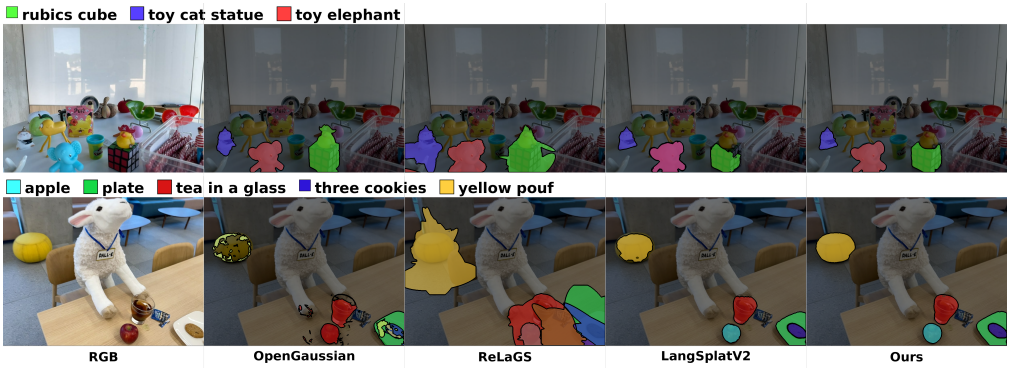

Figure 2: Qualitative comparison of open-vocabulary segmentation on LERF. In the queries shown, LEANSPLAT produces masks more focused on the target objects, with less background spillover and fewer scattered responses than some baselines.

Table 1: Open-vocabulary segmentation on LERF (mIoU, %). T-F denotes training-free methods.

Method	Venue	T-F	Figurines	Ramen	Teatime	Waldo	Mean
LERF	ICCV'23	×	38.6	28.2	45.0	37.9	37.4
LangSplat	CVPR'24	×	44.7	51.2	65.1	44.5	51.4
LangSplatV2	NeurIPS'25	×	56.4	51.8	72.2	59.1	59.9
3D VL-GS	ICLR'25	×	58.1	61.4	73.5	54.8	62.0
LaGa	ICML'25	×	64.1	55.6	70.9	65.6	64.0
CCL-LGS	ICCV'25	×	61.2	**62.3**	71.8	**67.1**	**65.6**
Occam's LGS	BMVC'25	✓	58.6	51.0	70.2	65.3	61.3
THGS	ACM MM'25	✓	57.3	43.5	68.3	50.7	54.9
ReLaGS	CVPR'26	✓	64.7	51.2	**81.0**	60.6	64.4
LEANSPLAT	—	×	**65.3**	52.0	77.5	**67.1**	65.5

Table 2: 3D semantic segmentation on ScanNet (%). ScanNet provides point-level annotations, so two LeanSplat results are reported. The unpruned row evaluates all points, while the pruned row evaluates only retained points.

Method	Venue	T-F	19 classes		15 classes		10 classes	
			mIoU	mAcc	mIoU	mAcc	mIoU	mAcc
LangSplat	CVPR'24	×	3.78	9.11	5.35	13.20	8.40	22.06
OpenGaussian	NeurIPS'24	×	24.73	41.54	30.13	48.25	38.29	55.19
LaGa	ICML'25	×	32.50	49.10	35.50	53.50	42.60	63.20
Occam's LGS	BMVC'25	✓	31.93	48.93	34.25	53.71	45.16	64.39
THGS	ACM MM'25	✓	34.39	50.74	39.61	57.07	46.38	64.74
ReLaGS	CVPR'26	✓	32.35	44.52	40.04	**60.59**	47.17	66.08
LEANSPLAT (pruned)	—	×	36.72	53.67	**40.97**	60.00	**50.14**	**69.83**
LEANSPLAT (unpruned)	—	×	**36.85**	**54.56**	40.28	59.76	48.74	69.01

5.3 ABLATION STUDY

We analyze the contribution of each component on LERF and ScanNet.

Module-combination ablation. Table 3 shows that iterative semantic-visibility learning improves the mean mIoU on LERF and the mIoU under every ScanNet class setting before pruning. On LERF, semantic pruning and shared prototypes further improve the mean mIoU. On ScanNet, shared prototypes also improve the mIoU under every class setting. Direct prototype-space retrieval matches the predictions obtained through high-dimensional feature reconstruction while avoiding feature recovery during querying.

Table 3: Ablation of language-branch components on LERF and ScanNet. All values are mIoU (%). For ScanNet, rows without pruning are evaluated on all points, whereas rows with pruning are evaluated on the retained points.

Components				LERF					ScanNet		
Sem. α	Prune	Proto.	Direct	Figurines	Ramen	Teatime	Waldo	Mean	19	15	10
				56.33	49.98	70.69	63.67	60.17	35.38	38.45	46.90
✓				61.74	**52.31**	74.98	64.02	63.26	35.92	39.26	48.00
✓	✓			61.02	52.28	75.04	66.41	63.69	36.23	39.69	48.91
✓	✓	✓		**65.29**	51.97	**77.46**	**67.10**	**65.45**	**36.72**	**40.97**	**50.14**
✓	✓	✓	✓	**65.29**	51.97	**77.46**	**67.10**	**65.45**	**36.72**	**40.97**	**50.14**

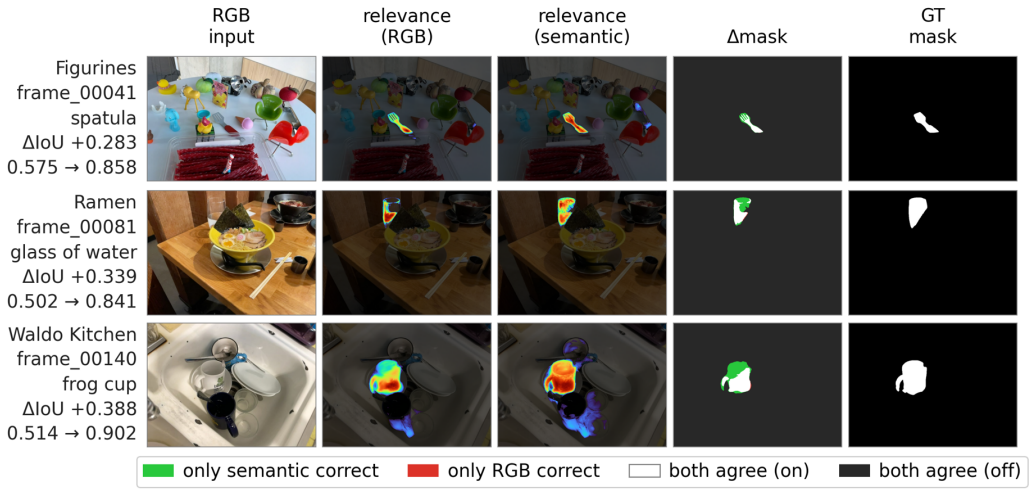

Figure 3: Green marks pixels predicted correctly only with semantic opacity, and red marks those predicted correctly only with RGB opacity. In the three queries shown, semantic opacity recovers more of the target regions and substantially improves IoU.

Semantic-opacity learning. Table 4 compares semantic-opacity optimization strategies on LERF.

Table 4: Ablation of semantic-opacity learning on LERF-OVS. Scene-level mIoU (%) is reported.

Continuous Optimization	Stage-wise Optimization	Entropy Regularization	Semantic Pruning	Figurines	Ramen	Teatime	Waldo	Avg.
✓				59.81	**52.66**	74.25	62.63	62.34
	✓			61.25	52.49	74.86	64.14	63.19
	✓	✓		**61.74**	52.31	74.98	64.02	63.26
	✓	✓	✓	61.02	52.28	**75.04**	**66.41**	**63.69**

Across the four LERF scenes, stage-wise optimization achieves higher average mIoU than continuous optimization. Entropy regularization further encourages the semantic opacity distribution toward low and high values, making low-contribution primitives more distinguishable from semantically important ones and thereby providing a clearer criterion for subsequent pruning. Building on this separation, semantic pruning removes 83.25% of the language primitives while segmentation performance remains stable and shows a slight improvement. With the same number of primitives retained in each scene, the setting based on semantic rendering significance maintains higher segmentation accuracy than the RGB rendering significance setting (63.69% versus 60.83% mIoU).

Prototype budget and storage. The prototype-budget comparison shows that a small codebook limits segmentation accuracy. As the codebook grows, accuracy gains gradually diminish while storage continues to increase. We therefore use a codebook budget that balances segmentation quality and storage.

Table 5 counts the language-branch payload: three prototype indices and one semantic opacity per retained primitive, plus one shared prototype codebook for each of the three levels. The per-primitive 512D language features are replaced by these indices. For $K = 256$, the per-primitive payload is $3 \times 1 + 4 = 7$ bytes, and the three codebooks contain $3 \times 256 \times 512$ float32 values. With a mean of $N = 215{,}357.5$ retained primitives per scene, the total storage is 2.94 MiB/scene.

Table 5: Sensitivity to the number of prototypes on LERF, reported in mIoU and language storage.

Number of prototypes K	Figurines	Ramen	Teatime	Waldo	Mean mIoU	Avg. language storage ↓
Raw 512D	61.02	52.28	75.04	66.41	63.69	1,261.9 MiB
32	34.64	49.45	47.07	54.60	46.44	**1.63 MiB**
64	55.28	**53.50**	70.52	**67.99**	61.82	1.81 MiB
128	62.79	52.66	74.89	67.39	64.43	2.19 MiB
256	**65.29**	51.97	**77.46**	67.10	**65.45**	2.94 MiB
512	64.10	51.64	76.23	65.73	64.43	5.05 MiB
1024	63.53	52.84	75.97	67.28	64.91	8.05 MiB

5.4 Query Efficiency

Table 6 compares the structural computation and measured latency of LeanSplat and LangSplatV2. Panel (b) compares the main structural query costs at a common resolution and number of texts.

Table 6: Query efficiency on LERF-OVS. (a) Steady-state single-query efficiency on A100. LangSplatV2 results are from the original paper. (b) Structural computation at 988×730 resolution with three semantic levels and five text prompts. Rasterization FLOPs are excluded. Channel counts describe intermediate output maps, not sparse coefficient updates. A dash indicates that the method does not perform the corresponding operation.

	(a) Measured efficiency		**(b) Structural computation**			
Method	Latency (ms) ↓	Throughput (q/s) ↑	Intermediate-map channels ↓	Dense 512D feature map	Dense reconstr. + pixel similarity ↓	Prototype–text LUT ↓
LangSplatV2	2.6	384.6	192	$3 \times 512 \times H \times W$	~ 153 GFLOPs	—
LeanSplat	**1.933**	**517.4**	**15**	—	—	~ 3.9 **MFLOPs**

Under the common setting in panel (b), LangSplatV2 rasterizes coefficient maps with 192 channels across three semantic levels, whereas LeanSplat directly rasterizes text-score maps with 15 channels in total. These channel counts describe the width of intermediate output maps, not the number of updates per primitive. LangSplatV2 uses sparse splatting and updates only four nonzero coefficients per primitive at each level. Its pixel-level high-dimensional feature reconstruction and text similarity cost approximately 153 GFLOPs. LeanSplat avoids these two computations, while constructing the prototype–text score table costs approximately 3.9 MFLOPs. These FLOP counts cover only the respective feature computations and do not represent total end-to-end query cost.

6 Conclusion

LeanSplat iteratively refines semantic visibility by alternating semantic-opacity optimization with multi-view language-feature re-aggregation. This signal guides the removal of low-contribution primitives and the construction of shared semantic prototypes, reducing both the number of language primitives and per-primitive feature storage. Querying computes and rasterizes text scores directly in prototype space without reconstructing the high-dimensional language feature field. On LERF, LeanSplat removes 83.25% of language primitives and achieves an average mIoU of 65.5%. With $K = 256$, average language storage is 2.94 MiB/scene. On LERF-OVS, query throughput exceeds 500 queries/s on an A100 GPU. These results show that the three sources of overhead can be reduced together while maintaining competitive semantic prediction quality.

AI USE STATEMENT

Generative AI tools were used solely to assist with writing and language polishing. All AI-assisted text was reviewed and revised by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Jaehun Bang, Jinhyeok Kim, Minji Kim, Seungheon Jeong, and Kyungdon Joo. LightSplat: Fast and memory-efficient open-vocabulary 3D scene understanding in five seconds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19812–19821, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Bang_LightSplat_Fast_and_Memory-Efficient_Open-Vocabulary_3D_Scene_Understanding_in_Five_CVPR_2026_paper.html.

Jiazhong Cen, Xudong Zhou, Jiemin Fang, Changsong Wen, Lingxi Xie, Xiaopeng Zhang, Wei Shen, and Qi Tian. Tackling view-dependent semantics in 3D language gaussian splatting. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2505.24746.

Jiahuan Cheng, Jan-Nico Zaech, Luc Van Gool, and Danda Pani Paudel. Occam's LGS: An efficient approach for language gaussian splatting. In *Proceedings of the British Machine Vision Conference*, 2025. URL https://bmvc2025.bmva.org/proceedings/694/.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Niessner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5828–5839, 2017. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Dai_ScanNet_Richly-Annotated_3D_CVPR_2017_paper.html.

Shaohui Dai, Yansong Qu, Zheyan Li, Xinyang Li, Shengchuan Zhang, and Liujuan Cao. Training-free hierarchical scene understanding for gaussian splatting with superpoint graphs. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 3673–3682, 2025.

Francis Engelmann, Fabian Manhardt, Michael Niemeyer, Keisuke Tateno, Marc Pollefeys, and Federico Tombari. OpenNeRF: Open set 3D neural scene segmentation with pixel-wise features and rendered novel views. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=SgjAojPKb3.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. LightGaussian: Unbounded 3D gaussian compression with 15x reduction and 200+ FPS. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2311.17245.

Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-Splat: Fast 3D gaussian splatting with sparse pixels and sparse primitives. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2412.00578.

Qiqi Hou, Randall Rauwendaal, Zifeng Li, Hoang Le, Farzad Farhadzadeh, Fatih Porikli, Alexei Bourd, and Amir Said. Sort-free gaussian splatting via weighted sum rendering. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.18931.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. URL https://pubmed.ncbi.nlm.nih.gov/21088323/.

Yuang Jia, Jinlong Wang, Junhong Lin, Ruiting Dai, and Wei Gao. CoSAG: Compact semantic anchor gaussians via training-free rate-distortion coding, 2026. URL https://arxiv.org/abs/2607.10237.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):139:1–139:14, 2023. doi: 10.1145/3592433. URL https://doi.org/10.1145/3592433.

Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. LERF: Language embedded radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19729–19739, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Kerr_LERF_Language_Embedded_Radiance_Fields_ICCV_2023_paper.html.

Jun-Seong Kim, GeonU Kim, Yu-Ji Kim, Yu-Chiang Frank Wang, Jaesung Choe, and Tae-Hyun Oh. Dr. Splat: Directly referring 3D gaussian splatting via direct language embedding registration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14137–14146, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Jun-Seong_Dr._Splat_Directly_Referring_3D_Gaussian_Splatting_via_Direct_Language_CVPR_2025_paper.html.

Hyunjoon Lee, Joonkyu Min, and Jaesik Park. CF3: Compact and fast 3D feature fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 27906–27916, 2025.

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3D gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://arxiv.org/abs/2311.13681.

Wanhua Li, Yujie Zhao, Minghan Qin, Yang Liu, Yuanhao Cai, Chuang Gan, and Hanspeter Pfister. LangSplatV2: High-dimensional 3D language gaussian splatting with 450+ FPS. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/feb14cd8a37e800f38edba497e847363-Abstract-Conference.html.

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

KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. CompGS: Smaller and faster gaussian splatting with vector quantization. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2311.18159.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3D gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://arxiv.org/abs/2401.02436.

Qucheng Peng, Benjamin Planche, Zhongpai Gao, Meng Zheng, Anwesa Choudhuri, Terrence Chen, Chen Chen, and Ziyan Wu. 3D vision-language gaussian splatting. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/98ed250b203d1ac6b24bbcf263e3d4a7-Abstract-Conference.html.

Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. LangSplat: 3D language gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20051–20060, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Qin_LangSplat_3D_Language_Gaussian_Splatting_CVPR_2024_paper.html.

Yansong Qu, Shaohui Dai, Xinyang Li, Jianghang Lin, Liujuan Cao, Shengchuan Zhang, and Rongrong Ji. GOI: Find 3D gaussians of interest with an optimizable open-vocabulary semantic-space hyperplane. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 5328–5337, 2024.

Jin-Chuan Shi, Miao Wang, Hao-Bin Duan, and Shao-Hua Guan. Language embedded 3D gaussians for open-vocabulary scene understanding. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5333–5343, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Shi_Language_Embedded_3D_Gaussians_for_Open-Vocabulary_Scene_Understanding_CVPR_2024_paper.html.

Lei Tian, Xiaomin Li, Liqian Ma, Hao Yin, Zirui Zheng, Hefei Huang, Taiqing Li, Huchuan Lu, and Xu Jia. CCL-LGS: Contrastive codebook learning for 3D language gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9855–9864, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Tian_CCL-LGS_Contrastive_Codebook_Learning_for_3D_Language_Gaussian_Splatting_ICCV_2025_paper.html.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pp. 6306–6315, 2017. URL https://mlanthology.org/neurips/2017/vandenoord2017neurips-neural/.

Sen Wang, Kunyi Li, Siyun Liang, Elena Alegret, Jing Ma, Nassir Navab, and Stefano Gasperini. Visibility-aware language aggregation for open-vocabulary segmentation in 3D gaussian splatting. In *Proceedings of the International Conference on 3D Vision*, pp. 1791–1800, 2026. doi: 10.1109/3DV69130.2026.00170. URL https://doi.org/10.1109/3DV69130.2026.00170.

Yanmin Wu, Jiarui Meng, Haijie Li, Chenming Wu, Yahao Shi, Xinhua Cheng, Chen Zhao, Haocheng Feng, Errui Ding, Jingdong Wang, and Jian Zhang. OpenGaussian: Towards point-level 3D gaussian-based open vocabulary understanding. In *Advances in Neural Information Processing Systems*, volume 37, pp. 19114–19138, 2024. URL https://arxiv.org/abs/2406.02058.

Yaxu Xie, Abdalla Arafa, Alireza Javanmardi, Christen Millerdurai, Jia Cheng Hu, Shaoxiang Wang, Alain Pagani, and Didier Stricker. ReLaGS: Relational language gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23826–23836, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Xie_ReLaGS_Relational_Language_Gaussian_Splatting_CVPR_2026_paper.html.

Shijie Zhou, Haoran Chang, Sicheng Jiang, Zhiwen Fan, Zehao Zhu, Dejia Xu, Pradyumna Chari, Suya You, Zhangyang Wang, and Achuta Kadambi. Feature 3DGS: Supercharging 3D gaussian splatting to enable distilled feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21676–21685, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Zhou_Feature_3DGS_Supercharging_3D_Gaussian_Splatting_to_Enable_Distilled_Feature_CVPR_2024_paper.html.

A Supplementary Material

A.1 Experimental Details

For LERF, semantic opacity is learned from the finest 2D mask level. ScanNet uses its only language-feature level. In the multi-level setting, each level aggregates its own 2D language features while sharing semantic opacity and the retained primitive set. The aggregated primitive features are L_2-normalized before semantic-opacity optimization. After pruning, only primitives in $\mathcal{I}_{\text{lang}}$ are rasterized. Semantic compositing weights and transmittance are recomputed over this set. Each level constructs its own prototype codebook and assignments.

Semantic-opacity training optimizes only the opacity logits with Adam at a fixed learning rate of 0.005, without learning-rate decay. At steps 3K and 6K, features are re-aggregated and the opacity parameters are reset. A new Adam optimizer is created for each stage, resetting its momentum state while keeping the learning rate at 0.005.

The feature L_2 term in Eq. 4 uses $\lambda_2 = 0.1$. The entropy weight uses the global training step t over $T = 9000$ iterations and increases linearly after step 6000:

$$\lambda_{\text{ent}}(t) = \begin{cases} 0, & t \le 6000, \\ 0.1(t - 6000)/3000, & 6000 < t \le 9000. \end{cases} \tag{A.1}$$

The entropy weight reaches 0.1 at step 9000. When stage-wise training resumes at step 6000, the schedule continues to use the global step rather than restarting.

Each LERF query uses one positive text and four fixed negative texts: `object`, `things`, `stuff`, and `texture`. Text scores are composited per pixel according to Eq. 11 before nonlinear relevancy is computed.

A.2 Ablation Study of the Iteration Budget for Continuous Semantic-Opacity Optimization

To test whether longer continuous training consistently improves performance, we compare checkpoints along the continuous schedule. Entropy regularization and pruning are disabled for all settings. Results are reported in Table 7.

Table 7: Scene-level mIoU (%) on LERF-OVS at continuous-training checkpoints.

Iter.	Figurines	Ramen	Teatime	Waldo	Avg.
9K	59.81	52.66	74.25	62.63	62.34
12K	**60.00**	52.70	74.02	**63.10**	62.46
15K	59.91	**52.77**	**74.07**	63.08	62.46

Mean mIoU remains nearly unchanged as training continues, while individual scenes show no consistent trend. Longer continuous optimization therefore does not yield reliable gains. We use a fixed 9K budget in the remaining experiments and adopt the stage-wise schedule under that budget (Table 4).

A.3 Semantic Opacity Analysis

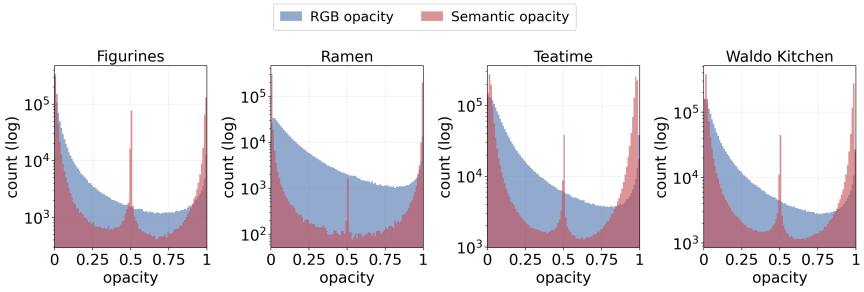

Figure 4: RGB and learned semantic opacity distributions for the same primitives on four LERF-OVS scenes. Blue denotes RGB opacity and red denotes semantic opacity. The vertical axis shows log counts. Counts are not normalized across panels, so the curves should be compared within each panel. The spike near 0.5 in semantic opacity comes from a small number of primitives that remain at their initialization and are never updated.

Figure 4 shows that learned semantic opacity is more concentrated near low and high values than RGB opacity. This change is not a uniform shift: some RGB-transparent primitives gain semantic opacity, whereas some RGB-visible primitives are suppressed. The primitive-wise rank correlation is also weak (Spearman $\rho = 0.167$). Together, these observations indicate that semantic opacity redistributes semantic contributions across primitives and cannot be explained by a simple monotone transformation of RGB opacity.

Effect of entropy regularization. A primitive is counted as near-binary when its semantic opacity is below 0.1 or above 0.9. In Figurines, Ramen, and Teatime, where paired distribution statistics are available, entropy regularization raises the near-binary fraction from roughly 16–32% to 83–97%. This shows that the entropy term pushes semantic opacity toward the extremes. The small mIoU change in Table 4 indicates that this distributional change does not produce a gain of comparable magnitude in segmentation accuracy.

A.4 Ablation Study on the Pruning Threshold

The effect of the rendering-importance threshold τ_{imp} on LERF-OVS is reported in Table 8.

Table 8: Effect of language-side pruning on LERF-OVS. The threshold ranges from no pruning to aggressive pruning. Per-scene and mean mIoU and the fraction of pruned language primitives (%) are reported. All values are in percent. The mean mIoU peaks at $\tau_{\text{imp}} = 0.1$ and drops at 0.2.

Setting	Figurines		Ramen		Teatime		Waldo		Mean	
	mIoU	Pruned	mIoU	Pruned	mIoU	Pruned	mIoU	Pruned	mIoU	Pruned[†]
None (baseline)	61.74	0.00	52.31	0.00	74.98	0.00	64.02	0.00	63.26	0.00
0.0001	61.99	57.27	52.54	47.60	75.04	20.83	64.01	30.00	63.40	33.57
0.001	**62.05**	67.14	52.57	51.45	**75.08**	28.10	64.06	39.10	63.44	41.53
0.01	61.98	78.70	52.60	65.76	74.92	52.61	64.40	62.63	63.48	62.17
0.1	61.02	89.20	52.28	78.03	75.04	80.49	**66.41**	84.86	**63.69**	83.25
0.2	60.19	92.19	**53.71**	82.49	74.79	85.78	62.80	89.35	62.87	87.76

[†] The overall pruning fraction is weighted by the number of pretrained primitives in each of the four scenes, rather than computed as an unweighted mean of scene-wise fractions. Before pruning, the pretrained RGB 3DGS contains $954{,}599$, $576{,}572$, $1{,}945{,}417$, and $1{,}665{,}673$ primitives in Figurines, Ramen, Teatime, and Waldo, respectively ($5{,}142{,}261$ in total, or $1{,}285{,}565.25$ per scene on average).

Pruning increases steadily with the threshold, while mean mIoU remains largely stable across a broad range. This suggests that performance is robust to threshold selection unless pruning becomes excessive. Pruning rates and per-scene accuracy changes vary, reflecting differences in scene-wise importance distributions. A noticeable decline occurs only when aggressive pruning removes primitives that still carry useful semantic evidence.

To test whether pretrained RGB visibility can replace learned semantic visibility for language-side pruning, primitives are ranked by RGB rendering significance. For each semantic pruning setting in Table 8, the number of retained primitives is matched exactly in each scene. Both strategies therefore use the same scene-wise primitive budget. Budget matching is needed because RGB rendering significance and the significance derived from semantic opacity have different numerical scales. Equal thresholds would not provide a comparable budget.

Table 9: Matched-budget comparison of RGB and semantic visibility on LERF-OVS. RGB pruning ranks primitives by RGB rendering significance. Each scene retains exactly the same number of primitives as in the corresponding semantic pruning setting in Table 8. Mean mIoU is averaged over four scenes. Overall pruning is computed as in Table 8. The unpruned row uses the full primitive set. Numeric matched settings are semantic pruning thresholds, not RGB thresholds.

Matched setting	Overall pruning (%)	RGB visibility mIoU	Semantic visibility mIoU
None (unpruned)	0.00	60.17	**63.26**
0.0001	33.57	60.30	**63.40**
0.001	41.53	60.35	**63.44**
0.01	62.17	61.13	**63.48**
0.1	83.25	60.83	**63.69**
0.2	87.76	58.83	**62.87**

The largely stable segmentation accuracy after substantial language-primitive removal indicates that many primitives in the pretrained set contribute little to the current semantic prediction. Pruning

by RGB rendering significance has limited effect on mIoU at moderate pruning levels, but causes a clearer decline as pruning becomes aggressive. By contrast, pruning by semantic rendering significance derived from semantic opacity maintains more stable segmentation accuracy under the same scene-wise primitive budgets. At an overall pruning ratio of 83.25%, the two settings achieve 60.83% and 63.69% mIoU, respectively.

A.5 Language-Storage Accounting

To compare language-branch storage, Table 10 separates each representation into the fraction of primitives rasterized at inference, the per-primitive payload, and scene-level shared data. All methods are accounted for at three semantic levels.

Table 10: Estimated language-branch storage for each method.

Method	Rasterized†	Payload / primitive	Scene-level constant	MiB/scene
LangSplat	100%	36 B (3 dims per level)	0.92 MiB (decoder Ψ)	45.1
LangSplatV2	100%	60 B (4 indices + 4 coefficients per level)	128 KiB (codebook)	73.7
Ours (unpruned 512D)	100%	6,144 B (512 dims per level)	—	7,532.6
Ours (pruned 512D)	16.8%	6,144 B (512 dims per level)	—	1,261.9
Ours (pruned, codebook)	16.8%	**7 B** (3 prototype indices + semantic opacity)	1.5 MiB (3 codebooks)	**2.94**

† Continuous features and model parameters use float32. Discrete indices use their actual integer storage width. All values are estimated from reported feature layouts or official implementations, not measured checkpoint sizes. Primitive fractions are relative to the pretrained RGB 3DGS.

For each setting, the MiB/scene value uses its mean primitive count across the four LERF scenes: this mean is multiplied by the per-primitive payload, then the scene-level constant is added. Shared multi-level data are counted once. Per-level data are summed.

LangSplat. Each semantic level stores three-dimensional latent features. The official implementation uses `encoder_dims` $= 256\,128\,64\,32\,3$ and `decoder_dims` $= 16\,32\,64\,128\,256\,256\,512$. At inference, one decoder is shared across the three levels. It contains $241{,}440$ parameters and occupies 0.92 MiB. The encoder is excluded from inference storage.

LangSplatV2. Each semantic level stores four sparse index–coefficient pairs, giving 60 B per primitive across all three levels. The levels share one codebook of size 64×512 ($K_v = 64$), which occupies 128 KiB per scene. Including the per-primitive payload, the total storage is 73.7 MiB/scene.

LeanSplat. Both 512D settings store per-primitive features at all three semantic levels. The prototype configuration uses one codebook per level, with $K = 256$ and $D = 512$, for 1.5 MiB across the three levels. Each retained primitive stores three prototype indices and one semantic opacity. At a pruning threshold of $\tau_{\text{imp}} = 0.1$, an average of approximately 215.4K primitives are retained across the four LERF scenes. The table uses this average for the pruned settings. The Raw setting stores per-primitive 512D features at all three levels without a codebook or indices.

A.6 Complexity Analysis of Direct Prototype-Space Retrieval

This section compares feature computation along the two query paths. Table 6 summarizes the results.

LangSplatV2 first produces a $K_v = 64$-channel coefficient map per semantic level through sparse splatting. It then reconstructs $D = 512$-dimensional language features at each pixel and computes their similarity to Q text embeddings. The reconstruction and similarity costs are

$$F_{\text{dec}} = 2LHWK_vD, \qquad F_{\text{sim}} = 2LHWDQ,$$

where L is the number of semantic levels and $H \times W$ is the image resolution. With $L = 3$, $Q = 5$, and $H \times W = 730 \times 988$, the two costs are approximately 141.8 and 11.1 GFLOPs, totaling 152.9 GFLOPs.

LEANSPLAT first computes the prototype–text similarity table $\mathbf{P} = \mathbf{C}T$. Its cost is

$$F_{\mathrm{LUT}} = 2LK_oDQ \approx 3.9\ \mathrm{MFLOPs},$$

where $K_o = 256$. This cost does not scale with image resolution. LEANSPLAT then uses prototype indices to obtain per-primitive text scores and produces $LQ = 15$ score-map channels. LangSplatV2 produces $LK_v = 192$ coefficient-map channels. Its sparse splatting updates only four nonzero coefficients per primitive at each level, or 12 across three levels. Thus, LEANSPLAT avoids per-pixel reconstruction of 512-dimensional features and the associated similarity computation. Nonlinear relevancy is still computed after pixel compositing.

These FLOP counts cover only feature reconstruction, pixel-level similarity, and prototype–text projection. They are not end-to-end query costs. Rasterization, index lookup, and nonlinear relevancy are excluded. At 988×730 resolution, explicitly materializing LangSplatV2's $3 \times 512 \times H \times W$ FP32 feature values would require approximately 4.43 GB. Kernel fusion can avoid writing this intermediate tensor, but it does not eliminate the reconstruction and similarity computation above.

A.7 LATENCY UNDER DIFFERENT PRUNING CONFIGURATIONS

Table 11 compares single-query efficiency across rendering-importance pruning thresholds.

Table 11: Single-query latency and throughput under different pruning configurations. Measurements are taken on an A100 PCIe 40 GB and an RTX 4090 and macro-averaged over 22 labeled views from four LERF-OVS scenes. Each query combines one positive text with four fixed negative texts. The five text-score channels are rasterized together, with 20 warm-up and 100 timed iterations recorded through CUDA events. Timing excludes text encoding, model loading, and file I/O.

		A100		RTX 4090	
Threshold τ_{imp}	Primitives	Latency (ms)↓	Throughput (q/s)↑	Latency (ms)↓	Throughput (q/s)↑
No pruning	5,142,261	4.550	219.8	2.609	383.3
0.0001	3,416,195	3.526	283.6	1.973	506.8
0.001	3,006,855	3.310	302.2	1.817	550.4
0.01	1,945,306	2.620	381.6	1.403	712.8
0.1	861,430	1.933	517.4	0.997	1003.4
0.2	629,535	**1.809**	**552.8**	**0.892**	**1121.2**

As the pruning threshold increases, fewer primitives participate in querying, and latency falls on both devices. At $\tau_{\mathrm{imp}} = 0.1$, queries are already substantially faster while the pruning-stage mean mIoU remains stable. Raising the threshold further reduces latency, but prediction quality begins to decline (Appendix A.4).

The gains from further pruning diminish as the retained set becomes smaller. Raising τ_{imp} from 0.1 to 0.2 removes about 27% of the remaining primitives, yet A100 latency falls by only about 6%. This indicates query costs that do not decrease in step with primitive count. Further latency reduction requires optimizing these operations. Batched-query results are reported in Appendix A.8.

A.8 BATCHED DIRECT PROTOTYPE-SPACE RETRIEVAL SCALING

Batch size B is the number of query groups processed together for the same view. Each group contains one positive text and four fixed negative texts. The groups share geometric projection, depth sorting, and semantic compositing weights. Table 12 compares query efficiency across batch sizes.

Table 12: Scaling behavior of batched direct prototype-space retrieval at $\tau_{\text{imp}} = 0.1$ on an A100 PCIe 40 GB and an RTX 4090. Each combination of scene, view, and batch size runs in a fresh process. The first run is discarded, followed by 20 warm-up and 100 timed iterations. Timing excludes text encoding, model loading, and file I/O. $B = 1$ still follows the batch path. Results for $B \leq 12$ are macro-averaged over 22 labeled views from four scenes. $B = 16$ covers only the 9 available views of Figurines and Waldo.

	A100			RTX 4090		
Groups B	Batch total (ms) ↓	Amortized (ms/query) ↓	Throughput (q/s) ↑	Batch total (ms) ↓	Amortized (ms/query) ↓	Throughput (q/s) ↑
1	2.244	2.244	445.6	1.179	1.179	848.2
2	2.970	1.485	673.5	1.284	0.642	1557.8
4	3.367	0.842	1188.0	1.870	0.467	2139.3
8	4.154	0.519	1925.8	2.340	0.293	3418.4
12	5.891	0.491	2037.0	3.357	0.280	3574.3
16[†]	6.331	0.396	2527.1	3.663	0.229	4368.3

[†] Figurines and Waldo only.

As batch size increases, total latency grows more slowly than the number of query groups, reducing amortized latency. At $B = 8$, latency per query is 0.519 ms on the A100 and 0.293 ms on the RTX 4090. Computation shared within a view can thus be amortized across query groups.

$B = 1$ still incurs the data organization and scheduling costs of the batch path, making it slower than the dedicated single-query path in Table 11. Outputs match sequential querying for $B \leq 8$. At $B = 12$ and $B = 16$, only small differences from floating-point accumulation order remain. The $B = 16$ row covers two scenes and is not directly comparable with the four-scene results.

A.9 LIMITATIONS

Scene-dependent pruning threshold. Scenes respond differently to pruning. An overly high threshold may remove primitives that still carry semantic evidence. This can weaken semantic coverage around object boundaries and reduce prediction quality. Adapting the threshold to each scene may better balance compression and prediction quality.

Uniform prototype budget. A fixed number of prototypes facilitates comparison, but may leave excess capacity in simpler scenes and constrain more complex ones. Allocating the prototype budget per scene is a direction for future work.

Training and evaluation scope. LEANSPLAT still requires language-branch optimization for each scene. Storage and query efficiency remain to be evaluated across more scenes and hardware configurations.